INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Ein Konzept für CORBA-basiertes Management by Delegation anhand typischer WWW-Managementszenarien

Albert Euba

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Maria-Athina Mountzia
	Alexander Keller

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Ein Konzept für CORBA-basiertes Management by Delegation anhand typischer WWW-Managementszenarien

Albert Euba

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Maria-Athina Mountzia
	Alexander Keller
Abgabedatum:	15. Mai 1997

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

München, den 15. Mai 1997

....

(Unterschrift des Kandidaten)

Zusammenfassung

Das Paradigma des *Management by Delegation* verfolgt durch die Flexibilisierung "traditioneller" Manager-Agent-Strukturen im Managementbereich, die effiziente Realisierung von Managementaufgaben unter Berücksichtigung des Gesamtkontexts. Die *Common Object Request Broker Architecture* CORBA, scheint als Basis eines solchen Konzepts ideal geeignet, da sie für die Interoperabilität von Anwendungen in verteilter und sogar völlig heterogener Umgebung konzipiert wurde.
Andererseits entstehen mit der zunehmenden Nutzung des *World Wide Web* vermehrt Wünsche nach umfassendem Management von WWW-Servern. Die Bemühung, unter CORBA ein *Management by Delegation*-Konzept am Beispiel von WWW-Server-Management zu realisieren, steht im Mittelpunkt dieser Diplomarbeit.

Dazu werden zu Beginn die notwendigen Grundlagen wie MbD und CORBA kurz erläutert. Anschließend erfolgt eine Top-Down-Betrachtung zu den Anforderungen des Managements von Webservern. Danach erfolgt während der Bottom-Up-Analyse zu Realisierbarkeit eines solchen Vorhabens über Logfiles, Konfigurationsdateien etc., die Zusammenführung der beiden Sichtweisen.
Die gefundenen Faktoren werden im Anschluß an die Untersuchungen mit Hilfe eines Entwicklungswerkzeugs in ein Objektmodell übertragen, das in mehreren Schritten verfeinert wird. Zum Abschluß ist eine prototypische Implementierung des entwickelten Konzepts unter Verwendung der Programmiersprache Java vorgesehen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Das World-Wide-Web (WWW) boomt. Immer häufiger stößt man in Werbeanzeigen, Zeitungsartikeln, sowie Rundfunk und Fernsehen auf Adressen (URLs) für Seiten im WWW. Mehr und mehr kommerzielle Anbieter entdecken das Web als Medium zur Verbreitung von Produktinformationen und Werbung, Service und Dokumentation und sogar zum Vertrieb.
Ein paar Seiten im WWW zu präsentieren ist durch die reichlich vorhandene, oftmals sogar kostenlos verfügbare Software zu diesem Zweck und den umfassenden Angeboten von Service-Providern nicht weiter schwierig. Wird die Anzahl von Dokumenten allerdings größer, übersteigt der Verwaltungsaufwand durch die entstehende Komplexität bald das vertretbare Maß. Geeignete Managementsysteme, die den entstehenden Aufwand effektiv reduzieren und die Verwaltung wesentlich vereinfachen, um sie nicht nur Experten zugänglich zu machen, existieren dagegen noch nicht im gewünschten Umfang.

Doch nicht nur spezielle Managementanwendungen, wie in diesem Falle das WWW-Management, sondern auch das Netz- und Systemmanagement in seiner Gesamtheit, befindet sich noch in der Entwicklung. Neugewonnene Erkenntnisse und variierende Problemstellungen führen zu neuen Konzepten und Lösungen.
So wird beispielsweise seit einiger Zeit mit dem Paradigma [YGY91] des *Management by Delegation* (MbD) eine Flexibilisierung der bisher üblichen Manager - Agent Strategien verfolgt. Das Ziel ist dabei eine Realisierung der Managementfunktionalität dort, wo sie unter Berücksichtigung des gesamten Umfeldes am effizientesten erbracht wird.

Gleichzeitig etablierte sich mit der *Common Object Request Broker Architecture* (CORBA) ein objektorientierter Standard für verteilte Anwendungsentwicklung in heterogenen Umgebungen. Aufgrund dieser Eigenschaften könnte diese Architektur als Grundlage für die Implementierung von MbD-Konzepten im Allgemeinen und eines auf dem MbD-Paradigma basierenden Managementsystems für WWW-Server im Besonderen dienen.

1.2 Aufgabenstellung

„*Ein Konzept für CORBA-basiertes Management by Delegation anhand typischer WWW - Managementszenarien*"; so lautet der genaue Titel dieser Diplomarbeit. Doch was verbirgt sich eigentlich dahinter, wie ist diese Aufgabenstellung genau zu verstehen?

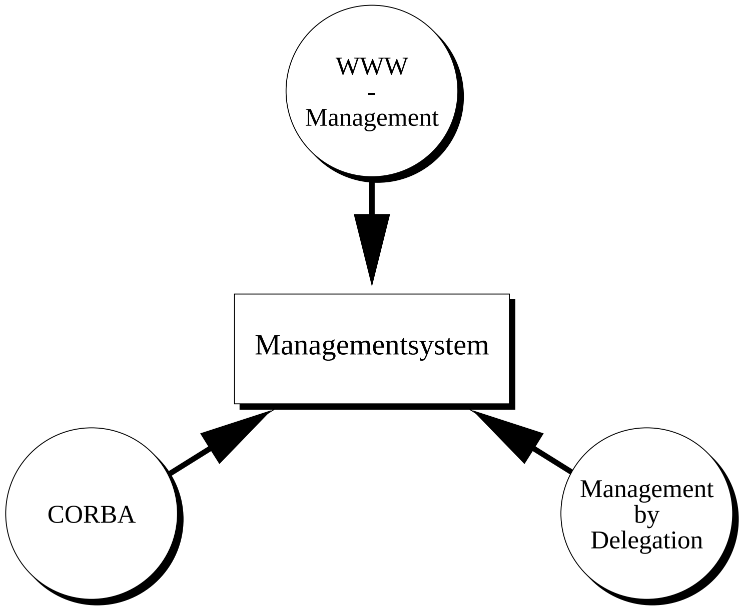

Abbildung 1.1: Die Einflußfaktoren der Aufgabenstellung

Das Ziel ist ein objektorientierter Entwurf und die anschließende prototypische Implementierung eines Konzepts für das Management von WWW-Servern. WWW-Clients, Proxys und Cached-Proxys liegen nicht im Fokus der Aufgabenstellung.
Die Anforderungen dafür sollen anhand typischer WWW-Managementszenarien ermittelt werden. Nach entsprechenden Untersuchungen zur Umsetzung dieser Anforderungen, wird daraus delegierbare Funktionalität abgeleitet, um ein Managementkonzept nach dem MbD-Paradigma, auf der Basis eines CORBA-konformen Object Request Brokers zu Realisieren.
Die Implementierung erfolgt in *Java*, um heterogene Umgebungen durch die Kombination von ortsunabhängigen CORBA-Schnittstellen mit plattformunabhängigem Programmcode bestmöglich zu unterstützen.

1.3 Gliederung

Zu Beginn wird in Kapitel 2 nach einem einführenden Beispiel der Internet-Draft über den Entwurf einer WWW-MIB behandelt und anschließend die theoretischen Grundlagen der Objektmodellierung, sowie für CORBA und Management by Delegation erläutert.

Danach befaßt sich Kapitel 3 mit den Anforderungen, die sich aus dem Wunsch nach dem Management von Webservern ergeben, also gewissermaßen eine Top-Down Betrachtung der Problematik. Diese wird im darauffolgenden Kapitel 4 mit einer Bottom-Up Analyse über die Realisierungsmöglichkeiten der gestellten Anforderungen zusammengeführt.
Aus dem ermittelten Ergebnis ergibt sich schließlich der Ansatz für ein Objektmodell. Das Design dieses Objektmodells wird in Kapitel 5 vom ersten Entwurf stufenweise bis zur endgültigen Version vollzogen. Eine CORBA-basierte Implementierung unter Verwendung von Java als Programmiersprache wird dann in Kapitel 6 behandelt.

Im Abschließenden Kapitel 7 werden die Ergebnisse der vorliegenden Arbeit Zusammengefaßt und ein kurzer Ausblick auf weitere interessante Aspekte im Zusammenhang mit der bearbeiteten Thematik gegeben.

Abbildung 1.2 veranschaulicht den Ablauf der einzelnen Schritte in den zentralen Kapiteln graphisch:

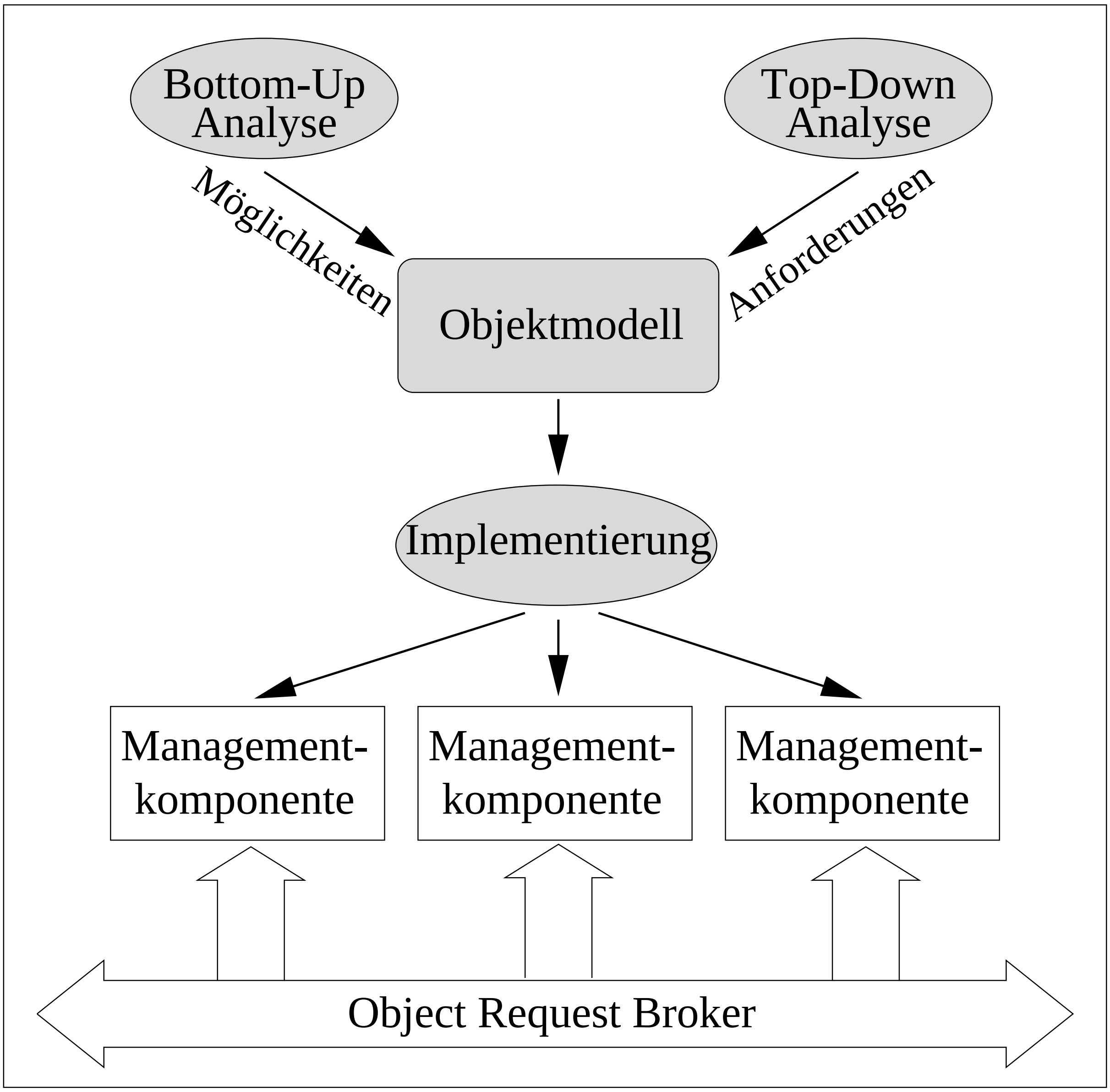

Abbildung 1.2: Die einzelnen Schritte in dieser Diplomarbeit

Kapitel 2

Grundlagen

2.1 Die konkrete Problemstellung am Beispiel

Zur Veranschaulichung der Problematik sei hier ein konkretes Beispiel aus dem Alltag erläutert:

Um im WWW als Anbieter aufzutreten, gibt es üblicherweise zwei Möglichkeiten: Über eine Standleitung angeschlossen an einen Internet-Provider in Eigenregie einen WWW-Server zu betreiben, oder diesen vom Provider auf dessen Rechnern betreiben zu lassen (Web-Space vs. Disk-Space).
Um den Aufwand in einem überschaubaren Rahmen zu halten, gehen viele – vor allem kleinere – Firmen den zweiteren Weg. Ein solcher Provider betreibt demzufolge eine größere Anzahl von Servern mit den Seiten und Diensten seiner Kunden. Dazu sind unter Umständen je nach Anforderungen verschiedene Softwarelösungen auf unterschiedlicher Hardware im Einsatz, es kann sich also um eine völlig heterogene Umgebung handeln.

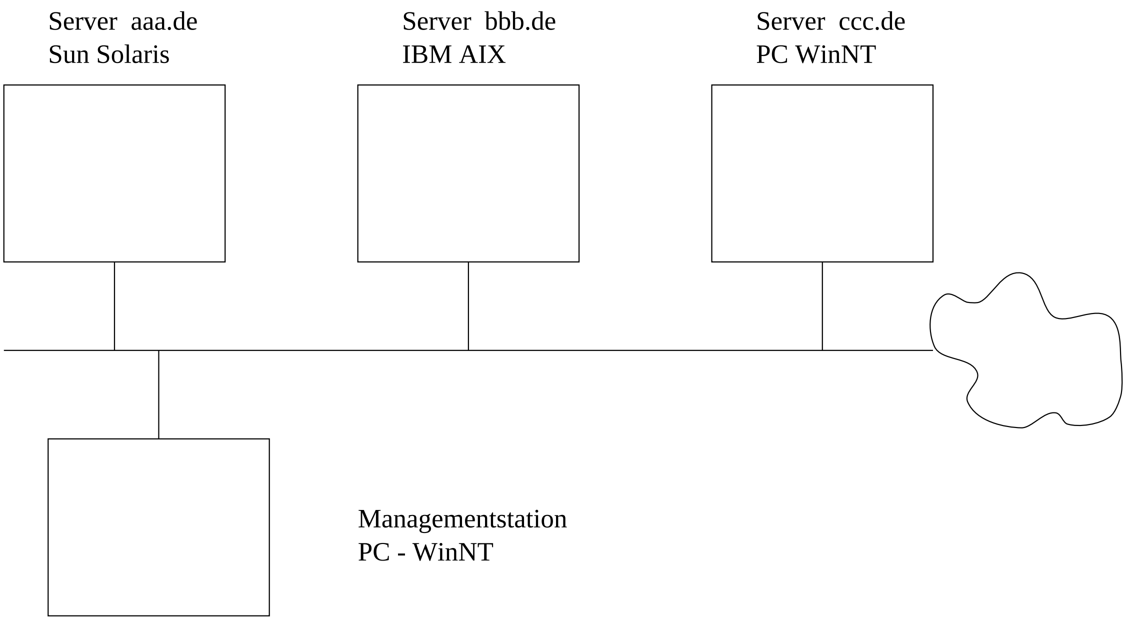

Abbildung 2.1: Heterogene Umgebung eines WWW-Service-Providers

Auf welche Probleme stößt ein solcher Provider nun während des Betriebs im allgemeinen, und Angesichts der Heterogenität im Besonderen?

Zuallererst ist zu gewährleisten, daß er den Überblick über das gesamte Angebot und vor allem den Linkstrukturen zwischen den Dokumenten behält, denn schließlich ist es ziemlich unangenehm, wenn das äußere Erscheinungsbild durch hoffnungslos veraltete Informationen, oder Links auf längst nicht mehr existente oder verschobene Dokumente getrübt wird.
Interessant dürfte für ihn auch die aufgetretene Nutzung des Servers sein. Wieviele Anfragen welcher Größe werden am Tag erledigt und wer stellt sie von woher? Die übertragenen Datenmengen sind evtl. für Abrechnungen erforderlich und hohe Zugriffsraten sind ein willkommener Werbefaktor für Neukunden – wie auch eine hervorragende Dienstgüte, die es ebenfalls zu kontrollieren gilt.
Außerdem muß er auch die Belastungen der einzelnen Server kennen, um die Performance zu optimieren und adäquate Hardware einsetzen zu können. Und für den Fall, daß etwas nicht so funktioniert wie es soll, steht er vor der Schwierigkeit, solche Fehler möglichst schnell zu erkennen und beheben.

Diese Komponenten sind typische Beispiele für Managementanwendungen. Sie in einem heterogenen Umfeld zu realisieren, ist allerdings keine triviale Aufgabe. Unterschiedliche Formate von benötigten Informationen, verschiedenste Werkzeuge, die nur für eine bestimmte Umgebung vorhanden sind und die Verteiltheit der Problemstellung stellen nur einige der typischen Hürden dar, die einem Zusammenspiel des Ganzen im Wege stehen. Interoperabilität und Plattformunabhängigkeit von geeigneten Managementanwendungen wären notwendig, um diese Thematik zu beherrschen.

2.2 Der WWW-MIB-Draft der IETF

Auf dem Weg zur Realisierung eines Softwaresystems, ist der erste Schritt meist die Suche nach vorhandenen Lösungen zu diesem oder ähnlichen Problemen. Im Internet-Draft *Definitions of Managed Objects for WWW Servers* [CWK96] versucht gegenwärtig eine Arbeitsgruppe der IETF (Internet Engineering Task Force) geeignete Managed Objects für WWW Server zu definieren. Die zum Bearbeitungspunkt vorliegende Version dieses Drafts ist vom März 1997 und kann gegenüber der endgültigen Fassung noch beliebig Veränderungen erfahren.
Daraus konnten wertvolle Anregungen gewonnen werden. Unter anderem sind einige Managed Objects daraus direkt, oder in abgewandelter Form in dieser Arbeit verwendet worden. Diese Managementinformationen sind im Kapitel 3 gekennzeichnet, ihre Zahl hält sich jedoch in Grenzen.

Während der Fokus dieser Arbeit auf dem Management von WWW-Servern liegt, existiert bei der WWW-MIB diese Einschränkung nicht. Darin sind Clients ebenso berücksichtigt. Bei Proxys sowie Cached Proxys ist dies zwar beabsichtigt, aber die entsprechenden Punkte wurden noch nicht bearbeitet. Zusätzlich ist die Ausführung so allgemein gehalten, daß auch andere Transferprotokolle wie z.B. FTP unterstützt werden können.

Besondere Rücksicht nimmt der Draft auch auf weitere relevante MIBs. Diese sind die NSM-MIB [KF94], sowie die SYSAPPL- [KS97] und APPL-MIB [KKPS97]. Die WWW-MIB ist derart gestaltet, daß sich sich in den jeweiligen Framework einfügt und diesem um WWW-spezifische Gesichtspunkte erweitert. Da diese Teile nur für ein SNMP-Management benötigten werden, sind sie für die vorliegende Arbeit bedeutungslos.

Der verwertbare Anteil gliedert sich in folgende Teile auf:

- `Systems group`
  Hier befinden sich die allgemeinen Angaben wie Administratorname, oder unterstützt Protokolle zu allen auf einem Rechner laufenden WWW-Komponenten.
- `Statistic group`
  Als Statistik werden die Informationen über den zu einer Komponente gehörenden Datenverkehr bezeichnet. Alle ein- und ausgehenden Anfragen und Antworten werden hier erfaßt.
- `Document group`
  Wie der Name vermuten läßt, finden sich hier Daten zu den übertragenen Dokumenten. Auch Filtermöglichkeiten zur Auswahl, welche Dokumente erfaßt werden, sind hier vorgesehen, um nicht unnötig viel Daten zu sammeln.
- `Error group`
  Diese Gruppe wurde in der vorliegenden Version noch nicht ausgearbeitet.

2.3 Objektorientierte Modellierung

Um bei größeren Systemen zu einem lauffähigen Programm zu gelangen, reicht es jedoch nicht aus nur die Thematik zu erfassen. Vor der Implementierung steht der Entwurf des Systems. Dazu gibt es aber eine Vielzahl von Hilfsmitteln. Verwendet wurden dabei zum einen die *Object Modeling Technique*, ein Vorgehensmodell zur objektorientierten Softwareentwicklung und das Entwicklungswerkzeug *Software through Pictures*, das dieses Vorgehensmodell optimal unterstützt.

2.3.1 Die Object Modeling Technique

Die *Object Modeling Technique* (OMT) ist wie bereits erwähnt ein Vorgehensmodell zur objektorientierten Softwareentwicklung, die sich dabei einer sprachunabhängigen graphischen Notation[1] bedient. Sie wurde 1991 von James Rumbaugh in [Rum91] eingeführt und hat seither große Verbreitung gefunden.

OMT besteht aus drei zusammengehörigen, aber unterschiedlichen Modellen, die die entscheidenden Aspekte eines System betreffen: Das Objektmodell repräsentiert die statischen Bestandteile, nämlich Klassendefinitionen, Beziehungen dazwischen und ähnliches. Es wird verwendet, um die Struktur der Objekte in einem System mit graphischer Unterstützung klarzulegen. Auf eben diesen Strukturen operieren die beiden anderen, das Dynamik- und Funktionsmodell. Sie beschreiben Kontrollstrukturen und Ablaufpläne, sowie die Interaktionen und funktionalen Abhängigkeiten zwischen Objekten.
Die zuletztgenannten spielen in dieser Arbeit nahezu keine Rolle. Die mit Abstand häufigsten Aktivitäten, die durchgeführt werden sind einfaches Abfragen und Setzen von Attributen nach Aufforderung. Interaktionen zwischen einzelnen Objekten sind eher trivial und beschränken sich hauptsächlich auf das Übermitteln von Daten. Eine nähere Betrachtung erscheint dabei wenig

[1] Eine kurze Übersicht der graphischen Notation findet sich im Anhang B

gewinnbringend. Aus diesen Gründen ist hier das Objektmodell das weitaus wichtigste, es erscheint ausreichend, um das gewünschte Managementsystem damit zu entwerfen.

Ein solches Objektmodell besteht im Wesentlichen aus Klassen und Beziehungen zwischen ihnen. Klassen bestehen aus Attributen und Methoden. Beziehungen können neben allgemeinen (Klasse *Sender* `sendet_Nachricht_an` Klasse *Empfänger*), auch die Spezialfälle Vererbung (Klasse *PKW* ist Subklasse von *Fahrzeug*), oder Enthaltensein (Klasse *Lenkrad* ist enthalten in *PKW*) darstellen, oder ganze Assoziationsklassen besitzen (Klasse *Bestellung* beschreibt die Beziehung `bestellt_Ware` zwischen *Kunde* und *Verkäufer*).
Aus einem Objektmodell können direkt die Klassendefinitionen für reale Programmiersprachen gewonnen werden. Dies wird bei entsprechenden Unterstützungwerkzeugen, wie dem im nächsten Abschnitt besprochenen *Software through Pictures* auch angeboten. Nicht zuletzt daraus begründet sich die Dominanz dieses Modells gegenüber den zwei Übrigen.

Zur konkreten Entwicklung eines Systems sieht OMT drei Phasen vor:

1. **Analyse**
   Hier wird die Problemstellung analysiert. Dabei werden alle relevanten Faktoren bestimmt, die für ein einfaches Objektmodell von Bedeutung sind.

2. **Design**
   Die Schrittweise Verfeinerung des ersten, aus der Analysephase gewonnenen Objektmodells, bis hin zur endgültigen Version.

3. **Implementierung**
   Übertragung des Objektdesigns in die gewählte Programmiersprache.

Der Verlauf dieser drei Phasen spiegelt sich exakt in der vorliegenden Arbeit wieder. Die Analyse erfolgt in den Kapiteln 3 und 4, in denen die Top-Down- und Bottom-Up-Analysen des WWW-Managements durchgeführt werden. Der Werdegang des Objektmodells, sowie die Implementierung folgt in den beiden nächsten Kapiteln.

2.3.2 Software through Pictures

Das bereits mehrmals genannte Softwarepakte *Software through Pictures* der Firma Interactive Development Environments unterstützt also, wie eingangs erwähnt, diese objektorientierte Modellierung nach der OMT-Methode [tP96b], vom ersten Entwurf bis hin zur Erzeugung der Klassendefinitionen, die damit für verschiedene objektorientiert Sprachen generiert werden können [tP96c]. Beispiele dafür sind C++, Smalltalk und Ada. Aber auch die Ausgabe von IDL-Schnittstellen zur Verwendung für CORBA wird angeboten. Dadurch erst ist StP für die Aufgabenstellung dieser Diplomarbeit geeignet.

Das StP-Paket besteht aus mehreren Komponenten, die zur Bearbeitung der Aufgabe nicht alle erfordelich sind. Im folgenden sind daher nur diejenigen erläutert, die in diesem Rahmen zur Anwendung gelangen.

Der StP Desktop

Er erscheint nach dem Programmaufruf und ist die Schaltzentrale des Softwarepaktets.
Von ihm aus werden alle weiteren Werkzeuge aktiviert, auch die später behandelten Objektmodelleditor und Klassentabelleneditor. Dazu wird das Icon *Model* geöffnet und der benötigten

Modelltyp ausgewählt. Im Menü *Commands* steht dann der Punkt *Start Editor* zur Auswahl, der wie der Name vermuten läßt, den entsprechenden Editor startet.

Auch die Codeerzeugung wird vom Desktop aus eingeleitet. Dazu muß aus dem Feld *Objects Model Diagrams* ein Modell ausgewählt werden. Anschließend wird aus dem Menü *Commands* die Codeerzeugung angestoßen.

Da mit StP mehrere Projekte gleichzeitig bearbeitet werden können, ist es notwendig nach dem Neustart das gewünschte Projekt im Menüpunkt File-Set Project/System zu spezifizieren. Daneben werden auch mehrere Benutzer unterstützt. Aus diesem Grund verweigert das System beim Editieren eines Modells das öffnen der zum gleichen Zeitpunkt bereits geöffneten Dateien. So wird ein gegenseitiges Überschreiben verhindert.

Der Object Model Editor

Er ist der hauptsächlich verwendete Editor. Mit ihm können die Objektmodelle erzeugt und verändert werden. Dazu stehen alle graphischen Möglichkeiten von OMT zur Verfügung. Eine ausführliche Beschreibung aller Komponenten würde hier allerdings zu weit führen. Das Handbuch [tP96d] ist dazu besser geeignet und ist auch als Onlinehilfe vorhanden.

Der Class Table Editor

Dieser ist zur Ergänzung des Object Model Editors gedacht. Hier können die Attribute und Operationen für einzelne Klassen definiert werden.
Außerdem besteht die Möglichkeit spezifische Einstellungen für bestimmte Programmiersprachen vorzunehmen, die nicht Teil des Objektmodells sind, sondern lediglich zur effizienteren Codeerzeugung benötigt werden. Ein Beispiel dazu wäre die Einstellung von `readonly` bei einer geplanten Ausgabe in IDL, die ansonsten nachträglich von Hand eingefügt werden müßte.

2.4 CORBA

Mit der *Common Object Request Broker Architecture* (CORBA) hat die OMG (Object Management Group) einen objektorientierten Standard für verteilte und heterogene Anwendungen festgelegt. In solchen Umgebungen die einfache und zuverlässige Zusammenarbeit von Applikationen zu gewährleisten, war dabei die Zielsetzung.

2.4.1 Die Object Management Architecture (OMA)

CORBA ermöglicht die Kooperation von Anwendungen, die in unterschiedlichen Programmiersprachen erstellt sind und auf verschiedensten Hardware- Betriebssystem- und und Netzwerkumgebungen laufen können. Das Kernstück der in Abb 2.2 skizzierten Object Management Architecture ist dabei der sogenannte Object Request Broker (ORB).
Über den ORB kommunizieren Client-Objekte mit Server-Objekten, deren physikalische Position und konkrete Implementierung für den Client völlig transparent ist. Um dies zu ermöglichen müssen lediglich die Schnittstellen solcher Server-Objekte in einer einheitlichen Sprache eindeutig definiert und dem ORB bekannt sein. Zu diesem Zweck wurde die sogenannte *Interface Definition Language* (IDL) etabliert. IDL ist objektorientiert und unabhängig von einer bestimmten Programmiersprache.

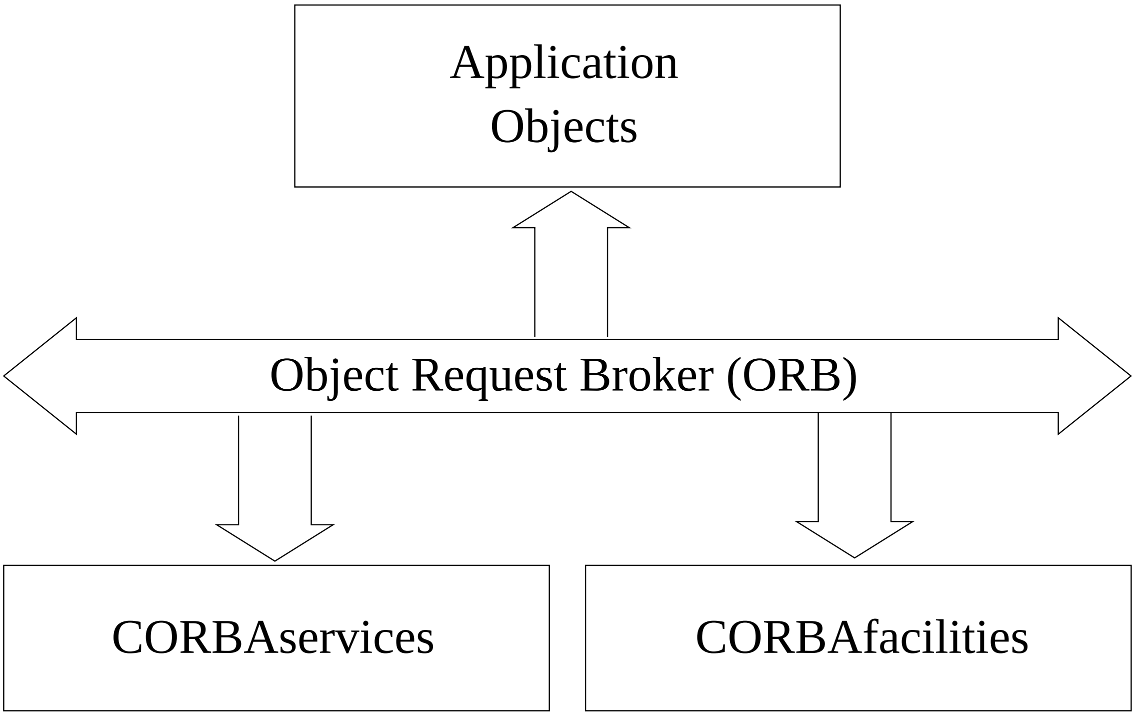

Abbildung 2.2: Die Object Management Architecture

Die CORBAservices stellen die grundlegenden Funktionen zur Verfügung, die für eine verteilte Programmierung auf Basis der OMA erforderlich sind. Ein Beispiel dafür wäre der *Naming Service*, der die eindeutige Benennung der Objekte gestattet, d.h. über die Bezeichnung ein Zielobjekt lokalisiert. Die Bereitstellung der genau festgelegten CORBAservices wird von jedem OMG-konformen ORB erwartet.
Die CORBAfacilities dagegen kann man eher im Sinne einer Klassenbibliothek betrachten. Sie stellen weiterführende Dienste zur Verfügung. Dabei muß zwischen vertikalen Facilities, die spezielle Funktionen für einen Bereich (z.B. Textverarbeitung) enthalten und horizontalen Facilities, die häufig verwendete Funktionen für verschiedene Bereiche anbieten, unterschieden werden. Die Bereitstellung von CORBAfacilities ist im Gegensatz zu den Services allerdings nicht zwangsweise gewährleistet.
Application Objects stellen schließlich die eigentlichen Anwendungen dar. Sie können ihrerseits weitergehende Dienste anbieten und andere benutzen.

2.4.2 Anwendungsentwicklung unter CORBA

Bei der Entwicklung von Anwendungen werden die Schnittstellendefinitionen also in IDL erstellt und anschließend von einem Compiler auf standardisierte Weise in die gewünschte Zielsprache übertragen. Im Moment sind solche IDL-Abbildungen (Mappings) für die Sprachen C, C++ und Smalltalk festgelegt. Für Java und einige andere Sprachen laufen gegenwärtig Standardisierungsverfahren. Bei dieser Übersetzung entstehen in der gewählten Sprache ein Stub Interface und das Skeleton (siehe auch Abb. 2.3) Das Stub Interface wird für die Entwicklung von Clients verwendet. Die Umsetzung der Zugriffe auf die Server-Objekte mit Hilfe des Stub Interface ist privat und je nach ORB verschieden. Das Skeleton gibt das Grundgerüst (daher der Name) für die Implementierung vor, die die Server-Objekte realisiert.

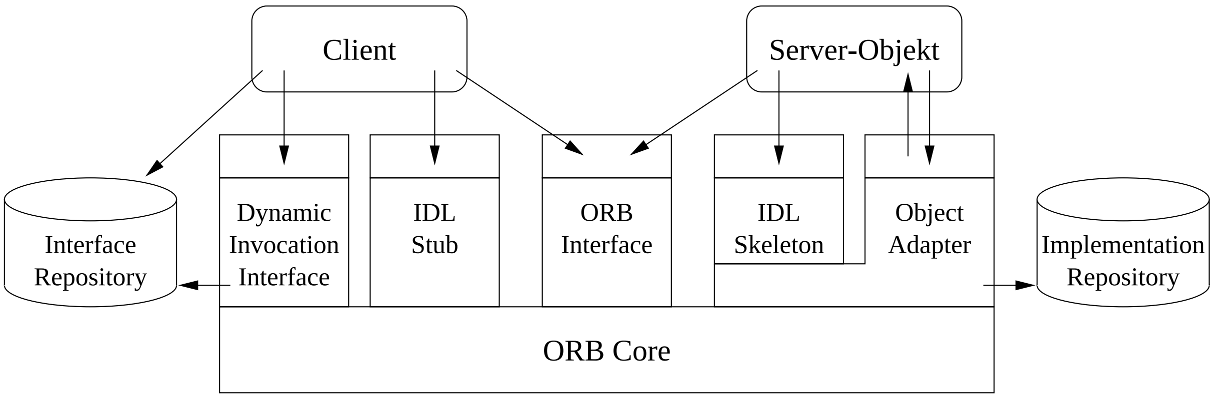

Abbildung 2.3: Aufbau eines ORB

Im ORB Core findet die eigentliche Kommunikation zwischen den Objekten statt (Parameterübergabe über das Netz, etc.), das ORB Interface stellt eine direkte Schnittstelle zum ORB Core dar, die bei allen ORB's identisch ist.
Das Dynamic Invocation Interface ermöglicht mit Hilfe des Interface Repository das Erfragen und Verwenden einer Objektschnittstelle zur Laufzeit, ohne daß diese zur Übersetzungszeit bereits bekannt war. Dadurch wird eine dynamische Aufrufschnittstelle realisiert. Seitens des Server-Objekts kann die Erzeugung eines solchen Aufrufs von einem direkten Aufruf über den IDL Stub nicht unterschieden werden.
Der Object Adapter stellt die Verbindung zwischen dem Objekt und dem ORB her. Er fügt die verschiedenen ORB-Implementierungen mit den standardisierten Skeletons zusammen, kann aber auch vom Objekt direkt verwendet werden.

2.4.3 OrbixWeb

Als Implementierungssprache ist in dieser Arbeit Java vorgesehen, da die Interoperabilität des CORBA-Konzepts durch die Portabilität einer plattformunabhängigen Sprache optimal ergänzt wird. Wie oben bereits erwähnt, ist gegenwärtig eine Anbindung für Java noch nicht standardisiert. Es existieren jedoch bereits einige proprietäre Lösungen zu diesem Problem.

NEO ist ein von Sun Microsystems entwickelter ORB für das Betriebssystem Solaris, der die CORBA-Programmierung in C++ unterstützt. Auf diesen ORB setzt die von der gleichen Firma frei erhältliche Erweiterung JOE auf. JOE ermöglicht eine CORBA-konforme Integration von Anwendungen, die in Java erstellt sind. Zu Beginn dieser Arbeit war die Verwendung von NEO und JOE als Entwicklungsplattform geplant. Nach genauerer Evaluierung erwies sich jedoch diese Kombination entgegen anderslautender Zusicherungen seitens Sun, nur für Clientseitige Entwicklungen unter Java als geeignet. Serverseitig existierte dafür – sofern überhaupt mit dem erworbenen Produkt durchführbar – keine dokumentierte Anbindung, was eine andere Produktwahl erforderlich machte.
Die Entscheidung fiel dabei auf OrbixWeb von IONA, das sowohl Client-, als auch Serverentwicklung unter Java zuläßt. OrbixWeb steht für Evaluierungszwecke zum kostenlosen Download

zur Verfügung und bietet eine hervorragende Onlinedokumentation.

Anwendungsentwicklung mit OrbixWeb

Bei einem Managementsystem entsprechen die Agenten, die die Informationen sammeln und verwalten (bei MbD auch selbst weiterverarbeiten) den Server-Objekten, die ihre Dienste den Clients, also Managern, bzw. anderen Servern, anbieten. Wie erläutert muß zunächst eine Schnittstelle für eine Klasse eines Server-Objekts entworfen werden. Diese trägt im folgenden den Namen `Agent.idl`. Nach dem Übersetzen mit dem zu OrbixWeb gehörenden IDL-Compiler sind folgende Dateien entstanden:

```
_AgentRef.java, Agent.java, _AgentHolder.java,
_boaimpl_Agent.java, _tie_Agent.java, _AgentOperations.java,
_dispatcher_Agent.java
```

Welche Bedeutung haben nun diese `.java`-Dateien, beziehungsweise die darin enthaltenen Klassen [ION97] und wie funktioniert das Zusammenspiel dazwischen?

Zur Clientseite gehören folgende Dateien, die keiner weiteren Bearbeitung bedürfen:

- `_AgentRef`
  Dies ist die Java Interface Definition für die Schnittstellen zum Server-Objekt, also der IDL Stub.
- `Agent`
  Hier sind die im obigen Interface definierten Methoden implementiert, sie werden bei der Übersetzung automatisch generiert. Der Zweck dieser Klasse ist die Verbindung des Clients mit dem ORB, sie stellt die private Schnittstelle vom Stub zum ORB dar.
- `_AgentHolder`
  Diese Datei beinhaltet ebenfalls automatisch generierten Code, der nur dazu dient, die inout/out Parameter der IDL-Definitionen zu realisieren, da eine direkte Umsetzung dieses Konzeptes zur Parameterübergabe für Java nicht möglich ist.

Zur Serverseite gehört folgendes:

- `_AgentOperations`
  Darin befindet sich die Java Interface Definition, die aus dem ursprünglichen IDL-File entstanden ist – das IDL Skeleton. Dieses Interface gilt es zu implementieren, um das Server-Objekt zu realisieren.
- `_tie_Agent` und `_boaimpl_Agent`
  Je nach der Art (TIE oder BOAImpl), in der das Interface Implementiert werden soll der ist eine dieser Dateien notwendig, die andere nicht.

`dipatcher.java` ist eine intern verwendete Klasse, mit der ein Entwickler nicht in Berührung kommt und die deshalb hier nicht weiter erläutert wird.

Implementierung von Client- und Server-Objekten

Zur Implementierung wird als erstes in `AgentImplementation.java` das vorgegebene Interface ausprogrammiert. Dies ist die Klasse, aus der die Objekte hervorgehen, die entsprechende Aufrufe durch den Client bearbeiten.
Um solche Clientaufrufe zu bedienen, muß noch eine Serverapplikation (`AgentServer.java`) geschrieben werden, in der dann ein solches Objekt konkret instantiiert wird. Der Server muß nun noch beim ORB angemeldet werden, dies geschieht bei OrbixWeb durch das `putit` -Kommando. Sobald diese Schritte erledigt sind, stehen die Server-Objekte zur Verwendung bereit.

Beim Erstellen eines Client (z.B. in der Datei `AgentClient.java`) muß ein Objekt der Interface-Klasse `_AgentRef` definiert werden, welches durch `bind()` mit dem gewünschten Server verbunden wird. Anschließend können dessen Methoden und Parameter über diesen aus den IDL-Schnittstellen entstandenen IDL-Stub benutzt werden.

Wichtig bei der Erstellung von Clients und Servern ist die Behandlung von Fehlern, die erst durch den Betrieb mit dem ORB entstehen. Typische Beispiele der eigens dafür vorgesehenen Exceptions wären eine fehlende Netzwerkverbindung zum gewünschten Objekt, oder das Fehlen eines Servers mit dem gewünschten Namen. Erst wenn dies berücksichtigt ist, kann eine Anwendung zuverlässig funktionieren.

2.5 Management by Delegation

Nachdem also durch CORBA eine Möglichkeit dazu besteht, wird nun betrachtet, was eine flexible Verteilung von Objekten für Vorteile bei der Entwicklung eines Managementsystems mit sich bringt.

Die Struktur nahezu aller gegenwärtig im Einsatz befindlicher Managementsysteme stellt sich wie in Bild 2.4 dar: Agenten sammeln vor Ort Informationen über die zu überwachenden Bestandteile (*Managed Objects*), die an den Manager übermittelt werden. Dieser erbringt die eigentliche Managementfunktionalität, indem er die benötigten Daten auswählt, bewertet und nach Bedarf verwendet.
Dieses Konzept verursacht einen ständigen Datenverkehr zwischen Manager und Agenten, um die Managementinformationen zu übermitteln. Mit einer ansteigenden Zahl von Managed Objects (in der vorliegenden Aufgabenstellung: eine größer werdende Zahl von Webservern) steigt auch die vom Managementsystem übertragene Datenmenge direkt proportional an. Bei einer festen Bandbreite des Netzwerkes bedeutet dies einen höheren prozentualen Anteil am Gesamtdatenverkehr. Diese ungenügende Skalierbarkeit kann theoretisch sogar soweit führen, daß das Netz bereits durch die Managementdaten überfordert ist.
Auch im Fehlerfalle kann dieses Konzept zu Schwierigkeiten führen. Sobald Probleme im Netzwerk auftreten, nimmt der Datenverkehr weiter zu, da der Manager eine Vielzahl von Informationen benötigt, um korrekt reagieren zu können. Es entsteht also genau dann die größte Belastung, wenn das Netz am wenigsten dazu in der Lage ist sie zu bewältigen. Im Extremfall gelangen die entscheidenden Daten überhaupt nicht zum Manager und das Problem bleibt bestehen.

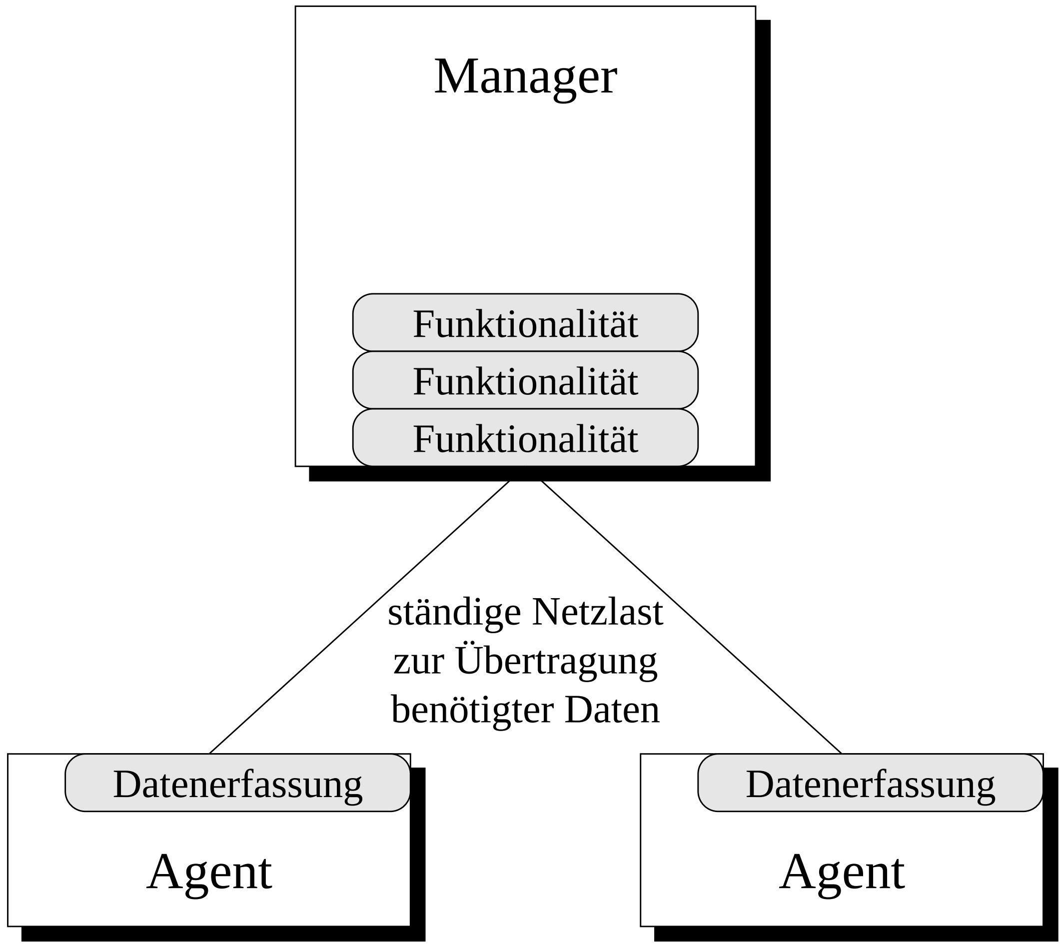

Abbildung 2.4: Manager/Agenten - Beziehung gebräuchlicher Managementkonzepte

Das *Management by Delegation* Paradigma bietet hier einen Lösungsansatz. Eine Verschiebung (Delegation) der Funktionalität vom Manager hin zum Agenten, wie in Abbildung 2.5 aufgezeigt, bedeutet weniger Datenverkehr, da die Informationen direkt vor Ort ausgewertet werden können, ohne den Weg über das Netzwerk zu beschreiten. Dadurch ist eine wesentlich bessere Skalierbarkeit gegeben.

Beim Auftreten eines Fehlers entfällt die zusätzliche Last aus dem selben Grund und erleichtert damit die Gegenmaßnahmen. Die lokalen Daten liegen auch dann vor, wenn das Netz völlig zusammengebrochen ist und ermöglichen zusammen mit den entsprechenden Funktionen auch unter diesen Umständen eine adäquate Fehlerbehandlung.

Wenn die Verzögerung durch den Transport der Daten wegfällt, kann entsprechend schneller auf eingetretene Ereignisse reagiert werden. Solche Zeitvorteile sind vor allem dann relevant, wenn es sich um Hochgeschwindigkeitsnetze handelt, bei denen ebensoschnelle Reaktionszeiten erforderlich sind.

Unterschieden wird dabei die dynamische und die statische Delegierung. Dynamisch bedeutet in diesem Zusammenhang die Verlagerung von Funktionalität zur Laufzeit, während statisch eine andere Festlegung der Funktionsaufteilung bei der Implementierung bezeichnet, die später nicht

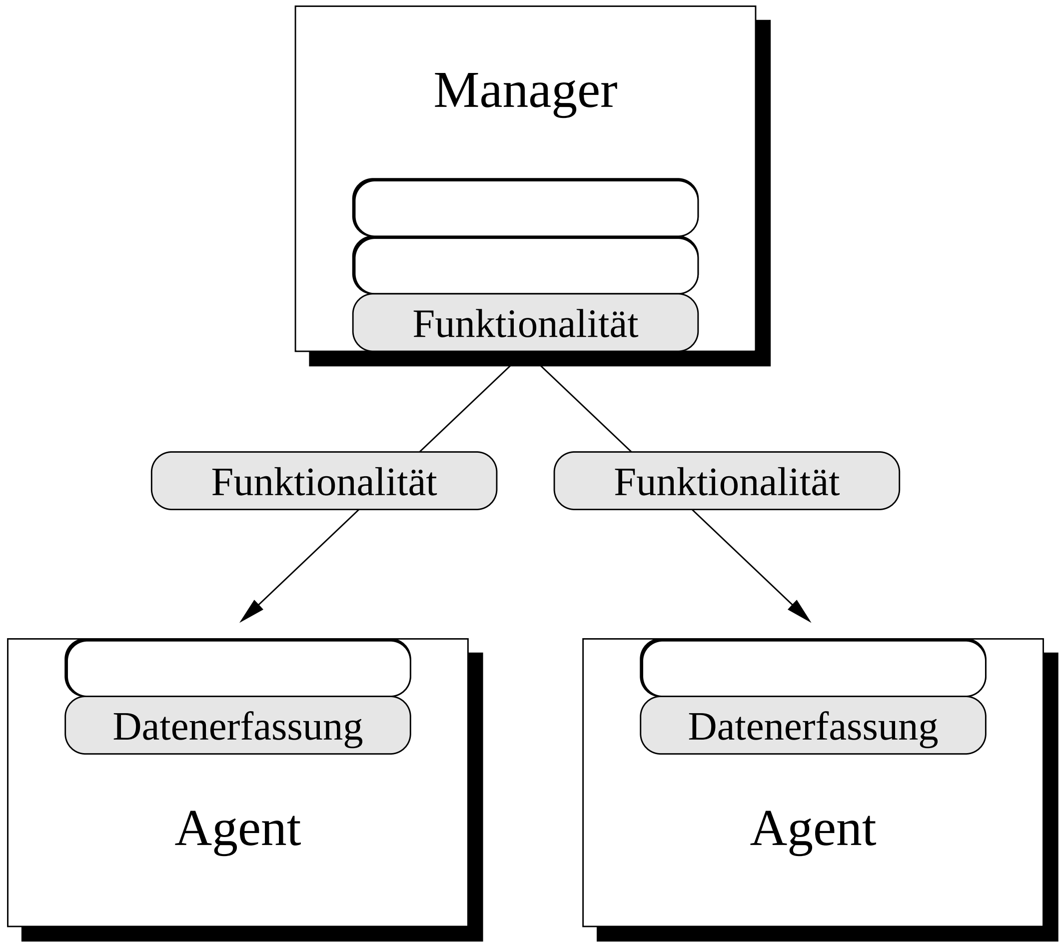

Abbildung 2.5: Verlagerung von Funktionalität vom Manager zu Agenten

mehr verändert werden kann. Mit einer dynamischen Vorgehensweise ließe sich beispielsweise ein Zuladen der jeweils gerade benötigten Funktionalität mit anschließender lokaler Ausführung realisieren. Sie ermöglicht eine wesentlich höhere Flexibilität als die statische Variante und benötigt dennoch nur die gerade erforderlichen Ressourcen, stellt aber ebenso höhere Anforderungen an die Verwaltung des Gesamtsystems.

Aber auch dieser Ansatz weist einige Tücken auf. Es kann nicht Alles beliebig delegiert werden. Gerade Anwendungen wie das Fehlermanagement benötigen oft die Übersicht über das ganze Netzwerk und nicht nur über eine Komponente damit der Fehler lokalisiert werden kann. Dies ist zwar ebenfalls lösbar, jedoch nur unter Verwendung von traditionellen Manager/Agenten - Beziehungen.

Aus den genannten Gründen müssen bei der Umsetzung eines Managementkonzepts nach diesem Paradigma Überlegungen angestellt werden, welche Funktionalitäten delegiert werden können und welche sinnvollerweise nicht.

Kapitel 3

WWW-Server Management

Die wachsende Verbreitung des Internet geht nicht zuletzt auf die Anwenderfreundlichkeit des World-Wide-Web (WWW) zurück. Hinter den bunten Seiten der Anbieter verrichten WWW-Server ihre Arbeit. Doch der Betrieb eines WWW-Servers bringt einen nicht unerheblichen Verwaltungsaufwand mit sich. Den überwiegenden Teil der dabei anfallenden Arbeiten kann man als typische Managementanwendungen klassifizieren. So wird mit der zunehmenden Anzahl von WWW-Servern auch der Bedarf nach Managementkonzepten für eben diesen Zweck stärker.

In dieser Arbeit soll deshalb der Betrieb von WWW-Servern in Hinblick auf ein mögliches Management, vor allem bezüglich Leistungs- und Fehlermanagement, näher untersucht und daraus ein Objektmodell entworfen werden, um mit diesem Wissen anschließend ein Managementsystem prototypisch zu implementieren

3.1 Überblick

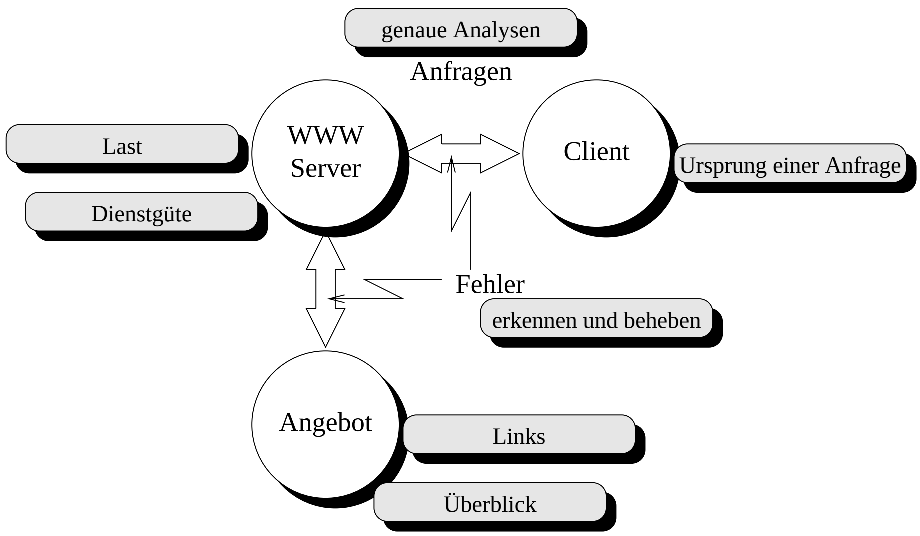

Abbildung 3.1: Aspekte beim Betrieb eines WWW-Servers

Für das Management eines WWW-Servers sind (siehe Abb. 3.1) folgende Aspekte von Bedeutung: Der Server selbst und die Anfragen an diesen, das zur Verfügung stehende Angebot, die Benutzer dieser Angebote und die dabei auftretenden Probleme. Das Betriebsziel liegt in der Nutzung: Man will für gewöhnlich möglichst viele und zufriedene Benutzer haben. Um das zu erreichen, müssen die Angebote attraktiv und die auftretenden Probleme gering gehalten werden. Dazu ist die Berücksichtigung folgender Punkte dienlich:

- Den Überblick über das eigene Angebot zu behalten. Hierbei sind nicht nur die genauen Daten aller Dokumente erforderlich, sondern auch die Linkstrukturen zwischen den Seiten.
- Das Sammeln von Informationen über die Benutzer des Servers.
- Eine Analyse des Serverbetriebs. Dazu zählen vor allem die Anfragen von Clients, die auftretende Last, die Steuerung der Leistungsfähigkeit und die Überwachung der Dienstgüte.
- Das Erkennen und Beheben von Fehlersituationen.

Zur genaueren Behandlung der oben genannten Punkte werden im Anschluß typische Probleme und Sichtweisen betrachtet, um daraus die jeweils managementrelevanten Informationen und Funktionalitäten zu extrahieren.

3.2 Die angebotenen Dokumente

Das eigene Angebot an WWW-Seiten zu überblicken, ist für einen zuständigen Administrator eine zentrale Aufgabe. Bei einem mächtigen Bestand von vielen tausend Dokumenten auf eventuell mehreren WWW-Servern ist eine Gesamtübersicht jedoch nur sehr schwer zu erlangen. Eine große Zahl verschachtelter Verzeichnisse, unter Umständen auf mehr als einem Rechner verteilt, enthält verschiedenste Dokumente, Anwendungen und Daten die im Laufe der Zeit erstellt und in die bestehenden Strukturen eingefügt wurden. Welche Anforderungen ergeben sich nun aus dieser Aufgabe?

3.2.1 Übersicht des Angebots

In erster Linie sind für eine Übersicht die nackten Zahlen und Namen, die ein Dokument beschreiben von Bedeutung. Jeder Versuch, sich ein Bild vom Ganzen zu machen, muß zwangsweise darauf zurückgreifen. Für diesen Zweck müssen die Daten über jede beliebige Datei auf bequeme und übersichtliche Weise in Erfahrung zu bringen sein. Auch eine Sortiermöglichkeit nach verschiedenen Gesichtspunkten und ausgewählte Selektionen bestimmter Dokumente wären vorteilhaft. Die folgende Auflistung nennt die einzelnen Punkte und begründet deren Bedeutung:

- Grundsätzlich müssen zwei Sichtweisen differenziert werden. Einerseits existiert lokal vor Ort eine Datei, eindeutig gekennzeichnet durch den physikalischen **Namen** und den **Pfad**, andererseits handelt es sich dabei um ein über das Netz durch seine **URL** verfügbares Dokument.
- Nichts ist der Akzeptanz eines Dienstes so abträglich wie eine unangemessen große Wartezeit. Nur allzuleicht wirkt sich die unpassende **Größe einer Datei** auf diese Wartezeit aus. Wenn die Übertragungsrate des Servers bedingt durch die Netzanbindung oder die

ständig hohe Anzahl von gleichzeitigen Benutzern, die sich die vorhandene Bandbreite teilen, sehr niedrig ist, ist der weitsichtige Verwalter bemüht, die Dokumente eher klein und damit schneller übertragbar zu halten.

- Bei manchen Dokumenten – ein typisches Beispiel hierfür wäre ein Terminplan – ist die **Aktualität** der darin enthaltenen Informationen der wichtigste Gesichtspunkt und es gilt soweit machbar, stets den neuesten Stand anzubieten. Ein älteres Datum kennzeichnet dabei auf den ersten Blick überholte Inhalte, die nicht im Sinne eines attraktiven Angebots sind. Je veränderlicher die Inhalte sind, desto entscheidender ist diese Information.
- Neben dem einfachsten Fall von statischem html-Code existieren weitere Arten von Dokumenten, wie z.B. durch Perl-Skripten dynamisch erzeugter Code oder eingebundene Grafiken. Diese **verschiedenen Typen** weisen auch unterschiedliche Charakteristika auf: dynamische Dokumente verursachen serverseitig mehr Last, Bilder sind unter Umständen weniger wichtig, falls nur zur optischen Aufbereitung verwendet, aber deutlich größer als Text allein.
- Der Name einer Datei, oder auch die komplette URL sind für gewöhnlich stark abgekürzt und tragen nur wenig dazu bei, die thematische Zugehörigkeit überschaubar zu machen. Bei manchen Dokumenten (für gewöhnlich nur bei statischen Seiten) ist es möglich einen **Titel** zu ermitteln. Dieser ist normalerweise aussagekräftiger und daher leichter verständlich und besser geeignet, den Überblick zu behalten

Um sich mit Hilfe des Managementsystems einen Überblick über die angebotenen Dokumente zu verschaffen, sollte man sie auch auch aus diesem heraus sowohl als Dokument, als auch als Quelltext betrachten können. Da für diesen Zweck Browser ebenso wie entsprechende Editoren zur Bearbeitung der Dokumente vorhanden sein dürften, ist es ausreichend, wenn sich die bevorzugten Werkzeuge für die gewünschten Dokumente aus dem Managementsystem heraus aktivieren lassen.

Managementinformation:

- **Name und Pfad** `[: string]`
- **URL** `[: string]` (→ **WWW-MIB**)
- **Größe** `[: integer]`
- **Aktualität** `[: date_and_time]`
- **Dokumenttyp** `[: string]` (→ **WWW-MIB**)
- **Dokumenttitel** `[: string]`

Managementfunktionalität:

- **Dokument/Quelltext betrachten**
- **Datei/Dokument bearbeiten**

3.2.2 Die Linkstruktur der Dokumente

Wie die Verzeichnisstruktur mit allen Dateien auf dem Server aussieht, ist allerdings nur die eine Seite der Medaille. Ein elementarer Bestandteil des WWW sind Hypertextverweise auf andere Dokumente, die sogenannten Links. Wie sich diese gesamte Linkstruktur auf einem Server darstellt, ist für einen Administrator genauso wichtig. Welche Seite verweist wohin – vor allem innerhalb des eigenen Servers – ist die dazu benötigte Information.

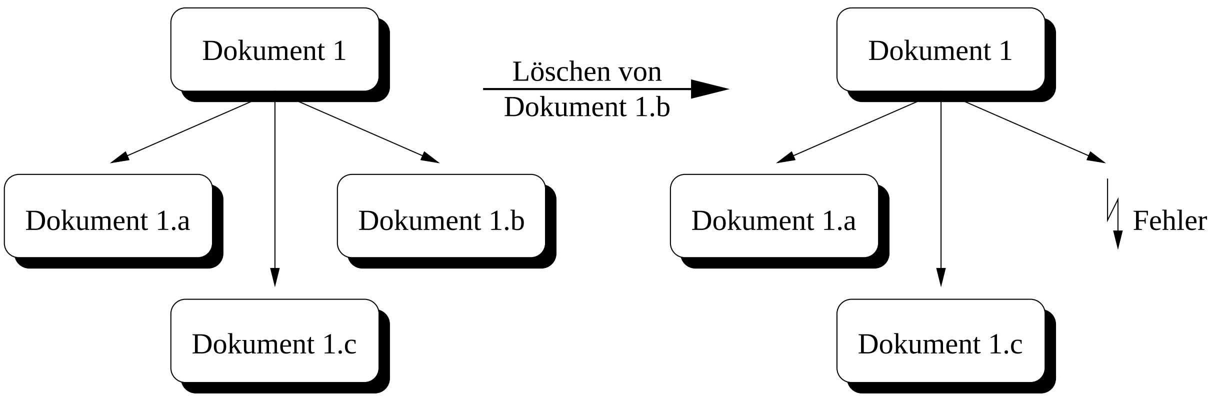

Abbildung 3.2: Fehlerhafter Link durch Löschen eines Dokuments

Beinhaltet eine Seite einen Verweis auf ein nicht mehr vorhandenes Dokument, oder einen sonstigen falschen Link, so sollte dieser entfernt bzw. korrigiert werden (siehe Abb. 3.2). Dazu muß die Konsistenz der Links automatisch überprüfbar sein, da eine manuelle Überprüfung nur mit großem Zeitaufwand durchführbar wäre. Um hier auf einfache Weise eine gewisse Konsistenz aufrechtzuerhalten, wäre es erforderlich, beim Löschen oder Verschieben eines Dokuments alle Links, die auf eine Seite verweisen, zu kennen, um sie entsprechend anzupassen. Der Verantwortliche weiß aber im Normalfall nicht, von wo aus auf eine Seite verwiesen wird und wo daher etwa bei einer Namensänderung Berichtigungen notwendig sind. Das Wissen um diese Links ist daher sinnvollerweise Bestandteil der Managementinformation.
Es wäre hier auch durchaus denkbar, für das Löschen oder Verschieben von Dokumenten Funktionen von Seiten des Managementsystems zur Verfügung zu haben, die das automatisch durchführen und somit die Korrektheit der Verweise besser gewährleisten.

Managementinformation:

- **Links auf andere Dokumente** `[: array of document]`
- **Links von anderen Dokumenten** `[: array of document]`

Managementfunktionalität:

- **Konsistenz der Links testen**
- **Datei/Dokument löschen**
- **Dokument verschieben**

3.3 Die Benutzer des Servers

Ein weiterer Aspekt beim Management eines Webservers sind dessen Benutzer – die Clients in der Client/Server-Architektur des WWW. Wer nutzt von wo aus und in welchem Umfang die Angebote? Und nicht immer ist die gesamte Angebotspalette zur Verwendung für die breite Öffentlichkeit vorgesehen. Beispielsweise für kostenpflichtige oder sicherheitsbeschränkte Anwendungen müssen autorisierte Benutzer unterschieden werden, deren Verwaltung und Abrechnung zweckmäßigerweise durch das Managementsystem unterstützt wird. Um dies zu ermöglichen und über die Clients auftretenden Fragen zu beantworten, sind einige Informationen nötig.

Woher stammen die Anfragen? Aus welchen Domains, oder bei interner Verwendung von welchen Rechnern und soweit ermittelbar von welchen Personen? Die Antworten auf diese Fragen geben beispielsweise Hinweise darauf, ob der Server vom gewünschten Zielpublikum genutzt wird. Vielleicht würde auch durch eine räumliche Verlagerung oder einen zweiten Server mit gespiegelten Inhalt der Netzverkehr wesentlich verringert. Entsprechende Ranglisten und gezielte Auswahlmöglichkeiten wären für solche Analysen optimal geeignet.
Auch der Zeitpunkt, wann ein Benutzer zuletzt auf dem Server aktiv war kann von Interesse sein. So etwa wenn nicht nur interessiert, wie viele Benutzer der Server insgesamt, sondern in den zurückliegenden xx Tagen oder Wochen etc. hatte. Genaue Aussagen darüber lassen deutliche Rückschlüsse auf den jeweiligen Bekanntheitsgrad zu.

Bei näherer Betrachtung von gebührenpflichtigen Anwendungen, etwa bei kommerziellen Informationsdiensten, zeigen sich einige spezielle Anforderungen. Wenn die Kosten nicht pauschal erhoben werden, sind je nach Abrechnungsart die jeweilige Anzahl der Zugriffe bzw. die übertragene Datenmenge jedes Benutzers für die Berechnung erforderlich. Um solche Aufgaben zu unterstützen, ist jedoch eine Anbindung an eine geeignete Datenbank kaum zu vermeiden. Dadurch wäre es dann auch möglich die autorisierten Benutzer geeignet zu verwalten (Passwörter, Rechte, etc.)

Aber auch bei nicht beschränkten Servern ist es notwendig, die Nutzung durch die Clients genauer zu analysieren. Wird der Server vom Anbieter selbst betrieben, dann ist es eine Möglichkeit für den Provider, die Kosten für die Bereitstellung des Internetanschlußes nach der von den Benutzern abgerufenen Datenmenge zu berechnen. Abgesehen davon, ist das Wissen um die gesamte umgesetzte Datenmenge auch zur Kapazitätsplanung unabdingbar und schon dadurch eine wichtige Information.

Managementinformation:

- **Hostname** [: `string`]
- **Name autorisierter Benutzer** [: `string`]
- **Anzahl Zugriffe** [: `integer`]
- **Übertragene Datenmenge** [: `integer`]
- **Zeitpunkt der Zugriffe** [: `date_and_time`]

3.4 Der Betrieb des Servers

Was sind die entscheidenden Komponenten beim Betrieb eines WWW-Servers – diejenigen, die für einen Administrator von Interesse sind? Es lassen sich hier mehrere Dinge ausmachen: Da sind zum einen die Anfragen der Benutzer. Die Abwicklung dieser Kommunikation stellt die eigentliche Aufgabe des Servers dar und ist daher auch von besonderer Wichtigkeit. Zum anderen verursachen diese Anfragen eine Last, die es einschließlich ihrer Bewältigung zu untersuchen gilt. Und schließlich darf die Dienstgüte (Quality of Service) nicht aus den Augen verloren werden. Sie ist ein gewichtiger Faktor bei der Bewertung durch die Benutzer. Diese drei Gesichtspunkte werden in den nachfolgenden Abschnitten genauer behandelt.

Daneben gibt es aber noch einige allgemeine Dinge, die einen Server genauer spezifizieren und nicht vernachlässigt werden sollten. An vorderster Stelle stehen dabei der physikalische Ort, an dem der Rechner aufgestellt ist, sein Domainname und die dazugehörige IP-Adresse, sowie die Portnummer die er benützt. Für den Fall von Störungen oder sonstigen Benachrichtigungsgründen wäre auch der Name und e-mail des Verantwortlichen (Webmaster) sinnvoll. Diese Angaben scheinen auf den ersten Blick eher von geringerem Interesse zu sein, denn sie verändern sich für einen Server normalerweise über längere Zeiträume nicht. Falls doch sollten die Änderungen aber mit der Unterstützung des Systems durchgeführt werden können und spätestens beim Betrieb mehrerer Server, ist es ohnehin notwendig diese Daten von jedem einzelnen zu kennen.
Ebenso grundlegend wie diese Angaben, ist eine Möglichkeit, den Server zu beenden und zu starten, z.B. um Konfigurationsänderungen wirksam werden zu lassen, oder im Fehlerfalle auf einen Absturz durch einen Neustart reagieren zu können.

Managementinformation:

- **Aufstellungsort** [: `string`]
- **Servername oder IP-Adresse** [: `string`] (→ **WWW-MIB**)
- **Portnummer** [: `integer`]
- **Administrator** [: `string`] (→ **WWW-MIB**)

Managementfunktionalität:

- **Starten und Beenden des Servers**

3.4.1 Anfragen und Antworten

Will man die Zugriffe von Clients auf den Server genauer betrachten, ergeben sich dabei vielfältige Fragestellungen. Die interessanteste Frage für den Betreiber eines Webservers ist dabei sicherlich nach dem Dokument, welches am häufigsten abgerufen wurde. Je öfter, desto größerer Bedarf besteht anscheinend nach diesem Angebot. Der Ausbau eines bereits gut frequentierten Gebietes würde vermutlich dessen Attraktivität weiter erhöhen, andererseits besteht wohl Nachholbedarf bei eher schwach genutzten Seiten. Ranglisten, wie oft insgesamt, oder wie oft in einem bestimmten Zeitraum ein Dokument angefordert wurde, sind die zumeist gewünschten Präsentationsformen. Aber auch detailliertere Angaben je Dokument, die zur Erstellung solcher

Ranglisten benötigt werden, z.B. neben der Häufigkeit auch die genauen Zeitpunkte wann es angefordert wurde, müssen zu erfahren sein.
Die Zeitdauer seit dem letzten Zugriff fügt sich hier ergänzend ein. Mit beiden Werten zusammen läßt sich ein guter Überblick über die gegenwärtigen Stärken und Schwächen bilden.

Interessant ist nicht nur die Anzahl von Anfragen, sondern auch eine präzise Aufschlüsselung nach den verwendeten Protokollen (FTP, HTTP, ...) und Methoden (GET, POST, ...), um die Zugriffe der Benutzer analysieren und darauf basierend Entscheidungen bezüglich der Realisierung des eigenen Angebots treffen zu können.

Nicht immer kann jede Anfrage beantwortet werden. Korrekt abgewickelte Verbindungen sollten von solchen zu unterscheiden sein, bei denen beispielsweise ein nicht vorhandenes Dokument angefordert wurde, oder die Zugriffsberechtigung nicht gegeben war. Zur Realisierung ist demnach auch der Ergebnisstatus einer Antwort in die Liste der managementrelevanten Informationen einzufügen.

Um Aussagen über die durchschnittliche Übertragungsgröße, oder den Durchsatz des Servers während einer bestimmten Zeit (beispielsweise Bytes/Stunde oder /Tag) zu treffen, ist die Datenmenge, die bei jeder Anfrage übermittelt wird, erforderlich. Diese Größe ist vor allem bei dynamischen Dokumenten wichtig, da je nach Anforderung variable Ausgaben erzeugt werden. Bei statischen Dokumenten würde diesbezüglich auch die Dateigröße aus Abschnitt 3.2.1 ausreichen.

Von Nutzen wäre auch zu wissen, wie lange eine Verbindung andauert. Um im Fehlerfalle Verbindungen nicht unnötig lange aufrechtzuerhalten wird eine maximale Verbindungsdauer angesetzt, nach deren Ablauf eine Verbindung ungeachtet des augenblicklichen Status vom Server abgebrochen wird. Bei der Übertragung umfangreicher Dokumente, oder sehr geringer Übertragungskapazität kann es allerdings vorkommen, daß dieses Limit überschritten wird, obwohl kein Fehler vorliegt. In einer ungünstigen Konstellation kann es sogar soweit gehen, daß ein Dokument nicht übermittelt werden kann, ohne daß die maximale Verbindungsdauer angehoben wird. Eine durchschnittliche Dauer der Verbindungen kann Auskunft geben, inwiefern die gewählte Grenze sinnvoll ist oder verändert werden sollte.

Managementinformation:

- **Anfragen je Dokument** `[: integer]` (→ **WWW-MIB**)
- **Zeitstempel der Anfragen** `[: date_and_time]` (→ **WWW-MIB**)
- **Letzter Zugriff je Dokument** `[: date_and_time]` (→ **WWW-MIB**)
- **Verwendete Methode** `[: string]` (→ **WWW-MIB**)
- **Verwendetes Protokoll** `[: string]` (→ **WWW-MIB**)
- **Antwortstatus** `[: integer]` (→ **WWW-MIB**)
- **Übertragene Datenmenge** `[: integer]` (→ **WWW-MIB**)
- **Durchschnittliche Übertragungsrate** `[: integer]`
- **Verbindungsdauer** `[: integer]`

- **Durchschnittliche Verbindungsdauer** `[: integer]`
- **Maximale Verbindungsdauer** `[: integer]`

3.4.2 Last und Leistung

Am einfachsten lassen sich über den längerfristigen Lastverlauf durch entsprechende Statistiken Aussagen treffen. Daraus lassen sich leicht verständliche Auflistungen oder Grafiken erzeugen, die auf einen Blick über unterschiedlichste Problemstellungen Aufschluß geben. Ein typisches Beispiel hierfür wäre wie in Abb. 3.3 dargestellt die zeitliche Verteilung der Verbindungen des Servers.

Abbildung 3.3: Ein Beispiel für die Anzahl von Verbindungen im Tagesverlauf

Der durchschnittliche Verlauf eines Tages zeigt eventuelle Spitzenbelastungen auf und ermöglicht so die geeignet Wahl von Zeiten für Verwaltungsarbeiten oder im Hintergrund laufende Jobs, die den Normalbetrieb nur wenig beeinträchtigen sollen. Ähnliches gilt für den Wochenverlauf. Hierdurch lassen sich günstige Zeitpunkte finden, an denen zeitlich nicht gebundene Unterbrechungen wie etwa das Herunterfahren des Servers für Wartungsarbeiten an der Hardware, so wenig Benutzer wie möglich stören.

Bei den meisten Servern – soweit unter Verwendung von mehreren Prozessen oder Threads realisiert – wird jede Anfrage von einem eigenen Prozeß verarbeitet, von denen jeder einen vollständigen Server darstellt. Um mehrere Anfragen gleichzeitig beantworten zu können und so eine höhere Leistung zu erhalten, müssen mehrere solcher Serverprozesse parallel ablaufen.
Soll die Leistungsfähigkeit beeinflußbar sein, ist eine Möglichkeit vonnöten, diese Prozeßanzahl, vor allem die der momentan nicht benötigten, nach Bedarf zu verändern (siehe Abb. 3.4). Laufen viele unbeschäftigte Server, werden Ressourcen vergeudet, um diese überflüssigen Prozesse zu realisieren. Eine zu niedrige Anzahl beschränkt aber die Leistung des Servers insgesamt erheblich, da zu oft kein freier Prozeß mehr vorhanden ist, um eine Verbindung zu akzeptieren und es bei mehreren gleichzeitigen Anfragen zu lange dauert, bis zu deren Bearbeitung genügend viele neue Prozesse gestartet sind. Eine flexible Einstellung dieser Werte ermöglicht es also erst, auf unterschiedliche Lastbedingungen geeignet zu reagieren. In einem gewissen Rahmen könnten diese Grenzen auch eigenständig vom Managementsystem geregelt werden.

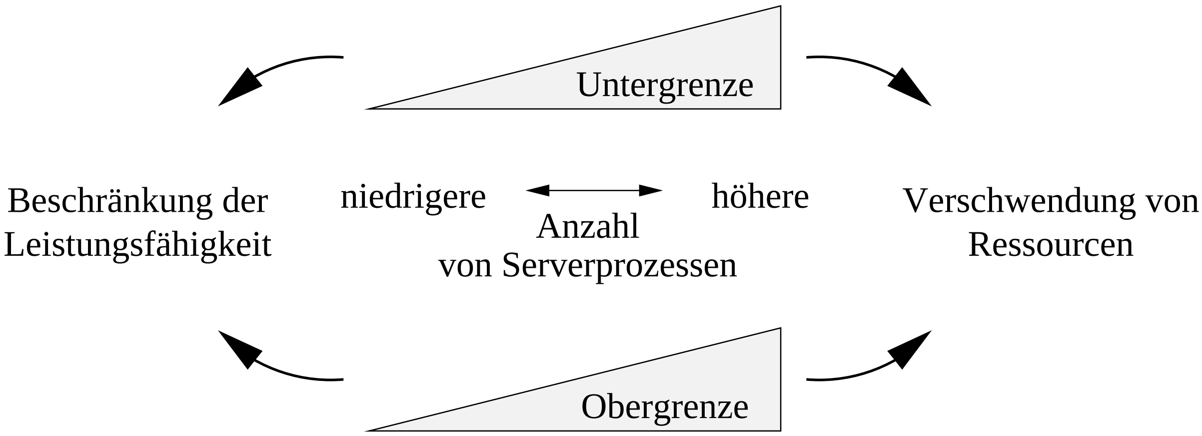

Abbildung 3.4: Problematik mit der Anzahl unbeschäftigter Serverprozesse im Leerlauf

Steigt die Belastung eines Servers an, werden auch zunehmend mehr Prozesse gestartet, auf diese Weise kann auf alle Anfragen parallel reagiert werden. Sinkt diese Last danach wieder ab, so bleiben dennoch mindestens soviele Prozesse aktiv, wie durch die maximale Anzahl unbeschäftigter Server vorgegeben ist. Die Belastung des Rechners nimmt dadurch nicht in gleichem Maße ab, sondern pendelt sich auf einen konstanten Wert ein. In der Praxis ist dieses Problem oft durch die Vergabe einer Lebensdauer je Prozeß gelöst. Mit der Zeit verringert sich also die Anzahl der aktiven Server selbständig, falls nicht infolge der Anfragen neue Prozesse gestartet werden. Diese maximale Lebensdauer ist demnach ein einfaches Mittel zur Lastregulierung. Es ist jedoch erforderlich diesen Wert der durchschnittlichen Belastung des Servers anzupassen (d.h. mehr Anfragen = höhere Lebensdauer).

Managementinformation:

- **Last (Anfragen/Zeit)** `[: integer]`
- **Maximale Anzahl von Serverprozessen** `[: integer]`
- **Maximale Anzahl von unbeschäftigten Serverprozessen** `[: integer]`
- **Minimale Anzahl von unbeschäftigten Serverprozessen** `[: integer]`
- **Lebensdauer von Serverprozessen** `[: integer]`

3.4.3 Dienstgüte (Quality of Service)

Neben der Analyse der Benutzung des Servers, gilt auch der bei dieser Benutzung aufgetretenen Dienstgüte (Quality of Service) besonderes Augenmerk. Je besser diese ist, umso weniger Gründe zur Beanstandung gibt es und umso wahrscheinlicher wird der Server öfter in Anspruch genommen.

Ein entscheidendes Qualitätsmerkmal ist die Verfügbarkeit. Die Fehlermeldung, daß der Server augenblicklich nicht erreichbar ist, gehört für den Benutzer mit zu den häufigsten und ist daher äußerst störend. Wie oft (Durchschnitt in Prozent) der Server ordnungsgemäß funktioniert –

verfügbar ist – ist demnach ein Wert, der für einen Administrator von großem Interesse ist. Um eine durchschnittliche Verfügbarkeit zu bestimmen, muß überprüft werden, ob der Server im Augenblick überhaupt ansprechbar ist oder nicht. Da diese Information daher bereits zur Verfügung stehen muß und im Falle eines Absturzes tunlichst etwas unternommen werden sollte, ist es durchaus sinnvoll, den Eintritt dieses Zustandes auch kenntlich zu machen.

Zur Güte eines Dienstes gehört auch, wieviel Zeit seine Benutzung in Anspruch nimmt. Diese Zeit wird festgelegt durch die Größe der zu übertragenden Datei (siehe auch Abschnitt 3.2.1 - Dateigröße) und die maximale Übertragungsrate des Servers, für gewöhnlich ein fest durch die Netzwerkanbindung vorgegebener Wert. Auf der anderen Seite spielt auch noch der Faktor Antwortzeit des Servers eine Rolle. Beim Abruf von statischen Seiten mag die dafür notwendige Zeit noch relativ klein sein, bei Überlast oder dem Start eines neuen Serverprozeßes, spätestens aber bei komplexen Datenbankabfragen fällt sie deutlicher ins Gewicht. Die Kontrolle der Antwortzeit wäre also durchaus eine brauchbare Komponente im Gesamtsystem.

Managementinformation:

- **Verfügbarkeit** `[: float]`
- **Serverstatus** `[: integer]` (→ **WWW-MIB**)
- **Übertragungsrate** `[: integer]`
- **Antwortzeit** `[: integer]`

3.5 Fehlermanagement

Fehler können zwar vermieden, aber nie vollständig verhindert werden. Wie lange es dauert bis ein Fehler beseitigt ist, hängt in erster Linie davon ab, wie schnell er erkannt wird, denn eine genaue Analyse und adäquate Fehlerbehandlung kann sich verständlicherweise erst danach anschließen. Von großem Vorteil ist dabei, wenn das System soweit realisierbar automatisch diese Aufgaben übernimmt, um Arbeits- und Reaktionszeit zu sparen.

Wird bei der Überprüfung des Serverstatus (Abschnitt 3.4.3) ein Absturz festgestellt, muß das eventuell noch aktive Programm beendet und anschließend neu gestartet werden. Eine Automatisierung bringt enorme Vorteile mit sich. Wenn sich der Administrator nicht regelmäßig des Servers bedient, stellt er dessen Funktionsunfähigkeit oft erst mit erheblicher Verzögerung fest – einmal ganz abgesehen von arbeitsfreien Zeiten oder Pausen. Die Zeitdauer bis zur Entdeckung und Behebung durch das Managementsystem dagegen, ist lediglich durch das gewählte Intervall der einzelnen Überprüfungen beschränkt.

Zu den häufigsten Fehlern zählen Anfragen nach nicht vorhandenen Dokumenten, die z.B. durch Tippfehler beim Laden eines Dokuments entstehen können. Allerdings ist es sehr offensichtlich, daß eine über längere Zeit oft wiederholte falsche Anfrage einen nicht korrekt gesetzten Link (etwa durch das löschen eines Dokuments) oder eine fehlerhaft generierte dynamische Verbindungsanforderung bedeuten, deren Ursache nicht zuletzt auf den eigenen Seiten zu suchen ist, denn kaum ein Benutzer wird ständig dieselbe falsche Anfrage stellen. Ein solcher Fehler würde durch eine Konsistenzprüfung, wie in Abschnitt 3.2.2 erläutert, vermieden.

Problematisch wird es, wenn der Server Zugriffsfehler meldet, d.h. er kann auf benötigte Dateien oder Daten nicht zugreifen. Dies kann zum Beispiel bei defekten Datenträgern oder Netzwerkproblemen auftreten, auf jeden Fall liegt die Ursache vermutlich nicht im Zuständigkeitsbereich des WWW-Managements. In einem solchen Fall ist die schnellstmögliche Benachrichtigung des Administrators durch das System die einzige Lösung.

Im Abschnitt 3.4.1 wurde bereits die maximale Verbindungsdauer erwähnt. Wenn eine Verbindung zu lange andauert wird sie vom Server abgebrochen, da von einer Störung ausgegangen wird. Wie oft sich solche Timeouts ereignen, sollte beobachtet werden. Falls diese Anzahl auf einem festgelegten Zeitraum zu hoch liegt, kann eine Erhöhung des Grenzwertes in Betracht gezogen werden. Dazu wäre eine einstellbare Obergrenze von Timeouts in einem solchen Zeitraum vorstellbar, bei dessen Überschreitung die maximale Verbindungsdauer automatisch heraufgesetzt wird.

Interessant sind auch noch Fehlermeldungen des Servers, die auf fehlende Zugriffsrechte hinweisen. Dafür gäbe es zwei Erklärungen: Ein User hat ohne Berechtigung, oder von einem nicht genehmigten Host aus eine Anfrage gestellt.

Da es darüber hinaus noch eine Vielzahl von Fehlermöglichkeiten gibt, wäre es sehr schwierig alle getrennt zu erfassen – und weil manche davon noch dazu recht selten auftreten auch nicht sehr empfehlenswert. Aus diesem Grunde sollten alle nicht anderweitig einzuordnenden Fehlermeldungen des Servers einfach nach Häufigkeit und genauem Wortlaut katalogisiert werden, um genauere Untersuchungen nach öfterem Auftreten eines identischen Fehlers wenigstens zu erleichtern. Des weiteren sollte auch jeder Fehler mit einem Zeitstempel versehen sein, um Fehlerhäufungen festzustellen und damit der jeweils letzte aufgetretene Fehler im Falle eines Serverabsturzes nachvollzogen werden kann.

Managementinformation:

- **Anzahl nicht vorhandener Dokumente** [: `integer`]
- **Anzahl von Zugriffsfehlern** [: `integer`]
- **Anzahl von Zeitüberschreitungen** [: `integer`]
- **Anzahl fehlender Zugriffsrechte** [: `integer`]
- **Anzahl sonstiger Fehler** [: `integer`]
- **Zeitpunkt des Auftretens** [: `date_and_time`]
- **Exakte Fehlermeldung** [: `string`]

Managementfunktionalität:

- **Überwachung des Serverstatus**
- **Starten und Beenden des Servers**
- **Steuerung der maximalen Verbindungsdauer**
- **Prüfung auf sich häufende Fehler**
- **Meldungen an den Administrator absetzen**

Kapitel 4

Realisierung der Managementanforderungen

Im vorhergehenden Kapitel wurden Anforderungen an das Management eines Webservers gestellt. Hier soll nun untersucht werden, welche Basis zur Verfügung steht, um Informationen dafür zu gewinnen und die gewünschte Funktionalität zu realisieren. Gleichzeitig muß darauf geachtet werden, wie sich diese Ergebnisse mit den bereits gewonnenen zusammenführen lassen (Verknüpfen der Bottom-up-Analyse mit der Top-down-Sicht).

Bei der Betrachtung von serverspezifischen Details wird als Referenz der *Apache HTTP-Server* [Gro] herangezogen. Dieser Server ist im UNIX-Umfeld sehr verbreitet. Er ist frei verfügbar und zur Verwendung in dieser Arbeit nicht zuletzt wegen seiner Stabilität und Flexibilität sehr gut geeignet.

4.1 Das Logfile für die Zugriffe

Jeder WWW-Server bietet die Möglichkeit, die Anfragen in einer Datei mitzuprotokollieren , um genaue Daten für Analysen über die Zugriffe und eventuell auftretende Probleme zur Verfügung stellen zu können.

Da sich unterschiedliche Formate für ein solches Logfile entwickelt haben, wurde es notwendig, sich auf einen einheitlichen Standard zu einigen. Dieses sogenannte *Common Log Format* [Con] wird von allen Servern unterstützt, wenngleich auch einige Server zusätzlich eigene Formate mit erweiterten Informationen anbieten.

Dieses standardisierte *Common Log Format* hält folgende Informationen über jeden erfolgten Zugriff fest:

- `remotehost`
  Der Hostname des Clients; Falls der DNS Hostname nicht verfügbar, oder DNS-Lookup nicht aktiv ist, steht hier die IP-Adresse. Der Hostname läßt sich zerlegen in:
  - Rechnername und
  - Domain.

  Daraus lassen sich die notwendigen Informationen zur Adresse des Clients aus Abschnitt 3.3 gewinnen.

- `rfc931`
  Der Username des Clients; Dieser wird aber nur bei Nachfrage des Servers und wenn er zur freien Verfügung steht übermittelt. Tatsächlich bleibt dieses Feld meist leer.

  Aufgrund des meist fehlenden Eintrags wäre eine Verwendung äußerst fragwürdig

- `authuser`
  Der Benutzername, mit dem sich der Client authentifiziert hat; Falls sich der Benutzer nicht authentifiziert hat, weil das angeforderte Dokument keiner besonderen Zugriffsrechte bedarf, oder es sich um einen anonymen User handelt, bleibt auch dieses Feld leer.

  Dieser Name kann verwendet werden, um die Anfrage eindeutig einem Benutzer (bzw. einer Benutzergruppe) zuzuordnen, z.B. falls es sich um kommerzielle Server handelt, bei denen sich der Benutzer authentifizieren muß, um die gewünschte Seite zu erhalten. Damit würde eine nutzungsbezogene Abrechnung möglich.

- `date`
  Datum und Uhrzeit der Anfrage; Wann genau ist der Zugriff erfolgt. Diese Angabe ist nicht zwingend erforderlich, sondern lediglich optional.

  Diese Angabe entspricht dem bei den Anfragen und auch dem bei den Benutzern geforderten Zeitstempel für die einzelnen Verbindungen.

- `"request"`
  Die Anfrage an den Server genau so wie sie vom Client empfangen wurde, eingeschlossen in zwei Anführungszeichen; Aus diesem Request kann man mehrere einzelne Komponenten isolieren:

  – Die verwendete Methode (`GET, GETHEAD, POST, ...`),
  – das verwendete Protokoll (`FTP, HTTP, WAIS, ...`),
  – das angeforderte Dokument und
  – eventuelle Parameter.

  Die ersten drei Angaben lassen sich direkt für weitere der im Abschnitt 3.4.1 über die Verbindungen des Servers geforderten Informationen verwenden. Gleichzeitig erlaubt es erst dieser Eintrag, die gesamten Daten eindeutig einem Dokument zuzuordnen.

- `status`
  Der Statuscode, der dem Client bei der Antwort übermittelt wurde; Unterscheidungen, ob die Anfrage korrekt beantwortet werden konnte, oder warum nicht.

  Dieser Code ist verwendbar für den Ergebnisstatus aus 3.4.1.

- `bytes`
  Die Länge der Antwort in Bytes;

  Ist für die ebenfalls in 3.4.1 behandelte Datenmenge geeignet.

Der Zeitpunkt des letzten Zugriffs auf ein bestimmtes Dokument kann durch einen Vergleich aller Zugriffszeitpunkte ermittelt werden.
Aus eben diesen Zeitangaben und den Werten für die übertragene Datenmenge zu den Antworten

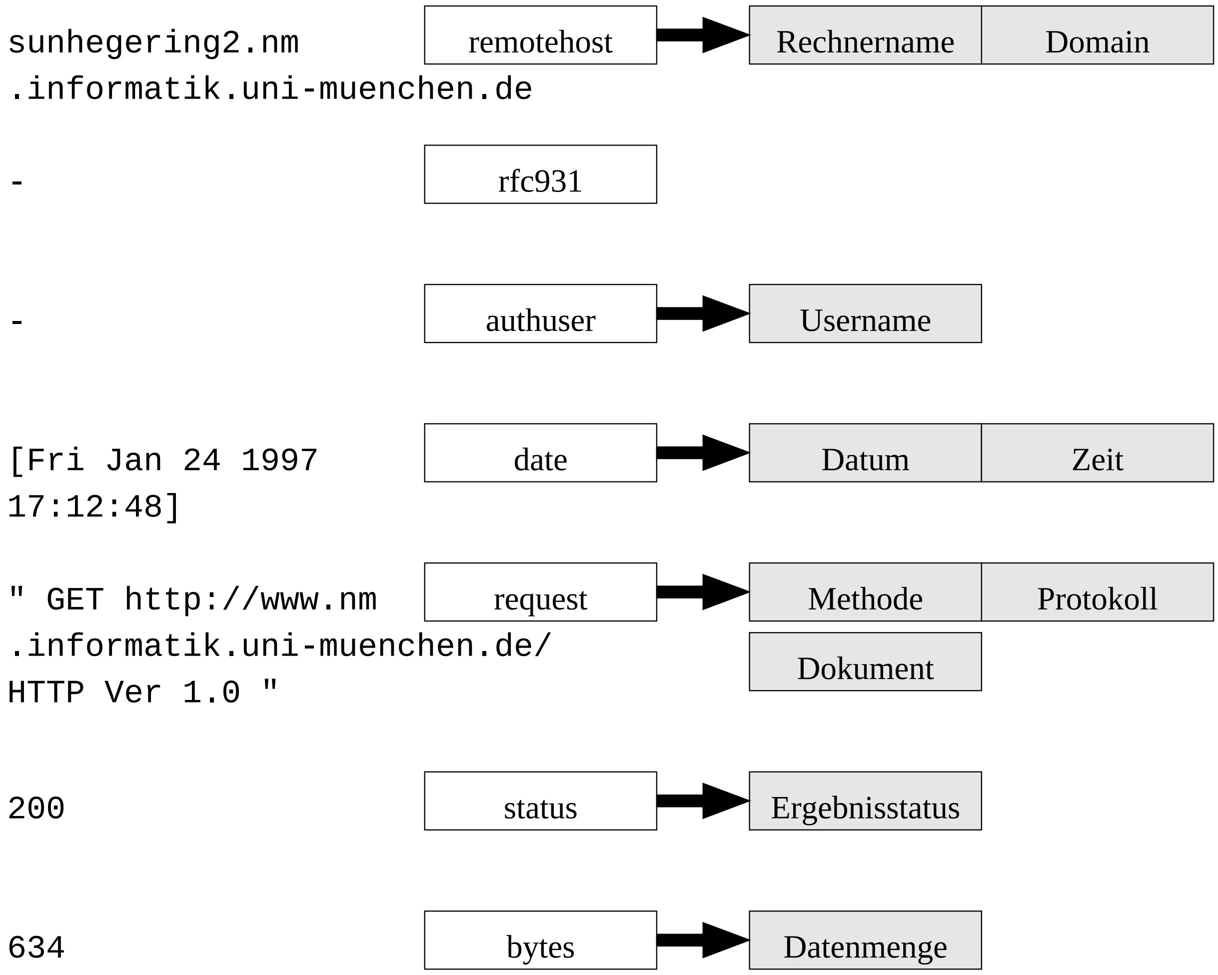

Abbildung 4.1: Zerlegung eines Eintrags des Common Log Formats

läßt sich auch die durchschnittliche Übertragungsrate berechnen, indem einfach die Datenmengen innerhalb einer Zeitspanne aufaddiert und durch die gewählte Zeitspanne dividiert werden:

$$\frac{\textit{übertragene Datenmenge innerhalb } 1h}{3600s} = \textit{durchschnittliche Übertragungsrate} \left[\frac{Bytes}{s}\right]$$

Um die Daten eines solchen Logfiles verwenden zu können, ist ein Werkzeug zur Auswertung erforderlich. In einem vorausgehenden Fortgeschrittenenpraktikum [Wim96] wurde ein solches Analysewerkzeug bereits entwickelt und steht für die Weiterverwendung zur Verfügung. Da die Implementierungssprache Perl verwendet wurde ist es denkbar, daß sich durch eine solche Anbindung Einschränkungen vor allem hinsichtlich der Portabilität ergeben, die ja in dieser Arbeit entscheidenden Stellenwert einnimmt. Da aber Perl zur Erstellung von CGI-Skripten auf Webservern weit verbreitet ist und so zumindest auf dem Rechner, der das Logfile erzeugt, vorhanden sein müßte, steht dieser Grund einer Verwendung nicht weiter im Wege.

Da für gut frequentierte Server das Logfile sehr schnell an Größe gewinnt muß es von Zeit zu Zeit zurückgesetzt werden. Wie dies genau zu geschehen hat entnimmt man am besten der zugehörigen Dokumentation. Es ist klar, daß die bisherigen Informationen über Verbindungen und Clients dem Managementsystem danach nicht mehr zur Verfügung stehen, sofern sie nicht anderweitig gesichert wurden.

4.2 Das Logfile für Fehler

Im Gegensatz zum standardisierten *Common-Log-Format* für die Zugriffsdaten, gibt es kein Standardformat für das festhalten von Fehlerinformationen. Das in diesem Abschnitt analysierte Format wird vom *Apache HTTP-Server* verwendet.

Als erster Eintrag sind Datum und Uhrzeit des Fehlers vermerkt. Diese Einträge entsprechen natürlich dem geforderten Zeitpunkt des Fehlers.

An diesen Eintrag schließt sich die konkrete Fehlermeldung an. Alle Fehler, die nicht das Fehlschlagen einer Anfrage betreffen, fallen nach der in 3.5 getroffenen Einteilung in die Kategorie Sonstige, da alle genannten Fehlerfälle eine korrekte Antwort verhindern. Alle solchen Einträge sind nach folgendem Muster aufgebaut:

access to `<file>` failed for `<clienthost>`, reason: `<reason>`

Der Unterschied liegt in den Fehlerursachen. So kennzeichnen die Meldungen `File does not exist` und `Connection timed out` die entsprechenden Fehlertypen.
Die Angaben `user <name> not found`, sowie `user <name>: password mismatch` und `Client denied by server configuration` stehen dagegen für fehlende Zugriffsberechtigung.
Alle anderen Fehlerursachen fallen schließlich in die Kategorie Zugriffsfehler: Hier konnte der Server nicht auf das gewünschte Dokument zugreifen, obwohl es existiert. Beispiele dafür sind die Meldungen `Symbolic link not allowed`, oder `file permissions deny server access`.

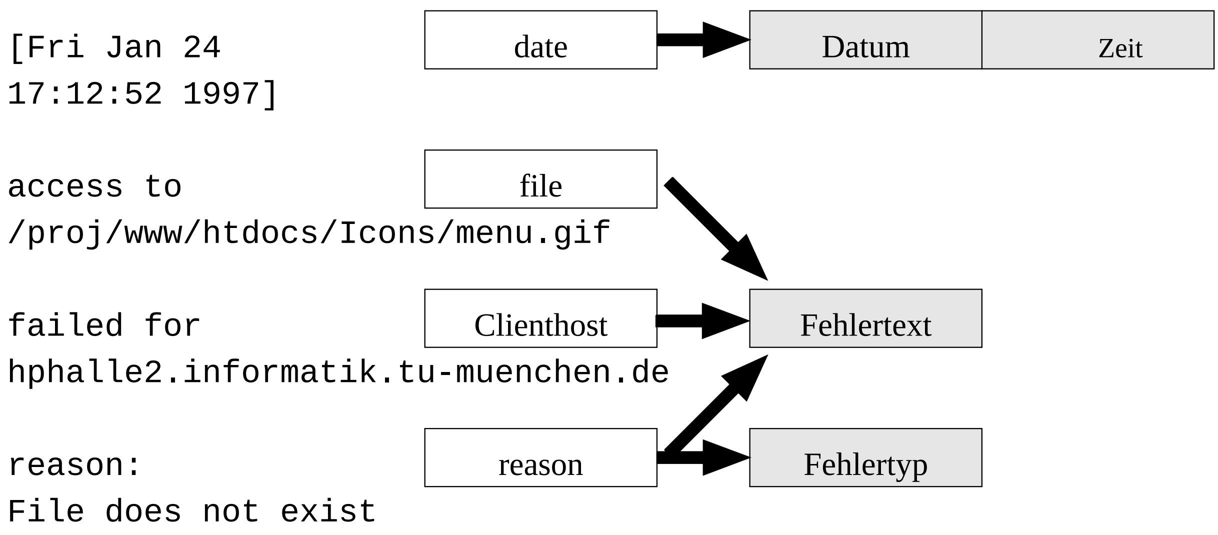

Abbildung 4.2: Auswertung eines Eintrags im Error Log

4.3 Die Konfigurationsdateien

Auch Die folgende Betrachtung der verschiedenen Konfigurationsdateien bezieht sich vollständig auf den *Apache HTTP-Server*. Die meisten Parameter stehen aber unter dem gleichen oder ähnlichen Namen auch auf nahezu allen anderen Servern zur Verfügung, da die zu lösenden Fragen ja unabhängig vom Produkt existieren.
Die drei Konfigurationsdateien `httpd.conf`, `srm.conf` und `access.conf` sind bei Apache in einem eigenen Verzeichnis (`conf/`) untergebracht. Die Zugriffskontrolle, die in letzterem definiert wird, kann zusätzlich für jedes Directory in dem sich Dokumente befinden über ein spezielles File `.htaccess` genauer festgelegt werden.

Da eine Vielzahl von Direktiven für die verschiedenen Konfigurationswünsche existieren, können hier nicht alle aufgeführt werden. Behandelt werden nur diejenigen, deren Verwendung bei der Realisierung der im vorigen Kapitel gestellten Anforderungen behilflich ist. Da sich diese ausschließlich in `httpd.conf` und `srm.conf` befinden ist die dritte Datei nur der Vollständigkeit halber aufgeführt. Theoretisch kann zwar jede Direktive in jeder Konfigurationsdatei stehen, aber in der Praxis wird die in Abb. 4.3 und den folgenden Abschnitten vorgestellte Trennung eigentlich immer eingehalten. Für eine weitergehende Analyse aller Möglichkeiten und der Durchführung solcher Änderungswünsche sei an dieser Stelle auf die Dokumentation des verwendeten Servers verwiesen.

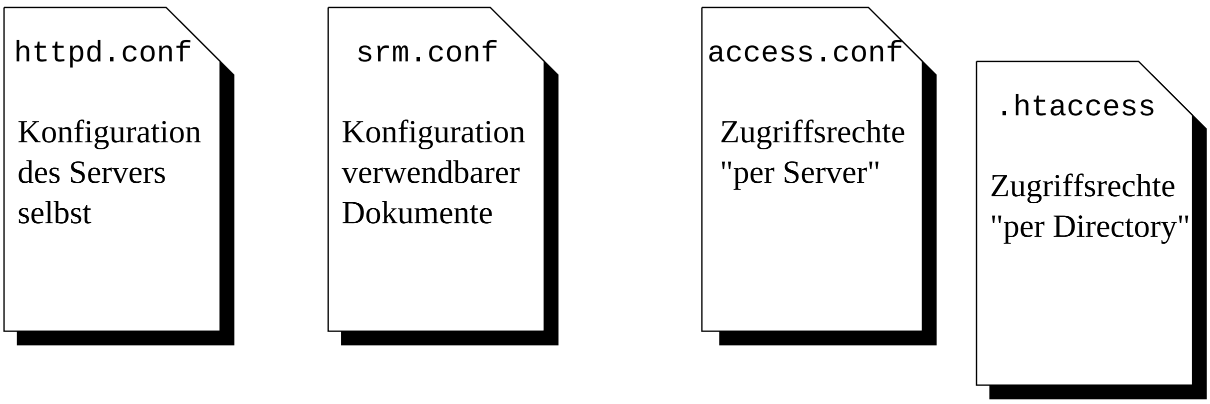

Abbildung 4.3: Aufgaben der Konfigurationsdateien

4.3.1 Konfiguration des Serverprogramms

Defaultmäßig wird beim Start des Servers die Datei `httpd.conf` ausgewertet, ein anderer Name kann jedoch auch durch eine Option in der Kommandozeile explizit angegeben werden. In dieser Datei befinden sich alle Parameter, die den Betrieb des Servers regeln. Darunter auch die folgenden, die von besonderem Interesse sind:

- `DocumentRoot`
  Default: `/usr/local/etc/httpd/htdocs`; Damit wird das Verzeichnis angegeben, von dem ausgehend der Server seine Dokumente findet.

  Diese Angabe kann verwendet werden, um die Dokumente zu lokalisieren und die Umsetzungen zwischen URL und absolutem Dateinamen zu realisieren.

- `ServerRoot`
  Default: `/usr/local/etc/httpd`; Damit wird das Verzeichnis spezifiziert, in dem sich der Server befindet.

  Relativ zu diesem Verzeichnis liegen die Unterverzeichnisse `conf/` und `logs/`, die die Konfigurations- und Logdateien enthalten.

- `Port`
  Default: 80; Portnummer, die der Server verwendet. 80 ist der Standardport für HTTP. Kann dieser Port nicht verwendet werden (unter UNIX ist die Verwendung von Ports unter 1024 Benutzern mit root-Berechtigung vorbehalten) werden meist die Ports 8000 oder 8080 verwendet.

  Dieser Eintrag ist wie die beiden folgenden direkt einem der Punkte aus Kapitel 3.4 zuzuordnen (hier: Portnummer).

- `BindAddress`
  Default: * (alle IP-Adressen); Angabe einer IP-Adresse, eine volle Internet-Domain, oder eines * für alle IP-Adressen des Rechners (falls der Servermaschine mehrere Adressen zugeordnet sind.)

  Ist hier eine eindeutige IP-Adresse oder ein Domainname spezifiziert, so ist dies der Name der Servers. Bei Verwendung mehrerer IP-Adressen für einen Server müssen die Namen auf einem anderen Wege ermittelt werden.

- `ServerAdmin`
  e-mail des Webmasters;

  Entspricht dem Administrator aus 3.4. Diese Mailadresse kann auch verwendet werden, um im Fehlerfalle Benachrichtigungen zu übermitteln.

- `TimeOut`
  Default: 1200 (Zeit in Sekunden); Begrenzt die maximale Zeitdauer einer Verbindung. Ist diese Zeit überschritten, wird die Verbindung vom Server beendet.

  Im Abschnitt 3.4.1 wird die maximale Verbindungsdauer erläutert, dies ist der entsprechende Konfigurationsparameter dazu.

- `MaxRequestsPerChild`
  Default: 0 (keine Begrenzung); Höchstzahl von Anfragen, die ein einzelner Prozeß bearbeiten darf. Nach Überschreitung dieser Zahl terminiert er. Eine Null als Wert bedeutet dabei, daß eine solche Grenze nicht beachtet werden muß.

  Eine deutlich zu niedrige Grenze verursacht bei vielbeschäftigten Servern unter Umständen unnötige Zusatzlast, da häufig neue Prozesse kreiert werden müssen. Andererseits verhindert ein Abschalten dieser Lebensdauerbegrenzung eine Selbstregulierung der Anzahl von Prozessen bei sinkender Last.

Die restlichen Punkte bieten einen Ansatz zur Steuerung der Anzahl von Serverprozessen, wie in 3.4.2 gefordert.

- `MaxClients`
  Default: 150; Maximale Anzahl von gleichzeitig Anfragen durch Clients, die bearbeitet werden können.

  Effektiv ist dies die maximale Anzahl von Serverprozessen, denn wenn nicht mehr gleichzeitig Anfragen bearbeitet werden, werden auch nicht mehr Prozesse zu diesem Zweck gestartet.

- `MaxSpareServers`
  Default: 10; Maximale Anzahl von unbeschäftigten Serverprozessen. Bearbeiten mehr als diese Zahl von Prozessen im Moment keine Anfragen, werden soviele beendet, bis diese Anzahl eingehalten wird.

  Kann bei stark frequentierten Servern erhöht werden, um plötzlich auftretende Spitzenlasten besser abzufangen. Bei schwach genutzten Servern dagegen, belastet ein hoher Wert den Rechner erheblich, da ständig zuviele Prozesse existieren.

- `MinSpareServers`
  Default: 5; Minimale Anzahl von unbeschäftigten Serverprozessen. Bearbeiten weniger als diese Zahl von Prozessen im Moment keine Anfragen, werden soviele neue gestartet, bis diese Anzahl erreicht wird.

  Ein zu niedriger Wert kann die Leistungsreserven stark beeinträchtigen, da bei mehreren gleichzeitigen Anfragen viel Zeit verloren geht, bis genügend Serverprozesse neu gestartet werden. Andererseits vergeudet ein hoher Wert entsprechen viele Ressourcen

4.3.2 Konfiguration des Dokumentangebots

Die Datei `conf/srm.conf` (relativ zum Pfad des Servers) beherbergte ursprünglich nur solche Direktiven, die logisch weder zur Konfiguration des Serverprogramms, noch zur Zugriffsregelung gehören. Im wesentlichen sind dies solche, die die Spezifikation von Dokumenten betreffen, die den Benutzern zur Verfügung stehen (Image-Maps, CGI-Skripten usw.). Lediglich Bestandteile also, die für die gestellten Anforderungen nicht herangezogen werden müssen. Inzwischen ist allerdings die Verwendung aller Konfigurationsmöglichkeiten in allen Konfigurationsdateien zulässig, wovon aber kaum Gebrauch gemacht wird, zumindest nicht dann, wenn die Einstellungen überschaubar bleiben sollen.

Die für die aus den Managementszenarien abgeleiteten Anforderungen notwendigen Parameter wurden bereits vollständig im vorigen Abschnitt erläutert. Unter den für gewöhnlich in dieser Datei festgelegten Werten befinden keine weiteren, die von Relevanz wären. Um jedoch die Möglichkeit zu behalten, alle genannten Parameter auch hier zu belegen, muß diese Datei ebenso wie `http.conf` ausgewertet werden.

4.3.3 Konfiguration der Zugriffsbeschränkung

Das Management der Zugriffsrechte für Dokumente ist ein interessantes Gebiet. Vor allem hinsichtlich der fortschreitenden kommerziellen Nutzung des WWW und firmeninternen Intranets mit verschiedenen Hierarchieebenen etc. gewinnt diese Thematik ständig an Bedeutung. Sie steht jedoch nicht im Fokus der vorliegenden Aufgabenstellung und würde auch den Arbeitsumfang überschreiten.

Wie bereits in Abschnitt 3.3 angesprochen wäre dafür eine Datenbankanbindung eine Lösungsmöglichkeit. Die Verwaltung der Zugriffsrechte und autorisierter Clients könnte so optimale Unterstützung erfahren. Angeboten werden soll hier nur eine Analyse der Zugriffe authentifizierter Benutzer im Rahmen der allgemeinen Möglichkeiten, nicht aber die Zugriffsregelungen selbst.

Im weiteren wird davon ausgegangen, daß sich in der Konfigurationsdatei `access.conf` keiner der benutzten Parameter befindet. Falls doch, sollte es für den Verantwortlichen im Grunde keine Schwierigkeit darstellen, die entsprechenden Teile in eine der beiden Dateien zu verschieben, die ausgewertet werden.

4.4 Sonstige Informationen

Ein Webserver ist kein abstraktes Etwas, sondern ganz gewöhnliche Software, die neben anderen Anwendungen auf einem Computer abläuft. Als Grundlage ist dazu ein Betriebssystem nötig und es können auch andere Programme zur Informationsgewinnung eingesetzt werden. Im folgenden wird betrachtet, was dadurch für das Management eines solchen Servers bereitgestellt werden kann.

An vorderster Stelle steht hier die Schnittstelle des Betriebsystems zum Filesystem des Rechners. Vom `DocumentRoot`-Pfad (siehe in 4.3.1) ausgehend sind alle Dateien, die als Dokumente angefordert werden können erreichbar. Dabei finden sich nahezu alle im Abschnitt 3.2.1 geforderten Punkte. Im einzelnen sind dies: Name, Pfad, letztes Veränderungsdatum (als Maß für die Aktualität), Größe und Typ der Dateien. Zusammen mit dem Servernamen ergeben Pfad- und Dateiname die URL des Dokuments.

Es stehen auch die jeweiligen Dokumente zur Verfügung, um daraus direkt Informationen zu gewinnen, wie beispielsweise den Titel. Dazu muß der Kopf (Head) der jeweiligen Dokumente durchsucht werden. Der Titel findet sich zwischen den entsprechenden Tags:

`<Title />` **Dokumenttitel** `</ Title>`

Die im Abschnitt 3.2.2 geforderten Links lassen sich ebenfalls durch die Durchsuchung der Dokumente finden. Zum Verfolgen und Testen der Linkstrukturen wurde in einem früheren Fortgeschrittenenpraktikum [Sch96b] bereits ein Werkzeug entwickelt, das zur Verwendung verfügbar ist. Als Programmiersprache wurde auch hier Perl verwendet, weshalb die bereits weiter oben gestellten Überlegungen zur Portabilität übernommen werden können. Dieses Werkzeug liefert Logdateien über die getesteten Links, die ausgewertet werden müssen. Genaueres dazu findet sich in [ES96], wo exakt diese Aufgabenstellung zur Bearbeitung kam.

Der Aufstellungsort des Rechners aus 3.4, auf dem ein Webserver läuft, kann offensichtlich nicht vom Managementsystem ermittelt werden. Diese Information muß vom Administrator selbst eingefügt werden.

4.5 Einschränkungen

Was jetzt noch fehlt, um alle Anforderungen zu erfüllen ist die Dauer der einzelnen Verbindungen, die auch die Grundlage zur Berechnung einer durchschnittlicher Verbindungsdauer bildet, und die Antwortzeit vom Eingang der Anforderungen bis zum Beginn der Übertragung. Diese beiden Werte können nicht auf einfache Weise ermittelt werden, hier wäre es notwendig, das Programm des Servers selbst über die zur Verfügung stehende Schnittstelle (API) zu erweitern. Erforderlich zur Bestimmung dieser Zahlen sind die Zeitpunkte des Eingangs einer Anfrage, sowie dem Beginn und Ende der zugehörigen Antwort.

Aufgrund des hohen Aufwands zur Gewinnung dieser beiden Werte über eine programmtechnische Anbindung des Managementsystems an die Server-API wird in der vorliegenden Arbeit auf die Durchführung verzichtet. Generell wäre es allerdings durchaus machbar, und in Hinblick auf die evtl. zusätzlichen Möglichkeiten die sich aus einer direkten Verbindung ergeben würden auch sinnvoll.

Als größte Einschränkung muß mit der Zielsetzung für ein allgemeingültiges WWW-Management die intensive Nutzung propietärer Lösungen betrachtet werden. Beschränkt man sich auf standardisierte Informationen bleiben im wesentlichen nur Logdateien im Common-Log-Format und die Dokumente des Servers. Für ein wirklich umfassendes Managementkonzept ist dies zu wenig.

Kapitel 5

Entwurf eines Objektmodells

In den beiden vorhergegangenen Kapiteln wurden zuerst die Anforderungen an das WWW-Management untersucht. Danach wurden die Möglichkeiten diese Anforderungen zu realisieren einer genaueren Analyse unterzogen. Das angestrebte Ziel war dabei, die daraus gewonnenen Kenntnisse in ein Objektmodell umzusetzen, was sich nun unmittelbar in diesem Kapitel anschließt. Von diesem Modell ausgehend, können dann im weiteren Verlauf die entsprechenden IDL-Schnittstellen bestimmt werden, um darauf basierend das geplante Managementsystem auf CORBA-Basis zu implementieren.

Begonnen wird bei der Modellierung mit einer direkten Übertragung der Anforderungen in Objektklassen. Diese werden dann in zwei weiteren Schritten nach objektorientierten Gesichtpunkten bis zu fertigen Entwurf weiterentwickelt.

5.1 Direkte Übertragung aus den Anforderungen

In diesem ersten Schritt werden die Punkte aus den Anforderungen direkt in ein sehr einfaches Objektmodell übertragen. Die extrahierten Managementinformationen werden dabei zu Attributen und -funktionalitäten zu Methoden

Begonnen wird mit der Festlegung der wohl entscheidendsten Klasse, des Servers selbst. In Abschnitt 3.4 wurden als grundlegende Attribute die Serveradresse, die Portnummer, die physikalische Position des Betriebsrechners und der zuständige Administrator identifiziert, sowie das Starten und Beenden als Methoden. Dies läßt sich wie in Abb. 5.1 geschehen, in eine Klassendeklaration umsetzen.

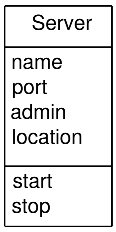

Abbildung 5.1: Die Klasse "Server"

Die in den Analysen über Last und Leistung, sowie Quality of Service beschriebenen Anforderungen lassen sich in identischer Weise in eigene Klassen Überführen. Jedoch sticht dabei sofort ins Auge, das all diese Punkte eigentlich nur Attribute sind, die die Eigenschaften des Servers näher charakterisieren, also auch in dessen Klasse aufgenommen werden sollten. Sie wird also wie in Abb 5.2 um diese Attribute erweitert.

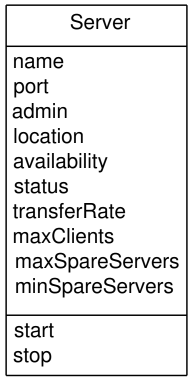

Abbildung 5.2: Die Klasse "Server" um zusätzliche Attribute erweitert

Bei den Verbindungen des Servers dagegen, sind andere Zusammenhänge gegeben. Sie werden vom Server aufgebaut (bzw. beantwortet), sind gewissermaßen ein Teil von ihm. Ein solcher Zusammenhang wird durch eine eigenständige Klasse und einer Enthaltenseinsrelation zwischen den beiden dargestellt, wie in Abb. 5.3 zu sehen.
Wird für jede Verbindung ein eigenes Objekt erzeugt, so genügt der Name des angeforderten Dokuments je Zugriff, um die Anzahl der Zugriffe auf jedes Dokument zu bestimmen.

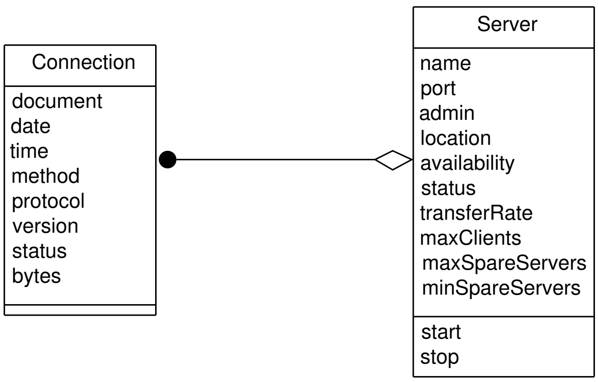

Abbildung 5.3: Die Klasse "Connection" mit der übergeordneten Serverklasse

Analog dazu läßt sich eine Klasse für die Dokumente einführen. Auch sie sind ein Bestandteil des Servers. Die Links zwischen den Dokumenten sind wiederum ein nähere Beschreibung der Dokumente und werden mit den damit zusammenhängenden Methoden in diese Klasse integriert.

Das Modell zu diesen beiden Klassen ist in 5.4 zu betrachten. (Die Tatsache, daß ein Server mehrere Dokumente enthalten kann, wird übrigens durch den ausgefüllten Punkt am Verbindungsende der Dokumentklasse dargestellt, die Enthaltenseinsrelation durch die Raute am Ende der Serverklasse)

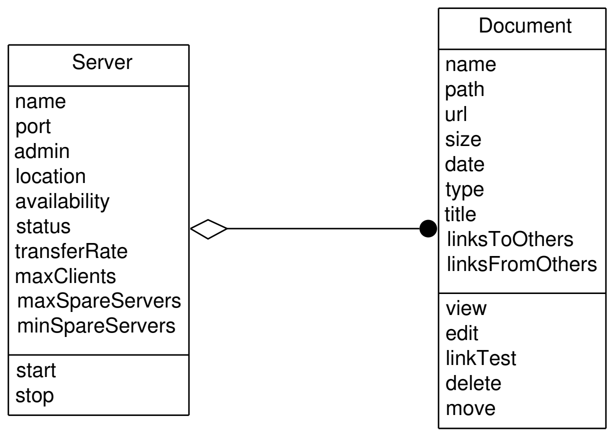

Abbildung 5.4: Auch die Dokumente sind ein Bestandteil des Servers

Wie in Abb. 5.5 gezeigt, gestaltet sich auch die Definition der noch fehlenden Klassen für die Clients und die aufgetretenen Fehler aus den geforderten Attributen recht einfach.

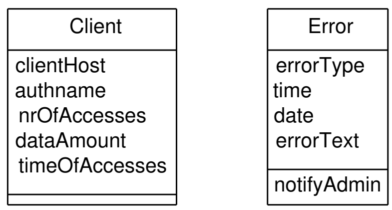

Abbildung 5.5: Die Klassen “Client“ und “Error“

5.2 Verfeinerung des bisherigen Modells

Im bisherigen Ansatz sind noch keinerlei Vererbungshierarchien enthalten und die Klassenaufteilung weist auch noch einige grobe Mängel auf. Diese Punkte sollen in einem zweiten Schritt verbessert werden.

Die Klasse `Document` spiegelt in der bisherigen Form die Sichtweise eines Dokuments als Datei nicht realitätsgemäß wieder. Tatsache ist, daß ein Dokument einen Spezialfall einer Datei darstellt, denn es gibt ja auch Dateien, die keine Dokumente sind. Demnach sollte diese Klasse auch von einer entsprechenden Superklasse abstammen, die die zu Dateien gehörenden Bestandteile der bisherigen Klasse enthält, wie etwa den Dateinamen, oder das Erstellungdatum.

Da die Methoden zum Betrachten und Editieren eines Dokuments, sowie das Testen von Links durch eigene Programme realisiert wird, setzten sie erst auf der Subklasse `Document` auf, während das Löschen durch das Entfernen einer Datei realisiert wird und daher von `File` abstammen muß. Die Funktionalität für `move` kann durch das Erzeugen einer neuen Datei und Löschen der alten erreicht werden. Beim Löschen und Verschieben von Dokumenten, muß die Konsistenz der Link aufrecht erhalten werden, dazu ist es nötig, die in `linksFromOthers` angegebenen Dokumente zu aktualisieren, oder wenigstens den Verantwortlichen davon in Kenntnis zu setzen, welche Dokumente dies sind.

Ein weiterer Schwachpunkt im bisherigen Entwurf ist der Eintrag von `type`. In objektorientierten Modellen werden solche Kategorien durch weitere Subklassen wiedergegeben. Dabei kann auch der Titel, der nur bei statischen Dokumenten vorkommt, in die entsprechende Subklasse mitgenommen werden.

Insgesamt ergibt sich damit die Darstellung aus Abbildung 5.6:

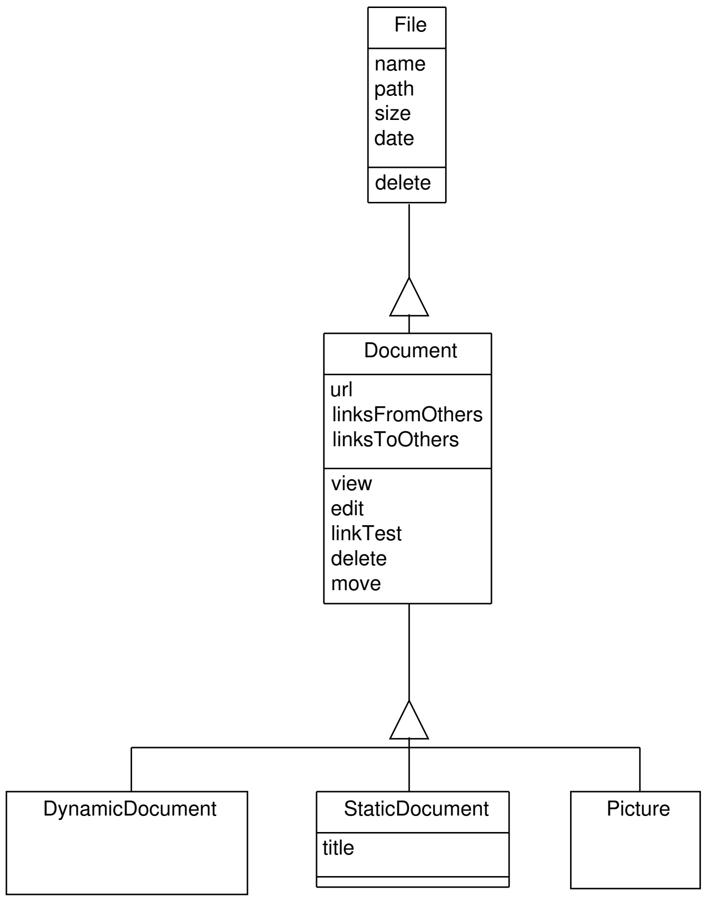

Abbildung 5.6: Aufteilung der Dokumentklasse in eine Vererbungshierarchie

Die meisten Attribute der Serverklasse stammen wie in Kapitel 4 gesehen aus den Konfigurationsdateien. Daher ist es sinnvoll, für diese Dateien eine eigene Klasse, mit genau den von daher stammenden Attributen einzuführen.

Auch weitere der geforderten Angaben stammen aus den Konfigurationsdateien und können ebenfalls in dieser Klasse eingebracht werden (z.B. Lebensdauer eines Prozesses als `maxNumber Requests`).

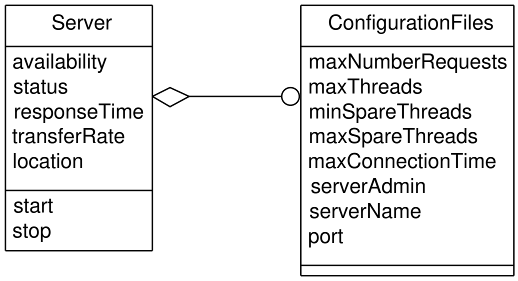

Abbildung 5.7: Auslagerung der Konfigurationsdaten

Nicht in der Abbildung 5.7 wiedergegeben ist, daß sich diese Klasse `KonfigurationFiles` von `File` abstammt, da es sich ja hier, wie der Name andeutet, ebenfalls um Dateien handelt.

Bei den Clients ist es gewünscht, die autorisierten Benutzer eigens zu Erfassen. Zu diesem Zweck diente in dieser Klasse der `authname`. Im objektorientierten Sinne ist es wesentlich besserer Stil, dafür eine eigene Subklasse einzuführen, mit eben diesem Namen als Attribut Ebenso wäre, wie in 5.8 abgebildet, eine zweite Subklasse für nicht autorisierte Benutzer sinnvoll.

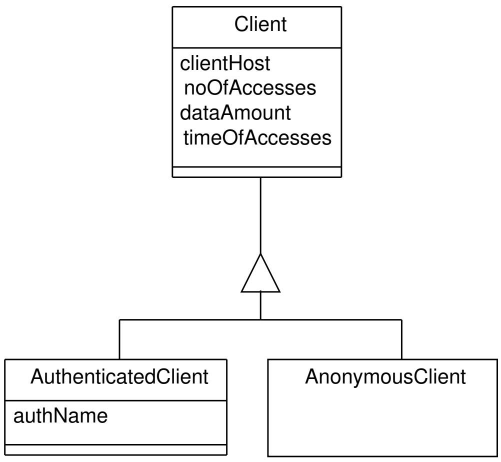

Abbildung 5.8: Die Vererbungshierarchie von "Client" zur Klassifizierung autorisierter Clients

Ähnliches gilt auch für die Fehlerklasse. Hier wird mit einem Attribut `errorType` zwischen verschiedenen Fehlertypen unterschieden. Eine Vererbungshierarchie ist an dieser Stelle wohl eher angebracht.

Wie bei den Realisierungsmöglichkeiten erläutert, besitzen die genauer kategorisierten Fehler die Gemeinsamkeit, daß dabei ein Dokument nicht korrekt, oder überhauptnicht übertragen wurde. Hier kann man eine weitere Zwischenklasse einführen, die als Attribut genau diesen Namen enthält. Die weiteren Fehler fügen sich anschließend als Subklassen an.

Damit wird aus der vormals primitiven Klasse folgende wesentlich verbesserte Struktur aus Abbildung 5.9:

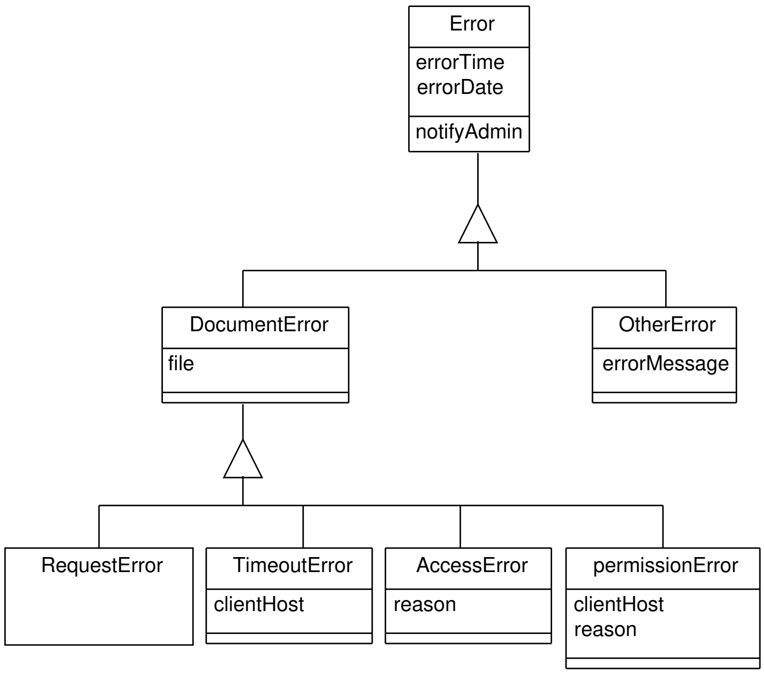

Abbildung 5.9: Die Klassen der “Error“-Hierarchie

5.3 Vervollständigung des Objektmodells

Inzwischen ist das Modell ziemlich weit gediehen. Im letzten Schritt soll in erster Linie das Zusammensetzen zu einer Gesamteinheit erfolgen. Vorher gilt es aber noch einige Ungereimtheiten auszuräumen und fehlende Details zu ergänzen.

Das größte Manko beim jetzigen Stand der Dinge liegt bei den Verbindungen. Die Verbindungen sind eigentlich keine eigenständige, für sich alleine existierende Klasse, sondern entstehen nur, wenn ein Client über einen Server auf ein Dokument zugreift. Dieser Sachverhalt wird in OMT durch eine Assoziationsklasse modelliert und ist in Abbildung 5.10 dargelegt.

Da durch die bei den Verbindungen entstehenden Objekte der Klasse `Connection` ermittelt werden kann, wie oft und wann ein Client Anfragen gestellt hat und welche Datenmengen dabei umgesetzt wurden, werden diese Attribute beim Client nicht mehr benötigt.

Eine 1:1 Beziehung zwischen `Connection` und `Document` stellt außerdem die bestehende Verknüpfung erst korrekt dar. Gegenüber einem entsprechendem Attribut in `Connection` eröffnet es daneben eine wesentlich bequemere Möglichkeit zur dateibezogenen Auswertung der Verbindungdaten.

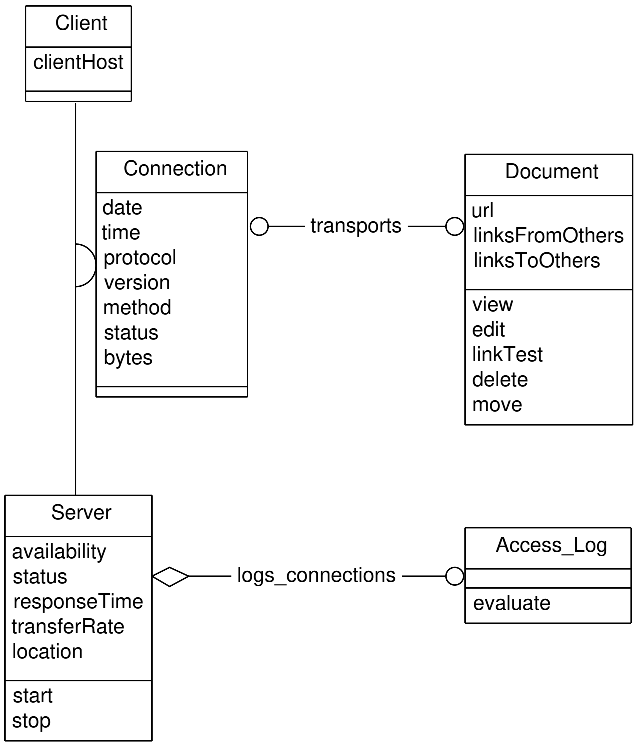

Abbildung 5.10: "Connection" als Assoziationsklasse zwischen "Client" und "Server"

Bisher noch nicht eingeführt wurde die in dieser Abbildung verwendete Klasse `Access_Log`. Hier werden vom Server die Daten der Zugriffe protokolliert.
Daneben existiert noch ein weiteres Logfile, in Abbildung 5.11 durch die Klasse `Error_Log` dargestellt, in dem durch den Server die Fehlerdaten festgehalten werden. Diese Dateien bilden nach Kapitel 4 durch ihre Auswertung eine wesentliche Grundlage zum Management eines Webservers und dürfen daher auch im Objektmodell nicht fehlen.

Das Zurücksetzen für die beiden Logdateien ist über eine gemeinsame Superklasse `LogFile` modelliert, die von `File` abstammt.

5.4 Zusammenfassung

Nachdem nun alle relevanten Komponenten mit ihren Attributen und Methoden modelliert sind, fügt sich das Bild zu einer Einheit zusammen. Die zentrale Klasse ist der Server. Er enthält Dokumente, wird durch zu ihm gehörende Konfigurationsdateien gesteuert und erzeugt Logfiles über die einzelnen Zugriffe von Clients auf Dokumente und dabei aufgetretene Fehler und Probleme.

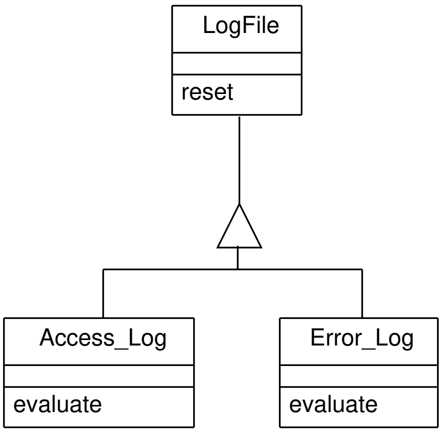

Abbildung 5.11: Der Klassenaufbau für die Logfiles

In den bisher abgebildeten Teilmodellen waren die Datentypen aufgrund der ständigen Weiterentwicklung von nebensächlicher Natur und wurden daher auch nicht angegeben. In der abschließenden Gesamtansicht 5.12 des Objektmodells wird dies nachgeholt.

Aus dem nun vorliegenden Modell lassen sich bei Bedarf die benötigten IDL-Schnittstellen [1] automatisch produzieren.

[1] Die von StP tatsächlich generierten IDL-Dateien sind im Anhang A angegeben

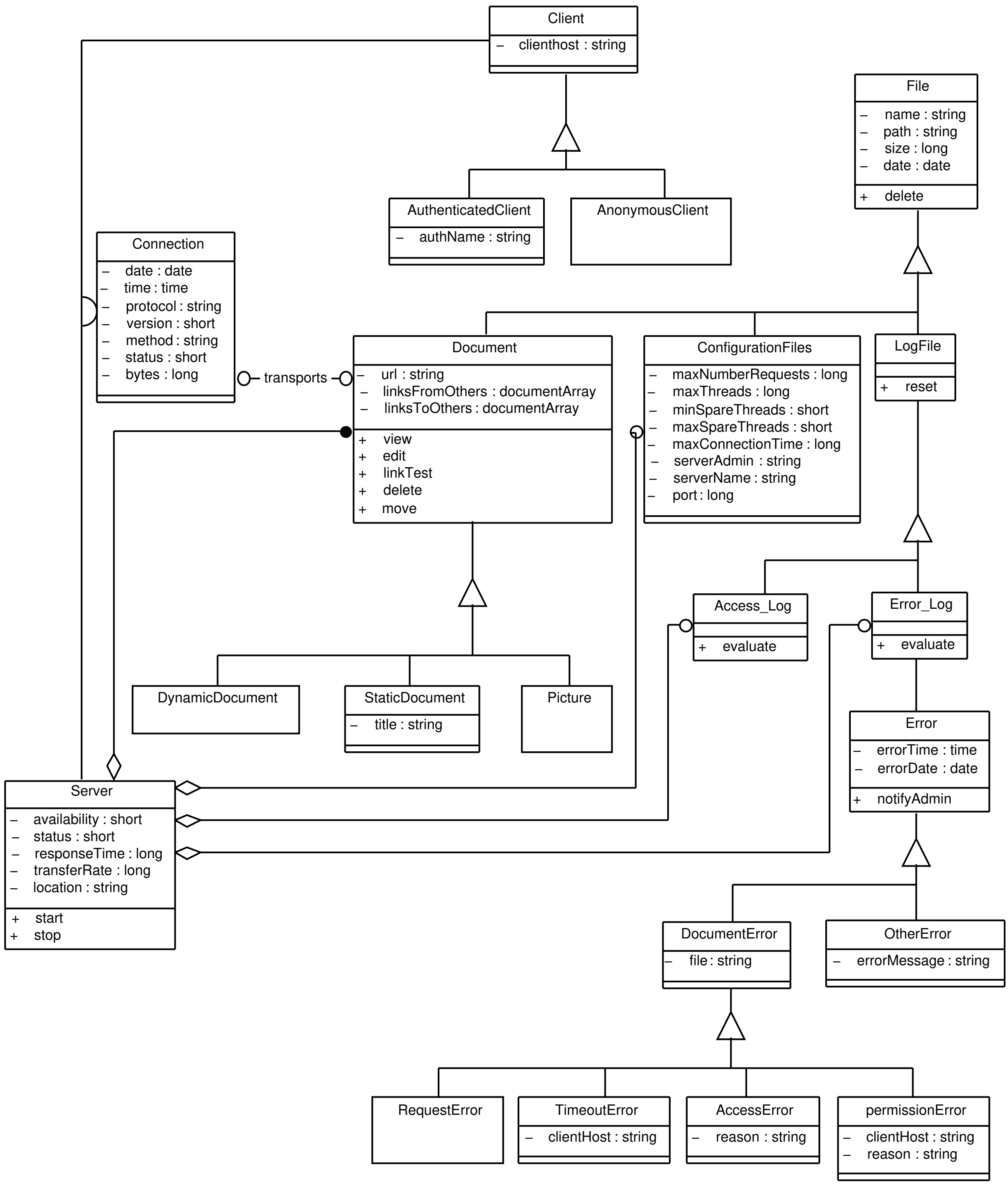

Abbildung 5.12: Das entstandene Objektmodell im Überblick

Kapitel 6

Implementierung

Nach der Erzeugung der IDL-Schnittstellen aus dem Objektmodell können die Objektklassen mit OrbixWeb, wie in Kapitel 2.4.3 beschrieben, direkt als Server in Java implementiert werden und stehen damit nach ihrer Registrierung zum Einsatz zur Verfügung.

Jedes solches Serverobjekt stellt ein real existierendes Objekt (ein *Managed Object*) dar, sei es ein Dokument, ein Eintrag im Logfile, oder der Server selbst. Aus diesem Grund ist es notwendig, die Objekte an dem Ort zu kreieren, an dem sich ihr reales Abbild befindet, also beispielsweise die Instanz einer Serverklasse dort, wo der Server tatsächlich läuft. Ansonsten wäre es etwas schwierig für den jeweiligen Agenten (das Serverobjekt), mit dem realen Gegenstück in Kontakt zu treten. Da diese Objekte ihre Methoden quasi "mitbringen" ist das die eigentliche Delegierung an sich: Funktionalität wird vom Manager hin zum Agenten verschoben, der in diesem Falle als *intelligenter Agent* bezeichnet wird.
Hierbei handelt es sich um statische Delegierung, da eine Änderung des Funktionsumfanges zur Laufzeit nicht möglich ist. Erst eine Veränderung des Modells und eine darauffolgende Neuimplementierung könnte den Funktionsumfang beeinflussen.

Sei der eigentlichen Manager im herkömmlichen Sinne, lediglich eine Bedienungsoberfläche, in welcher Form auch immer. Er läuft also dort, wo er vom Administrator gestartet wurde. Dynamische Delegierung kommt ins Spiel, wenn sogenannte *Mid-Level-Managers* zwischen den intelligenten Agenten und dem Manager geschaltet werden.
Ein solcher Mid-Level-Manager stellt ein über den ORB verschiebbares Modul dar, dem durch seine Implementierung ein gewisser Funktionsumfang zur Verfügung steht. Als Einteilung würden sich im konkreten Fall vier Module entsprechend den in Kapitel 3 festgestellten Bereichen anbieten: Zum Management der Dokumente, der Benutzer, des Serverbetriebs sowie der Fehler, wie in Abbildung 6.1 angedeutet.

In diesen Komponenten wird die Funktionalität erbracht, die nicht von den Agenten selbst erledigt wird, wie etwa die routinemäßige Kontrolle des Serverstatus mit anschließendem Neustart (der aber wiederum vom Serveragenten selbst durchgeführt wird) im Falle eines Absturzes.
Sie werden über eine Bedienungsoberfläche gesteuert und stellen über diese die gewünschte Information und Funktionalität zu Verfügung. Über den ORB ist der tatsächliche Ort der Ausführung völlig transparent und kann nach Belieben verändert werden.

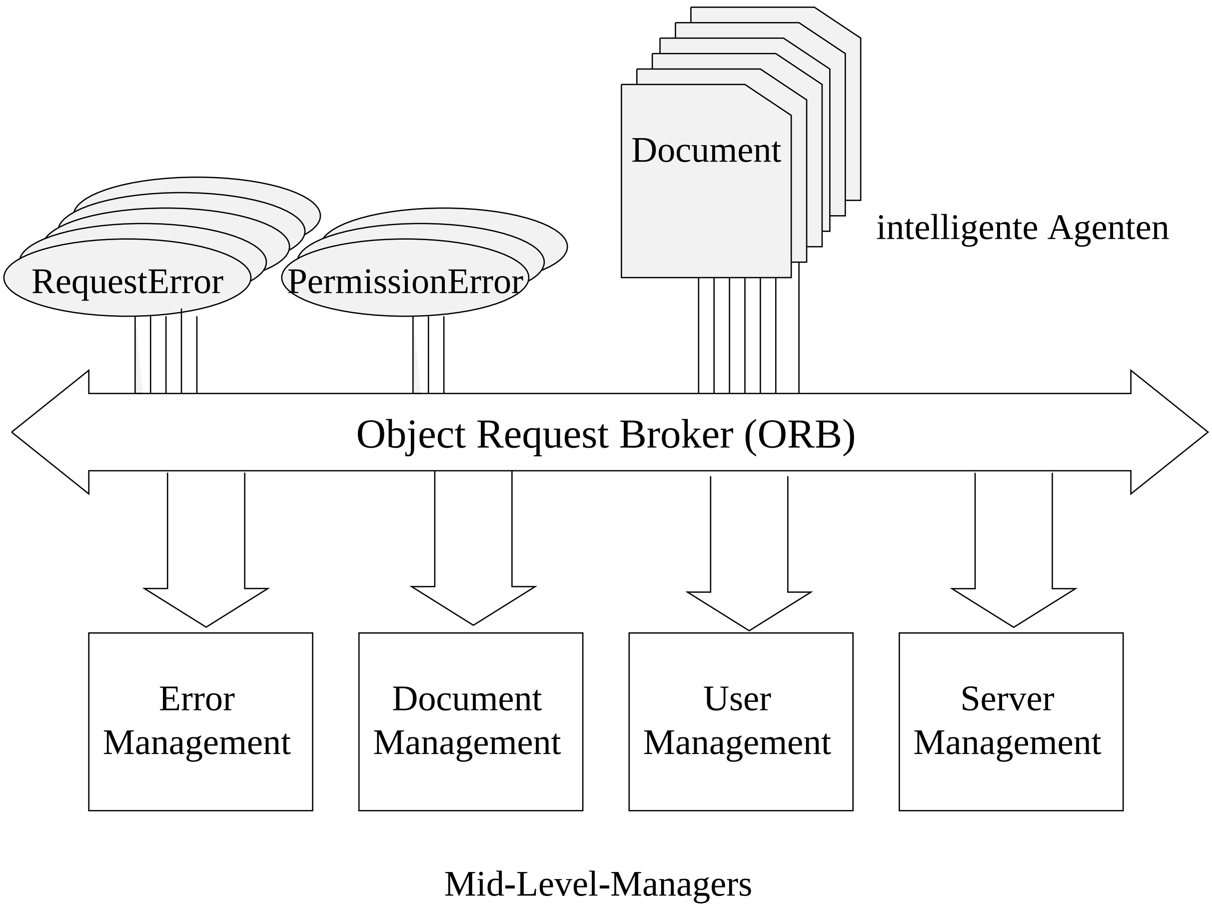

Abbildung 6.1: Realisierung des Managements von WWW-Servern nach dem MbD Paradigma auf der Basis von CORBA

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Anzahl der Benutzer und vor allem auch die der Anbieter im World-Wide-Web hat in den letzten Jahren stark zugenommen, ebenso wie der Umfang der einzelnen Angebote. Mit der Bemühung um ein qualitativ hochwertiges Angebot auf einem Webserver und einer zunehmenden Komplexität dieses Angebots, steigt auch der Verwaltungsaufwand der mit dem Betrieb verbunden ist stark an. Diese anfallenden Arbeiten sind meist typische Managementaufgaben, für die jedoch gegenwärtig kaum umfassende Konzepte existieren.

Daneben wurde im selben Zeitraum das Paradigma des *Management by Delegation* forciert, das eine Flexibilisierung der herkömmlichen Managementstrukturen verfolgt. Die Delegierung von Managementfunktionalität gestattet deren effiziente und zuverlässige Ausführung an dem unter der Berücksichtigung des Gesamtkontexts bestgeeignetsten Ort.
CORBA bietet sich als Basis hierfür förmlich an, da es für Interoperabilität von Anwendungen in verteilten und heterogenen Umgebungen konzipiert wurde. Damit läßt sich selbst dynamische Verlagerung von Funktionalität realisieren.

Das Ziel dieser Diplomarbeit war ein Konzept für das Management eines WWW-Servers nach dem MbD-Paradigma auf der Basis eines CORBA-konformen Object Request Brokers in der Programmiersprache Java. Hier wurde aufgrund der mangelnden Eignung der ursprünglich geplanten Kombination NEO/JOE von Sun, OrbixWeb von IONA als ORB verwendet.
Zur Gewinnung geeigneter Managementinformationen und -funktionalität wurde zuerst anhand typischer Szenarien eine Top-Down-Analyse der Anforderungen an das Management eines Webservers durchgeführt. Dabei ergab sich eine umfassende Sammlung von managementrelevanten Faktoren. Durch eine anschließende Bottom-Up-Analyse wurde untersucht, welche Möglichkeiten überhaupt zur Verfügung stehen, um das Management eines WWW-Servers erfolgreich zu bewerkstelligen. In erster Linie sind dies die Log- und Konfigurationsdateien des Servers, sowie Informationen, die aus den Dokumenten gewonnen werden können. Als Hauptproblem für ein allgemeine Lösung zeigen sich dabei fehlende Standards für Konfiguration und Error-Logging. Die parallel dazu durchgeführte Zusammenführung der beiden Analysen, ergab schließlich die Grundlage für das Managementkonzept.

Im nachfolgenden Kapitel wurde ein Objektmodell unter Zuhilfenahme des Sofwarepaktes *Software through Pictures* gestaltet. Dazu erfolgten mehrere Verfeinerungsschritte von der einfachen Umsetzung der gestellten Anforderungen, bis hin zu einem detaillierten, objektorientierten Modell eines Managementsystems. Die Implementierung wurde von StP durch die automatische Generierung von IDL-Schnittstellen aus dem erarbeiteten Objektmodell hervorragend unterstützt. Aufgrund der fortgeschrittenen Bearbeitungszeit konnte kein kompletter Prototyp mehr implementiert werden, sondern lediglich ein kleinerer Bestandteil, um die grundsätzliche Verfahrensweise zu demonstrieren.

Als Fazit läßt sich zum einen feststellen, daß CORBA tatsächlich sehr gut geeignet ist, um MbD-Konzepte zu realisieren. Andererseits ist ein wirklich umfassendes, allgemeines Konzept zum Management von Webservern wohl kaum realisierbar, sonder eher noch längere Zeit an proprietäre Lösungen gebunden.

7.2 Ausblick

Daß trotz eines bereits länger anhaltenden, starken Trends zur Nutzung des WWW immer noch keine wirklich weitgehenden allgemeinen Managementlösungen auf dem Markt verfügbar sind, zeigt deutlich, daß es sich dabei um ein nicht gerade triviales Problem handelt. Auch der Internet-Draft über eine WWW-MIB [CWK96] ist bisher nur einen kleinen Schritt in diese Richtung vorangekommen.

Für weitere Arbeiten auf diesem Gebiet steht Aufgrund der unvollendeten Implementierung wohl die Fertigstellung eines kompletten Prototypen an oberster Stelle.

Eine bessere Unterstützung der Benutzerverwaltung, in Kombination mit den in dieser Arbeit nahezu vollständig ausgeklammerten Sicherheitsaspekten wäre ebenfalls ein vielversprechender Ansatz.
Im Zuge verstärkter kommerzieller Nutzung, sind Sicherheitsfragen von Größter Wichtigkeit. Die Server bieten hier zum Zugriffsschutz weitgehend einheitliche Konzepte zur Regelung der Zugriffsrechte auf Dokumente (per Server - per Directory via `access.conf` bzw. `.htaccess` bei Apache) an, die der Server bei einer Anfrage auswertet. Das "Management" von solchen Berechtigungen wäre ein wichtiger Schritt, wenn Dokumente vor unberechtigtem Zugriff geschützt werden sollen.

Allgemein werden die Möglichkeiten zur Einrichtung autorisierter Benutzer relativ wenig genutzt. Es existieren zwar genügend Angebote, die nur für festgelegte Benutzergruppen verfügbar sind (z.B. Mitarbeiter, oder Anfragen innerhalb der eigenen Domain), aber erst durch geeignete Autentisierungsverfahren, beispielsweise mit Passwort über verschlüsselte Übertragung, ließen sich hier zusammen mit der eben erwähnten Regelung der Zugriffsrechte hochflexible Lösungen schaffen. Auch eine exakte benutzungsbezogene Abrechnung wäre durch entsprechende Datenbankanbindungen so realisierbar.

Was sich beim vorliegenden Entwurf noch besser ausbauen ließe, ist ein vollständiger Zugriff auf die Konfigurationsparameter, also auch jene, die hier nicht als managementrelevant erachtet wurden. Eine Erweiterung der entsprechenden Klasse bereitet keine Probleme und auch eine Anpassung auf die Parameter anderer Server als Apache sollte nach kurzer Einarbeitungsphase eigentlich keine Schwierigkeiten bereiten.

Zum Abschluß sei noch zu Bedenken gegeben, daß einfache, auch für “Laien“ verwendbare Managementsysteme den Boom in Sachen WWW vermutlich weiter verstärken würden, vor allem auf der Anbieterseite. Und die Qualität mancher Angebote würde sich sicherlich auch merkbar verbessern, damit man etwas weniger häufig auf solche Fehlermeldungen trifft, wie: *Sorry! Server down, try again later.*

Anhang A

Die IDL-Schnittstellen des Objektmodells

AccessError.idl

```
// StP -- created on Wed May 14 14:06:26 1997 for euba@sunhegering2 from system webman

#ifndef _AccessError_idl_
#define _AccessError_idl_

// stp class declarations
interface AccessError;
// stp class declarations end

// stp auto-include 5739
#include "DocumentError.idl"

// stp auto-include end

// stp class definition 5739
interface AccessError : DocumentError
{
// stp class members
    readonly attribute string reason;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Access_Log.idl

```
// StP -- created on Wed May 14 14:06:27 1997 for euba@sunhegering2 from system
webman

#ifndef _Access_Log_idl_
```

```
#define _Access_Log_idl_

// stp class declarations
interface Access_Log;
// stp class declarations end

// stp auto-include 7112
#include "LogFile.idl"

// stp auto-include end

// stp class definition 7112
interface Access_Log : LogFile
{
// stp class members
    void  evaluate();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

AnonymousClient.idl

```
// StP -- created on Wed May 14 14:06:26 1997 for euba@sunhegering2 from system webman

#ifndef _AnonymousClient_idl_
#define _AnonymousClient_idl_

// stp class declarations
interface AnonymousClient;
// stp class declarations end

// stp auto-include 7159
#include "Client.idl"

// stp auto-include end

// stp class definition 7159
interface AnonymousClient : Client
{
// stp class members
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

AuthenticatedClient.idl

```
// StP -- created on Wed May 14 14:06:27 1997 for euba@sunhegering2 from system webman

#ifndef _AuthenticatedClient_idl_
#define _AuthenticatedClient_idl_

// stp class declarations
interface AuthenticatedClient;
// stp class declarations end

// stp auto-include 7154
#include "Client.idl"

// stp auto-include end

// stp class definition 7154
interface AuthenticatedClient : Client
{
// stp class members
    readonly attribute string authName;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Client.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _Client_idl_
#define _Client_idl_

// stp class declarations
interface Client;
// stp class declarations end

// stp class definition 7140
interface Client
{
// stp class members
    readonly attribute string clienthost;
// stp class members end
};
// stp class definition end

// stp footer
```

```
#endif
// stp footer end
```

ConfigurationFiles.idl

```
// StP -- created on Wed May 14 14:06:28 1997 for euba@sunhegering2 from system webman

#ifndef _ConfigurationFiles_idl_
#define _ConfigurationFiles_idl_

// stp class declarations
interface ConfigurationFiles;
// stp class declarations end

// stp class definition 10193
interface ConfigurationFiles
{
// stp class members
    attribute long maxConnectionTime;
    attribute long maxNumberRequests;
    attribute short maxSpareThreads;
    attribute short minSpareThreads;
    attribute long maxThreads;
    attribute long port;
    attribute string serverName;
    attribute string serverAdmin;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Connection.idl

```
// StP -- created on Wed May 14 14:06:28 1997 for euba@sunhegering2 from system webman

#ifndef _Connection_idl_
#define _Connection_idl_

// stp class declarations
interface Connection;
// stp class declarations end

// stp class definition 8941
interface Connection
{
// stp class members
```

```
    readonly attribute short status;
    readonly attribute short version;
    readonly attribute string method;
    readonly attribute string protocol;
    readonly attribute date date;
    readonly attribute time time;
    readonly attribute long bytes;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Document.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _Document_idl_
#define _Document_idl_

// stp class declarations
interface Document;
// stp class declarations end

// stp auto-include 105
#include "File.idl"

// stp auto-include end

// stp class definition 105
interface Document : File
{
// stp class members
    readonly attribute string url;
    readonly attribute documentArray linksFromOthers;
    readonly attribute documentArray linksToOthers;
    void view();
    void delete();
    void edit();
    void linkTest();
    void move();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

DocumentError.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _DocumentError_idl_
#define _DocumentError_idl_

// stp class declarations
interface DocumentError;
// stp class declarations end

// stp auto-include 10212
#include "Error.idl"

// stp auto-include end

// stp class definition 10212
interface DocumentError : Error
{
// stp class members
    readonly attribute string file;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

DynamicDocument.idl

```
// StP -- created on Wed May 14 14:06:27 1997 for euba@sunhegering2 from system webman

#ifndef _DynamicDocument_idl_
#define _DynamicDocument_idl_

// stp class declarations
interface DynamicDocument;
// stp class declarations end

// stp auto-include 244
#include "Document.idl"

// stp auto-include end

// stp class definition 244
interface DynamicDocument : Document
{
// stp class members
// stp class members end
```

```
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Error.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _Error_idl_
#define _Error_idl_

// stp class declarations
interface Error;
// stp class declarations end

// stp class definition 1734
interface Error
{
// stp class members
    readonly attribute time errorTime;
    readonly attribute date errorDate;
    void notifyAdmin();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Error_Log.idl

```
// StP -- created on Wed May 14 14:06:26 1997 for euba@sunhegering2 from system webman

#ifndef _Error_Log_idl_
#define _Error_Log_idl_

// stp class declarations
interface Error_Log;
// stp class declarations end

// stp auto-include 7111
#include "LogFile.idl"

// stp auto-include end
```

```
// stp class definition 7111
interface Error_Log : LogFile
{
// stp class members
    void evaluate();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

File.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _File_idl_
#define _File_idl_

// stp class declarations
interface File;
// stp class declarations end

// stp class definition 104
interface File
{
// stp class members
    attribute string name;
    attribute string path;
    attribute long size;
    attribute date date;
    void delete();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Log_File.idl

```
// StP -- created on Wed May 14 14:06:25 1997 for euba@sunhegering2 from system webman

#ifndef _LogFile_idl_
#define _LogFile_idl_

// stp class declarations
```

```
interface LogFile;
// stp class declarations end

// stp auto-include 7108
#include "File.idl"

// stp auto-include end

// stp class definition 7108
interface LogFile : File
{
// stp class members
    void reset();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

OtherError.idl

```
// StP -- created on Wed May 14 14:06:26 1997 for euba@sunhegering2 from system webman

#ifndef _OtherError_idl_
#define _OtherError_idl_

// stp class declarations
interface OtherError;
// stp class declarations end

// stp auto-include 5717
#include "Error.idl"

// stp auto-include end

// stp class definition 5717
interface OtherError : Error
{
// stp class members
    readonly attribute string errorMessage;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Picture.idl

```
// StP -- created on Wed May 14 14:06:27 1997 for euba@sunhegering2 from system webman

#ifndef _Picture_idl_
#define _Picture_idl_

// stp class declarations
interface Picture;
// stp class declarations end

// stp auto-include 242
#include "Document.idl"

// stp auto-include end

// stp class definition 242
interface Picture : Document
{
// stp class members
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

RequestError.idl

```
// StP -- created on Wed May 14 14:06:28 1997 for euba@sunhegering2 from system webman

#ifndef _RequestError_idl_
#define _RequestError_idl_

// stp class declarations
interface RequestError;
// stp class declarations end

// stp auto-include 3306
#include "DocumentError.idl"

// stp auto-include end

// stp class definition 3306
interface RequestError : DocumentError
{
// stp class members
// stp class members end
};
// stp class definition end
```

```
// stp footer
#endif
// stp footer end
```

Server.idl

```
// StP -- created on Wed May 14 14:06:28 1997 for euba@sunhegering2 from system webman

#ifndef _Server_idl_
#define _Server_idl_

// stp class declarations
interface Server;
// stp class declarations end

// stp class definition 106
interface Server
{
// stp class members
    readonly attribute long responseTime;
    readonly attribute short availability;
    readonly attribute short status;
    readonly attribute long transferRate;
    readonly attribute string location;
    void start();
    void stop();
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

StaticDocument.idl

```
// StP -- created on Wed May 14 14:06:27 1997 for euba@sunhegering2 from system webman

#ifndef _StaticDocument_idl_
#define _StaticDocument_idl_

// stp class declarations
interface StaticDocument;
// stp class declarations end

// stp auto-include 243
#include "Document.idl"

// stp auto-include end
```

```
// stp class definition 243
interface StaticDocument : Document
{
// stp class members
    readonly attribute string title;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

TimeoutError.idl

```
// StP -- created on Wed May 14 14:06:28 1997 for euba@sunhegering2 from system webman

#ifndef _TimeoutError_idl_
#define _TimeoutError_idl_

// stp class declarations
interface TimeoutError;
// stp class declarations end

// stp auto-include 5745
#include "DocumentError.idl"

// stp auto-include end

// stp class definition 5745
interface TimeoutError : DocumentError
{
// stp class members
    readonly attribute string clientHost;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

permissionError.idl

```
// StP -- created on Wed May 14 14:06:26 1997 for euba@sunhegering2 from system webman

#ifndef _permissionError_idl_
#define _permissionError_idl_
```

```
// stp class declarations
interface permissionError;
// stp class declarations end

// stp auto-include 10216
#include "DocumentError.idl"

// stp auto-include end

// stp class definition 10216
interface permissionError : DocumentError
{
// stp class members
    readonly attribute string clientHost;
    readonly attribute string reason;
// stp class members end
};
// stp class definition end

// stp footer
#endif
// stp footer end
```

Anhang B

Kurzübersicht über die graphische OMT-Notation

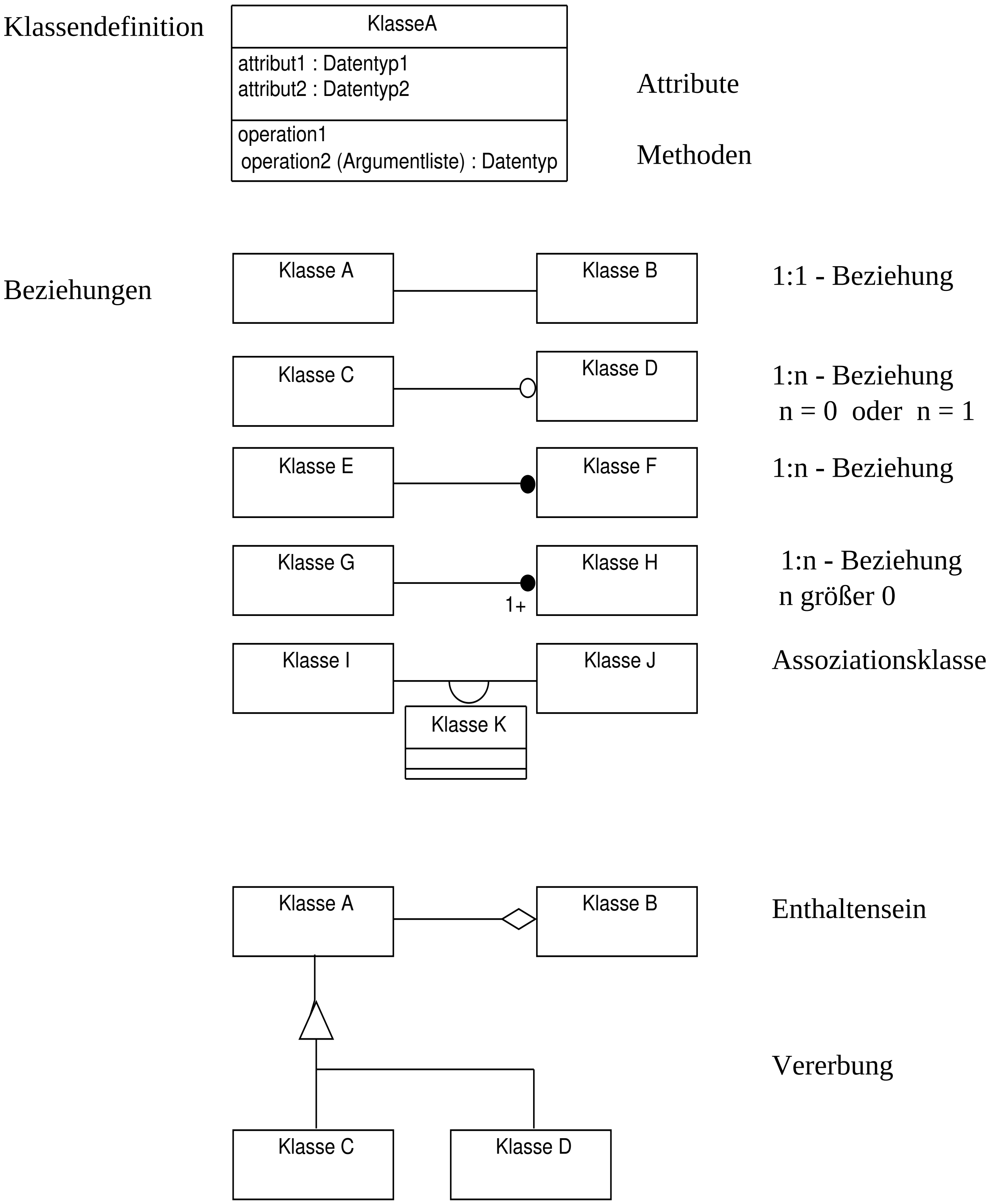

Abbildung B.1: Eine kurze Übersicht über die graphische Notation von OMT

Abbildungsverzeichnis

Literaturverzeichnis

[Con] World Wide Web Consortium. *The Common Logfile Format.* http://www.w3.org/hypertext/WWW/Daemon/User/Config/ Logging.html.

[CWK96] Jürgen Schönwälder Carl W. Kalbfleisch, Harrie Hazewinkel. *Definitions of Managed Objects for WWW Servers.* Internet Draft, IETF, März 1996.

[ES96] Albert Euba and Diana Stricker. *Implementierung eines Werkzeuges zur graphischen Darstellung von WWW-Link-Strukturen.* Fortgeschrittenenpraktikum, Institut für Informatik der Technischen Universität München, Juli 1996.

[GM95] James Gosling and Henry McGilton. *The Java Language Environment - A White Paper.* Sun Microsystems Computer Company, USA, 1995.

[Gro] Apache Group. *The Apache HTTP-Server.* http://www.apache.org/.

[GY95] German Goldszmidt and Yechiam Yemini. Distributed Management by Delegation. In *Proceedings of the 15th International Conference on Distributed Conputing Systems*, Juli 1995.

[ION97] IONA. *OrbixWeb Programming Guide.* http://www.iona.com/, 1997.

[Kal96] Carl W. Kalbfleisch. *Applicability of Standard Track MIBs to Management of World Wide Web Servers.* RFC 2039, OnRamp Technologies, November 1996.

[Kel96] Alexander Keller. Service-based Systems Management: Using CORBA as a Middleware for Intelligent Agents. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, 1996.

[KF94] S. Kille and N. Freed. *Network Services Monitoring MIB.* RFC 1565, ISODE Consortium, Innosoft, Januar 1994.

[KKPS97] C. Kalbfleisch, C. Krupczak, R. Preshun, and J. Saperia. *Application Management MIB.* draft-ietf-applmib-mib-02.txt, Verio, Empire Technologies, BMC Software, BGS Systems, März 1997.

[KS97] C. Krupczak and J. Saperia. *Definitions of System-Sevel Managed Objects for Applications.* draft-ietf-applmib-sysapplmib-07.txt, Empire Technologies, BGS Systems, März 1997.

[LP96] Laura Lemay and Charles L. Perkins. *teach yourself JAVA in 21 days.* Sams.net Publishing, 1996.

[MDR96] M. A. Mountzia and G. Dreo-Rodosek. Delegation of Functionality: Aspects and Requirements on Management Architectures. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, Oktober 1996.

[Mic96] Sun Microsystems. *Java: Programming for the Internet.* http://www.javasoft.com/, 1996.

[Mou96] Maria-Athina Mountzia. *Intelligent Agents in Integrated Network and Systems Management.* Institut für Informatik, Technische Universität München, 1996.

[Rum91] James Rumbaugh. *Object Oriented Modeling and Design.* Prentice Hall, 1991.

[Sch96a] J. Schönwälder. *Distributed Network Management by Delegation - A Standards Perspective.* Department of Computer Science, University of Twente, August 1996.

[Sch96b] Frank Schütz. *Implementierung eines Werkzeuges zur Konsistenzprüfung von HTML-Links.* Fortgeschrittenenpraktikum, Institut für Informatik der Technischen Universität München, März 1996.

[Sie96] John Siegel. *CORBA - Fundamentals and Programming.* Wiley Computer Publishing Group, 1996.

[Sir96] Holger Sirtl. *Objektorientierte Modellierung von Workstations für ein integriertes Systemmanagement.* Fortgeschrittenenpraktikum, Institut für Informatik der Technischen Universität München, Juli 1996.

[tP96a] Software through Pictures. *Core: Fundamentals of StP.* Interactive Development Environments, Februar 1996.

[tP96b] Software through Pictures. *Object Modeling Technique: Creating OMT Models.* Interactive Development Environments, Februar 1996.

[tP96c] Software through Pictures. *Object Modeling Technique: Generating Code.* Interactive Development Environments, Februar 1996.

[tP96d] Software through Pictures. *Object Modeling Technique: Getting Started with StP/OMT.* Interactive Development Environments, Februar 1996.

[Wim96] Peter Kai Wimmer. *Implementierung eines Analysewerkzeuges für Logdateien von WWW-Servern.* Fortgeschrittenenpraktikum, Institut für Informatik der Technischen Universität München, März 1996.

[YGY91] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In I. Krishnan and W. Zimmer, editors, *2nd International Symposium on Integrated Network Management.* Elsevier Science Publishers B. V. (North Holland), April 1991.