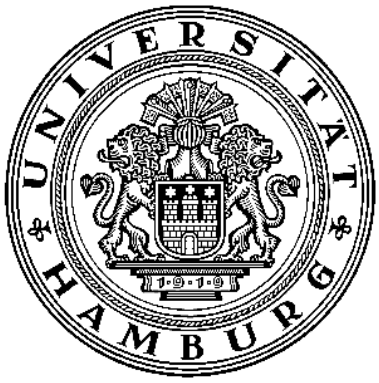

Universität Hamburg
FB Informatik

DIPLOMARBEIT

Das WWW als Informationssystem: Konzepte, Dienste, Defizite, Entwicklungen

25. September 1997

Andreas Bösch
Hallerstr. 3B, 6.Stock
20146 Hamburg

Betreuer:
Prof. Dr. Joachim W. Schmidt
Prof. Dr. Friedrich H. Vogt

Danksagung

An dieser Stelle möchte ich allen Menschen danken, die zum Gelingen dieser Arbeit beigetragen haben. Insbesondere danke ich meinen Eltern, die mir mein Studium ermöglichten und stets hinter mir standen.
Bei Gerald Schröder bedanke ich mich für viele Ideen, Anregungen und Kommentare und die insgesamt sehr gute Betreuung. Weiterhin danke ich Herrn Prof. Dr. Joachim W. Schmidt und Herrn Prof. Dr. Friedrich H. Vogt, für die Anregungen zu diesem Thema sowie für ihre Unterstützung bei der Durchführung der Arbeit.

Hamburg, 25. September 1997 Andreas Bösch

Typographie

In dieser Arbeit werden unterschiedliche Schriftarten verwendet, um die Lesbarkeit zu steigern. In der folgenden Tabelle sind alle verwendeten Schriftarten jeweils mit ihrer Bedeutung und einem Beispiel aufgeführt.

Schriftart	**Bedeutung**	**Beispiel**
Emphasized	wichtiger oder	*Intranet*
	fremdsprachlicher Begriff	*Remote Method Invocation*
Sans Serif	Firmen- oder	Netscape
	Produktname	Navigator

Inhaltsverzeichnis

1. Einleitung und Motivation **1**
1.1 Die Entwicklung des WWW 1
1.2 Aufgabenstellung 4
1.3 Aufbau der Arbeit 5

2. Anforderungen an verteilte Informationssysteme **7**
2.1 Funktionale Anforderungen 8
2.2 Architekturelle Anforderungen 11
2.3 Technische Anforderungen 13

3. Grundlagen des World Wide Web **15**
3.1 Adressierung im WWW 17
3.2 Hypertext Markup Language 18
3.3 Hypertext Transfer Protocol 19
3.3.1 HTTP Grundlagen 19
3.3.2 HTTP Methoden 20
3.4 Common Gateway Interface 21
3.5 Aktive Komponenten 22
3.5.1 Skripte 23
3.5.2 Programmcode 24

4. Bewertung vorhandener Konzepte und Technologien verteilter Informationssysteme **28**
4.1 Das WWW 28
4.1.1 Funktionale Anforderungen 28
4.1.2 Architekturelle Anforderungen 31
4.1.3 Technische Anforderungen 32
4.2 Andere verteilte Informationssysteme 33
4.2.1 Hyperwave 33
4.2.2 Netiva 35

5. Versionierung im WWW **38**
5.1 Terminologie und Konzepte 39
5.1.1 Datenmodelle 40
5.1.2 Unterstützung kooperativer Arbeit 40
5.1.3 Speichermechanismen 41
5.2 Anforderungen an Versionierungssysteme im WWW 41
5.2.1 Funktionale Anforderungen 42
5.2.2 Technische Anforderungen 43
5.3 Technologien 44
5.3.1 Versionierung durch redundante Datenhaltung 45
5.3.2 Versionierung durch HTML Erweiterungen 46
5.3.3 Versionierung durch Anbindung bestehender Systeme 49
5.3.4 Erweiterungen von HTTP 53
5.4 Erweiterung eines WWW Servers 56
5.4.1 SCCS 56
5.4.2 Jigsaw 58
5.4.3 Implementierung 58
5.5 Zusammenfassung 63

6. Konfigurationsmanagement im WWW **64**
6.1 Versionierung von Ankern 65
6.2 Konfigurationen im WWW 66
6.3 Realisierung von Kontexten 68
6.3.1 Kontextmanagement durch den Server 69
6.3.2 Kontextmanagement durch den Client 70

7. Verteilte Anwendungen im WWW **71**
7.1 Terminologie und Konzepte 73
7.2 Anforderungen an Kommunikationsmechanismen für verteilte Objekte im WWW 75
7.3 Technologien 77
7.3.1 CORBA 78
7.3.2 Java RMI 81
7.3.3 DCOM 83
7.4 Vergleich der Technologien 87

8. Entwicklungen des WWW **91**
8.1 Entwicklungen und Erweiterungen von HTML 91
8.1.1 HTML 4.0 91
8.1.2 HTML-Erweiterung mSQL 92
8.1.3 XML 93
8.2 Java 95
8.2.1 Pizza 95

8.2.2 PJava 97
8.2.3 Java Beans 98
8.3 Internet Protokoll der neuen Generation 100

9. Zusammenfassung **101**
9.1 Kommerzielle Relevanz 102
9.2 Weiterführende Arbeiten 102
9.3 Die Zukunft des WWW 103

Literaturverzeichnis **105**

Abbildungsverzeichnis

1.1 Das WWW als ein Dienst des Internet 2
1.2 Die Entwicklung des WWW 3

2.1 Anforderungen an verteilte Informationssysteme 8

3.1 Das WWW als ein Dienst des Internet 15
3.2 Ein typisches WWW-Szenario 16
3.3 JavaScript Code als Vermittler zwischen HTML und Java Code 24
3.4 Erstellung und Ausführung von Java Programmen 25

4.1 Architektur eines Hyperwave Systems 34
4.2 Architektur eines Netiva Systems 36

5.1 Revisionen und Varianten 38
5.2 Versionierungsansätze im WWW 44
5.3 Komponenten des WWRC 51
5.4 Prozesse während des *checkout* 52
5.5 Bereitstellung versionierter Textdaten für das WWW 56
5.6 Zusammenspiel von SCCS und Jigsaw 59
5.7 Einbindung neuer Java Klassen in Jigsaw 60
5.8 Ausschnitt der erweiterten Klassenhierarchie von Jigsaw 61

6.1 Zuordnung von versionierten Objekten zu Konfigurationen 65
6.2 Ein Kontext 67
6.3 Auswahl einer Version durch einen Kontext 68
6.4 Ein versionierter Kontext 69

7.1 Kommunikation zwischen verteilten Anwendungen 74
7.2 ORB versus RPC 75
7.3 Kommunikationsmechanismen in Java Anwendungen 77
7.4 Die Bestandteile eines *Object Request Broker* (ORB) 79
7.5 Integration von CORBA und WWW 81
7.6 Entfernte Methodenaufrufe in Java 82
7.7 Integration von Java RMI und WWW 83

7.8 Struktur eines DCOM Objektes 84
7.9 Verschiedene DCOM Server 86
7.10 OLE Automation Clients und ihre Server 86
7.11 Integration von DCOM und WWW 88

1. Einleitung und Motivation

Das Internet ist das größte Computer-Netzwerk der Welt [Scheller et al. 94]. War das Internet vor einigen Jahren noch nahezu ausschließlich im Bereich der Hochschulen verbreitet, drängen heute auch immer mehr Firmen auf einen Anschluß an einen der derzeit weltweit wachstumsstärksten Märkte. Die Internettechnologie wird aber nicht nur genutzt, um am weltweiten Internetverkehr teilzunehmen, sondern auch in lokalen Netzwerken von Unternehmen. Man spricht in diesem Zusammenhang von einem *Intranet* [Bichler 96]. Dies dient einerseits dazu, unterschiedlichste Plattformen und Systeme auf einer einheitlichen Basis zu verbinden und Dienste wie die elektronische Post (E-Mail) oder öffentliche Diskussionsgruppen (News) auch firmenintern zu nutzen, andererseits wird der optionale, spätere Übergang in das globale Internet sehr erleichtert, da die Intranet Technologie auf weit verbreiteten Standards (TCP/IP, Standardprotokolle für E-Mail und News) basiert. Der größte Vorteil eines Intranet ist seine Flexibilität, da alle Komponenten über Standardschnittstellen verbunden sind, d.h. der Systemdesigner kann jede benötigte Teilfunktionalität am jeweiligen Bedarf orientiert auswählen [Kossel 96].
Das World Wide Web (WWW) ist wie z.B. E-Mail, FTP und News ein Dienst des Internet. Das WWW ermöglicht mit Hilfe des HTT Protokolls die Übertragung von Texten, Grafiken und Programmen zwischen Servern und Clients (vgl. Abbildung 1.1, sowie Kapitel 3).
Die einfache Bedienbarkeit durch grafische Visualisierungswerkzeuge (engl. Browser) und Faktoren wie z.B. Plattformunabhängigkeit, globale Verbreitung und Erreichbarkeit machen das WWW für kommerzielle Anwendungen interessant, da es möglich ist, mit einem Mausklick Information, Waren und Dienstleistungen anzufordern bzw. zu bestellen. Im folgenden Abschnitt wird die Entwicklung des WWW in den letzten Jahren dargestellt und gezeigt, wie sich die Anforderungen an das WWW und der Begriff der „Information im WWW“ verändert haben.

1.1 Die Entwicklung des WWW

Die Anforderungen an das WWW haben sich im Laufe seiner Entwicklung verändert. Während das WWW am Anfang ein Veröffentlichungs- und Verteilungswerkzeug für Hypertextdokumente darstellte und Information lediglich vom Server zum Client jedoch nicht umgekehrt floß, bot sich mit der Entwicklung des einfach zu bedienenden grafischen WWW Browsers die Vision des globalen Marktplatzes, über den alle Arten von Dienstleistungen

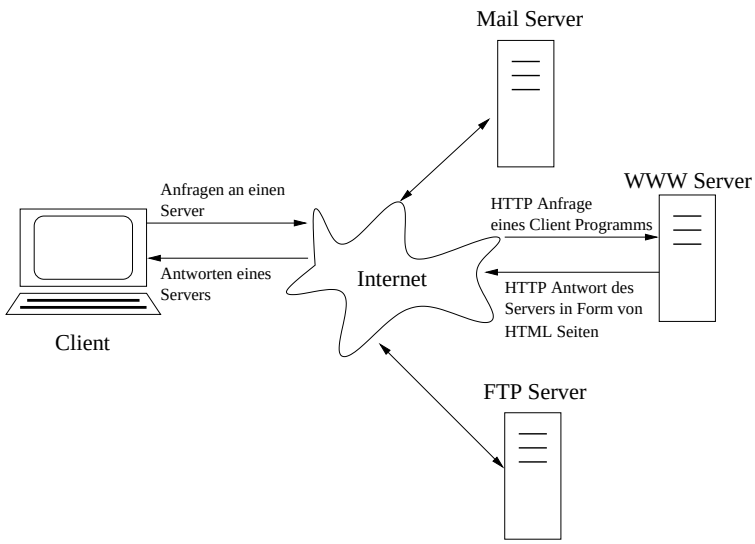

Abbildung 1.1: Das WWW als ein Dienst des Internet

und Gütern verbreitet werden können (vgl. Abbildung 1.2 aus [Orfali, Harkey 97]). Nach der Entwicklung sicherer Übertragungsprotokolle und elektronischer Abrechnungsverfahren wird diese Vision zunehmend Realität.

Es stellte sich jedoch heraus, daß die Basiskonzepte und -technologien des WWW, wie z.B. HTML, HTTP und CGI (vgl. Kapitel 3), die in sie gesetzten neuen Erwartungen nicht erfüllen können. HTML bietet im Vergleich zu professionellen Desktop Publishing Systemen nur eingeschränkte Präsentationsmöglichkeiten, HTTP erlaubt als zustandsloses Protokoll keine aus mehreren aufeinander aufbauenden Teilschritten bestehenden Transaktionen, das CGI erfordert hohe Berechnungskapazitäten auf dem ausführenden WWW Server.

Die Programmiersprache Java (vgl. Abschnitt 3.5.2) ermöglicht es, die Beschränkungen von HTML, HTTP und CGI zu umgehen, indem Java Anwendungen — sogenannte *Applets* — in HTML Seiten eingebunden werden und im Browser plattformunabhängig ausgeführt werden können. Java Applets können Verbindungen zu entfernten Java Anwendungen über ein eigens hierfür entwickeltes Protokoll — die *Remote Method Invocation* (vgl. Abschnitt 7.3.2) — aufnehmen. Mit Java ist es erstmals im WWW möglich, unbekannten Programmcode ohne Sicherheitsrisiken auf dem Rechner des Client auszuführen und die Umgebung des Browsers für verteilte Anwendungen zu nutzen.

Für die Nutzung des WWW als Basis für verteilte heterogene Anwendungen, deren Komponenten in verschiedenen Programmiersprachen geschrieben sind, reicht Java allein jedoch nicht aus. Eine Vermittlerrolle zwischen den Komponenten verteilter Anwendungen übernehmen Dienste wie z.B. CORBA oder DCOM, die die Zusammenarbeit heterogener Appli-

kationsobjekte erlauben (vgl. Kapitel 7). [1]
Zu der Vision des *Object Web* gehören neben der Nutzung des WWW als Basis für verteilte heterogene Anwendungen auch Komponententechnologien wie z.B. ActiveX oder Java Beans. ActiveX ermöglicht die Einbindung beliebiger Objekte in sogenannte *Container*. So ist z.B. die Anwendung Microsoft Word ein Container, in den Komponenten wie z.B. ein Formeleditor oder eine Rechtschreibüberprüfung eingebettet werden können. Ein anderes Beispiel für die Verwendung von Containern ist die Einbettung einer Microsoft Excel Tabelle in ein Microsoft Word Dokument. So ein aus mehreren Komponenten zusammengesetztes Dokument wird als *compound document* bezeichnet.

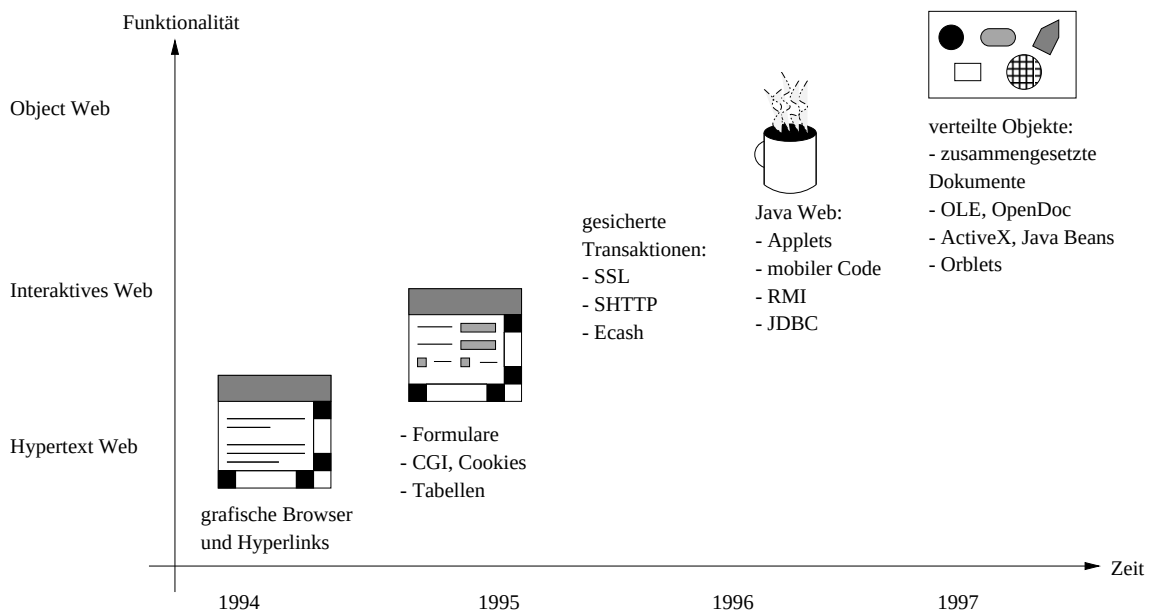

Abbildung 1.2: Die Entwicklung des WWW

Der Begriff der „Information im WWW“ hat sich im Laufe seiner Entwicklung verändert. Handelte es sich zu Anfang im Hypertext Web noch um statische Informationen, die für jeden Benutzer identisch waren, erlaubten Techniken wie HTML Formulare, CGI Anwendungen und HTML Tabellen die dynamische für jeden Client individuelle Erstellung von Information. So entwickelte sich das Hypertext Web zum interaktiven Web.
Die Verteilung von Programmen in Form von Skriptcode, Java Applets und ActiveX Komponenten führt zu einer neuen Dimension von Information, die die Ausführung unbekannten Programmcodes auf dem Rechner des Client ermöglicht. Dies ist wiederum die Grundlage für die Nutzung des WWW als Basis für verteilte Anwendungen und Komponententechno-

[1] Ein *Orblet* bezeichnet einen *Object Request Broker*, der in Form eines Java Applets zum Browser übertragen wird und dort die Kommunikation eines Applets mit einer entfernten Anwendung ermöglicht.

logien, die die Kommunikation und Verteilung heterogener Objekte in einem Object Web erlauben.

1.2 Aufgabenstellung

Die Anforderungen an das WWW und die im WWW zur Verfügung stehenden Inhalte haben sich geändert. Trotzdem basieren heutige WWW Anwendungen noch auf Konzepten, die für diese Art der Anwendung nicht vorgesehen sind, z.B. verteilte Datenbankanwendungen unter Benutzung von HTML, HTTP, CGI, Skripten und Datenbanksystemen. Die Benutzung dieser Technologien führt u.a. zu einer schlechten Wartbarkeit solcher Anwendungen, da ein Entwickler über Kenntnisse in allen verwendeten Technologien verfügen muß — er ist also gleichzeitig für Layout, Skriptprogrammierung, WWW Serveradministration, Datenbankadministration und Datenbankdesign zuständig. Soll z.B. das Layout einer dynamisch erzeugten HTML Tabelle geändert werden, muß der Autor neben HTML Wissen über Kenntnisse der Programmiersprache des zugehörigen CGI Skriptes verfügen.

Im ersten Teil der vorliegenden Arbeit wird untersucht, welche funktionalen Anforderungen von Informationsnutzern und Informationsautoren an verteilte Informationssysteme — insbesondere im Kontext des WWW — bestehen und wie diese sich auf die Anforderungen an die zugrundeliegende Architektur bzw. Technologie auswirken. Es wird dabei betrachtet, inwieweit Anforderungen (z.B. Konsistenzerhaltung, Redundanzfreiheit, Plattformunabhängigkeit, Skalierbarkeit), die für die Softwareentwicklung (vgl. z.B. [Matthes 93; Meyer 90; Balzert 82]) gelten, auf die Anforderungen an verteilte Informationssysteme übertragbar sind. In einem zweiten Schritt wird überprüft, inwiefern die derzeit vorhandenen Technologien im Kontext des WWW den im ersten Teil der Arbeit konkretisierten Anforderungen gerecht werden, bzw. in welchen Bereichen Defizite von WWW Konzepten liegen.

Im zweiten Teil der Arbeit werden ausgewählte Anforderungen im Detail betrachtet und Verbesserungen bestehender WWW Konzepte z.B. in Form von Erweiterungen von Protokollen oder Formaten zur Informationsrepräsentation erarbeitet bzw. vorgestellt.

Die Ziele der Arbeit sind also:

- Darstellung der Anforderungen von Informationsnutzern und Informationsautoren an die Erstellung, Speicherung, Wartung, Präsentation und Verteilung von Information in verteilten Informationssystemen;

- Vorstellung und Bewertung derzeit im WWW vorhandener Konzepte und Dienste zur Erstellung, Speicherung, Wartung, Präsentation und Verteilung von Information;

- Exemplarische Betrachtung ausgewählter Anforderungen an das WWW, sowie Erarbeitung, Darstellung und Vergleich geeigneter Konzepte und Technologien zu deren Erfüllung.

1.3 Aufbau der Arbeit

In dieser Arbeit wird keine allgemeine Einführung in das Internet gegeben, da sich mit dieser Thematik bereits andere Autoren ausführlich beschäftigt haben (bspw. [Scheller et al. 94], [Kehoe 94] oder [Gilster 95]).
Die Arbeit gliedert sich in drei Abschnitte. Der erste Abschnitt behandelt Anforderungen an verteilte Informationssysteme und die Vorstellung und Bewertung grundlegender WWW Technologien.

- Im Kapitel 2 werden allgemeine Anforderungen an die Erstellung, Speicherung, Wartung, Präsentation und Verteilung von Information als Pendant zu Programmen aus der Softwaretechnik erarbeitet. Dabei bilden funktionale Anforderungen von Informationsnutzern und Informationsautoren die Basis für die architekturellen und technischen Anforderungen an verteilte Informationssysteme. Es wird hier der Fragestellung nachgegangen, inwiefern sich u.a. Konzepte wie Konsistenzerhaltung, Redundanzfreiheit, Wartbarkeit, Skalierbarkeit oder Plattformunabhängigkeit aus der Softwaretechnik auf die Anforderungen an verteilte Informationssysteme übertragen lassen.

- In Kapitel 3 werden die Grundlagen des WWW (u.a. HTTP, HTML, CGI, Skriptsprachen, Java) vorgestellt, die die Basis für das Verständnis der nachfolgenden Kapitel bilden.

- Die Bewertung grundlegender WWW Technologien erfolgt anhand der in Kapitel 2 erarbeiteten Anforderungen in Kapitel 4. Außerdem werden hier einige Konzepte der WWW Erweiterung Hyperwave (ehemals Hyper-G) sowie der WWW Datenbankentwicklungsumgebung Netiva vorgestellt.

Im zweiten Abschnitt werden exemplarisch verschiedene Konzepte und ihre technologischen Ausprägungen untersucht, die zur Befriedigung der Anforderungen aus Kapitel 2 dienen können.

- In Kapitel 5 wird untersucht, welche Bedeutung das Management von Versionen von Ressourcen im Zusammenhang mit den in Kapitel 2 erläuterten Anforderungen wie Konsistenzerhaltung und Redundanzfreiheit für das WWW hat.

- Das Problem des Management von Konfigurationen im WWW ergibt sich aus der Versionierung einzelner WWW Ressourcen. In Kapitel 6 werden die grundsätzlichen Konzepte und technische Ansätze zur Bereitstellung von Konfigurationen im WWW vorgestellt.

- Kapitel 7 untersucht die Möglichkeiten zur Verwendung des WWW als Basis für verteilte Anwendungen und stellt drei Kommunikationsmechanismen vor, die die durch HTTP, CGI und HTML bestehenden Einschränkungen für verteilte Anwendungen im WWW überwinden.

Der dritte Teil der Arbeit liefert einen Ausblick auf aktuelle technische Entwicklungen des WWW.

- Neuere (noch) nicht standardisierte Erweiterungen und Weiterentwicklungen wichtiger Technologien im Kontext des WWW werden in Kapitel 8 erläutert. Unter anderem werden aktuelle Entwicklungen von HTML und Java beschrieben, die bereits einige der in Kapitel 2 beschrieben Anforderungen erfüllen.

Im abschließenden Kapitel 9 werden die Ergebnisse der Arbeit zusammengefaßt, deren kommerzielle Relevanz erläutert und ein Ausblick für weitere Forschungsaktivitäten gegeben.

2. Anforderungen an verteilte Informationssysteme

Unter einem *Informationssystem* (IS) versteht man ein Werkzeug zur Erfassung und Kommunikation von Information zum Zwecke der Erfüllung der Anforderungen seiner Benutzer [Vossen 95]. Ein IS umfaßt sämtliche zur Informationssammlung, -verwaltung, -benutzung und -distribution vorhandenen Ressourcen, wie z.B. die Daten in Datenbanken oder Dateisystemen, die Datenbank- und die Anwendungssoftware und die Hardware des Rechnersystems inklusive der verwendeten Speichermedien.
Der Begriff der *Information im WWW* hat sich in den letzten Jahren stark verändert und erweitert (vgl. Abschnitt 1.1). Bestand Information im WWW anfangs lediglich aus Hypertextdokumenten im HTML Format, kamen im Laufe der Zeit Bilder, Audio- und Videoinformation und ausführbarer Programmcode in diversen Formaten hinzu. Aus dieser Entwicklung ergibt sich die Frage, welche Anforderungen das WWW als verteiltes Informationssystem zu erfüllen hat.
In Abbildung 2.1 ist dargestellt, wie die Anforderungen von Informationsautoren und Informationsverbrauchern — im Folgenden als Autoren und Nutzer bezeichnet — auf die architekturellen und technischen Anforderungen an ein Informationssystem Einfluß nehmen. Für die Autoren ist hierbei insbesondere die Frage nach der Erstellung neuer Information und der Wartung bereits bestehender Information von Bedeutung. Die Erstellung und Wartung von Information steht in engem Zusammenhang mit der Speicherung von Information. Es ergeben sich hierbei z.B. Fragen wie:

- Muß neu erstellte Information vom Autor explizit gespeichert und im WWW veröffentlicht werden, oder ist neu erstellte Information implizit persistent und für andere Nutzer zugreifbar, wie z.B. beim Eintragen einer neuen Adresse in eine Datenbank?
- Kann die Wartung von Information konsistenzerhaltend erfolgen, oder führen nachträgliche Änderungen an einer Ressource zu Inkonsistenzen im System?
- Ist ein Autor berechtigt, eine Information zu ändern? Werden Zugriffsrechte mit einer Information gespeichert?

Der Autor beeinflußt desweiteren die Präsentation von Information, indem er sich entweder für proprietäre, plattformabhängige Formate zur Informationsrepräsentation entscheidet

oder eine weitgefächerte Nutzung mit Hilfe von offen standardisierten Formaten zur Informationsrepräsentation ermöglicht.

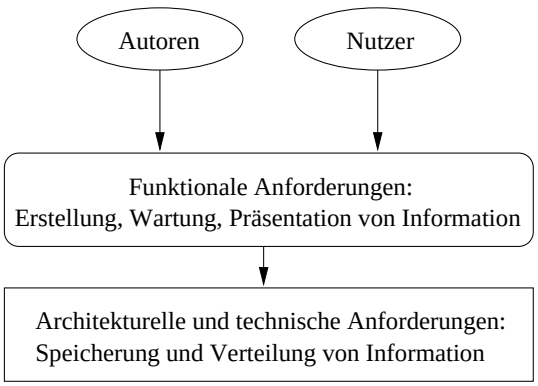

Abbildung 2.1: Anforderungen an verteilte Informationssysteme

Für den Nutzer ist die Präsentation der Information entscheidend, denn die Benutzerschnittstelle sorgt dafür, daß sich der Nutzer im System orientieren kann und daß er die gesuchte Information findet. Ihm ist es in der Regel nicht wichtig, wie die Information erstellt, gewartet, gespeichert und verteilt wird. Mit seinen Anforderungen an die Präsentation der vorhandenen Information bestimmt er die die Benutzerschnittstelle implementierenden Konzepte. Der Erfolg des WWW als einem Informationssystem für beliebige, nicht einschlägig vorgebildete Benutzer beruht nicht zuletzt auf einer ansprechenden und einfach zu bedienenden Benutzerschnittstelle.
Als Basis für die Bewertung der in Kapitel 3 vorgestellten Konzepte, Dienste und Technologien im WWW werden im Folgenden Anforderungen an verteilte Informationssysteme definiert. In vielen Fällen werden hierbei Parallelen zu Konzepten aus der Softwaretechnik wie z.B. Konsistenzerhaltung, Redundanzfreiheit, Skalierbarkeit oder Plattformunabhängigkeit aufgezeigt.
Dieses Kapitel gliedert sich in funktionale, architekturelle und technische Anforderungen an verteilte Informationssysteme. Die Anforderungen des Verbrauchers von Information — dem *Nutzer* — werden von den Anforderungen desjenigen unterschieden, der die Informationen bereitstellt — dem *Autor* (vgl. Abschnitt 2.1). Aus den funktionalen Anforderungen von Nutzern und Autoren ergeben sich architekturelle (vgl. Abschnitt 2.2) und technische (vgl. Abschnitt 2.3) Anforderungen an verteilte Informationssysteme.

2.1 Funktionale Anforderungen

Die funktionalen Anforderungen von Nutzern und Autoren verteilter Informationssysteme wie dem WWW sind unterschiedlich. Die erfolgreiche Nutzung eines Informationssystems zur Erfüllung einer Aufgabe erfordert die Bereitstellung folgender Konzepte:

Orientierung: In vielen Informationssystemen, die Inhalte mittels Hyperlinks — sogenannte *Hypermediasysteme* — verknüpfen, sehen Benutzer nur ein Dokument zur Zeit [Maurer 96]. In diesem Dokument sind Hyperlinks sicht- und aktivierbar, die zu weiterer Information führen. Es ist für den Nutzer jedoch nicht ersichtlich, in welchem Zusammenhang eine gerade betrachtete Information — z.B. in Form eines Hypertextdokumentes — mit den mit diesem Dokument zusammenhängenden Informationen steht. So ist es für den Nutzer nicht möglich zu erkennen, welche Hyperlinks von anderen Dokumenten zu dem aktuell angezeigten führen. Ein Benutzer kann also — bildlich gesprochen — in einer Menge von Informationen nur bis zur nächsten Kreuzung vorwärts sehen, ihm fehlt der Überblick. Notwendig sind in diesem Zusammenhang etwas wie *Informationslandkarten*, die Zusammenhänge zwischen einzelnen Informationen visualisieren und so eine Orientierung im Informationsraum ermöglichen.

Navigation: Es ist wünschenswert, daß ein Benutzer neben einem Überblick über alle verfügbaren Informationen auf einem Server auch eine direkte Zugriffsmöglichkeit auf diese erhält und nicht sequentiell nacheinander mehrere Hyperlinks verfolgen muß, um zu der gewünschten Information zu gelangen. Das Informationssystem sollte es sich desweiteren merken, daß bestimmte Informationen bereits betrachtet worden sind und Verweise auf diese Information entsprechend kennzeichnen, um so das zyklische Abrufen der selben Information zu verhindern. Wird die Information hingegen später verändert, ist die Kennzeichnung zu entfernen. Das spätere Wiederfinden einmal bereits besuchter Informationen erfordert das Speichern von Verweisen zu diesen.

Suchmöglichkeiten: Neben einer Volltextsuche, die bei der Suche nach Bildern nicht unbedingt Erfolg hat, ist es wünschenswert, daß ein Nutzer Metainformationen, wie z.B. Autor, Titel einer Ressource, Schlüsselworte, Typ der Information (Text, Bild, Audio, Video, Code, etc.) als Suchkriterien angeben kann. Der Suchraum sollte eingrenzbar sein, z.B. auf einen Server oder eine auf einem Server vorhandene Gruppe von Ressourcen.

Visualisierung von Änderungen: Das System sollte es ermöglichen, dem Nutzer Änderungen an für ihn interessanten Informationen automatisch mitzuteilen, z.B. durch den Einsatz von intelligenten Agenten, die im Auftrag des Nutzers regelmäßig Informationsangebote auf Neuigkeiten überprüfen und den Benutzer dann automatisch darüber informieren (z.B. per E-Mail).

Transparenz und Bedienbarkeit: Die Komplexität eines Informationssystems sollte sich nicht in der Benutzerschnittstelle widerspiegeln, sondern muß für den Benutzer unter einer ansprechenden, einfachen Benutzerschnittstelle verborgen bleiben.

Die weitergehenden Anforderungen der Autoren in verteilten Informationssystemen betreffen sowohl die initiale Erstellung von Informationen und deren Zusammenhängen sowie deren spätere Wartung und Überarbeitung:

Statische und dynamische Informationserstellung: Die in einem Informationssystem abrufbare Information kann sowohl statisch als auch dynamisch erstellt werden. Statische Information eignet sich für sich selten oder nie ändernde Inhalte bzw. Informationen, die für alle Nutzer identisch sind und nicht individuell aufbereitet werden müssen, wie z.B. Begriffsdefinitionen, Jahresberichte oder Dokumentationen von Industriestandards. Dynamische, sich oft ändernde Information ist weitaus häufiger anzutreffen, z.B. in Form von Adressen, Produktpreisen, Produktbeschreibungen, Börsenkursen, Flugplänen oder Programmcode. Ein Vorteil der dynamischen Erstellung von Information ist, daß auf individuelle Anforderungen des Nutzers eingegangen werden kann, wie z.B. den Flugplan einer bestimmten Fluglinie zu einem bestimmten Ziel zu einer bestimmten Zeit sehen zu wollen.

Autorisierte Erstellung und Wartung: In Systemen mit mehreren Autoren ist es wünschenswert, daß nicht jeder Autor Änderungen an den Dokumenten anderer vornehmen darf, bzw. daß nur bestimmte Bereiche des Informationsangebotes von dafür zugelassenen Bearbeitern geändert werden, z.B. wenn es um die Änderung von Produktpreisen in einer Datenbank geht. Desweiteren ist es von Vorteil, das Recht Hyperlinks zu einem Dokument hinzuzufügen von dem Recht es zu ändern zu trennen, denn dadurch ermöglicht man einem Bearbeiter, der nicht das Recht hat, ein Dokument zu ändern, eine Anmerkung zu erstellen und diese dann mit einem zuätzlichen Hyperlink von dem ursprünglichen Dokument aus zu referenzieren. Diese Anforderung hat direkte Auswirkungen auf die Speicherung von Information, da eine separate Speicherung von Hyperlinks und den durch sie verbundenen Informationen erforderlich wird. So könnten auch Hyperlinks mit eigenen Zugangsrechten versehen und ihre Verwendung nur bestimmten Nutzergruppen erlaubt werden.

Konsistenzerhaltende Wartung: Information bildet einen zu einem konkreten Zeitpunkt vorhandenen Wissensstand ab. Damit Information nicht veraltet bzw. ungenau oder falsch wird, muß sie wie andere Dinge, die ständigen Veränderungen unterliegen, gewartet werden. Statische Informationen, die z.B. in Form von HTML Dateien vorliegen, bauen mittels Hyperlinks Verbindungen zu anderen Informationen auf und werden selbst mittels Hyperlinks referenziert. Wird ein statisches Dokument geändert, kann es sein, daß die zu diesem Dokument vor der Änderung hergestellten Beziehungen ungültig werden, da evtl. die referenzierte Information gelöscht oder stark verändert wurde. Ziel eines verteilten Informationssystems muß es u.a. sein, die Konsistenz von bestehenden Beziehungen zwischen Informationen bei Änderungen zu erhalten. Ein aus der Softwaretechnik stammendes Konzept dafür ist das der Versionierung (vgl. Kapitel 5).

Automatische Erzeugung von Hyperlinks: Sofern in einem verteilten Informationssystem Hyperlinks zur Verknüpfung von Informationen genutzt werden, ist es für den Autoren wünschenswert, daß das System aufgrund von erkannten Schlüsselwörtern die Einsetzung von Hyperlinks zu der entsprechenden Ressource empfiehlt oder automatisch vornimmt. Da ein Autor in einem großen Informationssystem nicht alle vorhan-

denen Ressourcen kennen kann, ermöglicht das System so eine sinnvolle Vernetzung und Wiederverwendung der vorhandenen Information.

Trennung von Inhalt und Präsentation: Um die Unabhängigkeit von Information und der Präsentation der Information zu erreichen, ist es sinnvoll, Inhalt und Formatierung von Information voneinander zu trennen. Dadurch wird es möglich, Information in verschiedenen Präsentationsformaten wiederzuverwenden. Bei der Kombination von Inhalt und Präsentation z.B. durch den Einsatz von Hyperbildern — Bilder, die mit verschiedenen, mit der Maus auswählbaren, Hyperlinks versehen sind — wird die Nutzung der Information auf eine bestimmte Art der Benutzerschnittstelle festgelegt und ist z.B. für blinde Benutzer nicht möglich. Desweiteren erlaubt die Trennung von Inhalt und Präsentation so etwas wie *corporate identity*, indem das System für eine für alle Autoren gleiche Darstellung z.B. von Schriftarten, Schriftgrößen, Einrückungen, etc. sorgt.

Die Auswirkungen der Anforderungen von Nutzern und Autoren von Information auf die Architektur und die technologischen Grundlagen von verteilten Informationssystemen wird in den zwei folgenden Abschnitten untersucht.

2.2 Architekturelle Anforderungen

Die Korrektheit der in einem Informationssystem gespeicherten Daten bezeichnet man als deren Konsistenz oder Integrität. Konsistenzregeln bzw. Integritätsbedingungen definieren einen korrekten Datenzustand bzw. eine korrekte Datenzustandsveränderung [Jablonski 90]. Um die Konsistenz in einem Informationssystem zu erhalten, sollten folgende Forderungen erfüllt werden:

Objektbenennung: Die Benennung von Objekten (z.B. Dateien, Dokumente, Hyperlinks, Benutzer, Server) in verteilten Systemen sollte auf einer Trennung von Objektname, Objektadresse und Zugangsverfahren zu dem Objekt beruhen. Der Objektname identifiziert ein Objekt eindeutig, dies ist z.B. bei Büchern die ISBN Nummer. Die Adresse beschreibt den physikalischen Ort einer Instanz eines Objektes. Eine Adresse einer Datei auf einem Internetrechner wird z.B. durch die Internetadresse des Rechners und den Namen der Datei angegeben. Das Zugangsverfahren beschreibt, wie auf ein bestimmtes Objekt an einer bestimmten Adresse zugegriffen wird. Im Internet geschieht das durch Angabe eines bestimmten Übertragungsprotokolls (z.B. HTTP, FTP, News). Die Benutzung von Adressen und Zugangsverfahren zur Identifizierung von Objekten führt zur Abhängigkeit eines Objektes von seinem Speicherplatz. Beim Verschieben eines Objektes in ein anderes Verzeichnis ändert sich die Adresse und alle Verweise auf dieses Objekt werden ungültig, daher ist die Objektbenennung von der Objektadresse und dem Zugangsverfahren zu trennen.

Redundanzfreiheit: Man spricht von Redundanz, wenn das gleiche Objekt mehrfach in einer Datensammlung vorkommt. Dies kann zu einem inkonsistenten Zustand der be-

reitgestellten Information führen, wenn Änderungen an einem der mehrfach vorhandenen Objekte durchgeführt werden und ein Benutzer auf die nicht geänderten Objekte zugreift. Für die Wartung von Information ist es erforderlich, daß jede Information nur einmal gespeichert wird, damit Änderungen an nur einem Objekt vorgenommen werden müssen.

Versionierung: Bei der Wartung einer Ressource werden Verweise, die dieses Objekt referenzieren evtl. ungültig. Um dies zu verhindern, werden Versionierungssysteme eingesetzt, die die Erzeugung, Administration und Pflege mehrerer Instanzen einer Ressource über einen längeren Zeitraum ermöglichen (vgl. Kapitel 5).

Neben der Konsistenz der Information sind von einem verteilten Informationssystem folgende Anforderungen zu erfüllen:

Strukturierte Datenspeicherung: Insbesondere die Möglichkeit zur effektiven Suche nach Information anhand von Metainformation (z.B. Autor, Erstellungsdatum, Typ der Information) erfordert eine strukturierte Speicherung von Information samt ihrer Metadaten.

Skalierbarkeit: Die Skalierbarkeit betrifft zum einen die Performanz des Systems. Die Performanz sollte nicht von der Anzahl der gleichzeitigen Nutzer abhängig sein. Um dieser Anforderung gerecht zu werden, sollten Konzepte wie Replikation[1] und Zwischenspeicherung auf anderen Rechnern (*caching*) zur Verfügung stehen. Die Skalierbarkeit betrifft auch die Robustheit des Systems. Ein Informationssystem sollte nicht von einer einzigen Netzwerkverbindung erwarten, daß sie zu jeder Zeit vorhanden ist. Vielmehr sollte das System erkennen, wenn eine Verbindung oder ein Server ausgefallen ist und versuchen, Informationen dementsprechend über andere Verbindungen verfügbar zu machen.

Anpassung an variable Netzkapazitäten: Ein zunehmendes Problem von Internet Informationssystemen ist die zur Verfügung stehende Bandbreite. Die Übertragung von Audio- und Videodatenströmen muß in regelmäßigen Intervallen erfolgen, um eine verzögerungsfreie Darstellung der Information auf dem Zielrechner zu ermöglichen. Um die zu erreichen, sollte es die Möglichkeit geben, entweder Bandbreiten für Datenströme zu reservieren oder aber, je nach Netzbelastung, die übertragene Datenmenge zu reduzieren oder zu erhöhen.

Integrität: [Meyer 90] versteht unter Integrität „die Fähigkeit eines Softwaresystems, seine verschiedenen Komponenten (Programme, Daten, Dokumente) gegen unberechtigte Zugriffe und Veränderungen zu schützen". Systemsicherheitsdienste wie Authentifizierung, Zugangskontrolle und die Aufzeichnung von Benutzerinteraktionen (engl. *auditing*) werden zunehmend wichtiger für Informationssysteme, insbesondere in verteilten heterogenen Umgebungen [Rudloff et al. 95].

[1] Replikation: Datenbestände werden auf andere Rechner kopiert und Änderungen an den Daten zwischen den an der Replikation beteiligten Rechnern ausgetauscht.

Neben diesen Anforderungen müssen weitere Entscheidungen bezüglich der Architektur des zur Kommunikationsübertragung verwendeten Protokolls und der verwendeten Serverarchitektur gefällt werden.
Protokolle werden in verbindungslose und verbindungsorientierte Protokolle unterschieden. Für die Übertragung einzelner kurzer Informationspakete ist in der Regel eine verbindungsloses Protokoll sinnvoll. Dies führt zu einer einfachen Implementierung von Client und Server[2]. Ein verbindungsorientiertes Protokoll läßt — im Gegensatz zu einem verbindungslosen Protokoll — eine Verbindung so lange bestehen, bis sie explizit von einem der beiden Kommunikationsteilnehmer beendet wird. Das hat den Vorteil, daß für eine Folge von Datenübertragungen nur einmal eine Verbindung aufgebaut werden muß und diese dann für alle Datenpakete genutzt werden kann. Verbindungsorientierte Protokolle wie z.B. FTP und Hyper-G (vgl. Abschnitt 4.2.1) sind in der Regel effizienter als verbindungslose Protokolle, führen jedoch zu einer komplexeren Serverarchitektur [Maurer 96].
Man unterscheidet zwischen zustandslosen Servern und Servern, die Zustände von Verbindungen verwalten. Zustandslose Server behandeln jede Anfrage in einem separaten Unterprozeß. Der Zusammenhang einer Anfrage mit einer früheren Anfrage geht dabei verloren, so daß keine Kontextinformation verfügbar ist. Server für verbindungsorientierte Protokolle ordnen jedem Client einen Subprozess zu und ermöglichen so, daß Variablen, die in einer Anfrage instanziiert wurden, bei der nächsten Anfrage in der gleichen Verbindung den gesetzten Wert haben.

2.3 Technische Anforderungen

Aus den in Abschnitt 2.1 definierten Anforderungen von Nutzern und Autoren von Informationssystemen ergeben sich folgende technische Anforderungen:

Plattformunabhängigkeit: Plattformunabhängige Informationsformate erlauben in heterogenen verteilten Informationssystemen die Nutzung von Information unabhängig von den Werkzeugen zur Informationsdarstellung bzw. den verwendeten Betriebssystemen und Rechnerarchitekturen. Diese Anforderung gilt ebenso für die Softwareentwicklung, bei der die Plattformunabhängigkeit des Programmcodes die Portierung von Programmen auf andere Rechnerarchitekturen und Betriebssysteme erspart.

Kompatibilität: Unter Kompatibilität versteht [Meyer 90] „das Maß der Leichtigkeit, mit der Softwareprodukte mit anderen verbunden werden können“. Ein Informationssystem sollte also Schnittstellen bieten, die es erlauben, sowohl außerhalb des Systems bestehende Datenbestände zu integrieren als auch die im System vorhandene Informatione von anderen Systemen aus zu nutzen. Typische Schnittstellen von Informationssystemen dienen dem Zugriff auf externe Datenbanken oder erlauben mittels einer

[2] Alle heutigen Internet Informationssysteme sind Client/Server Systeme. Der Benutzer interagiert mit der Clientsoftware auf seinem Rechner, während der Server auf einem entfernten Rechner Informationen anbietet. Das Protokoll ist die Sprache, die Client und Server zur Anforderung und Übertragung von Information über ein Netzwerk benutzen. In Kapitel 3 wird die Client/Server Architektur des WWW genauer vorgestellt.

Anwendungsprogrammierschnittstelle (API) den Zugriff auf das System. Der Vergleich zu Softwareentwicklungssystemen, die z.B. für den Zugriff auf Datenbanken entsprechende Codebibliotheken bieten, liegt nahe.

Migration: Zur Realisierung intelligenter Agenten, die im Auftrag von Informationsnutzern für diesen interessante Informationen sammeln, ist es notwendig, daß ein Objekt — in diesem Fall eine parametrisierte Anforderung des Nutzers — *migriert*, d.h. seinen Ort wechselt [Mathiske 96]. Der jedem Objekt eigene Bearbeitungsstatus muß dabei persistent gespeichert werden, da ein Objekt durchaus einige Zeit inaktiv sein kann, z.B. wenn zur Bearbeitung nötige Information fehlt. Ein weiteres innovatives Szenario zum Einsatz des Migrationskonzeptes in Informationssystemen ist das des *workflow management*. Das Ziel von *Workflow–Management–Systemen* ist es, die Weitergabe und Bearbeitung von Geschäftsprozessen automatisch zu steuern, um so die Durchlaufzeit eines zu bearbeitenden Objektes zu optimieren und die bei manueller Weitergabe des Objektes — z.B. in Form von Akten — auftretenden Liegezeiten zu vermeiden.[3]

Ressourcenunabhängigkeit: Information in verteilten Informationssystemen hängt evtl. von anderer externer Information ab. Wird Information in einem verteilten Informationssystem zu einen anderen Rechner übertragen, auf dem die benötigte externe Information nicht vorhanden ist, müssen Konfliktlösungsstrategien vorhanden sein, wie z.B. die fehlende Information ebenfalls zu übertragen oder zu überprüfen, ob evtl. andere lokal vorhandene Information die nicht nachladbare externe Ressource ersetzen kann.

In Kapitel 3 werden die Grundlagen des WWW vorgestellt. Funktionale, architekturelle und technische Anforderungen bilden die Grundlage für die Bewertung des WWW als Informationssystem in Kapitel 4.

[3] Zum Migrationskonzept und dessen Realisierung in einem persistenten Objektsystem findet sich weitere Information in [Mathiske 96].

3. Grundlagen des World Wide Web

Das World Wide Web (WWW) ist einer der am stärksten frequentierten Dienste des Internet. Ein erster Prototyp entstand 1990 am CERN in Genf [CERN 96]. Zwei Jahre nach der offiziellen Vorstellung des WWW brachte die einfach zu bedienende Visualisierungssoftware (engl. *Browser*) MOSAIC den Durchbruch für das WWW, das mit Wachstumsraten von mehr als 1000 Prozent pro Jahr (Stand 1996) der am schnellsten wachsende Internet-Dienst ist.

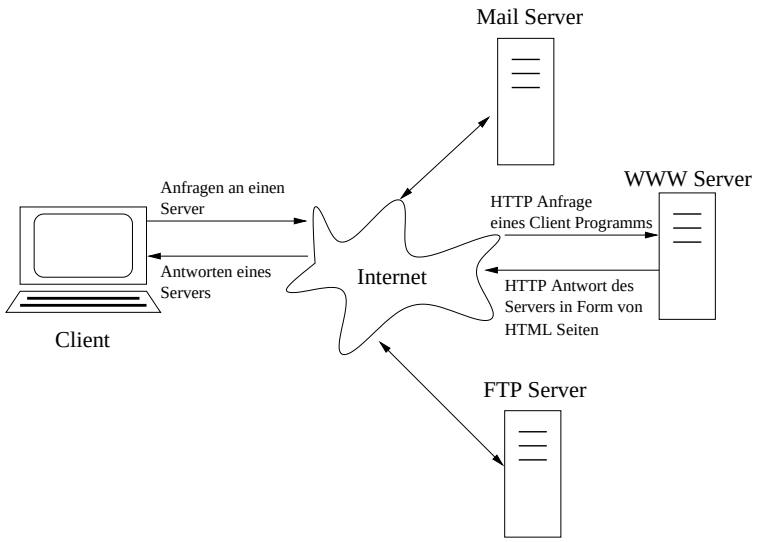

Abbildung 3.1: Das WWW als ein Dienst des Internet

Abbildung 3.1 stellt das Client/Server Konzept des WWW dar. Ein Browser auf dem Rechner des Client kann in Interaktion mit dem Benutzer Informationen von beliebigen Internet

Servern[1] anfordern. Der Server stellt dem Client die gewünschte Information zur Verfügung, sofern sie vorhanden ist und der Client berechtigt ist darauf zuzugreifen. Informationen liegen auf dem Server entweder in statischer Form (Dateien) vor oder werden von ihm dynamisch generiert. Nach der Übertragung der Information zum Client ist der Browser für deren Darstellung verantwortlich.
Verschiedene Arten von Informationen erfordern unterschiedliche Protokolle für ihre Übertragung. Das WWW ermöglicht mit Hilfe des verbindungslosen *Hypertext Transfer Protocol* (HTTP, vgl. Abschnitt 3.3) die Übertragung von Information zwischen WWW Servern und WWW Clients. Andere Internetdienste erfordern andere Protokolle. So wird zum Versand elektronischer Post das *Simple Mail Transfer Protocol* (SMTP) benutzt und zur Übertragung beliebiger Dateien das *File Transfer Protocol* (FTP). Mit der Entwicklung von neuen Browsern wie dem Netscape Navigator [Netscape 96] oder dem Internet Explorer von Microsoft [Microsoft 96] werden immer mehr Dienste des Internet, wie z. B. die elektronische Post (E-Mail) oder der Dateitransfer (FTP) unter einer Benutzerschnittstelle integriert und so für den Benutzer leichter zugänglich gemacht.
Anwendungen im WWW bestehen heute meist aus folgenden Komponenten: einem Browser auf der Client-Seite, einem Server mit optionalem CGI Skript und bspw. einem Datenbankserver (vgl. Abbildung 3.2) [December, Ginsburg 95; Bichler 96].

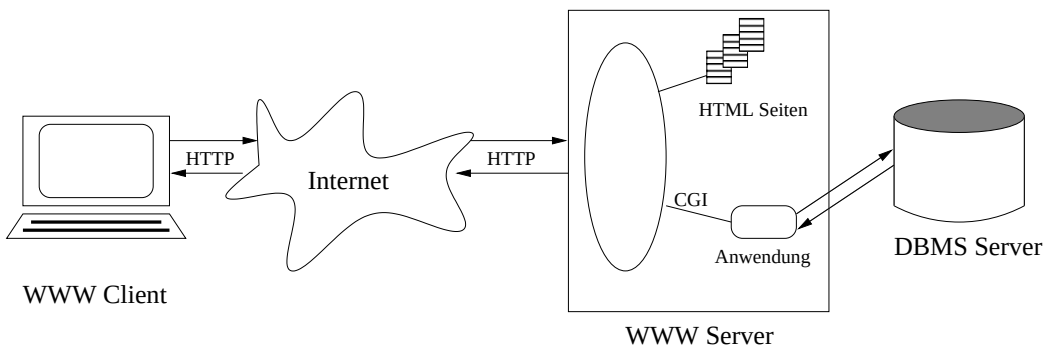

Abbildung 3.2: Ein typisches WWW-Szenario

Der Browser zeigt dabei vom Server an den Client übertragene Informationen an, die in Form von HTML Seiten entweder statisch als Dateien auf dem Server vorliegen oder automatisch aus dem Datenbestand einer Datenbank generiert werden. Die Verbindung zwischen WWW Server und Datenbank wird über das *Common Gateway Interface* (CGI) hergestellt. Ein weit verbreitetes Praxisbeispiel für eine solche Anwendung ist ein Händler, der seine Produktpalette in einer Datenbank verwaltet und über einen WWW Server einen Informationsservice für seine Produkte anbietet. Ein Benutzer fordert über seinen Browser Informa-

[1] Der Begriff Server wird hier zweideutig verwendet. Er bezeichnet einerseits einen physisch vorhandenen Rechner, der an das Internet angeschlossen ist und andererseits ein Softwareprogramm, daß auf diesem Rechner läuft. Es ist möglich, daß mehrere Serverprogramme auf einem Rechner aktiv sind.

tionen zu einem bestimmten Produkt an. Der WWW Server gibt die Parameter der Anfrage an das CGI Skript weiter, dieses baut eine Verbindung zur Produktdatenbank auf, fragt die entsprechenden Daten mittels eines SQL Kommandos ab, bereitet die von der Datenbank gelieferten Daten im HTML Format auf und schickt sie zum Benutzer zurück.
Nachfolgend werden die Grundlagen des WWW kurz vorgestellt. Die Adressierung von Daten im WWW erfolgt über *Uniform Resource Identifier* (vgl. Abschnitt 3.1). Abschnitt 3.2 erläutert die Grundlagen der Seitenbeschreibungssprache *Hypertext Markup Language* (HTML). Das *Hypertext Transfer Protocol* (HTTP) ist die Sprache, die WWW Clients und WWW Server benutzen, um miteinander zu kommunizieren (vgl. Abschnitt 3.3). Das *Common Gateway Interface* (CGI) ist die Schnittstelle zwischen einem WWW Server und einem beliebigen anderen Programm (vgl. Abschnitt 3.4).
Eine besondere Rolle spielen in diesem Zusammenhang Skriptsprachen (vgl. Abschnitt 3.5.1) wie z.B. JavaScript und VBScript, sowie die Programmiersprache Java (vgl. Abschnitt 3.5.2). Diese Sprachen werden nicht wie HTTP, HTML und CGI vom World Wide Web Konsortium (W3C) standardisiert[2], sondern sind kommerzielle Produkte, die in der jeweiligen Client- und Serversoftware ihrer Hersteller Verwendung finden.

3.1 Adressierung im WWW

Im WWW werden viele Dienste des Internet integriert und sind in einem Browser transparent zu benutzen. Da jedoch z.B. E-mail und FTP unterschiedliche Adressierungsschemata haben, wurde für das WWW das Adressierungskonzept der *Uniform Resource Identifier* (URI) entwickelt. URI sind die generische Menge aller Namen und Adressen, die als kurze Zeichenketten Ressourcen referenzieren. Sie schließen die Menge der *Uniform Resource Locator* (URL) und der *Uniform Resource Name* (URN) ein. Ein URN ist ein URI Schema, das einen Namen global eindeutig und persistent definiert. Ein URN ist im Gegensatz zu einem URL unabhängig vom Ort des benannten Objektes. Ein URL bezeichnet ein URI Schema, daß explizite Instruktionen für den Zugang zu Ressourcen im Internet enthält, es setzt sich aus dem Protokolltyp, dem Rechnernamen und den Pfadangaben zu der gewünschten Information zusammen. So könnte der Händler aus obigem Beispiel unter dem URL

http://www.unternehmen.de/cgi-bin/produktinfo

erreichbar sein. In dem URL wird das Hypertext Transfer Protokoll (HTTP, vgl. Abschnitt 3.3) zur Übertragung der gewünschten Information spezifiziert. Der Rechnername ist in diesem Fall *www.unternehmen.de*. Alternativ kann der Rechnername als IP Nummer angegeben werden. Falls der WWW Server nicht auf dem Standard WWW TCP Port 80 arbeitet, ist die Angabe einer Portnummer notwendig, z.B. *www.unternehmen.de:8001*. Der Pfadname zur gewünschten Information ist in diesem Beispiel */cgi-bin/produktinfo*. Ein Kunde, der diesen URL in seinem Browser aufruft, könnte als Antwort vom Server ein Formular erhalten, das es ihm ermöglicht, ein oder mehrere Produkte aus einer Liste

[2]Standards, Spezifikationen und Vorschläge zur Standardisierung sind unter *http://www.w3.org* zu finden.

auszuwählen, zu denen er weitere Information wünscht. Der Kunde in diesem Beispiel hat sich für das Produkt „Software“ entschieden und fordert vom Server unter folgendem URL weitere Information an:

http://www.unternehmen.de/cgi-bin/produktinfo?produkt=software

In diesem URL wird einem CGI Skript (vgl. Abschnitt 3.4) zusätzlich das Name=Wert Paar „produkt=software“ übergeben.

3.2 Hypertext Markup Language

Die *Hypertext Markup Language* (HTML) ist eine auf SGML (vgl. [Rieger 95; W3C 96b]) basierende Seitenbeschreibungssprache. HTML Texte können von einem SGML Parser gelesen und auf syntaktische Korrektheit überprüft werden [Dalitz, Hezer 95; W3C 96a]. Im WWW werden die Texte direkt an ein Anzeigeprogramm (engl. *Browser*) übergeben, das ihnen auf dem Bildschirm ein bestimmtes Aussehen zuordnet. Der Browser entscheidet über die Art der Darstellung anhand von Markierungen im Text (z.B. Titel, Überschrift, Listendarstellung, Formulare, etc.). Mit Hilfe von *Cascaded Style Sheets* (CSS) ist es möglich, die exakte Formatierung von HTML Seiten zu definieren. So lassen sich z.B. Schriftart, Schriftgröße, Schriftschnitt, etc. von HTML Markierungen vom Autor eines Dokumentes bestimmen und müssen nicht der willkürlichen Darstellung des Browser überlassen werden.
Da HTML Seiten aus ASCII Zeichen bestehen, können sie mit allen Texteditoren erstellt werden. Es gibt außerdem Programme, mit denen HTML Seiten nach dem Prizip des WYSIWYG (engl. *What You See Is What You Get*) erstellt werden können. Bei diesen Werkzeugen ist es nicht nötig, die spezielle Syntax der HTML Markierungscodes (engl. *markups*) zu kennen, da sie sich mit Hilfe von grafischen Symbolen oder Menüeinträgen in ein Dokument einfügen lassen.
Jeder Markierungscode beginnt mit einer offenen Klammer “<“, gefolgt seinem Namen. Daran schließt sich bei einigen Markierungscodes eine Liste von optionalen Attributen an. Ein *markup* wird durch eine schließende Klammer “>“ beendet.
Die meisten HTML Markierungscodes umschließen einen Text und andere Elemente eines Dokumentes mit einer Anfangs- und einer Endmarkierung. Eine Endmarkierung enthält vor seinem Namen einen Schrägstrich “/“ und niemals Attribute. Das folgende Beispiel definiert einen rechtsbündigen Absatz:

* Dies ist ein rechtsbündiger Absatz. *

HTML Seiten werden über sogenannte *Hyperlinks* miteinander verknüpft. Ein Hyperlink enthält den URL (vgl. Abschnitt 3.1) einer anderen WWW Ressource (HTML Seite, Bild, CGI Programm, etc.). Klickt ein Benutzer auf einen Hyperlink, wird die im URL angegebene Ressource angefordert. Der Hyperlink:

*Informationen zu unseren Produkten. *

wird von den meisten Browsern als unterstrichener Text dargestellt und referenziert eine HTML Seite, die bei Anwahl des Hyperlinks mit der Maus per HTTP angefordert wird.

3.3 Hypertext Transfer Protocol

Das *Hypertext Transfer Protocol* (HTTP) ist ein vom W3C standardisiertes, zustandsloses Protokoll zur Kommunikation zwischen WWW Clients und WWW Servern. Die Adressierung von Ressourcen erfolgt mittels URL's (vgl. Abschnitt 3.1). Das Verständnis des Protokolls dient nicht nur der Programmierung von Clients und Servern, sondern ist auch für die Programmierung von CGI Anwendungen (vgl. Abschnitt 3.4) unabdingbar [Spainhour, Quercia 96].

3.3.1 HTTP Grundlagen

HTTP Transaktionen folgen einem immer gleichen Schema, das sich sowohl beim Client als auch beim Server aus drei Teilen zusammensetzt:

- Der Client baut eine Verbindung zu einem bestimmten TCP Port — in der Regel Port 80 — auf. Er sendet eine Dokumentenanfrage (engl. *request line*) in Form eines HTTP Befehls, einer sogenannten *Methode* (vgl. Abschnitt 3.3.2), gefolgt von einer Ressourceadresse und einer HTTP Versionsnummer. So benutzt z.B. der Befehl:

  GET /welcome.html HTTP/1.0

  die Methode *GET*, um ein Dokument *welcome.html* anzufordern. Der Client benutzt dazu die HTTP Version 1.0.

- Als nächstes sendet der Client einen optionalen Nachrichtenkopf (engl. *request header*), um den Server über seine Konfiguration und die Dokumentenformate zu informieren, die er verarbeiten kann.

- Eventuell sendet der Client danach weitere zusätzliche Daten (engl. *request entity body*) zum Server, die dort zu einem verarbeitenden Programm übertragen werden.

Der Server antwortet darauf wie folgt:

- Die Antwort des Servers besteht aus einer Statuszeile (engl. *response line*), die die benutzte HTTP Version des Servers, eine Zustandskennung und deren Beschreibung enthält.

  HTTP/1.0 200 OK

  Die Zustandskennung ist eine dreistellige Dezimalzahl, die das Ergebnis des Servers auf die Anfrage des Client darstellt. Die Beschreibung ist lesbarer Text, der die Statuskennung erklärt.

- Als nächstes sendet der Server Informationen (engl. *response header*) über sich selbst und das gewünschte Dokument (z.B. Datum der letzten Änderung, Typ, Länge, etc.).

- War die Anfrage des Client erfolgreich, folgt das angeforderte Dokument (engl. *response entity body*). Dieses kann eine Kopie einer Datei oder die Antwort eines CGI Programms sein. Einige Server ermöglichen alternativ zu CGI Skripten die Einbindung von Skriptcode (vgl. Abschnitt 3.5.1) in HTML Dokumenten, der dann vom Server ausgeführt wird. Dies wird als *server-side scripting* bezeichnet.

Bei Verwendung von HTTP 1.0 bricht der Server die Verbindung zum Client ab und beendet die Transaktion, nachdem er die gewünschten Daten gesendet hat. Bei Nutzung von HTTP 1.1 bleibt die Verbindung zwischen Client und Server bestehen, um dem Client weitere Anfragen zu ermöglichen. Dies ist sinnvoll, da HTML Dokumente oft andere Dokumente (z.B. Bilder, Rahmen, Java Applets, etc.) enthalten und der wiederholte Verbindungsaufbau zum gleichen Server eingespart wird. Die Verbindung bleibt bestehen, solange sie nicht von Client oder Server explizit beendet wird.
Da HTTP ein zustandsloses Protokoll ist, bewahrt es keine Informationen von einer zur nächsten Transaktion. Jede Verbindung eines Client zu einem Server wird vom Server unabhängig von der vorhergehenden behandelt. Mechanismen, die die Emulation von individuellen Kommunikationszuständen zwischen Client und Server ermöglichen, sind von HTTP unabhängig und werden im Zusammenhang mit CGI Skripten (vgl. Abschnitt 3.4) verwendet.

3.3.2 HTTP Methoden

Eine Methode ist ein HTTP Befehl, mit dem die erste Zeile einer Client Anfrage beginnt. Nach dem Methodennamen folgt die Adresse des Zielobjektes. Folgende Methoden sind in der HTTP Version 1.1 gebräuchlich:

GET Die GET Methode fordert eine Information unter einer bestimmten Adresse vom Server an. Die Berechnung des Ergebnisses erfolgt durch den Server und kann auf unterschiedliche Art erreicht werden (Kopie einer Datei, Antwort eines CGI Programms, *server-side scripting*).

HEAD Diese Methode entspricht der GET Methode mit dem Unterschied, daß nur Metainformationen (z.B. Dateigröße, Datum der letzten Änderung, etc.) einer Ressource angefordert werden.

POST Die POST Methode erlaubt die Sendung von Daten vom Client zum Server in einer Clientanfrage. Die Daten werden im dritten Teil einer Clientanfrage — dem *request entity body* (vgl. Abschnitt 3.3.1) — an den Server gesandt. Dieser gibt die Daten an ein Programm an der spezifizierten Adresse weiter (vgl. Abschnitt 3.4).

Weitere Methoden sind u.a.:

PUT dient dazu, eine Ressource auf dem Server unter einem angegebenen URL zu speichern.

DELETE sorgt für das Löschen von Daten auf dem Server an einem spezifizierten URL.

OPTIONS fordert Informationen über auf dem Server verfügbare Kommunikationsoptionen an (z.B. eine Liste der vom Server unterstützten Methoden).

TRACE verlangt die intakte Rückgabe des *request entity body*. Dieser dient dann zur Fehlersuche.

Insbesondere die Methoden GET und POST sind für die Übertragung von Parametern und Formulareingaben von Benutzern an CGI Programme nötig.

3.4 Common Gateway Interface

Das *Common Gateway Interface* (CGI) ist der Teil eines WWW Servers, der mit anderen auf dem Server ablaufenden Programmen kommunizieren kann. Mit Hilfe des CGI kann der WWW Server ein beliebiges Programm aufrufen und diesem benutzerspezifische Parameter übergeben. Das Programm berechnet dann aufgrund dieser Parameter eine Ausgabe, die dem Benutzer in seinem Browser präsentiert wird [Gundavaram 96; Spainhour, Quercia 96].

Die Kommunikation eines Benutzers mit einem CGI Programm erfolgt über HTML Formulare. Diese Formulare können Benutzerschnittstellenelemente wie z.B. Textfelder, Auswahllisten, Optionsfelder, etc. enthalten und legen außerdem die zur Sendung des Formulars benutzte HTTP Methode fest. Klickt der Benutzer in einem Formular auf eine sogenannte "Submit" Befehlsschaltfläche, werden die Daten aus dem Formular bei Benutzung der HTTP GET Methode an den URL in der *request line* (vgl. Abschnitt 3.3.1) angehängt. Wird die POST Methode verwendet, werden die Daten im *request entity body* an den WWW Server übergeben. Der WWW Server reicht sie dann an ein CGI Programm in Form von Varible=Wert Paaren weiter. Dies geschieht bei Gebrauch der GET Methode über die UNIX Umgebungsvarible QUERY_STRING und bei Benutzung der POST Methode als Standard Eingabe (STDIN). Andere Umgebungsvariablen enthalten z.B. das Format und die Länge der Eingabe, die Rechneradresse des Client, die E-Mail Adresse des Client, den Namen des Servers, das Kommunikationsprotokoll, etc.

Aufgrund des zustandslosen HTT Protokolls gibt es keine unterschiedlichen Kommunikationszustände zwischen Client und Server. Aus diesem Grund muß bei einer neuen Anfrage ein Client einem Server zusätzliche Informationen über sich, bzw. früher gemachte Anfragen mitteilen, damit der individuell reagieren kann. Diese Informationen können sich in versteckten HTML Formularfeldern befinden, die in einer vorherigen Serverantwort generiert wurden oder in einer Datei (Netscape Cookies, s.u.) , die sich auf dem Rechner des Client befindet.

Netscape Cookies

Cookies sind ein von der Firma Netscape in ihrem Browser Navigator verwendeter Mechanismus zur persistenten Speicherung von clientspezifischen Informationen auf dem Rechner des Client. Diese Informationen werden beim wiederholten Zugriff auf einen URL automatisch vom Client einer Anfrage hinzugefügt und ermöglichen dem Server ein Personalisieren seiner Antwort für jeden Client. Cookies arbeiten auf folgende Weise:

1. Identifiziert ein CGI Programm einen neuen Benutzer, fügt es einen zusätzlichen *response header* (vgl. Abschnitt 3.3.1) seiner Antwort hinzu, der einen Bezeichner für diesen Benutzer enthält und andere Informationen, die der Server aus der Eingabe des Client entnimmt.

2. Aufgrund des zusätzlichen *response header* speichert der Browser diese Information in der Cookies Datei des Client.

3. Alle weiteren Anfragen des Browser an diese URL enthalten die Cookie Information als einen zusätzlichen *request header*.

4. Das CGI Programm benutzt diese Information, um eine Antwort zu generieren, die auf diesen spezifischen Benutzer zugeschnitten ist.

Cookies können außer vom Browser auch von in HTML Seiten befindlichen Skripten gelesen und geschrieben werden. JavaScript — ebenfalls ein Produkt der Firma Netscape — ist eine Skriptsprache, die in HTML Seiten verwendet werden kann.

3.5 Aktive Komponenten

HTML Dokumente bieten keine Möglichkeit, auf Benutzereingaben dynamisch zu reagieren. Tabelle 3.1 zeigt verschiedene Technologien, die auf der Client- wie auf der Serverseite eingesetzt werden können, um dynamisches Verhalten zu realisieren. Diese Technologien sind kommerzielle Produkte, die nicht vom W3C standardisiert werden und daher teilweise proprietär sind.

Client	Server
Client-Side-Scripting: JavaScript, VBScript, JScript	Server-Side-Scripting: JavaScript, VBScript, JScript
Java Applets	Java Servlets
ActiveX Controls	ActiveX Controls
Netscape Plug-Ins	

Tabelle 3.1: Aktive Komponenten im WWW Kontext

Die Technologien lassen sich konzeptuell in zwei Gruppen unterteilen. Die erste Gruppe bilden die Skriptsprachen JavaScript, VBScript und JScript. Der Skriptcode wird direkt in HTML Seiten eingefügt und vom Browser oder Server ausgeführt (vgl. Abschnitt 3.5.1). Die zweite Gruppe bilden Java, ActiveX und Netscape Plug-Ins, die kompilierten Code auf Client- und Serverseite einsetzen, um die Fähigkeiten von Browsern und Servern zu erweitern (vgl. Abschnitt 3.5.2).

3.5.1 Skripte

Skripte im WWW Kontext erfüllen den Zweck, die Fähigkeiten von HTML Seiten zu erweitern. Man unterscheidet *client-side scripting*, bei dem die Auswertung des Skriptcodes nach der Übertragung der HTML Seite vom Browser erfolgt, und *server-side scripting*, wobei der Skriptcode vor der Übertragung auf dem Server ausgeführt wird.

JavaScript

JavaScript ist eine interpretierte Programmiersprache der Firma Netscape und erlaubt die Verteilung von ausführbaren Inhalten über das WWW. Dies ist so noch kein Unterschied zu sogenannten Applets, die in der Programmiersprache Java ebenfalls über das WWW verteilt werden können. Während Applets jedoch in einer geschlossenen Umgebung ablaufen — der virtuellen Java Maschine (JVM) — kann JavaScript direkt in HTML Seiten verwendet werden und mit der Seite interagieren. JavaScript verfügt u.a. über folgende Fähigkeiten [Spainhour, Quercia 96]:

- Kontrolle des Erscheinungsbildes und des Inhaltes von HTML Dokumenten, z.B. lassen sich beliebige HTML Markierungscodes in Abhängigkeit von anderen Berechnungen in die Seite einfügen,
- Kontrolle des Verhaltens des Browser, z.B. existieren Methoden zur Erzeugung von Dialogboxen oder zum Öffnen und Schließen neuer Browser Fenster,
- Interaktion mit dem Inhalt von Dokumenten, z.B. lassen sich Benutzereingaben in HTML Formularen vor dem Abschicken des Formulars an den Server überprüfen und gegebenenfalls Neueingaben veranlassen,
- Interaktion mit dem Benutzer, z.B. kann, immer wenn ein Benutzer seine Maus an einer bestimmten Stelle des Dokumentes positioniert, eine Aktion des Browser ausgelöst werden,
- Lesen und Schreiben von Cookies,
- Ausführung beliebiger mathematischer Berechnungen.

Durch die Fähigkeit mit dem Inhalt von Dokumenten zu interagieren, kann JavaScript auch zur Steuerung von Java Applets (vgl. Abschnitt 3.5.2) dienen und somit die Rolle des Vermittlers zwischen einer HTML Seite und einem Java Applet annehmen (vgl. Abbildung 3.3) [Mintert 96].

Abbildung 3.3: JavaScript Code als Vermittler zwischen HTML und Java Code

VBScript und JScript

VBScript sowie JScript sind interpretierte Programmiersprachen der Firma Microsoft. JScript ist kompatibel zu JavaScript und hat den gleichen Umfang wie das von der Programmiersprache VisualBasic abgeleitete VBScript. Beide Sprachen verfügen über die o.g. Fähigkeiten von JavaScript.

3.5.2 Programmcode

Im Gegensatz zu Skriptcode (vgl. Abschnitt 3.5.1) wird Programmcode nicht in Form von Quelltexten ausgetauscht, sondern entweder als Bytecode (Java, vgl. Abschnitt 3.5.2.1) oder als Maschinencode (ActiveX, vgl. Abschnitt 3.5.2.2 / Plug-Ins, vgl. Abschnitt 3.5.2.3). Aufgrund der starken Unterschiede zwischen diesen beiden Modellen werden im folgenden die wesentlichen Merkmale von Java und ActiveX erläutert.

3.5.2.1 Java

Java ist eine objektorientierte C++-ähnliche Programmiersprache der Firma Sun Microsystems [Lemay, Perkins 96]. Seine große Popularität hat Java im wesentlichen der Tatsache zu verdanken, daß Java Programme nicht direkt in ausführbaren Maschinencode übersetzt werden, sondern in eine Zwischenrepräsentation (Bytecode), die auf jeder beliebigen Hardwareplattform verwendet werden kann, für die ein Java-Interpreter — die sogenannte *Java Virtual Maschine* — existiert (vgl. Abbildung 3.4). Mit der Plattformunabhängigkeit macht sich Java eines der wesentlichen Merkmale des WWW zu eigen. Im Vergleich mit Programmen in Maschinencode sind Java Programme langsam [Reibold et al. 97]. Abhilfe schafft hier ein sogenannter *Just in Time Compiler* (JIT), der Java Bytecode in Maschinencode übersetzt.
Es existieren drei verschiedene Ausprägungen von Java Programmen:

Anwendungen: Java Anwendungen sind Programme, die in der Sprache Java geschrieben und in der JVM ausgeführt werden.

Applets: Applets sind Java Programme, die in HTML Seiten eingebunden über das WWW übertragen und dann beim Client in einem Java-fähigen Browser ausgeführt werden.

Servlets: Servlets sind Java Programme, die Erweiterungen von Serverfunktionalität unabhängig von Server Software und der verwendeten Plattform erlauben [Paradies 97].

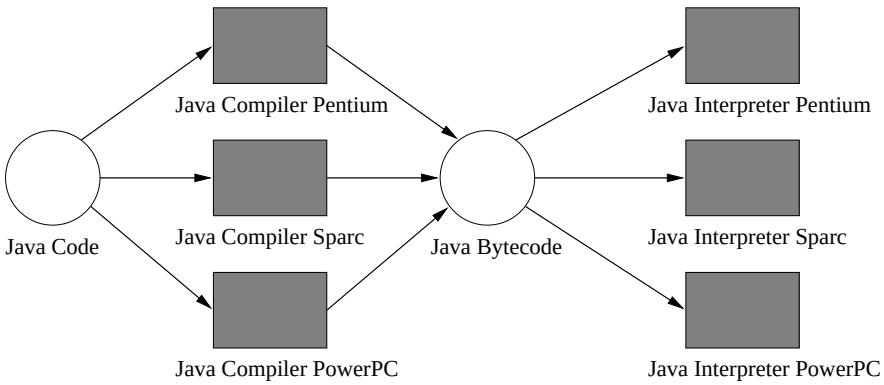

Abbildung 3.4: Erstellung und Ausführung von Java Programmen

Sie sind zu einer spezifischen Schnittstelle konforme Java Objekte, die im Gegensatz zu Applets keine grafische Benutzerschnittstelle haben.

Desweiteren unterstützt Java die drei entscheidenden Konzepte objektorientierter Programmiersprachen [Meyer 90]:

Kapselung: Daten und Methoden zur Verarbeitung der Daten sind zu einer Einheit in Klassen zusammengefaßt.

Vererbung: Von bereits bestehenden Klassen können neue abgeleitet werden. Die abgeleiteten Subklassen erben die Eigenschaften ihrer Superklassen und können um weitere Daten und Methoden erweitert werden.

Polymorphie: Ein Name in einem Programm (eine Variable) kann an Werte verschiedener Typen gebunden werden.

Java verfügt über umfangreiche Klassenbibliotheken, die sogenannten *Packages* [Schneider 96]. Neben den Basisklassen für grundlegende Datentypen wie Object, Integer, Float, String, etc. gibt es u.a. Packages für Applets, grafische Benutzerschnittstellen, Dateibehandlung und Netzwerkunterstützung. Eigene Klassen können von diesen Klassen erben und eigene Packages bilden.

Sicherheitsmechanismen

Mit Java Applets kommt unbekannter Programmcode — eventuell sogar unbemerkt — auf dem Rechner des Client zur Ausführung. Dies erfordert Sicherheitsmechanismen, die es verhindern, daß:

- lokale Dateien und Verzeichnisse geöffnet, gelesen oder verändert werden,
- Betriebssystembefehle aufgerufen werden,

- andere Programme gestartet werden,
- Netzwerkverbindungen zu anderen Rechnern außer dem Herkunftsrechner des Applet aufgebaut werden.

Java verhindert die o.g. Operationen auf vier Ebenen: der Programmiersprache selbst, der virtuellen Java Maschine (JVM), des Lademechanismus für Java Klassen und der Java Klassenbibliothek [Templ 96; Huber 96].

3.5.2.2 ActiveX

ActiveX ist die Antwort der Firma Microsoft auf Java [Reibold et al. 97]. Basis für ActiveX ist das unter Microsoft Windows verwendete OLE (engl. *Object Linking and Embedding*). OLE ermöglicht den Austausch von Daten zwischen verschiedenen Anwendungen. So ist es z.B. möglich, Daten einer Tabellenkalkulation als ein Objekt in ein Dokument einer Textverarbeitung einzubetten und so diese beiden Dokumente miteinander zu verknüpfen. Wenn die eingebettete Tabelle angewählt wird, wird das Tabellenkalkulationsprogramm gestartet, um die Daten zu bearbeiten. Werden andererseits unabhängig vom Textdokument die Tabellendaten geändert, übernimmt das Textdokument die Änderungen automatisch.
Grundlage von OLE ist das *Component Object Model* (COM), das festlegt, wie Softwareobjekte im Speicher vorliegen und Objekte miteinander kommunizieren können. Softwarekomponenten sind als austauschbare Teile eines Ganzen zu interpretieren. So läßt sich z.B. eine Komponente zur Rechtschreibprüfung eines Herstellers problemlos in eine Anwendung eines anderen Herstellers integrieren. DCOM (*Distributed Component Object Model*) bietet die Möglichkeit Softwarekomponenten im Netz verteilt zu halten und bildet die Basis für die Internet Variante ActiveX.
ActiveX Controls sind in Binärcode übersetzte Programme, die nach der Übertragung über das WWW auf dem Rechner des Client installiert und ausgeführt werden. Da ActiveX mit OLE und DCOM auf der 32 Bit Microsoft Windows Plattform basiert, ist es bis jetzt eine proprietäre Lösung. An einer Portierung von ActiveX und DCOM auf UNIX wird allerdings bereits gearbeitet [Reibold et al. 97]. ActiveX hat den Vorteil, daß durch die Basis OLE und COM bereits eine große Anzahl an Komponenten zur Verfügung stehen, die als ActiveX Controls im Internet verwendet werden können.

Sicherheitsmechanismen

ActiveX Controls haben uneingeschränkten Systemzugriff. Das ermöglicht einerseits die nahtlose Kommunikation mit anderen Anwendungen, andererseits ist eine mißbräuchliche Nutzung nicht auszuschließen. Eine Sicherheitsmaßnahme, die Microsoft für ActiveX Controls anbietet, ist deren Zertifizierung. Mit Hilfe eines mitgelieferten Zertifikates kann sich ein Anwender nach dem Laden eines ActiveX Controls davon überzeugen, daß diese Software mit derjenigen identisch ist, die der Entwickler abgeschickt hat. Dieser Mechanismus bietet jedoch keinen wirklichen Schutz vor Mißbrauch, wie ein von Microsoft zertifiziertes Acti-

veX Control zeigt, daß nach seinem Start das Windows System des Anwenders abschaltet [Reibold et al. 97].

3.5.2.3 Plug-Ins

Netscape Plug-Ins sind wie ActiveX Controls ausführbare Programme in Maschinencode und daher plattformabhängig. Sie können mit dem Netscape Navigator über definierte C/C++ Schnittstellen kommunizieren und sind in eine HTML Seite integrierbar. Plug-Ins reagieren auf MIME Typen von Dokumenten. Ist ein Dokument von einem bestimmten MIME Typ und ist dieser einem Plug-In zugeordnet, wird dieses gestartet und übernimmt die Behandlung bzw. Darstellung des Dokumentes.
Plug-Ins haben wie ActiveX Controls uneingeschränkten Systemzugriff und stellen somit eine Sicherheitsrisiko für den Benutzer und sein System dar. Während sich ActiveX Controls eventuell automatisch installieren, müssen Plug-Ins explizit vom Benutzer installiert werden.

4. Bewertung vorhandener Konzepte und Technologien verteilter Informationssysteme

Die grundlegenden Konzepte und Technologien des WWW (Adressierung mittels URLs, HTTP, HTML, CGI) sind bereits einige Jahre alt. In diesem Kapitel werden Konzepte und Technologien des WWW (vgl. Kapitel 3) anhand der in Kapitel 2 erarbeiteten Anforderungen an verteilte Informationssysteme beurteilt (vgl. Abschnitt 4.1).
Einige Vorteile der verteilten Informationssysteme Hyperwave und Netiva gegenüber dem WWW werden in Abschnitt 4.2 vorgestellt. Während Hyperwave über eigene Client- und Serversoftware verfügt, setzt Netiva einen Java fähigen WWW Browser als Clientsoftware voraus.

4.1 Das WWW

Das World Wide Web ist das Internet Informationssystem mit den seit Jahren größten Wachstumsraten. In diesem Abschnitt wird untersucht, inwiefern das WWW Anforderungen an verteilte Informationssysteme (vgl. Kapitel 2) erfüllt und wo Defizite grundlegender Konzepte und Technologien bestehen.

4.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen von Nutzern und Autoren verteilter Informationssysteme wie dem WWW sind unterschiedlich. Dies trifft insbesondere für das WWW zu, da nicht jeder Benutzer des Systems die Möglichkeit hat, selbst Informationen im WWW zu veröffentlichen. Im Folgenden werden die funktionalen Anforderungen des Nutzers an verteilte Informationssysteme auf das WWW übertragen und die dort vorhandenen Konzepte zu deren Erfüllung bewertet:

Orientierung: Im WWW sieht der Benutzer in der Regel nur eine HTML Seite. Insofern ist es für ihn nicht möglich, sich auf einem Server zu orientieren und zu erfahren, in welchem Kontext die gerade betrachtete Seite mit anderen Informationen auf dem

Server steht. Die Orientierung kann nur von dem Autor eines Informationsangebotes z.B. durch die Nutzung von HTML *frames* erleichtert werden. Hyperwave bietet für die Orientierung des Benutzers auf einem Server verschiedene vom Autor unabhängige Konzepte (vgl. Abschnitt 4.2.1).

Navigation: Im WWW ist die wichtigste Möglichkeit zur Navigation im Informationsraum das Verfolgen von Hyperlinks. Die Eingabe einer URL erlaubt den direkten Zugang zu Informationen deren Adresse bekannt ist. Browser bieten desweiteren Möglichkeiten zum Speichern von URL's. Diese können wie ein Lesezeichen in einem Buch später wieder aufgerufen werden. Desweiteren werden die in einer Sitzung besuchten Adressen in einer Liste gespeichert und sind während der Sitzung weiterhin zugänglich. Beim Beenden des Browser Prozesses geht diese Liste jedoch verloren, so daß sich der Benutzer frühzeitig entscheiden muß, ob er einen URL in seine Lesezeichenliste mit aufnehmen will. Hyperlinks, die bereits aufgerufen wurden, werden vom Browser in einer anderen Farbe dargestellt als Hyperlinks, die noch nicht verfolgt wurden. Hyperwave bietet im Zusammenhang mit seinen erweiterten Orientierungsfunktionen neben dem Verfolgen von Hyperlinks auch andere Navigationsmöglichkeiten an.

Suchmöglichkeiten: Die Suchmöglichkeiten im WWW sind nicht systemimmanent sondern werden mit Hilfe des CGI von externen Applikationen bereitgestellt, die auf Datenbanken zugreifen. Die Datenbanken werden durch das Abrufen von HTML Seiten und der Auswertung der darauf enthaltenen Texte gefüllt und speichern die URL der HTML Seite samt der gesammelten Informationen. Der Suchraum ist bei Suchmaschinen wie z.B. Lycos auf Audio- und Bilddateien bzw. verschiedene verwendete Sprachen einschränkbar. Suchmaschinen im WWW basieren auf der nachträglichen Auswertung von HTML Dateien, während in Systemen wie Hyperwave Informationen inklusive ihrer Metadaten (Autor, Datum, Stichworte, Typ der Information, etc.) in strukturierter Form in Datenbanken vorliegen. Dadurch sind die Suchergebnisse in diesen Systemen immer aktuell, da sie auf den Originaldaten beruhen, während die Ergebnisse von Suchmaschinen wie Lycos auf Zweitauswertungen der Originalinformation basieren und die Suchmaschine über Änderungen oder Löschungen von HTML Seiten nicht informiert wird.

Visualisierung von Änderungen: Die *push* Technologie erlaubt es dem Benutzer im WWW Informationen zu abonnieren. Ein Client erhält dann unter Zuhilfenahme einer speziellen Software automatisch die jeweils neusten auf einem Server vorliegenden Informationen. Das geschieht dadurch, daß die *push* Software beim Bestehen einer Netzwerkverbindung regelmäßig beim Server überprüft, ob dort neue vom Benutzer abonnierte Informationen vorliegen und diese dann automatisch anfordert. Solche Systeme werden z.B. zur automatischen Verteilung von Börsenkursen, HTML Seiten oder Software eingesetzt [Kirsch 97].

Transparenz und Bedienbarkeit: Die transparente Bedienbarkeit des WWW ist für den Benutzer gegeben, da er nur Hyperlinks verfolgt oder Formulare ausfüllt. Die Anfor-

derung und Übertragung der Informationen übernimmt der Browser, der auch für die Darstellung der Information sorgt, sofern das Datenformat bekannt ist.

Für Erstellung und Wartung von Information im WWW haben Autoren weitere Anforderungen:

Statische und dynamische Informationserstellung: Informationen im WWW können sowohl statisch als auch dynamisch erstellt werden. Statische Informationen liegen z.B. in Form von HTML Seiten, Bildern, Java Applets als Dateien auf dem Server vor und werden auf Anfrage eines Client ausgeliefert. Dynamische Informationen werden in der Regel in Form von HTML Seiten mit Hilfe der CGI Schnittstelle als Antwort auf eine individuelle Anfrage eines Client erzeugt (vgl. Abschnitt 3.4). Die CGI Schnittstelle wird auch für Client/Server Anwendungen im WWW genutzt, ist dafür aber nicht geeignet, da keinerlei Typinformation über die Parameter in Anfragen mitgeliefert werden und HTTP ein zustandsloses Protokoll ist.

Autorisierte Erstellung und Wartung: Mit der HTTP Methode *PUT* ist es möglich, Dateien vom Client zum WWW Server zu übertragen und dort zu speichern. Dies kann aber nur mit zugehörigen Sicherheitsmechanismen erfolgen, damit nicht jeder beliebige Benutzer eine WWW Seite ändern kann. Server, die die PUT Methode implementieren, bieten dementsprechend Authentisierungs- und Autorisierungmechanismen an. Da WWW Server statische Informationen im Dateisystem ablegen, muß der Systemadministrator darauf achten, daß dort nur befugte Autoren Schreibrechte haben.

Konsistenzerhaltende Wartung: Da die Erstellung von Informationen derzeit meist außerhalb des WWW (z.B. mit Hilfe von HTML Editoren, CGI Skripten und Datenbanken) erfolgt, müssen hier Mechanismen wie z.B. Versionierung eingesetzt werden, um bei Änderungen von Ressourcen die Konsistenz von Verweisen auf die alten Informationen zu erhalten. Kümmert sich ein Autor nicht um die Erhaltung der Konsistenz, z.B. dadurch daß er HTML Dateien umbenennt oder löscht, werden die Verweise auf diese Information ungültig und sind unbrauchbar.

Automatische Erzeugung von Hyperlinks: Die Wiederverwendung bereits bestehender Information durch die automatische Erzeugung von Hyperlinks wird im WWW nicht von System unterstützt. Der Autor muß die von ihm referenzierten Ressourcen und deren URL kennen, um einen Hyperlink zu erzeugen.

Trennung von Inhalt und Präsentation: Im WWW erlaubt HTML die Trennung von Inhalt und Präsentation. Während der Inhalt als ASCII Text und die Funktion des Inhalts (z.B. Überschrift, Aufzählung, Tabelle) als HTML Schlüsselworte definiert werden, sorgen die *style sheets* für die Festlegung der Formatierung des Inhalts. Liegen keine vom Autor definierten *style sheets* vor, übernimmt der Browser die Formatierung des Inhalts.

Die oben erwähnten Vorteile von Hyperwave werden in Abschnitt 4.2 näher erläutert.

4.1.2 Architekturelle Anforderungen

Die architekturellen Anforderungen an verteilte Informationssysteme aus Abschnitt 2.2 werden im Folgenden auf das WWW übertragen und die damit zusammenhängenden Konzepte und Defizite betrachtet.
Um die Konsistenz in einem Informationssystem zu erhalten, sollten folgende Forderungen erfüllt werden:

Objektbenennung: Die Identifizierung von Objekten erfolgt im WWW anhand ihrer URLs (vgl. Abschnitt 3.1). Objekte werden anhand ihrer Adresse und des Zugangsverfahrens referenziert. Bei Adreßänderungen kommt es so zu Inkonsistenzen. Würden stattdessen URNs zur Identifizierung von Objekten eingesetzt, könnten Hyperlinks unabhängig von der Adresse eines Objektes definiert werden und wären auch bei Adreßänderungen (z.B. Verschieben einer Datei in ein anderes Verzeichnis) weiterhin gültig.

Redundanzfreiheit: Das WWW bietet keine Möglichkeit die Redundanzfreiheit der Informationsinhalte sicherzustellen. Jede HTML Datei kann unter beliebig vielen Adressen den gleichen Inhalt haben. Dies ist im WWW ein Problem, wenn Informationen wie z.B. Adressen oder Telefonnummern von Organisationen auf mehreren statischen HTML Seiten verwendet werden. Eine konsistente Änderung dieser Dateien wird nicht vom WWW unterstützt. Bei dynamischen Informationen, die aus Datenbanken gewonnen werden, sind die Datenbanken für die Redundanzfreiheit der Information verantwortlich.

Versionierung: Das WWW bietet keine Konzepte und Technologien für die Versionierung von Ressourcen.[1] Verschiedene Ansätze wie z.B. Server- und Protokollerweiterungen werden in Kapitel 5 betrachtet.

Neben der Konsistenz der Informationen sind von einem verteilten Informationssystem folgende Anforderungen zu erfüllen:

Strukturierte Datenspeicherung: Die WWW Infrastruktur (URLs, HTTP, HTML, CGI) stellt keine Konzepte für die Speicherung der Informationsinhalte bereit. Während statische HTML Seiten als Dateien im Dateisystem gespeichert werden, bietet CGI die Verbindung von WWW Servern zu beliebigen anderen Programmen wie z.B. Datenbanken. WWW Server wie z.B. Jigsaw (vgl. Abschnitt 5.4.2) verwalten eigene Datenstrukturen der von ihnen bereitgestellten Information und bieten so den Zugriff auf strukturierte Metainformation.

Skalierbarkeit: Die Performanz des WWW hängt von vielen Faktoren ab (Bandbreiten zwischen Benutzer, Internet und Informationsanbieter, Verkehr auf dem Internet, Auslastung der Server). Um Engpässe bei der Leistung der Server zu umgehen, gibt es im

[1] Der neuste Standardisierungsvorschlag des WWW Konsortiums für HTML 4.0 unterstützt die Versionierung von HTML Dateien (vgl. Abschnitt 8.1.1).

WWW das Konzept der Zwischenspeicherung von Informationen auf *Proxy* Servern. Hierbei erreichen alle Anfragen des Client zuerst einen Proxy Server, der versucht, die Anfrage des Client aus seinem eigenen Zwischenspeicher zu bedienen, bevor er eine Anfrage an den in der Clientanfrage genannten Server stellt. Dadurch wird der Netzwerkverkehr reduziert und der Originalserver entlastet. Das auf dem Internet Protokoll basierende WWW ist es nicht an einzelne Netzwerkverbindungen gebunden. Die Umleitung des Netzwerkverkehrs erfolgt bei Ausfall eines Internetrechners automatisch über andere Rechner.

Anpassung an variable Netzkapazitäten: Ein Benutzer kann bei geringen Bandbreiten die Übertragung von Bildern und Java Applets mit Hilfe des Browsers unterdrücken. Eine automatische Anpassung wäre zusätzlich wünschenswert, z.B. mit Hilfe des IPv6 und des RSVP (vgl. Abschnitt 8.3).

Integrität: Das WWW unterstützt die sichere Übertragung von Informationen im WWW mit S-HTTP. Eine proprietäre Variante der sicheren Datenübertragung ist die *secure socket layer* (SSL) von Netscape. Einfache, unverschlüsselte Authentifizierung mittels Passwortabfragen bietet HTTP. WWW Server unterstützen das Protokollieren aller Zugriffe auf den Server (engl. *auditing*).

Die grundlegende Architektur des WWW — in Form des verbindungslosen HTT Protokolls und der zustandslosen WWW Server — führen zu Problemen bei verteilten Anwendungen, mit denen aus mehreren Teilschritten bestehende Transaktionen durchgeführt werden sollen (vgl. Abschnitt 3.4 und Kapitel 7). Für die einfache Übertragung von Information in Form von Dateien (HTML Seiten, Bilder, Java Applets) ist das WWW jedoch gut geeignet.

4.1.3 Technische Anforderungen

Die in Abschnitt 2.3 erarbeiteten technischen Anforderungen werden im Folgenden auf das WWW übertragen:

Plattformunabhängigkeit: Plattformunabhängigkeit ist im WWW durch plattformunabhängige Informationsformate (HTML, GIF, JPEG, Java, JavaScript) gegeben. Microsoft versucht mit ActiveX ein plattformabhängiges Codeformat im WWW zu etablieren. Dadurch entsteht für den Informationsanbieter hoher Aufwand, da er für jede Plattform eigene ActiveX Komponenten zur Verfügung stellen muß.

Kompatibilität: Die meistgenutzte Kompatibilitätsschnittstelle im WWW ist das CGI, da sie von allen Servern unterstützt wird und auf der Clientseite lediglich das Ausfüllen von HTML Formularen bzw. das Aufrufen von Hyperlinks erfordert. Die Java Applet Schnittstelle von Browsern erlaubt die sichere Ausführung von Programmcode auf dem Clientrechner. Schnittstellen zu beliebigen heterogenen Systemen stehen durch die Unterstützung von CORBA (vgl. Kapitel 7) im Netscape Browser Version 4.0 zur Verfügung. Auf der anderen Seite wird in neueren Betriebssystemen von Microsoft der

Browser mit der normalen Benutzerschnittstelle verschmelzen, so daß der Benutzer keinen Unterschied mehr zwischen lokaler und entfernter Information feststellt [Microsoft 96].

Migration: Migrationskonzepte sind in den WWW Technologien nicht vorhanden, da es sich um ein klassisches Client/Server System handelt. Es ist aber denkbar, daß die von Browsern unterstützte Programmiersprache Java mit solchen Konzepten und Bibliotheken versehen wird und der Benutzer migrierende Agenten mit seinen Aufträgen im WWW umherschicken kann, die ihn mit speziellen Informationen versorgen.

Ressourcenunabhängigkeit: Die Ressourcenunabhängigkeit ist im WWW Kontext insbesondere für Java Applets wichtig, die sich evtl. auf das Vorhandensein bestimmter benötigter Bibliotheken verlassen. Browser versuchen nicht lokal vorhandene Java Bibliotheken nachzuladen. Dieses Vorgehen ist auch für proprietäre Informationsformate denkbar, die auf dem Clientrechner nicht bekannt sind. So könnten bei unbekannten Informationsformaten vom Browser entsprechende Java Applets oder ActiveX Komponenten angefordert werden, die für die Darstellung der unbekannten Objekte sorgen.

Dadurch, daß die Datenspeicherung von Informationen außerhalb der Konzepte und Technologien des WWW stattfindet, kann das WWW in vielen Anwendungsfällen (z.B. Orientierung, Suchmöglichkeiten, Navigation) nicht mit auf Datenbanken basierenden Informationssystemen konkurrieren. In anderer Hinsicht wie z.B. der Plattformunabhängigkeit und der transparenten Bedienbarkeit ist es hingegen vorbildlich.

4.2 Andere verteilte Informationssysteme

In diesem Abschnitt werden ausgewählte Konzepte der Systeme Hyperwave und Netiva vorgestellt und gezeigt, inwiefern diese Vorteile gegenüber dem WWW bieten. Während Hyperwave (vgl. Abschnitt 4.2.1) einige grundlegend neue Konzepte (z.B. bidirektionale Hyperlinks) gegenüber dem WWW bietet, handelt es sich bei Netiva (vgl. Abschnitt 4.2.2) um ein System für Mehrbenutzerdatenbankanwendungen im WWW, das Beschränkungen und Komplexität von HTTP, HTML und CGI für Entwickler und Benutzer aufhebt.

4.2.1 Hyperwave

Hyperwave (ehemals Hyper-G) ist ein hypermediales Informationssystem für das Internet. Es wurde am Anfang seiner Entwicklung als ein eigenständiges „Internet Informationssystem der 2. Generation“ beschrieben [Dalitz, Hezer 95], während es heute, aufgrund des großen Erfolges des WWW, als eine Erweiterung des WWW gesehen wird [Maurer 96].
Die Architektur eines Hyperwave Systems ist in Abbildung 4.1 dargestellt. Es gibt spezielle Hyperwave Server und Clients sowie das sie verbindende *HyperG-Client Server Protocol* (HG-CSP). Ein Hyperwave Server ist auch über HTTP von WWW Browsern aus zugänglich.
Einige wesentliche Unterschiede zum WWW werden im Folgenden erläutert.

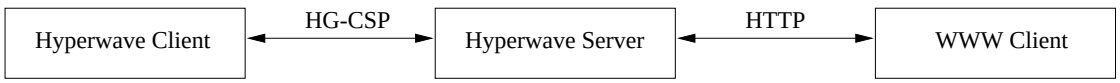

Abbildung 4.1: Architektur eines Hyperwave Systems

Objektbenennung

Ein Hyperwave Server kann Objekte beliebiger Art verwalten und speichert sie in einer Datenbank. Die Identifikation einzelner Objekte erfolgt mittels eines serverweit eindeutigen lokalen Objektbezeichners (OID). Im Zusammenhang mit der SID, die einen Server eindeutig bezeichnet, ergeben sich weltweit eindeutige globale Objektbezeichner, die in Hyperwave als *birthplace ID* bezeichnet werden. Anhand der OID wird ein Objekt in der Datenbank identifiziert und die zum Objekt gehörenden Metainformationen ausgewertet. Aus den Metainformationen wird die Adresse bzw. der Verzeichnispfad des Objektes gewonnen und letztlich das Objekt ausgeliefert.
Durch diese Art der Speicherung stehen umfassende Suchmöglichkeiten anhand der bereitgestellten Metainformationen (Autor, Titel, Schlüsselworte, Zugriffsrechte, Erstellungszeitpunkt, etc.) zu einem Objekt zur Verfügung. Desweiteren besteht Orts- und Adreßtransparenz. Referenzen auf Objekte bestehen immer nur aus der Kombination von SID und OID und müssen nicht geändert werden, wenn das Objekt seine Adresse ändert, wie es beim URL Konzept des WWW der Fall ist.

Strukturierung, Orientierung und Navigation

Während im WWW Informationsknoten nur mittels Hyperlinks strukturiert werden können, bietet Hyperwave wesentlich mehr Strukturierungsmöglichkeiten für die im System enthaltene Information. Hyperwave Objekte sind Instanzen spezifischer Klassen. Es gibt u.a. Klassen für Text-, Bild- und Klangdokumente. Ein der Strukturierung dienendes Objekt ist das Kollektionsobjekt. Eine *Kollektion* ist ein zusammengesetztes Objekt, das Dokumente oder andere Kollektionen enthält. Das Kollektionsobjekt ist mit einem Verzeichnis in einem Dateisystem vergleichbar. Die Kollektionen in einem Hyperwave System bilden einen gerichteten azyklischen Graphen — die Kollektionshierarchie —, dessen oberstes Objekt als *Wurzelkollektion* bezeichnet wird.
Eine Kollektionshierarchie wird in einem Hyperwave Client ähnlich einer Verzeichnisstruktur in einem zusätzlichen Fenster angezeigt. So kann ein Benutzer ein Dokument betrachten und gleichzeitig sehen, in welcher Kollektion er sich befindet und welche anderen Objekte in dieser Kollektion vorhanden sind. Außerdem kann er jederzeit in der Kollektionshierarchie Kollektionen öffnen und Dokumente aufrufen, die dann automatisch in einem anderen Fenster zur Dokumentendarstellung angezeigt werden. Folgt ein Benutzer einem Hyperlink in einem Dokument, wird nicht nur das jeweilige Dokument angezeigt, sondern auch die dazugehörige Stelle in der Kollektionshierarchie. Diese Orientierungs- und Navigationsmöglichkeiten sind jedoch mit WWW Browsern nicht zu nutzen sondern erfordern spezielle Hyperwave Clients.

Hyperlinks

Hyperlinks werden durch ihre Quell- und Zielanker definiert und nicht in Dokumenten selbst gespeichert. Der Quellanker ist der Startpunkt eines Hyperlinks und spezifiziert den Teil eines Dokumentes von dem aus der Hyperlink aktiviert werden kann. In einem Textdokument wird dieser z.B. hell unterlegt, während er in einem Video als Ausschnittrahmen dargestellt wird. Der Zielanker ist der Endpunkt eines Hyperlinks und beschreibt eine bestimmte Stelle in einem Dokument, die angezeigt wird, wenn man an diesem Knoten ankommt (z.B. ein spezifischer Absatz in einem Textdokument). Dokumente und Anker stehen in einer 1:n Beziehung, d.h. einem Dokument können mehrere Quellanker als auch mehrere Zielanker zugeordnet werden. Quell- und Zielanker stehen in einer n:1 Beziehung, d.h. viele verschiedene Quellanker können dem gleichen Zielanker zugeordnet werden. Soll ein Quellanker nicht zielgenau auf einen spezifischen Teil eines Objektes verweisen wird kein Zielanker für das Objekt angelegt. Der Hyperlink führt dann direkt vom Quellanker zu dem Dokument.
Das Wesentliche an diesem Konzept ist, daß es für den Client möglich ist, Hyperlinks in beide Richtungen zu verfolgen. Er kann ein Dokument zu einem Anker finden und andererseits alle Anker eines gegebenen Dokumentes. Da die Beziehung zwischen Quell- und Zielanker ebenfalls bidirektional ist, lassen sich auch alle Dokumente finden, die ein betrachtetes Dokument referenzieren. Mit bidirektionalen Hyperlinks ist es möglich die Konsistenz von Hyperlinks auch bei Änderungen an Dokumenten zu bewahren, z.B. indem der Autor eines Hyperlinks, der ein geändertes Dokument referenziert, über die Änderung informiert wird oder automatisch bei vorhandenen Rechten ein Quellanker entfernt wird.

4.2.2 Netiva

Netiva ist ein System zur Realisierung von Mehrbenutzerdatenbankanwendungen im WWW [Netiva 97b]. Es besteht aus einer Entwicklungsumgebung (Netiva *designer*) und einem Netiva WWW Server. Netiva Anwendungen sind Client/Server Datenbankanwendungen. Die Clientsoftware wird in Form von Java Applets per URL vom Netiva Server angefordert und dann in einem Browser ausgeführt. Die Java Applets kommunizieren mit dem Netiva Server und führen Datenbanktransaktionen durch (vgl. Abbildung 4.2).
Der Netiva Server hat folgende Aufgaben:

- Übertragung der Clientapplets zum Browser
- Management der Zustände der bestehenden Verbindungen
- Bereitstellung von Datenbankserverfunktionalität und Transaktionsmangement

Anwendungsentwicklung

Das Konzept von Netiva Anwendungen unterscheidet sich grundlegend von herkömmlichen relationalen Datenbankanwendungen [Lockemann, Schmidt 87], in denen zuerst die Tabellenstrukturen erstellt werden, um in einem zweiten Schritt bestimmte Sichten (Eingabeformulare, Abragen, Berichte) auf diese Strukturen zu entwickeln.

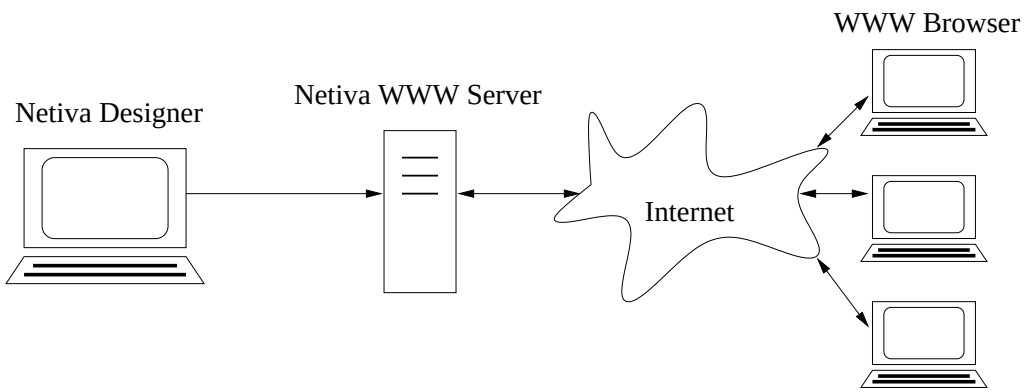

Abbildung 4.2: Architektur eines Netiva Systems

Die Anwendungsentwicklung von Netiva Datenbankanwendungen erfolgt in drei Schritten:

Gestaltung der Benutzerschnittstelle: Der Entwickler gestaltet per *drag and drop* sowohl Benutzerschnittstellenseiten (*front-end pages*) als auch Seiten, die nur für den Entwickler relevant sind (*back-end pages*). Während die *back-end pages* den Tabellen der Datenbank entsprechen, sind die *front-end pages* mit verschiedenen Sichten auf die Datenbanktabellen vergleichbar. Eine Seite kann z.B. Texteingabefelder, Schaltflächen, Tabellen und beliebige andere Benutzerschnittstellenelemente wie z.B. Grafiken und Hyperlinks enthalten.

Makroprogrammierung: Mit Hilfe einer VisualBasic ähnlichen Makroprogrammiersprache werden die *front-end pages* mit den *back-end pages* verknüpft. So kann z.B. ein Makro durch eine Schaltfläche aktiviert werden, das dafür sorgt, daß die in den Eingabefeldern gemachten Angaben in die Datenbank aufgenommen werden.

Bereitstellung der Anwendung auf dem Netiva Server: Aus *front-end pages*, *back-end pages* und Makros wird in einer einzelnen Datei eine Netiva Datenbankanwendung erzeugt. Außerdem wird eine HTML Seite generiert, die lediglich ein mit der entsprechenden Datenbankanwendung initialisiertes Netiva Clientprogramm in Form eines Java Applets referenziert. Eine Netiva Anwendung muß inkl. der zugehörigen HTML Seite in ein Verzeichnis des Netiva Servers kopiert werden und steht dann für Zugriffe aus dem WWW zur Verfügung.

Ein Netiva Entwickler kommt zu keinem Zeitpunkt mit HTTP, HTML, CGI oder Java in Berührung. Es werden keine Java Quellcodedateien erzeugt. Die gesamte Entwicklung findet im Netiva Designer statt. Neben der Neuentwicklung von Datenbanken können auch Datenbanken in anderen Formaten (z.B. Microsoft Access oder dBase) weiterverwendet werden, so daß mit Netiva Benutzerschnittstellen für bestehende Datenbanken im WWW verfügbar werden.

Bewertung

Anwendungen mit WWW Benutzerschnittstelle profitieren grundsätzlich davon, daß keine spezielle Clientsoftware installiert werden muß und daß neue Versionen der Clientanwendung automatisch verfügbar sind. Java Clientanwendungen bieten darüber hinaus den Vorteil der Dynamik, d.h. es können Überprüfungen von Eingabefeldern vorgenommen werden, ohne daß eine neue Verbindung zum Server erforderlich wird bzw. eine neue HTML Seite geladen werden muß.
Entwicklungsumgebungen wie der Netiva Designer setzen die bei VisualBasic begonnene Reihe der grafischen Entwicklungsumgebungen nahtlos für WWW Datenbankanwendungen fort. Datenbankanwendungen sind in sehr kurzer Zeit ohne Kenntnisse von HTML, HTTP, CGI, Skriptsprachen und Java zu entwickeln. Desweiteren ist es nicht notwendig, daß sich ein Netiva Entwickler mit Indizes, Fremdschlüsseln oder der Normalisierung von Datenbanken auskennt, da das System die Datenbankstruktur völlig transparent für den Entwickler aus den von ihm erstellten grafischen Benutzerschnittstellen generiert.
Ein entscheidender Nachteil von Netiva ist, daß das Komponentenmodell Java Beans (vgl. Abschnitt 8.2.3) noch nicht unterstützt wird. Dadurch würde es möglich, Java Beans beliebiger Hersteller in die Entwicklungsumgebung zu integrieren und die bisher beschränkte Funktionalität von Netiva zu erweitern.

5. Versionierung im WWW

Ausgehend von den Anforderungen an verteilte Informationssysteme im Kapitel 2 wird im Folgenden untersucht, welche Bedeutung das Management von Versionen von Ressourcen im Zusammenhang mit Anforderungen wie Konsistenzerhaltung und Redundanzfreiheit für das WWW hat.
Versionen sind verschiedene Instanzen einer Ressource (z.B. Text-, Bild-, Audio- oder Videodateien). Es werden zwei Ausprägungen von Versionen unterschieden [Hicks et al. 91]:

Revisionen: Eine Revision ist die Verfeinerung eines existierenden Objektes. In Abbildung 5.1 sind die Objekte 1.1 bis 1.4 eine Serie von Revisionen.

Varianten: Ein Objekt A ist eine Variante eines anderen Objektes B, wenn sich A auf einem zu B parallelen Entwicklungspfad befindet. In Abbildung 5.1 ist das Objekt 1.2.1.1[1] eine Variante des Objektes 1.3 und eine Revision des Objektes 1.2.

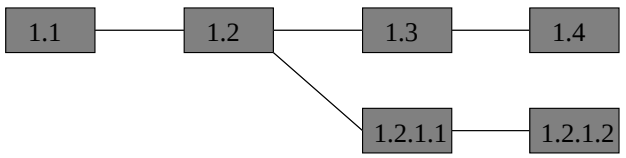

Abbildung 5.1: Revisionen und Varianten

Versionskontrolle erlaubt die Koexistenz verschiedener Versionen einer Ressource und ermöglicht deren individuelle und vergleichende Visualisierung, sowie ihre automatische oder unterstützte Zusammenführung und Differenzierung [Vitali, Durand 95]. Systeme zur Versionskontrolle werden z.B. in der Softwareentwicklung, bei computergestützter kooperativer Arbeit (CSCW) oder beim Abgleich verteilter Datenbanken eingesetzt.
Während das WWW heute — mit den technischen Grundlagen HTML 3.2 (vgl. Abschnitt 3.2) und HTTP 1.1 (vgl. Abschnitt 3.3) — ein passives Publikationmedium ist, können Versionierungssysteme es zu einem aktiven Kooperationsmedium machen, in dem Nutzer nicht

[1] Die Nummerierung erfolgt in diesem Fall so, daß es jederzeit möglich ist, weitere Varianten zu den Objekten 1.3 und 1.2.1.1 zu erzeugen, deren Nummern dem Schema 1.2.n.1 folgen würden, wobei n die n-te Variante zu Version 1.3 bezeichnet.

nur lesenden sondern auch schreibenden Zugriff auf Dokumente haben. Grundsätzlich sind zwei Motivationen für Versionskontrolle im WWW erkennbar:

Passive Benutzer des WWW möchten lesenden Zugang zu verschiedenen Versionen von Objekten haben, wie z.B. Dokumentationen von HTML, HTTP oder SQL Standards. Desweiteren sind Hyperlinks von einem Dokument zu seiner initialen, vorherigen, nächsten und neuesten Version bei Recherchen über die Historie einer Ressource nützlich. Von der Implementation eines Versionierungssystems muß ein nur lesender Benutzer keine Kenntnis nehmen, da es für ihn in der Regel nicht von Interesse ist, ob die Daten, die für ihn sichtbar sind, in verschiedenen Dateien gespeichert sind oder nach Anforderung durch ihn vor der Übertragung zuerst berechnet werden müssen.

Entwickler von WWW Inhalten benötigen das Management von Versionen für den Zugang zu Historien von Revisionen oder Varianten einer Ressource, für das parallele asynchrone Bearbeiten von Ressourcen oder als Grundgerüst für den kontrollierten Zugang zu Ressourcen. Editieren z.B. zwei Bearbeiter ein Dokument gleichzeitig, werden die Änderungen desjenigen, der das Dokument als erster sichert von den Änderungen des zweiten überschrieben und gehen verloren (engl. *lost update problem*). Dieses Problem läßt sich durch restriktiven Zugang zu Ressourcen — nur ein Bearbeiter pro Ressource zu einem bestimmten Zeitpunkt — oder durch das (evtl. automatische) Zusammenführen parallel entstandener Varianten einer Ressource lösen.

Bereits Ted Nelson, der im Jahr 1965 mit seinem System XANADU den Begriff Hypertext prägte, erkannte, daß Leser in Hypertextsystemen Verbindungen zu einem Dokument aufbauen, dieses dann überarbeitet wird und bei der Ersetzung des alten Dokumentes, die Verbindungen zu diesem Dokument eventuell ungültig werden. Aus diesem Grund werden in dem System XANADU Dokumente nie gelöscht, auch nicht, wenn neue Versionen eines bestehenden Dokumentes hinzugefügt werden [Nielsen 95].
Im Zusammenhang mit Hypermediadokumenten, wie man sie im WWW findet, wird Versionierung durch Probleme wie z.B. der Integrität von Hyperlinks oder der Konsistenz einer Menge von Dokumenten komplex [Hicks et al. 91]. Aus diesem Grund wird hier die Versionierung von Daten von der Versionierung von Strukturen getrennt. Dieses Kapitel behandelt die Handhabung mehrerer Versionen einer einzelnen Ressource, während in Kapitel 6 die Probleme des Konfigurationsmanagement versionierter Ressourcen betrachtet werden, die sich aus versionierten Hypermediadokumenten im WWW ergeben.
In Abschnitt 5.1 werden grundlegende Versionierungskonzepte dargestellt. Ziele und Anforderungen der Versionierung im WWW werden in Abschnitt 5.2 erläutert. In Abschnitt 5.3 erfolgt die exemplarische Betrachtung verschiedener technischer Realisierungen von Versionierung im WWW.

5.1 Terminologie und Konzepte

Ein *Versionierungssystem* ist ein Softwaremodul, das Beziehungen zwischen verschiedenen Instanzen eines Datenobjektes aufrechterhält, indem es für Benennung, Speicherung und

Retrieval der Instanzen sorgt. Jede Instanz ist eine *Version* des Objektes und steht mit den anderen Versionen in einer möglicherweise komplexen Struktur aus älteren, neueren oder parallel entstandenen Versionen in Beziehung [Vitali, Durand 95].
Die Versionierung von Daten bezieht sich auf den Prozeß des Management der Evolution und Entwicklung eines Datenobjektes. Die wesentlichen Konzepte beschreiben dabei das der Evolution zugrundeliegende Datenmodell, die Unterstützung kooperativer Arbeit an Datenobjekten sowie die verwendeten Mechanismen zur persistenten Speicherung der Datenobjekte.

5.1.1 Datenmodelle

Die in Hypermediasystemen zu versionierenden Daten sind in Knoten enthalten, die in verschiedenen Datenmodellen versioniert werden können. Die Modelle beschreiben, in welchen Beziehungen verschiedene Versionen eines Datenobjektes stehen. Man unterscheidet folgende drei Datenmodelle:

Lineare Sequenz: Versionierung wird im Rahmen einer linearen Sequenz von Revisionen unterstützt, d.h. Varianten können in diesem Modell nicht abgebildet werden.

Baumstruktur: Das Modell der Baumstruktur unterstützt sowohl Revisionen als auch Varianten und ermöglicht das parallele asynchrone Editieren von Objekten einerseits durch *zeitbasierte Versionierung*, die die Unterschiede zwischen mehreren aufeinanderfolgenden Revisionen eines Dokumentes abbildet, und andererseits durch *Autorbasierte Versionierung*, die Modifikationen verschiedener Varianten der selben Version erlaubt.

Gerichteter Azyklischer Graph: Dieses Modell bietet neben dem Management von Revisionen und Varianten das (evtl. automatische) Zusammenführen (engl. *merging*) mehrerer parallel entstandener Versionen von Objekten.

Der Inhalt einer spezifischen Version in allen drei Modellen wird durch die Operationen (z.B. Einfügungen und Löschungen) bestimmt, die auf den Knoten ausgeführt werden, die diese Version mit ihrer Initialversion — der sogenannten Wurzel — verbinden.

5.1.2 Unterstützung kooperativer Arbeit

Die Unterstützung kooperativer Arbeit erfolgt in Versionierungssystemen durch die Verhinderung simultaner Schreibzugriffe auf das gleiche Datenobjekt. Hierzu gibt es zwei unterschiedliche Ansätze:

Sperren von Objekten: Dieser Ansatz ist als *checkin/checkout* Prozess bekannt. Der Änderungsprozess eines Objektes beginnt mit dem Sperren des Objektes durch den *checkout* Prozess für andere Benutzer als den Initiator des *checkout*. Nachdem dieser seine Änderungen ohne Zugriffskonflikte vorgenommen hat, führt der *checkin* Prozess zur Erzeugung einer neuen Revision des Objektes und die Sperrung wird aufgehoben, so daß andere Benutzer das Objekt editieren können.

Zusammenführen von Objekten: Dieser Ansatz erlaubt das gleichzeitige Editieren eines Objektes durch mehrere Bearbeiter. Das führt zu der Erzeugung von Varianten, die in einem *merge* Prozess zu einer neuen, allen zusammengeführten Varianten gemeinsamen Revision vereinigt werden.

Beide Konzepte treten auch in Kombination auf, so daß falls ein weiterer Bearbeiter versucht, eine gesperrte Ressource zu ändern, er eine Warnung erhält, daß die Ressource bereits in Bearbeitung ist, er aber trotzdem Änderungen durchführen darf, die dann zu einer Variante führen.

5.1.3 Speichermechanismen

Versionierungssysteme speichern keine vollständigen Kopien von Revisionen und Varianten von Dateien sondern verwenden Delta-Speicherungstechniken, die nur die Menge der Differenzen aufeinanderfolgender Versionen speichern. Dadurch ist die Menge an benutztem Speicherplatz proportional zu der Menge der an den Dateien vorgenommenen Änderungen [Bolinger, Bronson 95] und die verschiedenen Versionen sind frei von Redundanzen. Man unterscheidet dabei zwei Verfahren [Hicks et al. 91]:

Separate Delta-Dateien: Informationen, die benötigt werden, um eine Revision in eine andere zu transformieren, werden in separaten Dateien gespeichert. Jeweils eine Datei gehört zu einer Revision. RCS — ein System für die Versionskontrolle von Softwareprojekten — verwendete früher einen Delta-Mechanismus, der die jeweils aktuellste Revision einer Datei vollständig speichert, während in den Delta-Dateien die Informationen gespeichert werden, die die Wiederherstellung älterer Revisionen aus neueren Revisionen erlauben [Hicks et al. 91]. Heute speichert RCS alle Versionen in einer Archivdatei aus der die spezifischen Versionen bei Bedarf wiederhergestellt werden [Bolinger, Bronson 95].

Archivdatei: Bei der Verwendung von Archivdateien werden die Informationen für alle Revisionen eines Objektes in einer Datei — dem *Archiv* — gespeichert. Eine bestimmte Revision wird wiederhergestellt, indem die Datei berechnet wird und nur die für die gewünschte Revision zutreffenden Modifikationen (z.B. Löschungen und Einfügungen) vorgenommen werden. Dieses Verfahren wird u.a. von SCCS (vgl. Abschnitt 5.4.1) und VTML (vgl. Abschnitt 5.3.2) verwendet.

Datenmodelle, Unterstützung kooperativer Arbeit sowie Speicherverfahren für das Management der Evolution einzelner Ressourcen bilden die Grundlage für die Versionierung komplexer, aus mehreren Ressourcen bestehender Strukturen (vgl. Kapitel 6).

5.2 Anforderungen an Versionierungssysteme im WWW

In diesem Abschnitt werden Anforderungen an Versionierungssysteme im WWW aufgestellt, nach denen die in Abschnitt 5.3 vorgestellten Versionierungsansätze abschließend be-

wertet werden. Die Anforderungen werden unterteilt in funktionale Anforderungen (vgl. Abschnitt 5.2.1) und technische Anforderungen (vgl. Abschnitt 5.2.2).

5.2.1 Funktionale Anforderungen

Die im folgenden dargestellten Anforderungen beschreiben die notwendige Funktionalität eines Versionierungssystems im WWW. Wie am Anfang des Kapitels bereits erwähnt, haben passive Benutzer und Entwickler unterschiedliche Anforderungen an die Versionierung von Datenbeständen im WWW. Für den passiven Benutzer stehen folgende Anforderungen im Vordergrund:

Unterstützung stabiler, versionierter Hyperlinks: Es sollte für den Benutzer möglich sein, Links zu der initialen, einer spezifischen oder der aktuellen Version einer Ressource zu verfolgen und zu speichern, ohne daß diese Links nach der Entwicklung einer neuen Version ungültig oder falsch werden.

Visualisierung von Versionstopologien: Die Metainformationen (z.B. Autor, Datum der letzten Änderung, etc.) aller existierenden Versionen einer Ressource müssen in einem Standardformat von allen Clients genutzt werden können (z.B. HTML-Text [Reuter et al. 96]), damit sich ein Benutzer über die Entwicklung einer Ressource informieren kann und anhand der Metainformationen eine für ihn interessante Version auswählen kann.

Zugang zu einer auf dem Server definierten Standard-Version: Wird keine spezielle Version eines Objektes vom Client angefordert, liefert der Server eine Standard-Version (z.B. die aktuellste). Dies ist der einfachste Weg die Kompatibilität mit Clients zu wahren, die keine Versionierung unterstützen.

Zugang zu „verwandten" Versionen: „Verwandte" Versionen sind z.B. die erste Version eines Dokumentes (Wurzelversion), die vorausgegangene und die nachfolgende Version eines Dokumentes und die Standard-Version.

Die darüber hinausgehenden Anforderungen eines Entwicklers hinsichtlich der Versionierung von Ressourcen im WWW sind:

Unterstützung unsichtbarer Wartung von Ressourcen: Entwickler von WWW Ressourcen muß es möglich sein, unsichtbar für den Benutzer einer Ressource neue Versionen dieser Ressource zu erzeugen und sie erst für den Benutzer zugreifbar zu machen, wenn die Bearbeitung abgeschlossen ist, bzw. er die neue Version öffentlich freigibt. Das kann z.B. so erfolgen, daß ein Entwickler eine bestimmte Version einer Ressource per HTTP in seinen Browser lädt, sie dann lokal editiert und nach Abschluß der Änderungen wieder an den Server sendet, der eine neue Version erzeugt und diese dann öffentlich zugänglich macht.

Unterstützung paralleler asynchroner Entwicklung: Durch die Speicherung mehrerer Varianten einer Ressource und der späteren Zusammenführung zu einer gemeinsamen Revision wird das parallele asynchrone Arbeiten mehrerer Entwickler an einer Ressource möglich.

Zugangskontrolle zu Ressourcen: In einigen Fällen ist das parallele asynchrone Bearbeiten einer Ressource nicht erwünscht. Der Entwickler benötigt dann die Möglichkeit Ressourcen zu sperren. Dies muß aber kein Editieren der gesperrten Ressourcen nach sich ziehen, da Ressourcen evtl. nur gesperrt werden sollen, um Änderungen an diesen zu verhindern und so die Konsistenz mit einer gerade in Bearbeitung befindlichen Ressource zu bewahren.

Je nach Einsatzzweck des Versionierungssystems muß sowohl die vom lesenden Benutzer als auch die vom Entwickler geforderte Funktionalität via WWW zugänglich sein. Es ist z.B. denkbar, daß die versionierten Ressourcen per WWW nur für den lesenden Benutzer bereitgestellt werden sollen, es aber aufgrund der entstehenden Sicherheitsrisiken keine Möglichkeit gibt, neue Versionen per WWW Zugriff zu erzeugen. Soll dagegen die weite Verbreitung und einfache Bedienung des WWW für CSCW genutzt werden, muß auch die vom Entwickler geforderte Funktionalität per WWW Zugriff zur Verfügung stehen (vgl. z.B. Abschnitt 5.3.3).

5.2.2 Technische Anforderungen

Versionierungssysteme für den Einsatz im Zusammenhang mit WWW Ressourcen finden eine Reihe von vorgegebenen Rahmenbedingungen (z.B. verschiedene Browser, Server, Datenformate, unterschiedliches Anwenderwissen etc.) vor. Um sich diesen Bedingungen anzupassen, sollten Versionierungssysteme folgende technische Anforderungen erfüllen:

Kompatibiltität: Ein Versionierungssystem im WWW ist im Idealfall kompatibel zu vorhandenen Browsern und WWW Servern und erlaubt deren Weiterverwendung unabhängig vom Versionierungssystem. Das bedeutet u.a., daß spezifische Versionen via URL addressierbar sein sollten. Dadurch wird es möglich, von vorhandenen Browsern auf beliebige versionierte Ressourcen zuzugreifen. Würde hingegen die Versionsinformation in einem neu definierten HTTP *request header* (vgl. Abschnitt 3.3) übergeben, müßten Browser und WWW Server erweitert werden, um diesen zu senden bzw. zu verarbeiten.

Flexibiltiät: Ein Versionierungssystem sollte in der Lage sein, beliebige Typen von vorhandenen Ressourcen zu versionieren und z.B. nicht auf HTML Dateien beschränkt sein, da der multimediale Anteil im WWW vorhandener Ressourcen (Bilder, Audio, Video) ständig zunimmt.

Effizienz: Es ist wünschenswert, daß die zwischen Browser und Server übertragene Datenmenge zur Abfrage bzw. Übertragung von Revisionen möglichst klein gehalten wird.

So sollte es möglich sein, durch Angabe einer Revisionsnummer eine spzifische Version vom Server anzufordern und nicht eine komplette Archivdatei vom Server laden zu müssen, aus der der versionierungsfähige Browser die gewünschte Version extrahiert. Das Versionierungssystem sollte für effiziente Bennenung, Speicherung und Retrieval spezifischer Instanzen von Ressourcen sorgen.

Transparenz und Benutzbarkeit: Ein nur lesend zugreifender Benutzer muß von der Implementation des Versionierungssystems keine Kenntnis erlangen. Für ihn ist die Möglichkeit der direkten und einfachen Auswahl verschiedener Versionen von Ressourcen relevant.

Funktionale und technische Anforderungen bilden die Grundlage der Bewertung der im folgenden betrachteten Ansätze für Versionierungssysteme im WWW.

5.3 Technologien

In diesem Abschnitt werden verschiedene Technologien für die Versionierung von Ressourcen im WWW exemplarisch vorgestellt. Zuerst wird in Abschnitt 5.3.1 die Möglichkeit zur Versionierung ohne die Unterstützung spezieller Techniken und Werkzeuge dargestellt. Die weiteren betrachteten Technologien setzen auf verschiedenen Ebenen des WWW an (vgl. Abbildung 5.2).

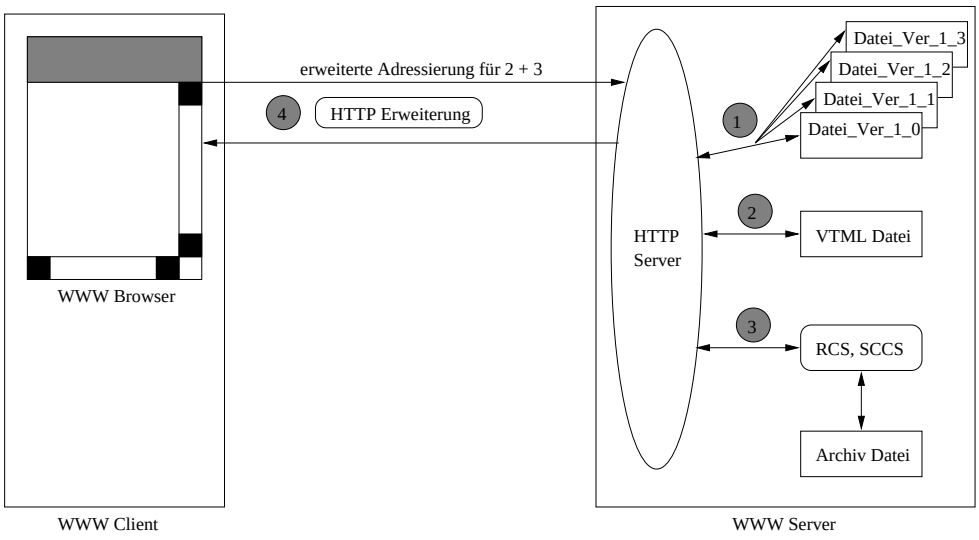

Abbildung 5.2: Versionierungsansätze im WWW

VTML ist eine Dokumentenbeschreibungssprache für die Speicherung von Versionsinformationen (2) und kann in HTML und SGML Dokumenten verwendet werden (vgl. Ab-

schnitt 5.3.2). Der dritte Ansatz (3) setzt auf die Anbindung eines bereits vorhandenen Versionierungssystems über die CGI Schnittstelle eines WWW Servers (vgl. Abschnitt 5.3.3). Eine weitere Basis (4) für die Versionierung von WWW Inhalten bietet die Erweiterung des HTT Protokolls (vgl. Abschnitt 5.3.4).

5.3.1 Versionierung durch redundante Datenhaltung

Die Aufgabe eines Versionierungssystems ist die Bereitstellung eines Zugangs zu sämtlichen Revisionen und Varianten einer Ressource. Im dateibasierten WWW Zusammenhang bedeutet dies, daß alle Änderungen an Ressourcen in eigenen Dateien gespeichert und diese dann eindeutig benannt werden müssen, um die eindeutige Referenzierung einer bestimmten Version zu ermöglichen. Der Zugriff durch den Benutzer erfolgt dann mittels Angabe eines für jede Version unterschiedlichen Dateinamens (vgl. Abbildung 5.2, (1)). Diese Art der Versionierung erfolgt also unabhängig von Browser, WWW Servern und dem HTT Protokoll.

Anwendungsbeispiel

Ein versioniertes Dokument unterscheidet sich in diesem Fall intern nicht von einem nicht versionierten Dokument, da die Metainformationen über den Versionierungsgraphen außerhalb des Dokumentes gesammelt werden müssen. Die verschiedenen Versionen werden nur über ihren Dateinamen (z.B. *datei_Ver_1_0.html*, bzw. *datei_Ver_1_1.html*) einer Versionshistorie zugeordnet. Der Anwender ruft diese Dateien per URL ab. Die Realisierung einer abfragbaren Versionstopologie erfolgt durch die Generierung einer speziellen Datei (z.B. *datei_Ver_Hist.html*) sobald eine neue Version einer Ressource erstellt wird. Diese Versionshistorie enthält Metainformationen über die verfügbaren Versionen einer Ressource sowie Hyperlinks zu diesen. Setzt man CGI Skripte ein, ist es auch möglich, die Versionshistorien zu ihrem Abfragezeitpunkt neu zu berechnen.

Bewertung

Anhand der in Abschnitt 5.2 aufgestellten Anforderungen an Versionierung im WWW wird nun die Versionierung durch redundante Datenhaltung bewertet.

Funktionale Anforderungen: Die Anforderungen des passiven Benutzers sind durch schematisch benannte versionierte Ressourcen, bzw. Hyperlinks erfüllbar. Die Anforderungen des Entwicklers erfordern die Programmierung von Skripten, die die nötige explizite Benennung neuer und die evtl. Umbenennung älterer Versionen, die Verwaltung von Metainformation (Datum der Änderung, Bearbeiter, Kommentare, ...) und die Verhinderung von parallelen Schreibzugriffen auf eine Ressource übernehmen. Desweiteren ist ein Skript nötig, das Informationen über die Versionstopologie einer Ressource erzeugt und z.B. im HTML Format bereitstellt.

Technische Anforderungen: Volle Kompatibilität mit existierenden Browsern und Servern ist gegeben, da diese nichts von der Existenz eines solchen Versionierungssystems erfahren müssen. Der große Vorteil dieses Ansatzes liegt in seiner Flexibilität

gegenüber verschiedenen Datenformaten (Texte, Bilder, Audio, Video). Da die Metainformationen getrennt von den versionierten Ressourcen verwaltet werden, spielt deren Datenformat keine Rolle. Die Effizienz ist in Bezug auf die Zugriffsgeschwindigkeit auf Daten optimal, da keine Berechnungen zur Wiederherstellung einer spezifischen Version durchgeführt werden müssen. Die Speicherung ist nicht effizient, da nicht nur die Unterschiede von Version zu Version gespeichert werden, sondern ganze Kopien von Dateien. Die Speicherung einer WWW Struktur in mehreren Versionen erfordert so einen hohen Bedarf an Speicherplatz und führt zu Redundanzen. Transparenz und einfache Benutzbarkeit sind gegeben, da der passive Benutzer nur Dateien mit unterschiedlichen Namen per URL abruft.

Zusammenfassend kann festgestellt werden, daß, wenn die nötigen Werkzeuge bereitstehen, die den Entwickler neuer Versionen von Ressourcen von den Verwaltungsaufgaben (z.B. Bennung, Verwaltung der Metainformationen, Sperrung) entlasten, dieser Ansatz volle Flexibilität in Bezug auf versionierte Datenformate und verwendete Browser und WWW Server bietet. Der Nachteil des großen Speicherplatzverbrauches für die redundante Datenspeicherung ist insbesondere im Zusammenhang mit Bildern und Grafiken sowie Multimediadaten nicht zu unterschätzen.

5.3.2 Versionierung durch HTML Erweiterungen

HTML Erweiterungen für die Integration spezieller Information in HTML Dokumente sind insbesondere im Datenbankbereich (vgl. Abschnitt 8.1.2) sowie bei der Integration von Skripten (vgl. Abschnitt 7.3.3) verbreitet. Es ist möglich, mit Hilfe von SGML Kommentaren beliebige Informationen in HTML Dokumente einzubinden und diese dann durch ein Programm auf dem Server oder dem Client auswerten zu lassen. Prinzipiell hat dieses Vorgehen den Nachteil, daß der Ansatz von einem HTML oder SGML Dokument abhängig ist und z.B. nicht auf Bilddateien übertragen werden kann. Desweiteren müssen Client und/oder Server erweitert werden, damit sie die zusätzlichen Informationen auswerten können.
Ein Beispiel für die Einbindung von Versionierungsinformationen in HTML oder SGML Dokumente ist die *Versioned Text Markup Language* (VTML).

Versioned Text Markup Language

VTML (*Versioned Text Markup Language*) ist eine Dokumentenbeschreibungssprache für die Speicherung von Versionsinformationen [Vitali, Durand 95]. Das Hauptziel von VTML ist es, die asynchrone Zusammenarbeit mehrerer Autoren bei der Erzeugung und Editierung von Textdokumenten zu ermöglichen.
VTML unterstützt das Datenmodell des azyklischen Graphen. VTML bietet keine Sperrkonzepte jedoch das Zusammenführen mehrerer Varianten zu einer gemeinsamen Folgerevision (*merge*) und unterstützt so das asynchrone, parallele Arbeiten an Ressourcen. Mehrere Versionen einer Ressource werden in einer Archivdatei gespeichert und bei Bedarf aus dieser wiederhergestellt.

Beschreibung von VTML

VTML Daten werden in SGML Kommentaren in einem HTML oder SGML Dokument untergebracht. So können sie von nicht VTML-fähigen Werkzeugen ignoriert werden. Das VTML Schlüsselwort *INS* kennzeichnet Einfügungen und das Schlüsselwort *DEL* Löschungen. Vor der Visualisierung eines Dokumentes muß ein VTML Parser dieses in Abhängigkeit von der zu visualisierenden Version filtern.
VTML Dokumente werden ähnlich HTML Dokumenten gekennzeichnet durch

<!--{VTML *Attribute*}--> VTML Dokument <!--{/VTML}-->

Die Attribute beschreiben allgemeine Charakteristika des Dokumentes und müssen zumindest den Namen des Dokumentes und die aktuelle Versionsnummer enthalten, die automatisch angezeigt wird, wenn der Benutzer keine Versionsnummer in seiner Anfrage spezifiziert. Attributlisten bezeichnen die Attribute, die für mehrere Operationen in einem Dokument Gültigkeit besitzen. Hierbei sind zumindest der Name des Autors, die Version und das Datum der Modifikation anzugeben. Während der Editierung sollte die Version als CURRENT, und das Datum als NOW angegeben werden. Die tatsächlichen Werte werden dann während des Festsetzens einer neuen Version, dem sogenannten "check in" automatisch (z.B. von einem Serverskript) eingesetzt. Im Beispiel werden zwei Attributlisten benutzt, um nicht bei jeder INS bzw. DEL Operation die ansonsten nötigen Angaben zu Autor, Versionsnummer und Datum wiederholen zu müssen.

<!--{ATTR ID=1 vers=1 author=„fabio" date=„Jul 16, 1996"}-->
<!--{ATTR ID=2 vers=CURRENT author=„david" date=NOW}-->
<!--{INS ATTR=1}--> Dies ist <!--{DEL ATTR=2}--> dein
<!--{/DEL}--> <!--{INS ATTR=2}--> mein <!--{/INS}--> Dokument.
<!--{/INS}-->

INS und DEL Operationen können beliebig geschachtelt werden. Daten, die mit DEL gekennzeichnet sind, werden während der Anzeige ignoriert. Ein VTML Dokument enthält also alle Informationen, um den Inhalt jeder Version zu berechnen.
Neben den Attributlisten und der Kennzeichnung der Einfüge- und Löschoperationen lassen sich auch mehrere parallel entstandene Varianten einer Version zu einer gemeinsamen neuen Version zusammenfassen.
Desweiteren ist es möglich, Operationen, die ein anderes Dokument betreffen, zu speichern. Dies ist sinnvoll, wenn man keinen direkten Zugang zu dem Dokument hat, daß man editieren möchte. Bei der jetzigen Implementation von VTML müssen die extern gespeicherten Modifikationen vor dem anzuzeigenden Dokument den Parser durchlaufen, dessen Algorithmus hier vereinfacht dargestellt wird (aus: [Vitali, Durand 95]):

doPrint(content)
*char *content*

```
/* SIMPLIFIED */
{
  if ((type == INS) || (type == MERGE))
     if (in_chain (version, verstbd))
          printf("%s", content) ;
  else
     if ((!in_chain (version, verstbd)) &&
        (in_chain (lastins, verstbd)))
          printf("%s", content) ;
}
```

Während der grammatischen Analyse durch den Parser wird ein Stapel von Versionen erzeugt. Immer wenn ein neuer VTML Bezeichner wie z.B. INS oder DEL gefunden wird, wird dieser auf den Stapel gelegt und der Textinhalt untersucht. Handelt es sich bei dem Typ des Bezeichners um INS oder MERGE, wird geprüft, ob die Version des Bezeichners (*version*) ein Vorfahre der anzuzeigenden Version (*verstbd*) ist. Ist dies der Fall, wird der Inhalt angezeigt. Handelt es sich um ein DEL, wird geprüft, ob die Version des Bezeichners kein Vorfahre ist (und daher in der anzuzeigenden Version zu löschen ist) und ob die Version des Textes in die das DEL eingefügt wurde ein Vorfahre des anzuzeigenden Textes ist. Ist dies der Fall, wird der Inhalt angezeigt. Die Vereinfachung des obigen Verfahrens besteht in der Ausklammerung der rekursiven Aufrufe, die nötig sind um verschachtelte Operationen, die in *content* vorhanden sein könnten zu behandeln.

Anwendungsbeispiel

Die Autoren schlagen zwei verschiedene Anwendungsmöglichkeiten vor. Zum einen könnte ein WWW Browser um Bearbeitungsfähigkeiten erweitert werden, der dann auf dem Clientrechner neue Versionen von VTML Dokumenten erzeugen würde und diese zum Server sendet und dort ablegt. Der Client würde also eine komplette VTML Datei laden, sie lokal editieren und wieder zum Server schicken.[2]

Die andere Anwendungsmöglichkeit von VTML ist ein Serverskript. Es könnte ein neuer Inhaltstyp (*content-type*) von HTTP Nachrichten für VTML definiert werden, der dann mittels HTTP zwischen Browser und Server verhandelt wird. Wenn ein nicht VTML-fähiger Browser eine VTML Datei unter dem URL „http://my.host/dir/file.vtml“ anfordert, berechnet der Server die aktuelle Version des Dokumentes und schickt sie als HTML Datei dem Client. Fordert der Client eine spezifische Version unter dem URL „http://my.host/dir/file.vtml/1.3“ an, generiert das Serverskript die entsprechende Version und liefert sie an den Client aus.

[2]Die Autoren äußern sich in ihrem Artikel nicht zu den bei gleichzeitiger Editierung eines Dokumentes entstehenden Konflikten, wenn diese neuen Versionen auf dem Server abgelegt werden. Dort müßte zumindest ein Serverskript aktiv sein, das Schreibkonflikte behandelt oder das das gleichzeitige Editieren von VTML Dateien durch mehrere Bearbeiter verhindert.

Bewertung

Anhand der Anforderungen an Versionierung im WWW aus Abschnitt 5.2 wird nun die Eignung von HTML Erweiterungen am Beispiel von VTML als Ansatz für Versionierung im WWW bewertet.

Funktionale Anforderungen: Die funktionalen Anforderungen des lesenden Benutzers lassen sich z.B. durch eine Implementation von VTML als Serverskript erfüllen. Die Addressierung würde in der oben beschriebenen Art und Weise erfolgen und entweder die aktuelle oder eine spezifische Version liefern. Das Skript müßte desweiteren in die Lage versetzt werden, aus den in einer VTML Datei enthaltenen Metainformationen eine Versionshistorie in Form einer HTML Datei zu erzeugen, die die einzelnen Versionen einer Ressource mit ihren Metadaten zugreifbar macht. Sollen Anwender per WWW neue Versionen eines VTML Dokumentes erzeugen können, müßte entweder der Client wie oben beschrieben erweitert werden oder das Serverskript muß in die Lage versetzt werden, eine vom Client zum Server übertragene neue Version in ein vorhandenes VTML Dokument einzufügen und dabei evtl. auftretende Konflikte zu behandeln.

Technische Anforderungen: Beschränkt sich der Benutzer im WWW auf den lesenden Zugriff, können um Versionierungsinformationen erweiterte HTML Dateien von jedem Browser gelesen werden. Eine Servererweiterung ist in jedem Fall notwendig, um das Wiederherstellen spezifischer Versionen von Dateien aus den Archivdateien zu ermöglichen. Generell sind Systeme wie VTML auf HTML/SGML Dokumente beschränkt und lassen sich nicht auf Bilder oder Datenströme wie Audio und Video übertragen. Transparenz und Benutzbarkeit ist bei oben beschriebener Adressierung versionierter Daten für den lesenden Benutzer gegeben. Für den Entwickler ist die Unterstützung durch Werkzeuge unabdingbar, da es ihm nicht zuzumuten ist, per Hand VTML Schlüsselworte in ein Dokument einzufügen.

Ansätze wie VTML haben gegenüber der Versionierung durch redundante Datenhaltung den Vorteil, daß sie die Versionsinformation in dem versionierten Dokument speichern. So lassen sich mehrere Versionen in einem Dokument ablegen, Speicherplatz wird effizient genutzt und es treten keine Redundanzen auf. Die größte Einschränkung des Ansatzes ist die Beschränkung auf HTML/SGML Dateien. Die vom Speicherplatzbedarf wesentlich anspruchsvolleren Datenformate für Bilder, Audio und Video lassen sich mit diesem Ansatz nicht versionieren.

5.3.3 Versionierung durch Anbindung bestehender Systeme

Die Anbindung bereits bestehender Versionierungssysteme wie z.B. RCS oder SCCS an das WWW erfolgt in der Regel über die CGI Schnittstelle des WWW Servers oder eine Erweiterung der Serversoftware. Funktionalität und Flexibilität hängen dann prinzipiell nur von dieser Versionierungssoftware ab. In der Praxis müssen WWW Schnittstellen für die benötigten Funktionalitäten geschaffen werden.

Das WWRC (s.u.) ist ein Versionierungssystem für die Unterstützung kooperativer Arbeit mehrerer Entwickler an einem Softwareprojekt und wird durch HTML Formulare und CGI Skripte realisiert, die ein vom WWW unabhängiges Versionierungssystem nutzen. Bei der Servererweiterung in Abschnitt 5.4 wird dagegen nur die vom lesenden Benutzer benötigte Funktionalität unterstützt und der Zugriff auf versionierte Ressourcen erfolgt über erweiterte URLs.

WWRC

Das WWRC (World Wide Revision Control) System dient der Versionierung von Quellcodedateien bei der Softwareentwicklung [Reuter et al. 96]. Es ermöglicht die Benutzung des Versionierungssystems RCE — eine Weiterentwicklung von RCS [Bolinger, Bronson 95] — über beliebige WWW Browser ohne daß dem Entwickler direkter Zugang zu den verwalteten Dateien gegeben werden muß. Dies ist insbesondere in Unternehmen bei kurzzeitig eingestellten freien Mitarbeitern oder Heimarbeitern von Vorteil, da diese keine Benutzerrechte auf dem Rechner des Versionierungssystems bekommen müssen und dadurch das Sicherheitsrisiko des von außen wählbaren Zugangs zum Dateisystem vermieden wird.
Es handelt sich hierbei nicht um ein allgemeines Versionierungssystem für das WWW sondern um ein System zur verteilten Bearbeitung von Quellcodedateien. So ist es z.B. nicht möglich, ohne eine Hilfsanwendung spezifische Versionen anhand eines URL anzufordern. Bei diesem Ansatz werden jedoch die Vorteile und Probleme einer Clienterweiterung für das verteilte asynchrone Bearbeiten von Dokumenten deutlich.

Aufbau des WWRC

Das WWRC System setzt sich aus folgenden Komponenten zusammen (vgl. Abbildung 5.3):

RCE: Die Revision Control Engine (RCE) bietet eine Anwendungsprogrammierschnittstelle (API), die die Funktionalität der RCE von außen zugänglich macht.

CGI Anwendung: Die CGI Anwendung übernimmt die Kommunikation zwischen WWW Server und der RCE.

WWW Client und Server: Dies können beliebige WWW Browser (z.B. MOSAIC, Navigator, Internet Explorer, etc.) und WWW Server (z.B. Apache, Jigsaw, etc.) sein.

Hilfsanwendung[3] beim Client : Die Hilfsanwendung auf der Seite des Client ermöglicht die Übertragung großer Dateien sowohl vom als auch zum Server, sowie deren korrekte Visualisierung auf dem Client Rechner.

Funktionsweise

Der Benutzer macht sämtliche Eingaben für das System über die Schnittstelle seines Browsers in dem er Formulare ausfüllt. Mit der Hilfe von Formularen ist es möglich, notwendige

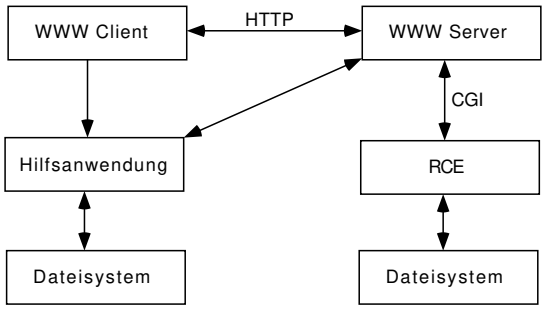

Abbildung 5.3: Komponenten des WWRC

Kontextinformationen über eine erweiterte URL an das CGI Programm weiterzugeben und so eine verbindungsorientierte Kommunikation zwischen Client und Server zu simulieren (vgl. Abschnitt 3.4).
Die Arbeitssitzung eines Benutzers des WWRC Systems besteht aus folgenden Schritten:

Anmeldung: Bevor ein Benutzer mit dem WWRC System arbeiten kann, muß er sich bei diesem mit Benutzernamen und Passwort anmelden.

Archivdatei auswählen: Nach der erfolgreichen Anmeldung hat der Anwender u.a. die Möglichkeit eine Archivdatei auszuwählen, in der die verschiedenen Versionen einer Ressource enthalten sind.Der Versionierungsgraph einer Archivdatei wird bei Bedarf als formatierter Text visualisiert, wobei sich der besseren Übersicht wegen Subgraphen aus- und wieder einblenden lassen.

Spezifische Version auswählen: Wird eine Revision oder Variante ausgewählt, erhält der Anwender mehr Informationen darüber wie z.B. Erstellungsdatum, Autor, den aktuellen Zustand und Kommentare.

Datenübertragung zum Client: Ist eine Version ausgewählt, ist es möglich diese zur Bearbeitung aus der Versionskontrolle herauszunehmen (engl. *checkout*) und zum Rechner des Anwenders zu übertragen. Im ersten Schritt einer Anfrage für ein *checkout*, führt der Server ein lokales *checkout* durch, indem er die Datei in einem für jeden Benutzer spezifischen Verzeichnis ablegt. Der Antwort des Servers setzt sich u.a. aus dem Dateinamen, der Verzeichnisposition, dem speziellen Typ "rceco" und der Datei selbst zusammen. Der spezielle Typ der Datei führt dazu, daß der Browser des Client die o.g. Hilfsanwendung ausführt. Die Hilfsanwendung liest den Antwortkopf des Servers, bestimmt so Name und Verzeichnis der neu zu erzeugenden Datei und legt den Inhalt der Datei dort ab. Danach wird eine Empfangsbestätigung an den Server gesendet, woraufhin dieser die lokale Kopie der Datei zur Löschung freigibt. War die Datenübertragung nicht erfolgreich, ist es für den Benutzer noch immer möglich, die Datei noch

einmal anzufordern, ohne daß evtl. zwischenzeitlich entstandene Änderungen die zu übertragende Datei beeinflussen.

Bearbeitung: Die Bearbeitung der Datei erfolgt lokal auf dem Rechner des Client.

Datenübertragung zum Server: Um eine Datei wieder unter die Versionskontrolle des WWRC zu stellen (engl. *checkin*), schickt der Server eine Datei an den Client, in welcher die von der Hilfsanwendung zu übertragende Datei bestimmt wird. Die Hilfsanwendung wertet diese Angaben (Dateiname, Verzeichnisangabe) aus, liest die entsprechende Datei ein und baut eine direkte Verbindung zum Server auf, der die Datei in einem speziellen Verzeichnis ablegt. Aus diesem Verzeichnis des Servers findet dann das *checkin* in die zugehörige Archivdatei statt.

Abbildung 5.4 visualisiert die während des *checkout* ablaufenden Prozesse.

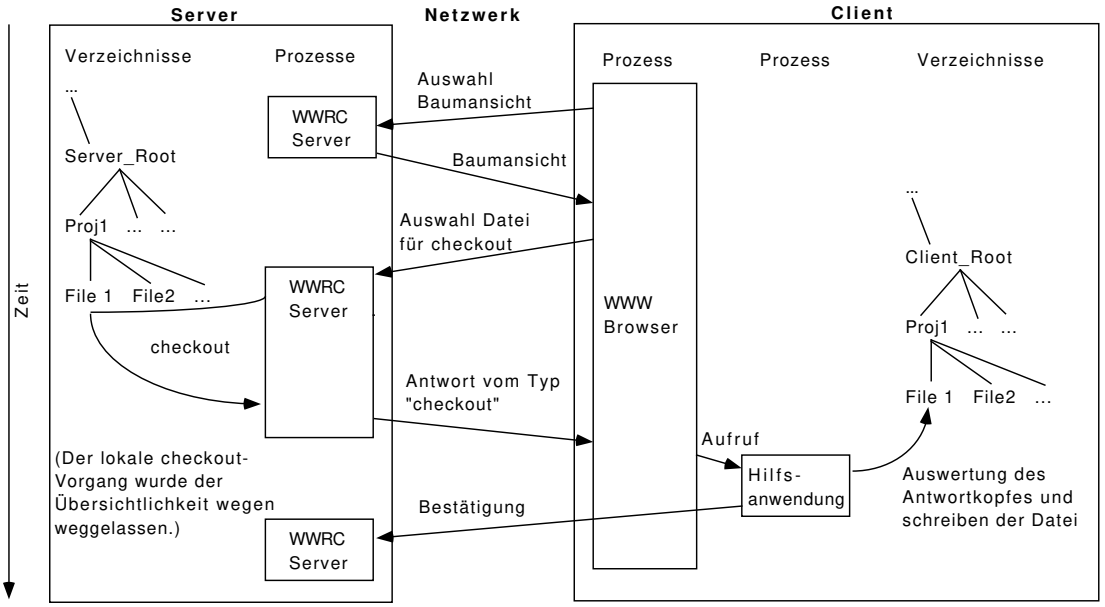

Abbildung 5.4: Prozesse während des *checkout*

Bewertung

Das WWRC ist auf die Versionierung von Programmcode zugeschnitten und erhebt nicht den Anspruch ein allgemeines Versionierungssystem für das WWW zu sein, daher kann die Bewertung nicht nach den in Abschnitt 5.2 erarbeiteten Kriterien erfolgen. Stattdessen soll hier allgemein die Nutzung von Clienterweiterungen zur Versionierung von Ressourcen am Beispiel des WWRC bewertet werden.

Die Clienterweiterung im WWRC System ist lediglich für den Entwickler neuer Versionen notwendig. Geht es nur darum, versionierte Daten abzufragen und zu visualisieren, reicht das Mittel der erweiterten Adressierung versionierter Ressourcen wie in Abschnitt 5.3.2 vorgestellt aus.
Eine plattformabhängige Clienterweiterung schränkt ihre Einsatzfähigkeit im WWW ein. Unabhängiger wäre die Realisierung als Java Applet. Hier ergibt sich jedoch das Problem der Sicherheitseinschränkungen von Applets, die nicht auf die Festplatte eines Clientrechners zugreifen dürfen. Ein Plug-In als Java Anwendung setzt das Vorhandensein eines Java Interpreters voraus, entlastet aber zumindest von der Notwendigkeit verschiedenen Code für die unterschiedlichen Plattformen ausliefern zu müssen.
Vorteile bietet das WWRC bei der Unterstützung von Versionskontrolle und kooperativer Arbeit dadurch, daß es verteilte Versionskontrolle ermöglicht, ohne daß die Entwickler direkten Rechnerzugang zu den Archivdateien haben müssen. Ein weiterer Vorteil ist die Nutzung des WWW als Basis für computerunterstützte kooperative Arbeit (CSCW), da der Zugang zum System so nahezu von jedem Rechner aus möglich ist.

5.3.4 Erweiterungen von HTTP

In diesem Abschnitt werden zwei Vorschläge für Erweiterungen des HTT Protokolls vorgestellt und abschließend kritisch betrachtet. Während der Vorschlag von [Whitehead 96] ausschließlich auf die Versionierungsunterstützung für das WWW zielt, beschreiben [Cunningham, Faizi 96] in ihrem Vorschlag ein Protokoll für das verteilte Verfassen und Versionieren von Dokumenten, von dem in diesem Zusammenhang aber nur die die Versionierung betreffenden Abschnitte erläutert werden.

Protokoll für das verteilte Verfassen und Versionieren von Dokumenten

Die HTTP Erweiterung nach [Cunningham, Faizi 96] setzt sich aus der Beschreibung der funktionalen Definition, der Syntax der HTTP Anfrage, sowie der Reaktion des Servers auf die Anfrage für u.a. folgende neue HTTP Methoden zusammen:

CHECKOUT fordert eine Ressource vom Server an, überträgt sie zum Client und kann die Datei bei Bedarf für andere Benutzer sperren. Als Attribute müssen angegeben werden, ob die Datei gesperrt werden soll und welche Version der Ressource von der Methode betroffen ist.

CHECKIN überträgt eine Ressource zum Server. Attribute sind: die Version der zu übertragenden Ressource, der Status (aufheben oder bestehen lassen) der Sperrung der Ressource sowie Kommentare des Bearbeiters zu dieser Version.

UNLOCK enfernt eine Sperre einer Ressource. Attribut ist die spezifische Version der Ressource.

LOCK sperrt eine Ressource für andere Benutzer. Attribut ist die spezifische Version der Ressource.

DEFAULT definiert eine spezifische Version einer Ressource als die Standardversion der Ressource, die z.B. als Antwort auf eine GET Anfrage eines Client übertragen wird. Attribut ist die spezifische Version der Ressource.

REVLOG fordert Metadaten (z.B. Versionshistorie, Aufzeichnungen über CHECKOUT und CHECKIN Operationen, etc.) über Versionierungsinformationen einer Ressource an. Ein Attribut gibt an, ob Metadaten zu einer spezifischen Version, zu allen Versionen oder nur zu einem bestimmten Zweig einer Versionshistorie angefordert werden.

LABEL weist einer Version einer Ressource einen Textbezeichner zu. Attribute sind hier die spezifische Version sowie der Textbezeichner.

Das Attribut *Content-Version* referenziert die von den Methoden eventuell zu spezifizierende Version der Ressource. Die andere Möglichkeit eine Version in einer Anfrage zu spezifizieren ist das Attribut *Content-Label*. Dieses erhält als Wert einen zuvor mit der Methode LABEL zugewiesenen Textbezeichner. Die Methode LABEL ermöglicht dem Entwickler das Erzeugen von Kontexten (vgl. Kapitel 6) indem spezifischen Versionen mehrerer Ressourcen derselbe Bezeichner zugewiesen wird. Beim späteren Retrieval der Ressourcen — z.B. durch GET oder HEAD (vgl. Abschnitt 3.3) — gibt der Client anstatt einer spezifischen Version den Kontext in Form des Attributes *Content-Label* an und der Server ermittelt anhand dessen die spezifische Version der auszuliefernden Ressource.

WWW Versionierungsunterstützung

Während in der HTTP Erweiterung nach [Cunningham, Faizi 96] die Spezifizierung einer Version über die Attribute der neuen HTTP Methoden erfolgt, wird bei dem Vorschlag von [Whitehead 96] ein URL versioniert, indem ein Parameter an den URL angehängt wird. So würde z.B. die Version 3.0 einer Ressource *.../foo.html* als *.../foo.html;v=3.0* referenziert werden.
Mit der Methode FLAG können verschiedene Kennungen (Flags) für Ressourcen gesetzt, gelöscht und abgefragt werden. Es gibt EDIT und LOCK Flags. Die FLAG Methode führt als Gegenstand der Anfrage (engl. *request entity body*, vgl. Abschnitt 3.3.1) immer eine Liste aller für diese Ressource aktiven Kennungen mit sich. Die Liste enthält Paare von Benutzerkennungen und die für den jeweiligen Benutzer gesetzten EDIT bzw. LOCK Flags. Die EDIT Kennung hat die Funktion anzuzeigen, daß eine Ressource gerade von dem genannten Benutzer bearbeitet wird und ermöglicht dadurch einen Versionierungsstil, der das gleichzeitige Editieren von Dokumenten durch mehrere Benutzer im Sinne von CSCW erlaubt. Das erfordert jedoch ein nachträgliches Zusammenführen (engl. *merging*) von parallel entstandenen Version. Die LOCK Kennung wird von der unten vorgestellten HTTP Methode LOCK gesetzt, verbietet gleichzeitiges Editieren einer Ressource durch mehrere Benutzer und markiert diese als gesperrt. Die weiteren Methoden sind folgende:

LOCK sperrt den Zugriff auf einen URL mittels der HTTP PUT Methode (vgl. Abschnitt 3.3.2), für alle Benutzer außer den Inhaber der die Sperre beantragt hat. Ist

in dem URL eine spezifische Version angegeben, gilt die Sperre nur für diese Version, ansonsten werden alle Versionen der Ressource gesperrt.

UNLOCK hebt eine bestehende Sperre auf und löscht das LOCK Flag des betreffenden Ressource URL.

USE entspricht der o.g. Methode DEFAULT und definiert eine spezifische Version einer Ressource als die Standardversion der Ressource, die z.B. immer dann als Antwort auf eine GET Anfrage übertragen werden soll, wenn die Version der Ressource nicht explizit spezifiziert ist.

Die Versionierung aus mehreren Elementen bestehender Strukturen (vgl. Kapitel 6) kann durch die Verwendung von Konfigurationsdateien erreicht werden. Eine Konfigurationsdatei enthält eine Liste von URL Paaren bei denen jeweils ein URL ohne Versionsinformation einem URL mit Versionsinformation zugeordnet ist. So könnte z.B. der URL *http://www.foo.bar/entry.html* dem versionierten URL *http://www.foo.bar/entry.html;v=3.0* zugeordnet sein. Die Konfigurationsdatei wird in den Browser des Client geladen und wirkt fortan als Filter, der nicht versionierte URLs automatisch durch versionierte ersetzt.

Bewertung

Im folgenden wird der Ansatz der Versionierung durch HTTP Erweiterungen im Hinblick auf die in Abschnitt 5.2 aufgestellten Anforderungen kritisch betrachtet.

Funktionale Anforderungen: Die Funktionalen Anforderungen sind durch die Definition neuer HTTP Methoden und Attribute erfüllbar. Die Implementation erfordert jedoch die Entwicklung neuer Browser die diese Methoden senden können und neue Server die die entsprechenden Funktionen bereitstellen.

Technische Anforderungen: Die Kompatibilität mit bestehenden Browsern und Servern ist nicht gegeben. Flexibilität und Effizienz liegen in der Hand des Herstellers des Serversoftware. Der Ansatz gibt hier keinerlei Einschränkungen vor. Transparenz und einfache Benutzbarkeit hängen von der Benutzerschnittstelle der neu zu entwickelnden Browser ab.

Beide Ansätze bieten die Versionierung aus mehreren Elementen bestehender Strukturen — sogenannter *Kontexte* (vgl. Kapitel 6). Im ersten Ansatz ist die Verantwortung dafür zwischen Client und Server aufgeteilt. Der Client gibt den gewünschten Kontext in seiner Anfrage an und der Server versucht anhand dessen die entsprechende Version der Ressource zu bestimmen. Im zweiten Fall entstehen Kontexte durch Konfigurationsdateien, die Hyperlinks dem Kontext entsprechend versionieren. Der Server kennt also nur versionierte Ressourcen jedoch keine Kontexte (da ja die Konfigurationsdateien selbst wieder Ressourcen sind). Für diese Lösung ist eine Erweiterung der Browser notwendig, die die Konfigurationsdateien lesen und verwalten müssen.

HTTP Erweiterungen ermöglichen die flexibelste und weitgehendste Unterstützung von Versionierungskonzepten und erfordern andererseits den größten Aufwand zu ihrer Realisierung, da neben dem Protokoll selbst sowohl Client als auch Server erweitert werden müssen.

5.4 Erweiterung eines WWW Servers

Im folgenden wird die vom Autor prototypisch implementierte Anbindung des *Source Code Control Systems* (SCCS) an den WWW Server Jigsaw des WWW Konsortiums vorgestellt. Das Ziel der Anbindung war die praktische Umsetzung der Anforderungen des passiven Nutzers versionierter Information. Es sollten dabei vorhandene Dienste exemplarisch an einen WWW Server angebunden werden.
Das Umfeld der Servererweiterung und die beteiligten Komponenten sind in Abbildung 5.5 dargestellt. Der Client fordert mittels HTTP eine Ressource vom WWW Server Jigsaw an. Dieser bestimmt anhand der URI der Anfrage, ob es sich hierbei um eine versionierte Ressource handelt. Ist das der Fall, werden entsprechende SCCS Befehle ausgeführt, die die entsprechende Version aus dem Dateisystem extrahieren. Diese wird dann per HTTP an den Client übertragen. Handelt es sich nicht um eine versionierte Ressource, kommuniziert der Server direkt mit dem Dateisystem.

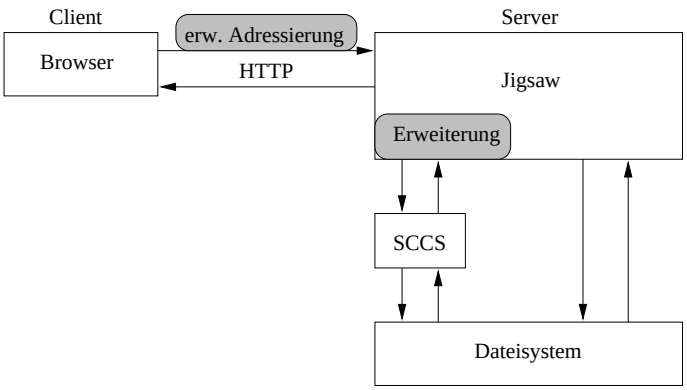

Abbildung 5.5: Bereitstellung versionierter Textdaten für das WWW

Nach einer kurzen Einführung in SCCS (vgl. Abschnitt 5.4.1), wird Architektur und Funktionsweise des in Java implementierten objektorientierten WWW Servers Jigsaw erläutert (vgl. Abschnitt 5.4.2). Die Verbindung beider Systeme wird in Abschnitt 5.4.3 beschrieben.

5.4.1 SCCS

Das Source Code Control System (SCCS) ist ein Versionierungssystem, daß das Datenmodell des azyklischen Graphen verwendet und sowohl Revisionen, Varianten als auch das Zusammenführen mehrerer Versionen unterstützt (vgl. Abschnitt 5.1). Die verschiedenen

Versionen einer Ressource werden in einer Archivdatei gespeichert und bei Bedarf wiederhergestellt. Wird z.B. eine neue Ressource *foo.html* unter die Kontrolle von SCCS gestellt, legt SCCS im Verzeichnis der Ressource ein Unterverzeichnis *SCCS* an — sofern dies noch nicht existiert — und erzeugt in diesem Unterverzeichnis eine *Archivdatei* mit dem Prefix *s.* namens *s.foo.html*. In dieser Archivdatei werden alle Versionen der Ressource inklusive der Metadaten wie Änderungsdatum, Bearbeiter, Kommentare, etc. gespeichert. Die Ursprungsdatei *foo.html* enthält zu jedem Zeitpunkt die aktuellste Version der Ressource.
SCCS verfügt nicht über eine Anwendungsprogrammierschnittstelle (API), daher müssen andere Systeme oder Werkzeuge die SCCS Funktionalität über die im Folgenden vorgestellten Befehlszeilen nutzen.

Befehlszeilen

SCCS akzeptiert Befehlszeilen im klassischen UNIX Format [Bolinger, Bronson 95]. Eine Befehlszeile besteht aus einem Befehlsnamen, optionalen Parametern und einem oder mehreren Dateinamen. Ein optionaler Parameter beginnt mit einem Bindestrich, dieser unterscheidet ihn von einem Dateinamen. Danach folgt ein einzelner Buchstabe, der die Option identifiziert, und in einigen Fällen eine Zeichenfolge, die als Wert für die Option dient. Die Befehlszeile

prt –s foo.html

sorgt dafür, daß auf der Standardausgabe Metainformationen (z.B. die Versionsnummer, das Datum der letzten Änderung, der Benutzername des Bearbeiters dieser Version, etc.) über sämtliche Versionen der Datei *foo.html* ausgegeben werden. Die Option *–s* führt zur Unterdrückung der Kommentarinformationen in der Ausgabe. Die Antwort des SCCS Systems auf die obige Befehlszeile ist folgende:

SCCS/s.foo.html:

D 1.2.1.1 97/04/21 15:15:20 boesch 5 2 00000/00000/00023

D 1.4 97/04/21 14:50:24 boesch 4 3 00000/00000/00023

D 1.3 97/04/21 14:42:31 boesch 3 2 00000/00000/00023

D 1.2 97/04/21 14:35:27 boesch 2 1 00000/00000/00023

D 1.1 97/04/21 14:33:34 boesch 1 0 00023/00000/00000

Die Anbindung des WWW Servers Jigsaw an SCCS erfolgt über Aufrufe von Befehlszeilen und der Auswertung von deren Ausgabedaten.

5.4.2 Jigsaw

Jigsaw ist ein in Java (vgl. Abschnitt 3.5.2.1) geschriebener objektorientierter WWW Server des WWW Konsortiums. Er profitiert von der Plattformunabhängigkeit jeder in Java geschriebenen Software und ist ohne Änderungen auf allen Plattformen, auf denen ein Java Bytecode Interpreter zur Verfügung steht, lauffähig [Klute 96].

Der Kern von Jigsaw ist seine optimierte interne Datenrepräsentation. Der Server betrachtet jedes Objekt, das er ausliefern kann, z.B. eine Datei oder ein Verzeichnis, als eine *Ressource*. Zu jeder Ressource gehören Metainformationen wie Datentyp, Datum der letzten Änderung, Verfallsdatum, Zugriffsschutz, etc. Alle Ressourcen sind persistente Objekte und werden in einem Cache gehalten. Wird eine Ressource wiederholt abgefragt, kann Jigsaw diese Anfragen aus seinem Cache bedienen und muß bei ausreichendem Hauptspeicher keine Festplattenzugriffe durchführen. Jigsaw definiert eine Reihe von Klassen, die verschiedenste Ressourcen implementieren. So ist z.B. die Klasse *w3c.jigsaw.resources.Resource* die Oberklasse, aller vom Server ausgelieferten Objekte. Subklassen dieser Klasse sind z.B. *w3c.jigsaw.resource.FileResource*, die Dateien auf Ressourcen abbildet und *w3c.jigsaw.resource.DirectoryResource* die Verzeichnisse auf Ressourcen abbildet. Einträge innerhalb einer Verzeichnisressource sind nicht beschränkt auf Dateien oder weitere Verzeichnisse sondern können beliebige Subklassen von *w3c.jigsaw.resources.Resource* sein. Es ist möglich Ressourcen zu erzeugen, die keine Entsprechung im Dateisystem haben. So beantwortet eine Instanz der Klasse *w3c.jigsaw.resources.RelocateResource* eine HTTP GET Anfrage (vgl. Abschnitt 3.3), indem sie den WWW Client dazu auffordert, die gewünschten Daten unter einem anderen URL zu suchen.

Alle Ressourcen sind durch den Server konfigurierbar. Dies geschieht mit Hilfe von HTML Formularen zu den einzelnen Ressourcen. So lassen sich z.B. bei einem Dokument *Overview.html* HTTP Attribute wie Sprache (*Content-Language*), Verfallsdatum (*Expires*), Typ des Inhaltes (*Content-Type*), etc. ändern. Änderungen an Ressourcen wirken sich zunächst nur auf den Cache des Servers im Hauptspeicher aus und werden erst später abhängig von Anzahl und Größe der übrigen Ressourcen auf die Festplatte ausgelagert.

Für die Wahl von Jigsaw als WWW Server war die dynamische Erweiterbarkeit von Java Anwendungen ausschlaggebend. Es ist für den Administrator eines Jigsaw Servers möglich, eigene Java Klassen zur Laufzeit in den Server einzubinden. Dies geschieht z.B. indem man sogenannte Verzeichnisschablonen definiert. Verzeichnisschablonen beschreiben, wie Verzeichnisse mit einem bestimmten Namen auf Ressourceninstanzen abgebildet werden. Dabei wird der Name eines Verzeichnisses einer Java Klasse zugeordnet. Enthält ein vom Client angefragter URL diesen Verzeichnisnamen übernimmt die zugeordnete Java Klasse die weitere Bearbeitung der Anfrage. Die im Folgenden dargestellte Implementierung basiert auf diesem Konzept und gibt ein Anwendungsbeispiel einer Verzeichnisschablone.

5.4.3 Implementierung

Die Implementierung der Erweiterung des WWW Servers Jigsaw erfolgt in zwei Schritten:

1. **Aufbau von SCCS Datenstrukturen:** Aus dem Aufruf von SCCS Befehlszeilen und deren Auswertung werden Instanzen von Java Klassen erzeugt, die den Zustand eines SCCS Verzeichnisses, einer SCCS Archivdatei und einer einzelnen Version aus einer Archivdatei abbilden.

2. **Erzeugung von Jigsaw Ressourcen:** Alle Objekte, die Jigsaw ausliefert, sind Ressourcen (vgl. Abschnitt 5.4.2). Auf Basis der SCCS Datenstrukturen werden Jigsaw Ressourcen erzeugt.

Abbildung 5.6 stellt das Zusammenspiel von SCCS und Jigsaw dar. Der Server erhält eine HTTP Anfrage und gibt diese an die zuständige Ressourceklasse weiter. Diese sendet ihrerseits eine Anfrage nach geeigneten SCCS Datenstrukturen an die Abstraktion des SCCS Systems. Die SCCS Klassen rufen SCCS Befehle auf und erzeugen aus der Antwort des SCCS geeignete Java Datenstrukturen, aus denen die zuständigen Ressourceklassen HTTP Antworten generieren, die der Server an den Client weiterleitet.

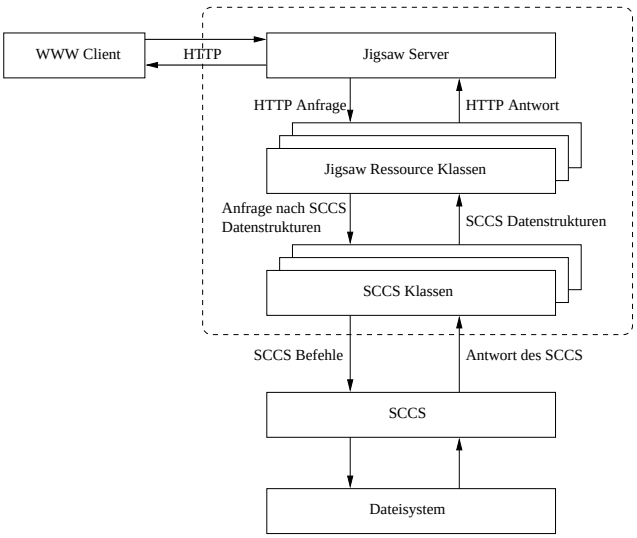

Abbildung 5.6: Zusammenspiel von SCCS und Jigsaw

Die Erweiterung des WWW Servers Jigsaw erfolgt über die Einbindung von Java Klassen in den Server. Steht eine Datei unter der Kontrolle von SCCS, befindet sich im Verzeichnis der Datei ein Unterverzeichnis, daß die Versionierungsinformationen enthält. Diese Tatsache wird bei der Implementierung genutzt, indem der WWW Server Jigsaw mittels einer neu definierten Verzeichnisschablone (vgl. Abschnitt 5.4.2) so konfiguriert wird, daß er die Bearbeitung von Anfragen, die in der URI der Anfrage die Zeichenkette "SCCS" enthalten einer eigens dafür angelegten Java Klasse überträgt. Die Einbindung der entsprechenden Klasse in den Server erfolgt über ein HTML Formular, das vom Server bereitgestellt wird (vgl. Abbildung 5.7).

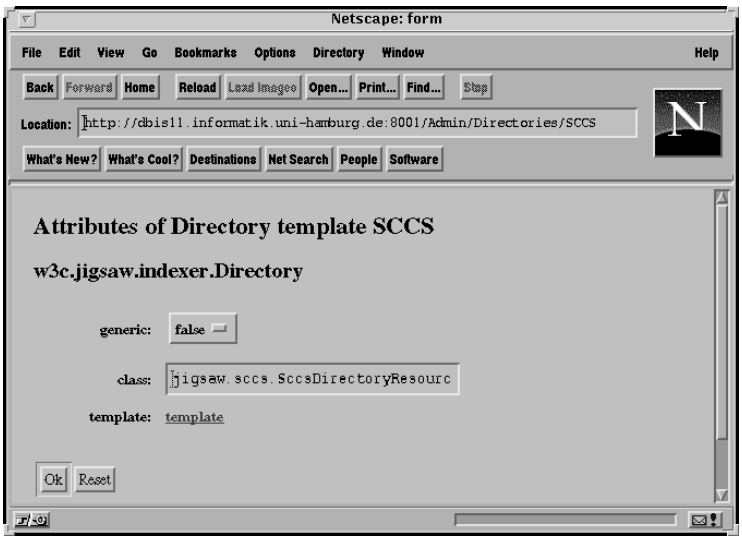

Abbildung 5.7: Einbindung neuer Java Klassen in Jigsaw

Der Kontext der neu in den Server integrierten Klasse *ab.jigsaw.sccs.SccsDirectoryRessource* ist aus einem vereinfacht dargestelltem Ausschnitt der Klassenhierarchie des Servers ersichtlich (vgl. Abbildung 5.8). Jedes vom Server ausgelieferte Objekt ist eine Ressource (vgl. Abschnitt 5.4.2) und daher eine Instanz der Klasse *w3c.jigsaw.resources.Resource* oder einer ihrer Subklassen wie z.B. *w3c.jigsaw.resources.DirectoryResource*, die wiederum Superklasse der Klasse *ab.jigsaw.sccs.SccsDirectoryRessource* ist.
Die neue Ressourceklasse *ab.jigsaw.sccs.SccsDirectoryRessource* stellt die Verbindung zwischen den vorhandenen Serverklassen und der Abstraktion des SCCS her (vgl. Abbildung 5.6).

Konventionen für die Benennung von Versionen

Die Spezifizierung von Versionen einer Ressource erfolgt wie bei [Whitehead 96] durch eine Erweiterung der URI Syntax. Die Nutzung eines zusätzlichen HTTP Nachrichtenkopfes in der Anfrage zur Spezifikation der Version einer Ressource würde u.a. eine Erweiterung der Client Software (dem Browser) erfordern und wurde verworfen, um die Unabhängigkeit der Servererweiterung zu erhalten. Daten werden in dieser Implementierung u.a. über folgende URI Schemata angesprochen:

Protokoll://Rechnername/Pfad/Datei Dies stellt den üblichen Abruf einer Ressource dar.

Protokoll://Rechnername/Pfad/SCCS Der Zugriff auf ein SCCS Verzeichnis gibt dem Client in Form einer HTML Seite Informationen darüber, welche Dateien in welchen Versionen unter dem angegebenen Pfad vorhanden sind.

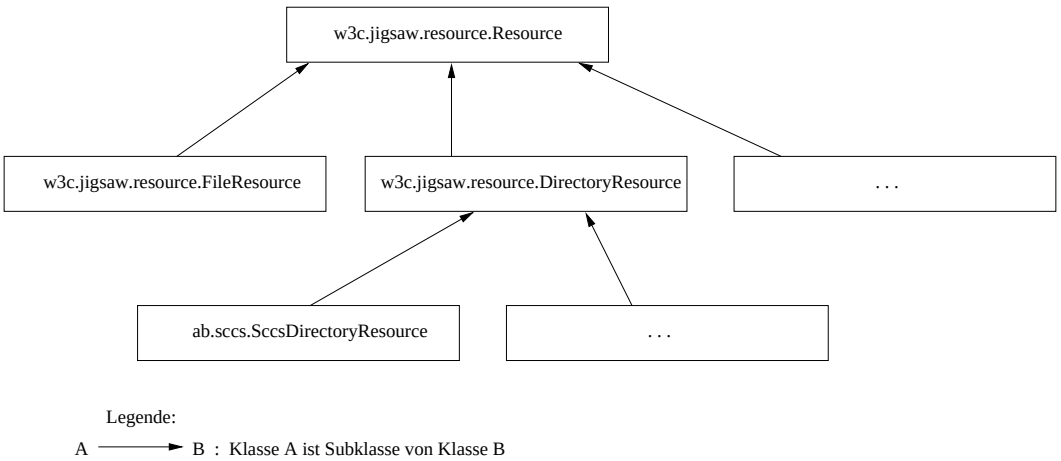

Abbildung 5.8: Ausschnitt der erweiterten Klassenhierarchie von Jigsaw

Protokoll://Rechnername/Pfad/SCCS/Datei Der Zugriff auf eine Datei in einem SCCS Verzeichnis liefert dem Client die Versionstopologie dieser Datei als HTML Seite.

Protokoll://Rechnername/Pfad/SCCS/Datei;version=x.y Dies stellt den Zugriff auf eine spezifische Version einer Datei dar. Dabei entspricht die Versionsnummer "x.y" dem Format der SCCS Versionsnummern.

Die Behandlung von HTTP Anfragen

Jigsaw Objekte behandeln HTTP Anfragen (vgl. Abschnitt 3.3) nach Ressourcen im wesentlichen in zwei Methoden:

lookup: Es wird geprüft, ob die von der Anfrage betroffene Ressource vorhanden ist. Ist das nicht der Fall wird eine Fehlermeldung erzeugt und an den Client ausgeliefert.

perform: Ist die Ressource vorhanden, wird die in der Anfrage enthaltene HTTP Methode auf der Ressource ausgeführt.

Durch Ableitung der Klasse *ab.jigsaw.sccs.SccsDirectoryRessource* von *w3c.jigsaw.resources.Resource* werden diese Methoden vererbt und in der Subklasse überschrieben.
Die *lookup* Methode der Klasse *ab.sccs.SccsDirectoryRessource* ist die Schnittstelle zum SCCS. Sie prüft, ob eine und wenn ja welche Benennungskonvention für versionierte Daten verwendet wird und ob die gewünschten Daten vorhanden sind. Dazu bedient sie sich der Hilfe der Klasse *ab.sccs.SccsDirectory*, die das Bindeglied der Java Datenstrukturen zum SCCS (vgl. Abbildung 5.6) bildet. Nach der Überprüfung wird die Zielressource der Anfrage bestimmt und — falls noch nicht vorhanden — erzeugt. Die *perform* Methode ruft

entsprechend der in der *request line* (vgl. Abschnitt 3.3) enthaltenen HTTP Methode die Methodenimplementation der durch die *lookup* Methode bestimmten Ressourceklasse auf. Die Implementation der HTTP GET Methode sorgt für die Erstellung einer HTTP Serverantwort, die dieser an den Client überträgt.
Bei einem nicht vorhandenen SCCS Verzeichnis sowie einer in einem vorhandenen SCCS Verzeichnis nicht vorhandenen Datei wird eine Fehlermeldung erzeugt und an den Client übertragen. Im Fall einer nicht vorhandenen Version einer Ressource wird die Versionstopologie der Ressource ausgeliefert. An dieser Stelle sind auch andere Konfliktlösungsstrategien denkbar, die z.B. bei einer nicht vorhandenen spezifischen Version die aktuellste oder die initiale Version einer Ressource ausliefern. SCCS erlaubt für diese Zwecke die Definition einer Standardversion, die immer dann ausgewählt wird, wenn keine Version einer Ressource spezifiziert wird.

Bewertung

Im folgenden wird der Ansatz der Versionierung durch Anbindung vorhandener Systeme an WWW Server im Hinblick auf die in Abschnitt 5.2 aufgestellten Anforderungen am Beispiel der vorgestellten Servererweiterung kritisch betrachtet.

Funktionale Anforderungen: Die vorgestellte Implementierung zeigt, daß die funktionalen Anforderungen des nur lesenden Benutzers im Gegensatz zu z.B. einer HTTP Erweiterung relativ einfach zu realisieren sind. Sollen per WWW auch neue Versionen erzeugt werden, sind aufwendigere Lösungen nötig, da nicht nur Information vom Server zum Browser sondern auch umgekehrt fließen muß. Eine Lösung dafür wären z.B. HTML Formulare und verarbeitende CGI Skripte (vgl. Abschnitt 5.3.3). Eine andere Möglichkeit wäre die Realisierung mit Hilfe von mehreren in HTML *Frames* untergebrachten Java Applets, die für die Kommunikation mit dem Server sorgen und Änderungen propagieren.

Technische Anforderungen: Die oben beschriebene Implementierung ist nur zu dem Jigsaw WWW Server kompatibel. Sofern sich allerdings die Java Servlet API (vgl. Abschnitt 3.5.2) bei den Herstellern von Serversoftware durchsetzt, ist es mit deren Nutzung möglich, eine plattformunabhängige und in jeden Server integrierbare Anbindung von Versionierungssystemen wie z.B. RCS und SCCS vorzunehmen. Die Flexibilität in Bezug auf die versionieren Datenformate sowie die effiziente Datenspeicherung hängen nur von dem zugrundeliegenden Versionierungssystem ab. Transparenz und einfache Bedienbarkeit sind gegeben, da der nur lesende Benutzer lediglich einen erweiterten URL angeben muß, um eine versionierte Ressource anzufordern. Somit läßt sich das oben vorgestellte System von jedem Browser nutzen.

Die Anbindung vorhandener Versionierungssysteme an vorhandene WWW Server stellt in dieser Reihe einen relativ einfach realisierbaren Ansatz dar. Vorhandene Systeme werden integriert und müssen nicht von Grund auf neu entwickelt werden, wie z.B. bei einer HTTP Erweiterung.

5.5 Zusammenfassung

In diesem Kapitel wurde gezeigt, daß Versionierungssysteme neben ihren eigentlichen Aufgaben — wie z.B. der Bereitstellung von Historien von Objekten, der Unterstützung von kooperativer Arbeit mehrerer Entwickler an Objekten und der effizienten Verwaltung von Metadaten und Speicherplatz — u.a. Unterstützung für die Konsistenzerhaltung von Hyperlinks und die Redundanzfreiheit bei der Speicherung von WWW Ressourcen bringen können.
Nach einer Einführung in die grundlegenden Konzepte und die Terminologie wurden verschiedene Ansätze zur Versionierung im WWW betrachtet sowie eine eigene Implementation vorgestellt. Die Bewertung der verschiedenen Ansätze erlaubt die Auswahl einer geeigneten Versionierungsstrategie je nach beabsichtigtem Einsatzfeld.
Versionierung durch redundante Datenhaltung (vgl. Abschnitt 5.3.1) bietet eine je nach gewünschter Funktionalität einfache Möglichkeit für die Bereitstellung eines sehr flexiblen Versionierungssystems. Der Aufwand für die Programmierung der verwaltenden Skripte sowie der Verbrauch von Speicherplatz sind jedoch nicht zu unterschätzen.
Sollen lediglich HTML bzw. SGML Dokumente versioniert werden, bieten sich Ansätze wie VTML (vgl. Abschnitt 5.3.2) an, die alle Versionen einer Ressource in einem Dokument speichern und über die erweiterte URL Adressierung die Möglichkeit bieten, transparent auf versionierte Dokumente zuzugreifen. Da für VTML jedoch keine kommerziellen Werkzeuge (z.B. Editoren und Parser) vorliegen, ist die Anbindung eines bereits vorhandenen Versionierungssystems (vgl. Abschnitt 5.3.3 oder Abschnitt 5.4) wie RCS oder SCCS einfacher realisierbar, da hier lediglich vorhandene Systeme integriert werden müssen.
Die umfassendste Funktionalität und Flexibilität bieten HTTP Erweiterungen (vgl. Abschnitt 5.3.4), die allerdings durch die Erstellung neuer Browser und WWW Server auch den größten Implementierungsaufwand erfordern.
Im Zusammenhang mit der Hypertextstruktur des WWW sind technische Lösungen für das Konfigurationsmanagement von weitergehendem Interesse. Konfigurationen entstehen, wenn bestimmte Versionen von Ressourcen Beziehungen zu spezifischen Versionen anderer Ressourcen z.B. über Hyperlinks aufbauen (vgl. Kapitel 6).

6. Konfigurationsmanagement im WWW

Im vorherigen Kapitel wurde gezeigt, daß die Versionierung von Ressourcen im WWW die Stabilität von Hyperlinks unterstützt, indem einzelne Hyperlinks versioniert werden und so auf eine unveränderliche spezifische Version einer Ressource zeigen. Diese Ressource steht unter der Kontrolle eines Versionierungssystems, das Änderungen an der Ressource in neuen Versionen ablegt, anstatt die vorhandene Instanz zu ändern.

Versionierung in Hypermediasystemen erfordert nicht nur die Verwaltung von Revisionen und Varianten einzelner Objekte beliebigen Typs, sondern auch das versionierte *Konfigurationsmanagement* von Strukturen und Beziehungen, die zwischen diesen bestehen [Hicks et al. 91]. Konfigurationsmanagment kontrolliert die Evolution eines Produktes u.a. durch die Identifikation seiner Komponenten und Veränderungen [Feiler 91].

Eine *Konfiguration* umfaßt eine Menge von Objekten innerhalb eines Systems und ihre Beziehungen untereinander. So ist z.B. eine Menge von Klassen, die ein Softwaresystem bilden, eine Konfiguration. Im WWW kann man z.B. die Struktur einer Softwaredokumentation, die sich aus HTML Dokumenten und Bildern zusammensetzt, als Konfiguration auffassen. Solche Konfigurationen können im WWW auch andere Objekttypen enthalten, z.B. Audio- oder Videodateien, Java Klassen, CGI Skripte, etc. Benutzer des WWW möchten lesenden Zugang zu verschiedenen Versionen von Objekten haben, wie z.B. Dokumentationen von HTML, HTTP oder SQL Standards. Diese Objekte können aus mehreren Teilobjekten zusammengesetzt sein und Konfigurationen bilden. Für den Benutzer ist es wichtig, daß er in solchen Konfigurationen kontextabhängige Hyperlinks zur Verfügung gestellt bekommt, d. h. in Objekten vorkommende Hyperlinks müssen auf die in diesem Kontext richtigen Versionen anderer Objekte zeigen.

Werden die einzelnen Objekte einer Konfiguration versioniert, muß festgelegt werden, welche Version eines Objektes zu einer Konfiguration gehört. Abbildung 6.1 zeigt drei Objekte, die jeweils in mehreren Versionen vorliegen. Die Konfigurationsbeschreibung legt fest, welche Versionen zur Konfiguration gehören. Die Bestimmung der zu einer Konfiguration gehörenden spezifischen Versionen der einzelnen Komponenten kann auf zwei Arten erfolgen:

statisch: Es wird zum Erstellungszeitpunkt der Konfigurationsbeschreibung festgelegt, welche Version eines Objektes zu einer Konfiguration dazugehört.

dynamisch: Die Auswahl der zu einer Konfiguration gehörenden spezifischen Version eines Objektes wird erst zum Auswertungszeitpunkt der Konfigurationsbeschreibung bestimmt. So ist es möglich, festzulegen, daß z.B. immer die aktuellste Version eines Objektes zu einer Konfiguration gehört.

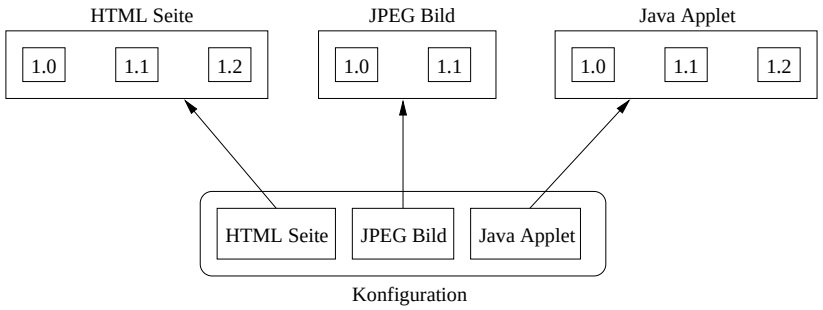

Abbildung 6.1: Zuordnung von versionierten Objekten zu Konfigurationen

Werkzeuge für das Konfigurationsmanagement bieten Möglichkeiten zur eindeutigen Identifikation der Komponenten einer Konfiguration und der Beziehungen zwischen ihnen. Das Konfigurationsmanagementwerkzeug *make* [Herold 94] z.B. benutzt dazu eine Beschreibung der Konfiguration in Form einer speziellen Datei — dem *makefile*.
Im Folgenden wird zuerst die eindeutige (evtl. dynamische) Spezifikation verschiedener Versionen einzelner Objekte im WWW als Voraussetzung für das Konfigurationsmanagement erläutert (vgl. Abschnitt 6.1). In Abschnitt 6.2 wird die Notwendigkeit des Konfigurationsmanagement für WWW Ressourcen motiviert. Danach werden verschiedene Möglichkeiten zur Realisierung von Konfigurationsmanagment im WWW dargestellt (vgl. Abschnitt 6.3).

6.1 Versionierung von Ankern

Anker sind die Endpunkte von Hyperlinks. Anker können einerseits einen ganzen Knoten in einem Netzwerk bezeichnen, wie z.B. eine HTML Seite oder ein Bild. Zum anderen können Anker eine bestimmte Stelle in einem Knoten bezeichnen (*interner Anker*). Auf der Ebene der nicht internen Anker kann die Versionierung statisch oder dynamisch mit Hilfe von Attributen erfolgen, z.B. könnte ein Anker, der stets dynamisch auf die aktuelle Version eines Objektes zeigen soll, ein Attribut *version=head* haben:

* Link zum aktuellen Dokument *

Eine ältere Version wäre statisch mit der Angabe eines Datums oder einer Versionsnummer zu referenzieren, z.B.:

* Link zu der Version 1.3 des Dokumentes *

Nicht interne Anker lassen sich wie oben beschrieben mit Hilfe eines erweiterten URL versionieren. In HTML (vgl. Abschnitt 3.2) werden interne Anker als Markierung einer bestimmten Stelle in einem Dokument definiert:

* Ein beliebiger Text*

Befindet sich dieser Anker z.B. in einem Dokument "doc.html" wird er von einem anderen Dokument referenziert mit:

* Link zu einem beliebigen Text*

Eine sichere Referenzierung von internen Ankern ist nur möglich, wenn explizit die Version des den internen Anker umgebenden Objektes angegeben wird. Dies könnte folgendermaßen erfolgen:

* Link zu einem beliebigen Text*

Wird keine spezifische Version in der Referenz auf den internen Anker angegeben, ist es möglich, daß die vom Konfigurationsmanagementwerkzeug ausgewählte Version (z.B. die neuste) diesen internen Anker nicht mehr enthält oder er mit einer anderen Semantik besetzt ist.

6.2 Konfigurationen im WWW

Im WWW werden Konfigurationen durch die Verbindung einzelner Objekte mittels Hyperlinks hergestellt. Jede HTML Seite, die Hyperlinks enthält, baut explizit eine Beziehung zu anderen Ressourcen auf und ist, wenn sie von anderen Hyperlinks referenziert wird, implizit Teil einer oder mehrerer solcher Beziehungen. Der Autor eines Hyperlinks kann mit diesem zwei unterschiedliche Intentionen verfolgen:

Eine allgemeine Referenz bezieht sich auf das allgemeine Thema eines Zielobjektes (z.B. einer HTML Seite). Hierbei ist die Version des Objektes nicht relevant.

Eine spezifische Referenz bezieht sich auf den exakten Inhalt des Zielobjektes. Die Version des Zielobjektes ist hier entscheidend für die Integrität der Referenz.

Ein Beispiel soll die obige Definition verdeutlichen. Der im folgenden Beispiel enthaltene Hyperlink referenziert eine spezifische Version der HTML Spezifikation nämlich die Version 2.0 und ist somit eine spezifische Referenz.

HTML ist in der Version 2.0 standardisiert.

Spezifische Referenzen erfordern bei einer Abfrage einer Ressource exakt den gleichen Zustand der Ressource, den diese zum Zeitpunkt der Erstellung der Referenz hatte. Während in obigem Beispiel die Ersetzung der Ressource durch einen neuen HTML Standard den Hyperlink ungültig machen würde, bliebe er im nächsten Beispiel weiter gültig, da er hier die Intention einer allgemeinen Referenz verfolgt.

HTML ist eine auf SGML basierende Seitenbeschreibungssprache.

Um die Integrität spezifischer Referenzen zu bewahren, muß der jeweilige Hyperlink entweder explizit versioniert werden — z.B. durch eine versionierte URL:

HTML ist in der Version 2.0 standardisiert.

— oder der Hyperlink muß Teil eines Kontextes sein. Ein *Kontext* ist eine Menge spezifischer Versionen von Links, die als eine Einheit betrachtet, benannt und referenziert werden können [Hicks et al. 91]. In Abbildung 6.2 ist ein Kontext mit dem Namen „WWW 1995“ abgebildet. Er enthält eine Menge spezifischer Versionen von Hyperlinks.

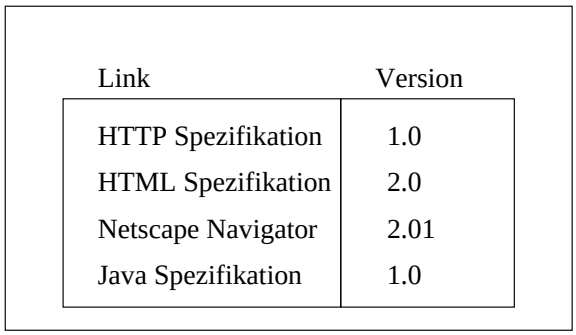

Abbildung 6.2: Ein Kontext

Der Kontext könnte dann z.B. auf Basis der Quellressource des Hyperlinks die passende Version der Zielressource bestimmen. In Abbildung 6.3 ist zum einen die Beziehung von Quelle, Hyperlink und Ziel ohne die Unterstützung von Kontexten visualisiert und zum anderen die Spezifikation einer Version eines Hyperlinks unter Zuhilfenahme eines Kontextes.
Bewegt sich ein Benutzer z.B. durch eine bestimmte Version einer aus vielen Dokumenten bestehenden Softwaredokumentation, wird jeder Hyperlink durch den Kontext auf die entsprechende Version des Zielobjektes umgeleitet. Der Vorteil der Verwendung von Kontexten besteht also darin, daß die in den einzelnen Ressourcen enthaltenen Hyperlinks keine Versionsinformationen enthalten müssen und keine Änderungen an den Hyperlinks einer Ressource erforderlich werden, wenn eine neue Version der Ressource erzeugt wird.
Angenommen ein Dokument „http://foo.bar/www.html“ enthält Hyperlinks zu den im Kontext „WWW 1995“ enthaltenen Zielen. Der Kontext sorgt dann für die Auswahl der richtigen Version der Zielressource. Das heißt, die im Dokument „http://foo.bar/www.html“ enthaltenen Hyperlinks sind nicht versioniert (z.B. durch Angabe einer Versionsnummer „.../html-Spez.html;version=2.0“) sondern enthalten lediglich die üblichen Links auf das Zielobjekt

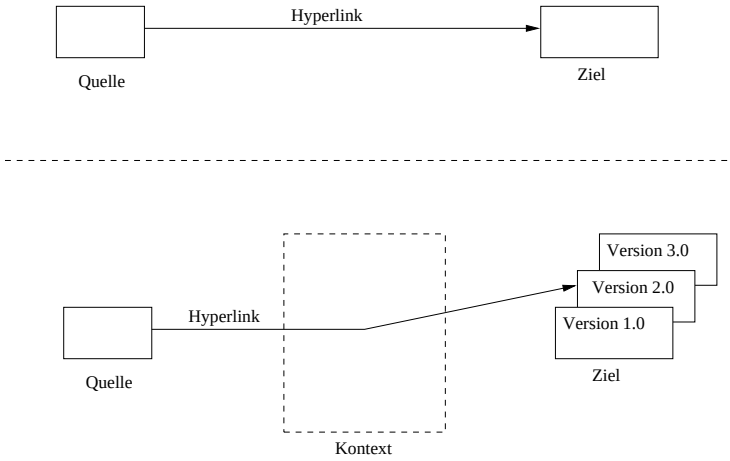

Abbildung 6.3: Auswahl einer Version durch einen Kontext

(z.B. „.../htmlSpez.html“). Wird das Dokument „http://foo.bar/www.html“ überarbeitet, müssen lediglich die Versionen der Links im Kontext „WWW 1995“ geändert werden. Allerdings gibt es dann keinen Zugriff mehr auf alte Versionen des Kontextes und der Name des Kontextes ist semantisch nicht mehr korrekt. Es ist also wünschenswert, daß Kontexte selbst ebenfalls versioniert werden.
In Abbildung 6.4 ist das Dokument „http://foo.bar/www.html;version=1.0“ dem Kontext „WWW;version= 1.0“ zugeordnet und die enthaltenen Hyperlinks werden entsprechend dem Stand des Jahres 1995 versioniert. Das Dokument „http://foo.bar/www.html;version=3.0“ hingegen wird in diesem Beispiel dem Kontext „WWW;version=3.0“ zugeordnet, damit die enthaltenen Hyperlinks entsprechend dem Stand des Jahres 1997 versioniert werden.[1]
Möglichkeiten zur Realisierung des Kontextmanagement werden im nächsten Abschnitt vorgestellt.

6.3 Realisierung von Kontexten

Das Problem des Konfigurationsmanagement ergibt sich als Folge der Integration von Versionierungsmechanismen in Hypermediasysteme wie dem WWW. Ein *Kontext* ist eine Menge spezifischer Versionen von Links, die als eine Einheit betrachtet, benannt und referenziert werden können. In diesem Abschnitt werden Ansatzmöglichkeiten[2] zur Realisierung von Kontexten untersucht.

[1] Die schwarzen Punkte in der Abbildung stellen weitere mögliche Versionen des Dokumentes „http://foo.bar/www.html“ bzw. des Kontextes „WWW“ dar.
[2] Eine Implementation dieser Ansätze könnte zum Gegenstand weitergehender Untersuchen genutzt werden (vgl. Abschnitt 9.2).

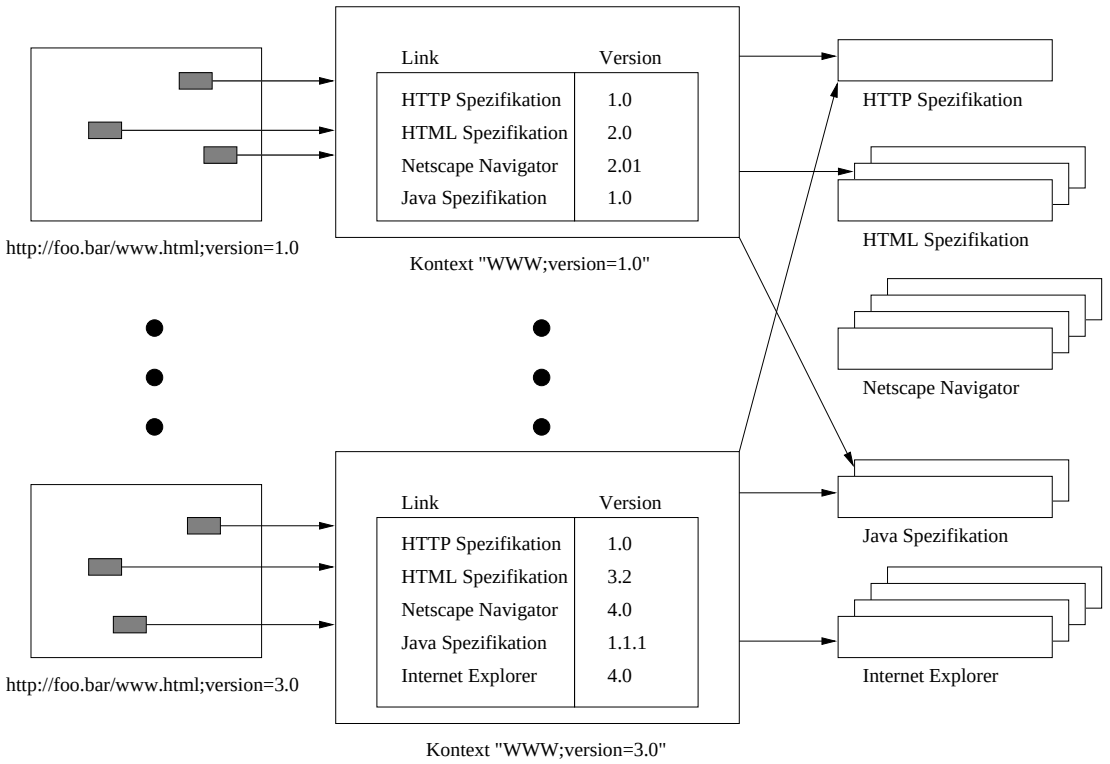

Abbildung 6.4: Ein versionierter Kontext

6.3.1 Kontextmanagement durch den Server

Ein Kontext auf dem Server besteht aus einer Menge von nicht versionierten URLs, die jeweils einer versionierten URL zugeordnet sind. Erhält ein Server eine Anfrage nach einer Ressource, muß der Client als zusätzlichen Parameter der Anfrage den Namen eines Kontextes mitliefern, der für diese Anfrage gelten soll. Danach prüft der Server, ob die URL der Anfrage des Client in dem spezifizierten Kontext enthalten ist. Ist dies der Fall, wird anstatt der URL der Anfrage die dieser im spezifizierten Kontext zugeordnete versionierte URL vom Server mit der HTTP Methode betraut.

Die Realisierung wäre z.B. möglich, indem beim erstmaligen Betreten eines Kontextes der Server dem Client einen Netscape Cookie (vgl. Abschnitt 3.4) mit dem Namen des Kontextes übergibt, der bei jedem Aufruf des Servers wieder an diesen zurückgegeben wird. Dieser kann dann mit Hilfe eines CGI Skriptes oder einer erweiterten Serverfunktionalität Anfrage und Kontext auswerten und die entsprechende Ressource ausliefern.

Die Zuordnung von spezifischen Versionen zu Kontexten kann im *Revision Control System* (RCS) mit Hilfe von symbolischen Namen vorgenommen werden [Bolinger, Bronson 95]. Ein symbolischer Name kann genau einer Version einer Ressource zugeordnet werden. Einer

Version können jedoch mehrere symbolische Namen gegeben werden. Im Zusammenhang mit Kontexten in Hypermediasystemen heißt das, daß jede Version einer Ressource in mehreren Kontexten enthalten sein kann, ein Kontext jedoch nur eine Version einer Ressource beinhalten darf. Ein symbolischer Name bezeichnet hier alle zu einem Kontext gehörenden Versionen in verschiedenen Archivdateien. Ein CGI Skript oder eine Servererweiterung muß zur Bestimmung der spezifischen Version einer Ressource den in der Anfrage mitgelieferten Kontextnamen mit dem symbolischen Bezeichner der verschiedenen Versionen der angeforderten Ressource vergleichen und die entsprechende Version identifizieren und ausliefern.

6.3.2 Kontextmanagement durch den Client

Das Kontextmanagment durch den Client kann wie in Abschnitt 5.3.4 beschrieben mit Hilfe von Konfigurationsdateien erfolgen, die nicht versionierte URLs versionierten URLs zuordnen und wie ein Filter die jeweiligen Anfragen des Client beeinflussen.
Die automatische Ersetzung einer URL einer Anfrage beim Client ist ein potentielles Sicherheitsrisiko für den Benutzer, da er beim automatischen Laden und Anwenden von Konfigurationsdateien nicht sicher sein kann, daß er tatsächlich die von ihm gewünschte Ressource erhält, da die Konfigurationsdatei URLs automatisch auf andere Rechneradressen umleiten kann. So sollte durch den Benutzer definierbar sein, ob eine Konfigurationsdatei eine komplette URL ersetzen oder nur Versions- bzw. Kontextinformation hinzufügen darf.

7. Verteilte Anwendungen im WWW

Das WWW bietet aufgrund seiner offenen, plattformunabhängigen Standards für den Informationsaustausch und der vielfach vorhandenen kostenlosen Software für Client und Server eine gute Basis für die Integration bestehender und neuer Anwendungen unter einer Benutzerschnittstelle. Verteilte Anwendungen können das WWW nutzen, um die Portierung von Clientsoftware auf verschiedene Betriebssystemplattformen zu vermeiden.
Verteilte Anwendungen bestehen aus mindestens zwei Komponenten, die miteinander kommunizieren. So ist das WWW eine verteilte Anwendung, bei der der Browser für die Darstellung der vom Server per HTTP übertragenen HTML Dokumente verantwortlich ist. Der Server ist für Lokalisierung, Retrieval, Berechnung, Layout — z.B. bei Datenbankzugriffen mit Hilfe von CGI Skripten (vgl. Kapitel 3) — und Sendung der angeforderten Information zuständig. Da heute im WWW viele Arbeitsplatzcomputer über eigene Rechenkapazitäten verfügen, ist diese Verteilung des Rechenaufwandes suboptimal, weil die Ressourcen des Client bei der Berechnung der Daten ungenutzt bleiben.
Die Nutzung der Kombination von HTML Formularen, dem HTT Protokoll und CGI Applikationen für verteilte Anwendungen im WWW hat den Nachteil, daß keinerlei Information über die Datentypen einer Anfrage übertragbar sind. Stattdessen besteht diese aus Zeichenketten, die das CGI Programm interpretieren und mit Semantik besetzen muß. Es kann also keine typsichere Parameterübergabe zugesichert werden, da weder Client noch Server überprüfen können, ob die übergebenen Parameter der Schnittstellendefinition entsprechen. Die in Abschnitt 7.3 vorgestellten Technologien für verteilte Anwendungen ermöglichen die typsichere Übergabe von Parametern in der Kommunikation zwischen Client und Server.
Voraussetzung für eine bessere Verteilung des Rechenaufwandes sowie der eventuellen Reduzierung der Netzwerkzugriffe durch lokal durchgeführte Berechnungen ist die Ausführung von Programmcode auf dem Rechner des Benutzers. Die in Abschnitt 3.5 vorgestellten Technologien erlauben die Einbettung von Programmcode in HTML Seiten, der auf dem Benutzerrechner ausgeführt wird. Hierbei sind zwei Varianten der Programmexekution zu unterscheiden:

Lokale Ausführung: Nach dem Laden des Programmcodes vom Server erfolgt dessen Ausführung auf dem Clientrechner, ohne daß eine direkte Kommunikation zwischen dem geladenen Clientcode und einem auf einem anderen Rechner laufenden Programm

stattfindet. Ein Beispiel hierfür ist ein Java Applet, das eine Animation auf dem Clientrechner ausführt, oder ein JavaScript Programm, das die Korrektheit von Benutzereingaben in einem HTML Formular überprüft.

Verteilte Ausführung: Ein über das WWW geladenes Clientprogramm kommuniziert mit einem oder mehreren Serverprogrammen auf einem entfernten Rechner. Dies ist z.B. der Fall bei einem Java Applet, das eine Verbindung mit einer Datenbank auf einem Server aufbaut und Operationen auf der Datenbank ausführt.

Die lokale Ausführung kommt in Fällen in Betracht, in denen das Programmobjekt mobil ist. Dies ist z.B. bei Datenbanken und großen Programmobjekten nicht der Fall und auch nicht gewünscht, da dadurch dem Benutzer evtl. die gesamte Informationsbasis eines Unternehmens zur Verfügung gestellt würde. Andererseits ist bei der Aufteilung großer Programme in mehrere kleine Komponenten die Übernahme von Teilfunktionalität durch den Client, wie z.B. der grafischen Aufbereitung der Datensätze einer Datenbankabfrage erstrebenswert. Dies hat bei wiederholter Nutzung des Codes durch den Client eine Entlastung von Server und Netzwerk sowie eine bessere Ausnutzung der Clientressourcen zur Folge.
Die Kommunikation in verteilten Anwendungen erfordert oft das Management individueller Kommunikationszustände, die sich aus sequentiell aufeinanderfolgenden Vorgängen ergeben. Ein klassisches Beispiel dafür ist ein Buchungsvorgang für eine Urlaubsreise, der sowohl eine Flugbuchung, eine Hotel– sowie eine Mietwagenreservierung umfaßt [Kreuz, Lange 96]. Diese drei voneinander getrennten Vorgänge stehen in einem unmittelbaren zeitlichen und örtlichen Zusammenhang und erfordern eine zusammenhängende transaktionale Behandlung, da, falls einer der genannten Vorgänge nicht durchführbar ist, die evtl. schon erfolgten Buchungen automatisch zurückgenommen werden müssen. Da HTTP ein verbindungsloses Protokoll ist, werden verschiedene Zustände der Kommunikation zwischen Client und Server nicht unterstützt, vielmehr wird jede Anfrage eines Client unabhängig von der vorhergehenden behandelt. Andere verteilte Anwendungen wie z.B. Telekonferenzanwendungen und Mehrbenutzerspiele erfordern inhärent einen hohen Grad an Benutzerinteraktion und einen verbindungsorientierten Kommunikationsmechanismus zwischen den beteiligten Programmobjekten.
Komponentenmodelle wie z.B. COM erlauben die einfache Komposition von Softwarekomponenten verschiedener Hersteller zu kompletten Applikationen. Der Vorteil der Komponentenmodelle liegt in dem hohen Abstraktionsgrad des vorhandenen Codes. Komponenten für Benutzerschnittstellen, Applikationsalgorithmen und Datenverwaltung können vom Anwender mit Hilfe von Skriptsprachen und grafischen Editoren zu vollständigen Applikationen zusammengesetzt werden [Bösch 95]. Komponentenmodelle benötigen für die Interaktion zwischen Komponenten auf dem gleichen Rechner, genau wie für Zugriffe auf im Netzwerk verteilte Komponenten einen Kommunikationsmechanismus.
Bessere Ressourcennutzung in verteilten Anwendungen, Typisierung von Parametern in der Kommunikation zwischen Client und Server, immobile Objekte, das Management von Kommunikationszuständen und die Unterstützung von Komponentenmodellen sind Gründe für die Notwendigkeit zusätzlicher Mechanismen zur Kommunikation verteilter Objekte im

WWW. Diese Mechanismen erlauben die Nutzung bereits vorhandener und eventuell sehr umfangreicher Programme und Daten auf entfernten Rechnern über das WWW, dessen grundlegende Kommunikationstechnologie (CGI und HTTP, vgl. Kapitel 3) für die Bereitstellung statischer Informationen und einfacher formulargesteuerter Anwendungen geeignet ist.
Nach einer Einführung in Terminologie und Konzepte verteilter Anwendungen (vgl. Abschnitt 7.1) werden in Abschnitt 7.2 Anforderungen an Kommunikationsmechanismen für verteilte Objekte erarbeitet. Darauf aufbauend werden im folgenden drei kommerziell relevante Ansätze zur Kommunikation von verteilten Objekten im WWW vorgestellt. Die *Common Object Request Broker Architecture* (CORBA) erlaubt die plattform– und sprachunabhängige Kommunikation zwischen beliebig im Netzwerk verteilten Applikationen (vgl. Abschnitt 7.3.1). Die *Remote Method Invocation* (RMI) ist eine Klassenbibliothek der Programmiersprache Java (vgl. Abschnitt 3.5.2) und ermöglicht das Referenzieren von Java Objekten und den Aufruf ihrer Methoden über ein Netzwerk (vgl. Abschnitt 7.3.2). Das *Distributed Component Object Model* (DCOM), das bisher auf die Microsoft Windows Plattform beschränkt ist (Stand Juli 1997), unterstützt die Kommunikation zwischen Binärcodekomponenten auf verteilten Rechnern (vgl. Abschnitt 7.3.3).

7.1 Terminologie und Konzepte

In verteilten Anwendungen versteht man unter einem *Server* einen Prozeß, der auf Kommunikationsanforderungen eines anderen Prozesses – des *Clients* – wartet und auf diese reagiert [Mathiske 96]. Es spielt dabei keine Rolle, ob Client und Server auf dem gleichen oder verschiedenen Rechnern laufen.
Damit Client und Server miteinander kommunizieren können, stellt der Server eine Schnittstelle mit Operationen zur Verfügung, die der Client transparent in seinen Programmtext einbetten und aufrufen kann. Die Implementation der Schnittstelle wird durch den Server bereitgestellt (vgl. Abbildung 7.1).
Der Client ruft die auf dem Server implementierten Operationen mit Hilfe eines *Stub* auf. Ein Stub nimmt Funktionsaufrufe des Client entgegen, konvertiert diese inklusive aller übergebenen Parameter in eine zur Netzwerkübertragung geeignete serielle Repräsentation (*marshalling*) und leitet sie für den Client transparent über einen Kommunikationsmechanismus an den Server weiter. Auf der Serverseite rekonstruiert der *Skeleton* den Funktionsaufruf und die Parameter aus der seriellen Repräsentation (*unmarshalling*) und leitet die entsprechenden Funktionsaufrufe an die Funktionsimplementation weiter. Der Server führt die Funktion aus und übergibt das Ergebnis an den Skeleton. Der Skeleton übernimmt das *marshalling* des Ergebnisses und leitet dieses über einen Kommunikationsmechanismus an den Client weiter, dessen Stub das *unmarshalling* des Ergebnisses und die Ergebnisübergabe an den Client vollzieht.
Stub und Skeleton werden aus einer gesondert zur Implementation zu definierenden Schnittstellenbeschreibung mit Hilfe eines speziellen Compilers generiert. Wird die Schnittstelle z.B. durch das Hinzufügen neuer Operationen oder Signaturmodifikationen geändert, ist für die

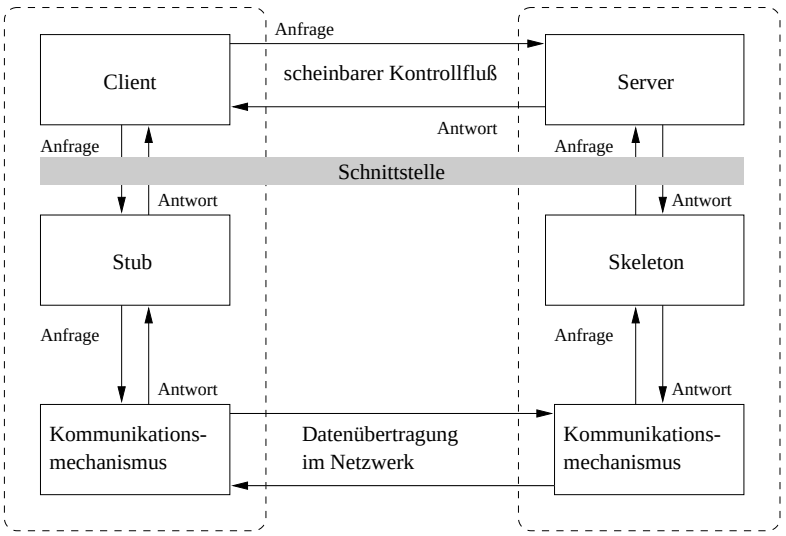

Abbildung 7.1: Kommunikation zwischen verteilten Anwendungen

Nutzung der neuen Funktionalität ein neuer Generierungsvorgang erforderlich.
Ziel dieses mehrschichtigen Kommunikationsmodells ist die Abstraktion von den Kommunikationsvorgängen zwischen Client und Server. Der Client erhält über ein Stellvertreterobjekt (*Proxy*) die Illusion, direkt mit dem Server zu kommunizieren. Der Proxy verfügt über die gleichen Schnittstellen wie die Implementation des Serverobjektes. Diese werden vom Client angesprochen und der Proxy leitet die Anfragen des Clientobjektes mit Hilfe des Stub über das Netzwerk an die Implementation des Objektes weiter. Für den Client entsteht so der Eindruck, er würde mit einem lokalen Objekt kommunizieren. Der Applikationsprogrammierer muß sich nicht um Details der Kommunikation wie z.B. Datenformate und Protokolle kümmern, sondern kann entfernte Dienste transparent nutzen.
Der Vorgang des Aufrufens von auf entfernten Rechnern vorhandenen Operationen wird als *entfernter Methoden-* oder *Prozeduraufruf* bezeichnet.

ORB versus RPC

Man unterscheidet zwei Arten des entfernten Methodenaufrufes. Ein *Object Request Broker* (ORB) unterstützt das klassische Objektmodell, indem Methodenaufrufe an eine spezifische Instanz einer Objektklasse weitergeleitet werden, die über ihre privaten Daten in Form von Attributen verfügt. Ein *Remote Procedure Call* (RPC) greift hingegen auf eine Funktion zu, die nicht über spezifische Instanzdaten verfügt. Beim RPC werden die zu bearbeitenden Daten in der Regel in Form von Parametern übergeben bzw. referenziert, während beim ORB Ansatz spezifische Instanzen von Objekten mit einem individuellen Zustand angesprochen werden.

Desweiteren bieten ORBs das Konzept des Polymorphismus, d. h. unterschiedliche Objektklassen verhalten sich bei Aufruf der gleichen Methode unterschiedlich. Abbildung 7.2 (nach [Orfali, Harkey 97]) verdeutlicht die Unterschiede zwischen den beiden Konzepten.

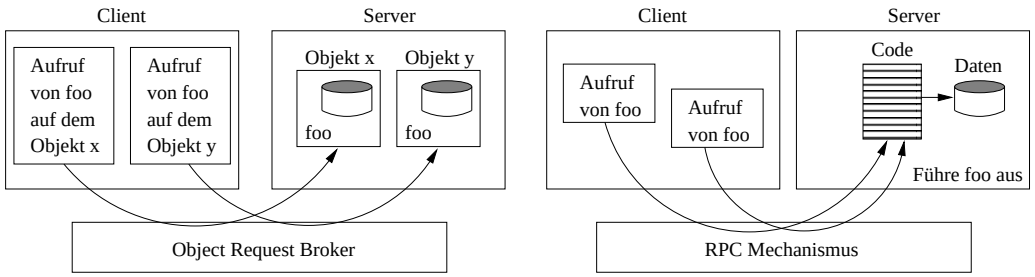

Abbildung 7.2: ORB versus RPC

7.2 Anforderungen an Kommunikationsmechanismen für verteilte Objekte im WWW

Auf Basis von [Richtsmeier 97] und [Schwarz 97] werden im folgenden Anforderungen an Kommunikationsmechanismen für verteilte Objekte im WWW aufgestellt:

Behandlung der Heterogenität von Plattformen: Die an einem verteilten System beteiligten Rechner stammen in der Regel von verschiedenen Herstellern und benutzen unterschiedliche Betriebssysteme und Datenformate. Damit Objekte auf unterschiedlichen Plattformen miteinander kommunizieren können, muß ein einheitliches Datenaustauschformat definiert werden.

Behandlung der Heterogenität von Programmiersprachen: Damit Systeme, die in unterschiedlichen Programmiersprachen implementiert wurden, miteinander kommunizieren können, müssen Schnittstellendefinition und Implementierung voneinander getrennt werden. Ein Kommunikationsmechanismus kann dann anhand der Schnittstellenbeschreibungen Methodenaufrufe an die Implementierung weiterleiten.

Integration mit bestehender WWW Infrastruktur: Es muß möglich sein, Objekte ähnlich wie HTML Seiten über das WWW zu verteilen, mit den verteilten Objekten die Fähigkeiten der Browser zu erweitern sowie eine Interaktion zwischen einer HTML Seite und einem integrierten Objekt herzustellen.

Lokale Sicherheit des Client im WWW: Ein WWW Client muß sich auf die sichere Ausführung eines unbekannten über das WWW geladenen Programms verlassen können. So sollte es für einen Client möglich sein, sein System vor unerwünschten

Zugriffen auf seine lokale Festplatte zu schützen, das Starten von anderen Anwendungen zu verhindern und den Aufbau beliebiger Netzwerkverbindungen zu entfernten Rechnern zu verbieten.

Entwicklungsunterstützung: Neben leicht handhabbaren Kommunikationsdiensten sind Systemdienste wie z.B. Transaktionsmanagement, Lizensierung von Aufrufen verteilter Objekte, Speicherung von Objekten, etc. wünschenswert. Außerdem werden Werkzeuge benötigt, die die einfache Integration von verteilten Objekten in HTML Seiten ermöglichen sowie Generatoren für Stubs und Skeletons aus Schnittstellenbeschreibungen.

Dynamisches Finden und Aufrufen von Objekten: Softwarekomponenten müssen sich dynamisch zur Laufzeit zu einem Gesamtsystem zusammenfügen lassen und Methodenaufrufe sollten zur Laufzeit aus Schnittstellenbeschreibungen generierbar sein, um die Austauschbarkeit von Objekten zu gewährleisten, da zum Übersetzungszeitpunkt von Client und Server nicht immer bekannt ist, welche Schnittstellendefinitionen zur Kommunikation verwendet werden. Handelt es sich bei dem Server z.B. um einen Interpreter, der zur Laufzeit neue Objektbeschreibungen verarbeitet und so neue Objektschnittstellen bereitstellt, muß es möglich sein, diese neuen Objekte von einem entfernten Rechner aufzurufen. Clients können z.B. Vermittlungsdienste[1] benutzen, um Objekte in einem Netzwerk zu finden. Da den Clients nach dem Finden eines Objektes zur Benutzung desselben in der Regel kein Stub zur Verfügung steht, muß es Mechanismen geben, die den Client mit Metainformationen (z.B. Methodennamen, benötigte Parameter und deren Typen) über das Objekt versorgen und es ihm erlauben das Objekt dynamisch aufzurufen. Dieses Vorgehen ist insbesondere im WWW im Rahmen einer wirtschaftlichen Nutzung von verteilten Objekten wünschenswert, z.B. zur Nutzung der preiswertesten Druck- oder Konvertierungsdienste, die evtl. zum Erstellungszeitpunkt des Client noch gar nicht existierten.

Eine weitere — insbesondere im WWW wichtige — Anforderung ist die Kommunikation über die Grenzen von *Firewalls*. Eine Firewall entscheidet anhand einer festgelegten Sicherheitsstrategie, welche Anfragen von außen in ein an das Internet angeschlossenes Firmennetz gelangen und welche Pakete das Firmennetz in umgekehrter Richtung verlassen dürfen. Um die Kommunikation zwischen verteilten Objekten auch über die Grenzen von Firewalls hinweg zu ermöglichen, gibt es zwei Ansätze. Der erste Ansatz basiert auf einem Proxy Mechanismus, bei dem auf dem Firewall-Rechner eine Stellvertreteranwendung für den eigentlichen Objektserver eingerichtet wird. Dieser Proxy entscheidet anhand einer Zugangskontrolliste, ob eine Anfrage an das auf dem internen Server vorhandene Objekt gestattet oder abgelehnt wird. Der zweite Ansatz bettet Anfragen an einen Objektserver in HTTP Anfragen ein. Dies wird als *Tunneling* bezeichnet. Da auf einer Firewall in der Regel ein HTTP Proxy vorhanden ist, erspart man sich so die Installation und Pflege eines zusätzlichen Proxy. Allerdings erfordert das *Tunneling* sowohl auf der Client- als auch

[1] Der CORBA *Trader Service* erlaubt Objekten ihre Dienste öffentlich bekannt zu machen und für Aufträge anzubieten. Der Dienst ist vergleichbar mit einem Branchentelefonbuch wie z.B. den „Gelben Seiten".

auf der Serverseite den Einsatz von Kodierern bzw. Dekodierern, die die jeweiligen Kommunikationsprotokollaufrufe in HTTP Aufrufe ver- bzw. entpacken. Die im Abschnitt 7.3 betrachteten Technologien unterstützen neben dem Proxy Mechanismus das *Tunneling* von Anfragen an einen Objektserver in HTTP Anfragen, daher wird diese Anforderung nicht im Einzelnen näher betrachtet.
Die o.g. Anforderungen bilden die Grundlage für die Bewertung der im Folgenden vorgestellten Technologien in Abschnitt 7.4.

7.3 Technologien

Java ist u.a. aufgrund seiner Plattformunabhängigkeit, des Sicherheitskonzeptes, der Codemobilität und der Unterstützung grafischer Benutzerschnittstellen eine geeignete und weitverbreitete Technologie zur Verteilung von Programmobjekten über das WWW (vgl. Abschnitt 3.5.2). Mit Schnittstellen zu Datenbanken (*Java Database Connectivity*, JDBC), der Anbindung an CORBA Systeme und entfernten Methodenaufrufen (RMI) stehen Kommunikationsmechanismen zur Realisierung verteilter Anwendungen zur Verfügung. Die jeweils verwendeten Schnittstellen und Protokolle zur Datenübertragung sind in Abbildung 7.3 dargestellt.

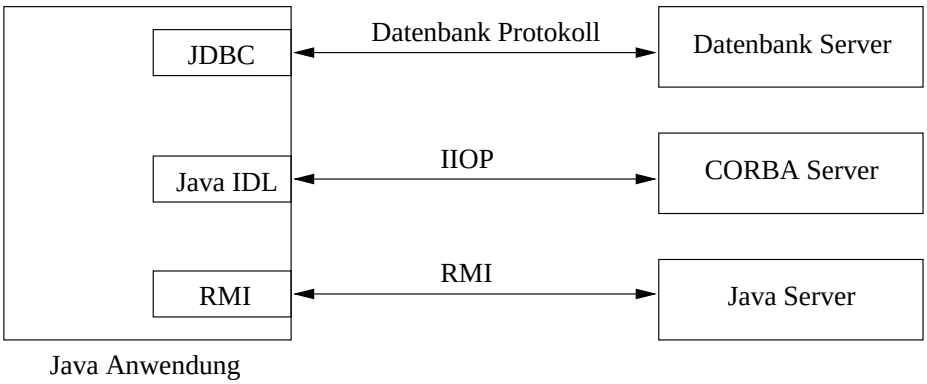

Abbildung 7.3: Kommunikationsmechanismen in Java Anwendungen

Während JDBC[2] auf die Kommunikation zwischen SQL Datenbanken und Java Anwendungen beschränkt ist und RMI (vgl. Abschnitt 7.3.2) die Kommunikation zwischen beliebigen Java Anwendungen übernehmen kann, bietet die CORBA (vgl. Abschnitt 7.3.1) Anbindung durch ihre Sprachunabhängigkeit und Verfügbarkeit auf diversen Plattformen den weitreichendsten Ansatz. Im Gegensatz zu den Java Ansätzen, die auf plattformunabhängigem Bytecode basieren, stellt das *Distributed Component Object Model* (DCOM,

[2] JDBC wird in dieser Arbeit nicht näher betrachtet, da hier allgemeine Kommunikationsmechanismen von Interesse sind, die nicht an einen bestimmten Typ von Server gebunden sind. Interessierte Leser finden weitere Informationen in [Kramer 97; Hamann 96; Orfali, Harkey 97].

vgl. Abschnitt 7.3.3) einen sprachunabhängigen Kommunikationsmechanismus für verteilte Binärcodeobjekte zur Verfügung.

7.3.1 CORBA

Die *Common Request Broker Architecure* (CORBA) ist ein Standard der *Object Management Group* (OMG) zur Kommunikation zwischen heterogenen Softwarekomponenten. Diese Komponenten können sich auf verschiedenen Rechnern befinden, in verschiedenen Programmiersprachen und auf unterschiedlichen Betriebssystemen implementiert sein. Die Spezifikation von CORBA [OMG 95] definiert neben einem grundlegenden Objektmodell, eine Schnittstellenbeschreibungssprache (IDL), in der die Schnittstelle zwischen Client und Server definiert wird (vgl. Abschnitt 7.1), die verschiedenen Bestandteile eines *Object Request Broker* (ORB) inklusive der jeweiligen Anwendungsprogrammierschnittstellen (API), Protokolle für die Interoperabilität zwischen verschiedenen Implementationen unterschiedlicher Hersteller von ORBs sowie Abbildungen der IDL auf einige Programmiersprachen (u. a. C, C++ und Smalltalk). An einer standardisierten Abbildung der IDL auf die Programmiersprache Java wird noch gearbeitet (Stand Juli 1997). Einige Hersteller von ORBs bieten jedoch bereits IDL Compiler für Java an.

Die Bestandteile eines ORB

Ein *Object Request Broker* (ORB) ist der Vermittler von Methodenaufrufen zwischen entfernten Objekten. Er ist für die Suche des Objektes, das die erforderlichen Dienste erfüllt, für die Weitergabe der Parameter, den Aufruf der Methoden und die Rückgabe von Ergebnissen zuständig [OMG 95]. In Abbildung 7.4 (aus [Orfali, Harkey 97]) sind die Bestandteile eines ORB dargestellt:

Client IDL Stub: Ein Client IDL Stub entsteht aus der Übersetzung einer in IDL abgefaßten Schnittstellenbeschreibung in die jeweilige Zielprogrammiersprache (z.B. C, C++, Smalltalk, Java, etc.). Der dabei entstehende Stub definiert, in welcher Weise ein Client in der Schnittstelle definierte Dienste einer Objektimplementation in der von ihm gewählten Programmiersprache aufruft. Der Stub ist ein lokales Stellvertreterobjekt für das (evtl. entfernte) Serverobjekt und übernimmt die Serialisierung und Deserialisierung von Parametern. Der Client IDL Stub wird zum Übersetzungszeitpunkt des Clientprogramms eingebunden.

Dynamic Invocation Interface (DII): Mit Hilfe des DII ist es für den Client möglich, Objekte zu benutzen, die erst nach der Übersetzung des Clients erstellt werden. CORBA definiert Standardprogrammierschnittstellen, die das Retrieval von Serverschnittstellen, die Generierung von Parametern und den dynamischen Aufruf eines entfernten Objektes erlauben.

Interface Repository: Im Interface Repository werden IDL Definitionen wie in einer Datenbank persistent abgelegt. So können Clients zur Laufzeit für beliebige Objekte

Informationen über den Aufbau der zugehörigen Schnittstellen einholen, unabhängig davon, ob diese bei der Entwicklung des Clientprogramms bereits bekannt waren.

ORB Interface: Das ORB Interface stellt lokale Dienste des ORB zur Verfügung wie z.B. die Konvertierung einer Objektreferenz in eine Zeichenkette und umgekehrt, diese Operation kann für die persistente Speicherung von Objektreferenzen nützlich sein.

Server IDL Skeleton: Der Server IDL Skeleton stellt statische Schnittstellen zu einem von einem Server exportierten Dienst bereit. Ein Skeleton wird wie ein Stub mit Hilfe eines IDL Compilers erzeugt.

Dynamic Skeleton Interface (DSI): Das DSI erlaubt als Pendant zum DII die Bereitstellung von Objektimplementationen für Schnittstellen, die zur Übersetzungszeit des Servers noch unbekannt sind. Dies ist z.B. der Fall, wenn es sich bei dem Server um einen Interpreter handelt, der während seiner Laufzeit Programme einliest und so erst von neuen Objektklassen, deren Schnittstellen und Implementationen erfährt.

Object Adapter (OA): Der OA übernimmt die Vermittlung zwischen ORB und Objektimplementation, er generiert und interpretiert Objektreferenzen, gibt Methodenaufrufe weiter, aktiviert und deaktiviert Objekte. Er registriert außerdem die von ihm unterstützten Klassen und ihre Instanzen im Implementation Repository.

Implementation Repository: Im Implementation Repository werden die von einem Server unterstützten Klassen sowie deren Instanzen abgelegt. Es ermöglicht dem ORB die Lokalisierung und Aktivierung von Objekten.

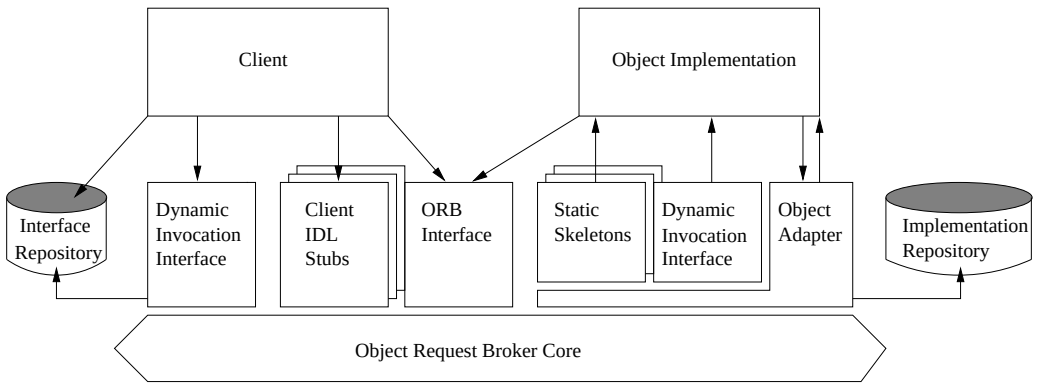

Abbildung 7.4: Die Bestandteile eines *Object Request Broker* (ORB)

CORBA definiert eine Architektur und eine Reihe von Anwendungsprogrammierschnittstellen, die ein ORB bereitstellen muß. Die Implementation der Architektur und der APIs ist herstellerspezifisch. ORBs verschiedener Hersteller können nicht ohne zusätzliche Vereinbarungen miteinander kommunizieren. Um Objekte über die Grenzen eines ORBs hinaus

nutzen zu können, sind Protokolle für die Kommunikation zwischen den ORBs verschiedener Hersteller erforderlich.

Interoperabilität zwischen verschiedenen ORBs

Im Zusammenhang mit dem WWW ist das *Internet inter-ORB Protocol* (IIOP) von besonderem Interesse. IIOP ist eine konkrete Realisierung des abstrakten *General Inter-ORB Protocol* (GIOP), das eine standardisierte Transfersyntax und eine Menge von Nachrichtenformaten für die Kommunikation zwischen ORBs spezifiziert. Genau wie die CORBA IDL nicht benutzt werden kann, um komplette Anwendungen zu programmieren und eine Abbildung auf eine konkrete Programmiersprache erfordert, erfordert das GIOP eine Abbildung auf ein konkretes Transportprotokoll. Das IIOP spezifiziert, wie GIOP Nachrichten über TCP/IP Verbindungen ausgetauscht werden.

Objektdienste

Die CORBA Spezifikation ist Basis der *Object Management Architecture* (OMA), die ebenfalls von der OMG entwickelt wird. In dieser Architektur werden zusätzlich zu der Transport- und Übertragungskomponente ORB unter anderem standardisierte Objektdienste zur Verfügung gestellt, die über in IDL spezifizierte Schnittstellen verfügen. Es gibt u.a. Dienste, die die Speicherung von Objekten auf verschiedenen Medien unterstützen, die Abrechnungsmechanismen für die Nutzung verteilter Objekte anbieten, für das Transaktionsmanagement, etc. [OMG 97d].

WWW Integration

Die Integration von CORBA Anwendungen und dem WWW kann mit Hilfe der Programmiersprache Java erfolgen. Java kann dynamisch Code von einem Server aus dem Netzwerk laden. So ist es nicht nötig, Komponenten einer CORBA Anwendung vor der Ausführung lokal zu installieren, da Java neben der Anwendung selbst auch den Client IDL Stub sowie den ORB[3] über das WWW dynamisch zum Client übertragen kann (vgl. Abbildung 7.5).
Nachdem Applet, Client IDL Stub und evtl. der ORB zum Client übertragen wurden, steht neben dem klassischen Zugang zu externen Anwendungen über HTTP und CGI mit dem IIOP ein weiteres speziell für verteilte Anwendungen entworfenes Protokoll für Java Applets zur Verfügung. Die Kommunikation von einem CORBA Server zu bereits vorhandenen Systemen (*legacy systems*) wie z.B. Datenbanken oder Transaktionsmonitoren übernimmt wiederum der ORB bzw. IIOP.
CORBA und IIOP fügen sich aufgrund ihrer offenen Standardisierung in die offenen Standards des WWW ein, während die im Folgenden betrachteten Ansätze RMI und DCOM den Entwickler verteilter Anwendungen in plattform- oder programmiersprachenabhängige

[3] Netscape integriert in seine kommende Version des Navigator einen ORB, so daß dieser bei Verwendung des gleichen ORB zur Erstellung der CORBA Clientanwendung nicht mehr vom WWW Server geladen werden muß.

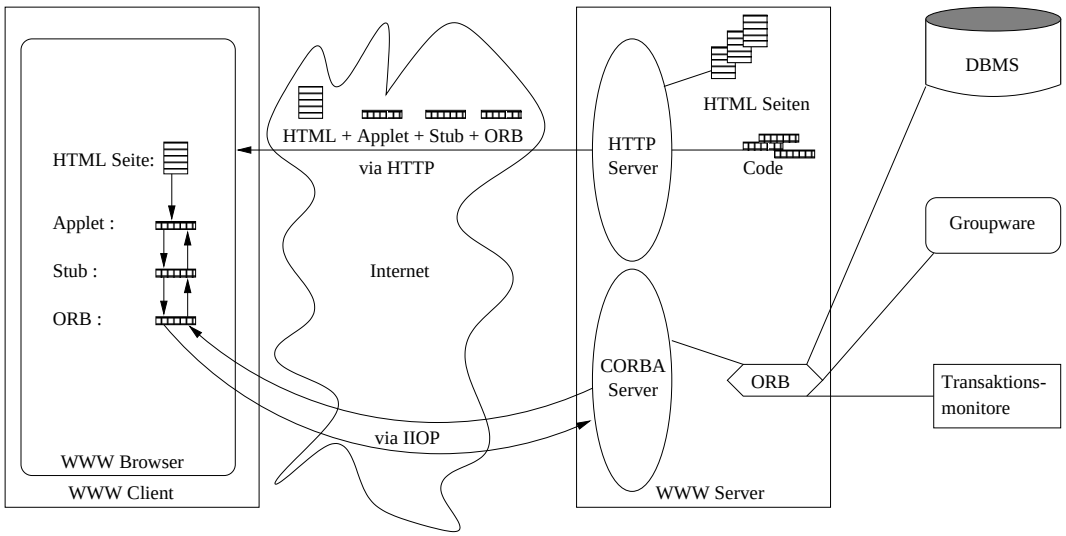

Abbildung 7.5: Integration von CORBA und WWW

Abhängigkeiten führen können, deren Verhinderung von vornherein das erklärte Ziel von WWW und CORBA war.

7.3.2 Java RMI

Die Version 1.1 des Java API enthält mit der *Remote Method Invocation* (RMI) Klassenbibliothek die Fähigkeit, Java Objekte auf entfernten Rechnern zu referenzieren und deren Methoden aufzurufen. Es handelt sich bei RMI um einen Mechanismus, der die Semantik von entfernten Prozeduraufrufen (*Remote Procedure Call*, RPC) auf Objektaufrufe in verteilten Objektsystemen abbildet [Sun Microsystems 96].
Eine in einem Java *Interface* definierte Schnittstelle beschreibt die Methoden eines entfernten Objektes, die von einem Client aufgerufen werden dürfen. Diese Schnittstelle wird in einer oder mehreren Klassen implementiert. Nach der Übersetzung dieser Klassen mit einem speziellen RMI Compiler steht dem Client ein Stellvertreterobjekt (*Stub*) zur Verfügung, das das entfernte Objekt auf dem Clientrechner lokal abbildet und alle Methodenaufrufe an die Objektimplementation auf dem Server weiterleitet. Neben dem Stub wird aus der Serverimplementation ein *Skeleton* generiert, das die Methodenaufrufe des Client auf dem Server an die Objektimplementation weiterleitet und Berechnungsergebnisse an den Client zurückgibt. Stub und Skeleton tauschen hierbei Objekte mit Hilfe der *Object Serialization Specification* (OSS) aus, die das Serialisieren von Java Objekten ermöglicht [Hendrich 97]. Neben dem Austausch von Objekten zwischen Client und Server können auch Objektreferenzen als Parameter in Methodenaufrufen übergeben werden.
Das Auffinden einer Objektimplementation übernimmt ein nicht persistenter Namensdienst:

die *Registry*. Dabei handelt es sich um eine Laufzeitdatenbank, die nach ihrem Start mit den Namen und Adressen von Objekten gefüllt wird. Wird der Registry-Prozeß beendet, gehen sämtliche darin vorhandenen Einträge verloren. Dieser Dienst erlaubt die Registrierung und das Retrieval von Serverobjekten anhand von Namen, z.B. erfolgt die Registrierung eines Objektes "this" unter dem Namen "my object" folgendermaßen:

java.rmi.Naming.rebind("my object", this);

Der Client benutzt ein URL Schema, um auf dieses Objekt zuzugreifen:

ObjRMI myObject = (ObjRMI) java.rmi.Naming.lookup("rmi://www.foo.bar/my object");

Die Registry ist als ein RMI Serverobjekt auf dem gleichen Rechner aktiv, wie die Objekte, die es verwaltet (vgl. Abbildung 7.6, (1)). Es verfügt nicht über Beschreibungen der verwalteten Objekte, sondern gibt lediglich dem Client anhand einer URL eine Referenz auf den Skeleton (3) der Implementation, der enfernte Methodenaufrufe ermöglicht. Der Client ruft dann über seinen Stub (2) die entfernten Objekte auf. Mehrere Registry Objekte können parallel auf einem Rechner aktiv sein. Die RMI Klassenbibliothek enthält weitere Klassen, die es dem Client und dem Server ermöglichen, eine Referenz auf Registry Objekte zu erhalten.

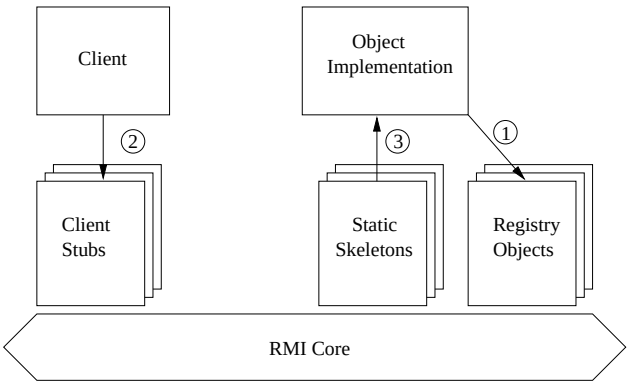

Abbildung 7.6: Entfernte Methodenaufrufe in Java

Die Beseitigung nicht mehr referenzierter Objekte durch Freigabe des von ihnen belegten Speicherplatzes (engl. *Garbage Collection*) wird bei entfernten wie lokalen Objekten automatisch und für den Programmierer transparent durchgeführt. In verteilten Sytemen stellt sich immer die Frage, wann ein Serverobjekt entfernt werden darf. RMI benutzt zur Beantwortung dieser Frage einen Zähler, der die aktiven Referenzen in Form von Client/Server Verbindungen in TCP/IP Sitzungen überwacht. Solange eine Referenz zu einem entfernten Objekt besteht, kann dieses nicht entfernt werden.

WWW Integration

Im WWW erlaubt RMI die Kommunikation von Java Applets und entfernten Java Serverprogrammen. Die Integration des RMI in das *Java Development Kit* (JDK) ab der Version 1.1 wird das zusätzliche Laden der RMI Funktionalität bei der Benutzung von Applets überflüssig machen, sobald das JDK 1.1 von den verbreiteten Browsern unterstützt wird (vgl. Abbildung 7.7).

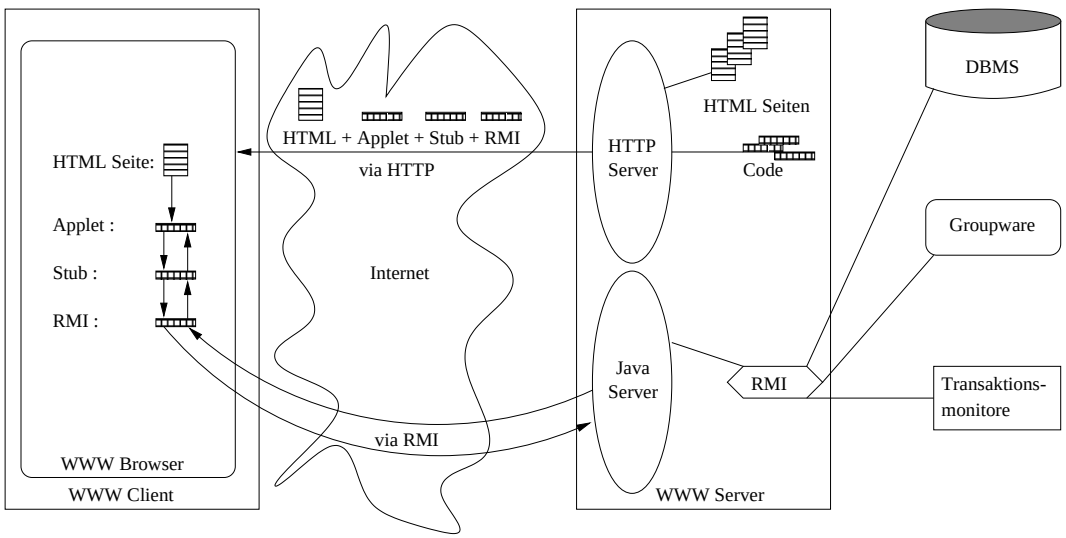

Abbildung 7.7: Integration von Java RMI und WWW

RMI erfordert auf der Serverseite die Kapselung von bereits vorhandenen Anwendungen (*legacy systems*) in Java Code. Im Gegensatz dazu erlaubt CORBA die Kommunikation verteilter Objekte unabhängig von der verwendeten Programmiersprache.

7.3.3 DCOM

Komponententechnologie ist insbesondere auf Microsoft Windows Betriebssystemen in Form von *OCX Controls* weit verbreitet und erlaubt dort die Wiederverwendung binären Codes in integrierten Entwicklungsumgebungen wie z.B. Microsoft VisualBasic und Anwendungsprogrammen wie z.B. Microsoft Word. ActiveX ist der Nachfolger der OCX Technologie und erlaubt die Verteilung von Softwarekomponenten über das WWW (vgl. Abschnitt 3.5.2). ActiveX basiert auf dem *Distributed Component Object Model* (DCOM).
DCOM ist die Antwort der Firma Microsoft auf die Frage nach einem Mechanismus für die Vermittlung von Funktionsaufrufen zwischen verteilten binären Objekten. DCOM ist eine Erweiterung des *Component Object Model* (COM) [Microsoft Corp. 97e]. COM definiert, wie binäre Softwarekomponenten und ihre Klienten miteinander auf einem einzelnen Rechner kommunizieren können. DCOM erweitert COM um die Möglichkeit zur Kommunikation

zwischen Komponenten auf verschiedenen Rechnern und basiert auf dem Konzept entfernter Prozeduraufrufe (vgl. Abschnitt 7.1). DCOM definiert eine eigene Objektterminologie, die sich grundlegend von klassischen Objektmodellen unterscheidet, wie sie von CORBA und RMI verwendet werden.

DCOM Objekte

Wie in CORBA und RMI werden Objektschnittstellen von ihrer Implementation getrennt. Dies geschieht mit einer Schnittstellenbeschreibungssprache (IDL), die nicht zur CORBA IDL kompatibel ist. Ein DCOM Objekt — auch als ActiveX bezeichnet — ist eine plattformabhängige Softwarekomponente, die eine oder mehrere DCOM Schnittstellen unterstützt. Eine DCOM *Schnittstelle* ist eine Sammlung von Methoden. Um auf eine solche Schnittstelle zuzugreifen, benutzen Clients Zeiger auf ein Feld von Funktionszeigern. Dieses Feld wird als virtuelle Tabelle (*vtable*) bezeichnet. Die in der virtuellen Tabelle enthaltenen Zeiger referenzieren die Implementation der jeweiligen Methoden (vgl. Abbildung 7.8).

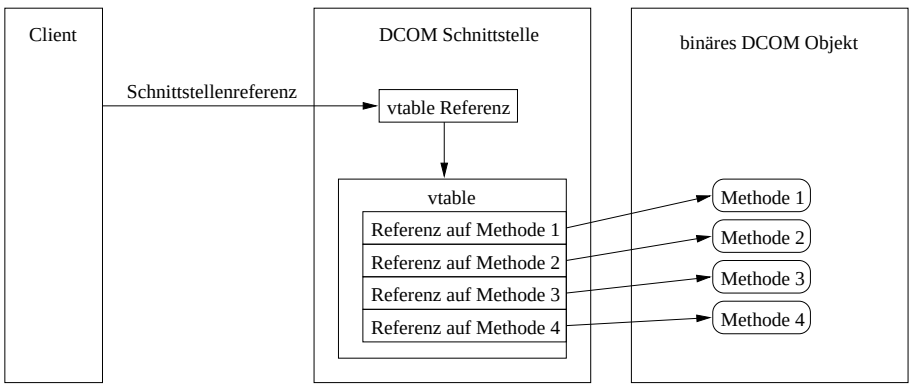

Abbildung 7.8: Struktur eines DCOM Objektes

DCOM Schnittstellen sind Kontrakte zwischen Client und Server, sie kapseln ein Objekt und bieten Clients die Möglichkeit, über einen auf der Tabelle von Zeigern basierenden API auf das Objekt zuzugreifen. Jede Schnittstelle wird zur Laufzeit über einen global eindeutigen Schnittstellenbezeichner (IID) in Form einer 128-Bit Zahl referenziert, die mit Hilfe einer Funktion des DCOM API erzeugt wird. Diese Funktion benutzt das aktuelle Datum, die aktuelle Zeit, die ID der Netzwerkkarte und einen Hochfrequenzzähler zur Erzeugung einer global eindeutigen 128-Bit Zahl.
Ein DCOM Objekt ist eine Laufzeitinstanz seiner Klasse. Eine DCOM Klasse implementiert eine oder mehrere Schnittstellen und wird ebenfalls durch eine 128-Bit Zahl (*CLSID*) global eindeutig identifiziert. In DCOM sind Objekte nicht von ihren Klassen zu unterscheiden, da ein Client mit einem Objekt nur über dessen IIDs interagiert. Es ist für einen Client nicht möglich, eine spezifische Instanz eines DCOM Objektes zu referenzieren, da DCOM keine

eindeutigen Objektreferenzen unterstützt. Ein DCOM Objekt ist kein Objekt im klassischen objektorientierten Sinn, das über einen bestimmten internen Zustand verfügt, sondern eine Sammlung von semantisch zusammengehörigen Funktionen.

DCOM Server

Ein DCOM Server liegt entweder in Form einer dynamisch bindbaren Bibliothek (*Dynamic Link Library*, DLL) oder einer ausführbaren Datei vor und enthält eine oder mehrere DCOM Klassen, die jeweils durch ihre CLSID bezeichnet sind. Server werden in der Systemregistrierung (*Registry*) der Microsoft Windows Betriebssysteme mit ihrer CLSID und dem Pfad des Programmes eingetragen, das diesen Server implementiert. Bei der Anfrage eines Client sucht DCOM nach der CLSID des Servers in der Systemregistrierung und ruft das entsprechende Programm auf. Der Server ist dann für die Erzeugung eines Objektes der gewünschten Klasse verantwortlich. Es werden drei Arten von DCOM Servern unterschieden:

Integrierte Prozeß Server werden im gleichen Prozeßraum wie ihre Clients ausgeführt. Clients können ohne intermediäre Systemkomponenten auf andere Komponenten im gleichen Prozeß zugreifen. Unter den Betriebssystemen Microsoft Windows und Microsoft Windows NT werden diese Komponenten als DLLs direkt in den Prozeß des Client integriert.

Lokale Server werden auf dem gleichen Rechner in einem von ihren Clients separaten Prozeßraum ausgeführt. Die Kommunikation zwischen Komponenten in unterschiedlichen Prozessen auf dem gleichen Rechner und Betriebssystem erfolgt über den sogenannten *Lightweight Remote Procedure Call* (LRPC) Mechanismus. Lokale Prozeß Server werden durch ihre eigene (.EXE) Datei gestartet.

Entfernte Server werden in einem separaten Prozeß auf einer entfernten Maschine und evtl. einem anderen Betriebssystem ausgeführt als ihr Client. Clients benutzen einen DCE[4] ähnlichen RPC Mechanismus, um mit entfernten Servern zu kommunizieren. Entfernte Server liegen wie Lokale Server in einer eigenen ausführbaren Datei vor.

In Abbildung 7.9 (aus [Orfali, Harkey 97]) sind die verschiedenen Kommunikationsmöglichkeiten für DCOM Objekte dargestellt. Die schwarzen Punkte, die jeweils mit den Serverobjekten verbunden sind, stellen dabei deren DCOM Schnittstellen dar. Der Client sieht bei jeder der o. g. Serverarten nur eine Schnittstellenreferenz, es ist für ihn nicht ersichtlich, wo sich ein bestimmtes Objekt befindet. Proxy Objekte auf der Client Seite und Stubs auf der Server Seite sind mit den CORBA Client IDL Stubs bzw. den Server IDL Skeletons vergleichbar: sie erlauben statische Aufrufe von — evtl. entfernten — DCOM Objekten.

Dynamische Aufrufe von DCOM Objekten

Neben statischen Aufrufen von Objekten über Proxy und Stub Objekte ist es möglich, Methodenaufrufe dynamisch zur Laufzeit zu erzeugen und auszuführen. Diese Techniken

[4] *Distributed Computing Environment* [Open Group 97a]

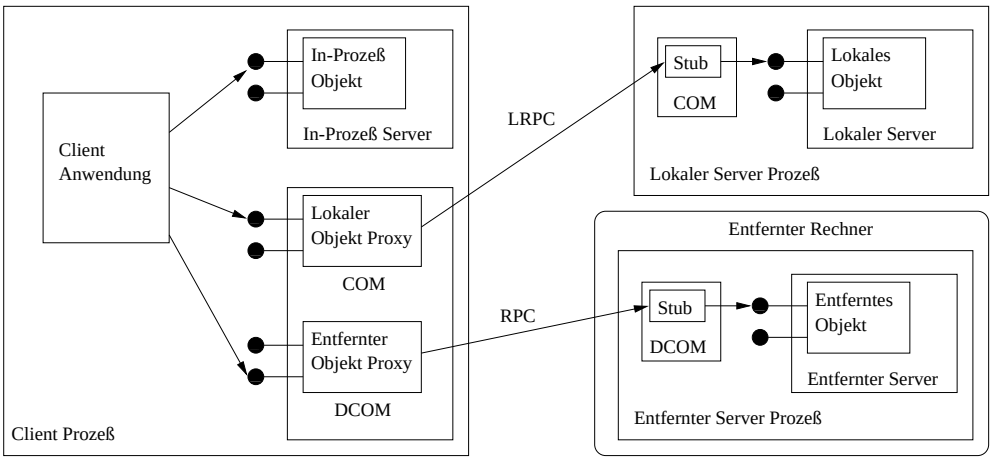

Abbildung 7.9: Verschiedene DCOM Server

bilden die Grundlage für die OLE[5] Automation, die es Clients mit Hilfe von Skriptsprachen erlaubt, dynamisch Methoden aufzurufen, die die Inhalte von Objekten manipulieren. OLE Automation erlaubt es einem Programm — dem OLE *Automation Controller* — ein anderes Programm — den OLE *Automation Server* — zu steuern [Lassesen 95]. Ein OLE *Automation Server* stellt in einer Laufzeitdatenbank (*Type Library*), die mit dem CORBA Interface Repository vergleichbar ist, Beschreibungen seiner Schnittstellen, Methoden und Eigenschaften[6] zur Verfügung (vgl. Abbildung 7.10).

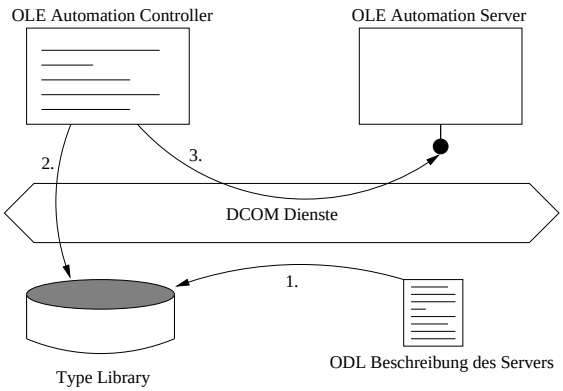

Abbildung 7.10: OLE Automation Clients und ihre Server

Aus der Beschreibung der Serverobjekte — abgefaßt in der DCOM *Object Definition Lan-*

[5] *Object Linking and Embedding* [Brockschmidt 96]
[6] Eigenschaften sind mit Attributen im klassischen Objektmodell vergleichbar.

guage (ODL), einer Untermenge der DCOM IDL — werden die entsprechenden Metadaten erzeugt und in einer Type Library abgelegt (1). Ein Client eines Automation Server wird als *Automation Controller* bezeichnet. Er kann die Namen von unterstützten Methoden und Eigenschaften eines Servers mit Hilfe der Type Library bestimmen (2) und nachfolgend Server Objekte erzeugen, deren Methoden aufrufen sowie deren Eigenschaften setzen und abrufen (3).

WWW Integration

Ein ActiveX ist ein DCOM Objekt [Orfali, Harkey 97]. ActiveX Komponenten sind sowohl beim WWW Client als auch beim WWW Server einsetzbar, sofern sie für die jeweils vorhandene Plattform übersetzt wurden. Dies hat den Vorteil des gegenüber Java Applets/Servlets schnelleren Laufzeitverhaltens, birgt aber auch die Gefahr der mißbräuchlichen Nutzung, da ActiveX Programme uneingeschränkten Systemzugang haben. Sie dürfen u. a. externe Anwendungen starten und können diese per Skriptsprache fernsteuern. In [Donnerhacke, Peter 97] wird gezeigt, wie eine per WWW zu einem Clientrechner übertragene ActiveX Komponente automatisch Überweisungsaufträge im Namen des Benutzers ausführt.
ActiveX Komponenten können ähnlich wie Java Applets in HTML Seiten eingebunden und so über das Internet verteilt werden. Auf der Serverseite können ActiveX Komponenten neben ihrem Einsatz als herkömmliche DCOM Server auch als Dienstleister für einen *Active Server* genutzt werden [Cluts 97]. Ein Active Server ist in der Lage, in HTML Seiten enthaltene Skripte auszuführen. Diese erweiterten HTML Seiten werden als *Active Server Pages* bezeichnet [Behme 97]. In den *Active Server Pages* kann neben VBScript sowohl JScript als auch JavaScript als Skriptsprache benutzt werden. Diese Skripte können Instanzen von ActiveX Server Komponenten (auch als OLE Automation Server bezeichnet, s. o.) erzeugen und diese mit einer bestimmten Aufgabe wie z.B. einer Datenbankanfrage betrauen. Über die Active Server Pages lassen sich also HTML Seiten mit bereits bestehenden Anwendungen verbinden. Die Ausgabe einer Active Server Page ist reines HTML. Die sich durch Active Server Pages ergebenden Möglichkeiten sind mit den Fähigkeiten von CGI Anwendungen vergleichbar, mit dem Unterschied, daß der Serverprozeß selbst die Auswertung von Parametern übernimmt und, falls eine betroffene ActiveX Server Komponente als DLL vorliegt, er die komplette Berechnung einer Anfrage übernehmen kann, ohne einen externen Prozeß starten zu müssen.
Die Übertragung von ActiveX Komponenten zum Client via HTTP und deren Kommunikation via DCOM mit einem entfernten DCOM Server ist in Abbildung 7.11 dargestellt.

7.4 Vergleich der Technologien

Auf Basis der Anforderungen aus Abschnitt 7.2 werden im Folgenden die oben vorgestellten Technologien gegenübergestellt.

Behandlung der Heterogenität von Plattformen: CORBA konforme ORBs werden von verschiedenen Herstellern für unterschiedliche Plattformen und Betriebssysteme

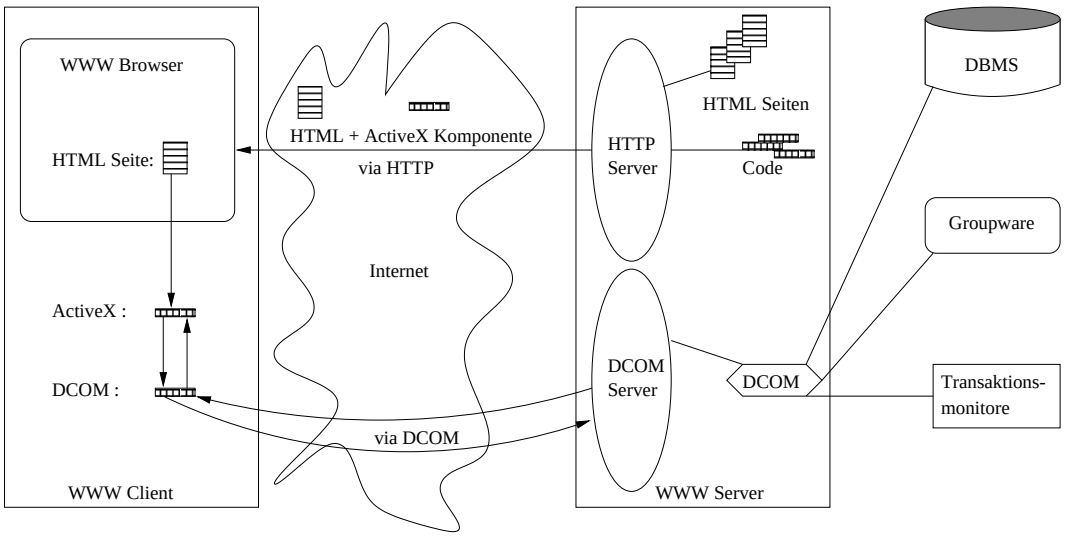

Abbildung 7.11: Integration von DCOM und WWW

angeboten. Über das IIOP können ORBs verschiedener Hersteller miteinander kommunizieren. RMI steht auf allen Plattformen zur Verfügung, für die ein Java Interpreter und die RMI Bibliotheken existieren. DCOM unterstützt derzeit nur die Betriebssysteme Windows 95 und Windows NT, eine Beta Version für Unix ist von der Software AG verfügbar.

Behandlung der Heterogenität von Programmiersprachen: Mit Hilfe von Schnittstellenbeschreibungen in einer speziellen Schnittstellenbeschreibungssprache (IDL) trennen CORBA und DCOM Schnittstellen von Objekten von deren Implementierung. Ein IDL Compiler erzeugt aus den Beschreibungen Stubs und Skeletons in Programmiersprachen, für die Übersetzungsvorschriften (*language mappings*) bestehen. RMI ist ausschließlich für die Interaktion zwischen Java Softwarekomponenten entwickelt worden, daher werden Schnittstellen als Java *Interfaces* definiert.

Integration mit bestehender WWW Infrastruktur: RMI und CORBA sind im WWW mit Hilfe von Java Applets verteilbar, die von allen namhaften Browsern unterstützt werden. Java Applets können nicht direkt mit einer HTML Seite interagieren, sondern erhalten in einem Browser Fenster einen rechteckigen Bereich zugeteilt, den sie dann selbst verwalten. Die Interaktion mit der umgebenden HTML Seite kann über JavaScript erfolgen (vgl. Abschnitt 3.5.2). DCOM Objekte sind als ActiveX Binärcodeobjekte derzeit ausschließlich im Microsoft Internet Explorer und im Netscape Navigator auf der 32-Bit Microsoft Windows Plattform verwendbar. Die Verbindung zu HTML Seiten wird ebenfalls über Skriptsprachen hergestellt.

Lokale Sicherheit des Client im WWW: Während CORBA und RMI Anwendungen im WWW von dem Sicherheitskonzept der in der Programmiersprache Java verfaßten Applets profitieren, unterliegen ActiveX Objekte keinerlei Sicherheitsbeschränkungen und können beliebige Festplattenzugriffe durchführen, andere Anwendungen starten und beliebige Netzwerkverbindungen aufbauen.

Entwicklungsunterstützung: Alle drei betrachteten Technologien erzeugen werkzeuggestützt Stubs und Skeletons. Die Entwicklungsunterstützung von CORBA ORBs hängt von der jeweiligen Herstellerimplementation ab. In der Regel stehen Generatoren für Stubs und Skeletons und einige zusätzliche Systemdienste zur Verfügung. Java bietet mit seinen Klassenbibliotheken eine stetig wachsende Anzahl von zusätzlichen Diensten für RMI Anwendungen. Interessant zu bemerken ist, daß der Transaktionsdienst von Java (JTS) auf dem *Object Transaction Service* der OMG basiert [Curtis 97]. DCOM verfügt nicht wie CORBA in der *Object Management Architecture* über eine begleitende Definition von Systemdiensten.

Dynamisches Finden und Aufrufen von Objekten: CORBA und DCOM unterstützen das dynamische Finden und Aufrufen von Objekten mit Hilfe von Laufzeitdatenbanken, die Metadaten über verfügbare Objekte enthalten. Ein Client kann so passende Objekte finden, sich über deren Methoden und die zum Aufruf notwendigen Parameter informieren und dynamisch eine Anfrage an das Objekt stellen. RMI bietet hier lediglich die Möglichkeit des dynamischen Ladens von Stubs.

Die drei betrachteten Technologien spielen ihre Vorzüge jeweils in bestimmten Anwendungsfällen aus. So bietet Java RMI die nahezu nahtlos integrierte Verwendung von entfernten Java Objekten. Der Entwickler muß keine spezielle IDL Syntax lernen, er kann serialisierbare Java Objekte als Parameter von entfernten Methodenaufrufen verwenden, er nutzt die automatische Speicherbereinigung von nicht referenzierten entfernten Objekten, etc. Wenn es aber darum geht, Objekte anderer Programmiersprachen einzubinden, führt RMI schnell zur Abhängigkeit von der Programmiersprache Java, da für ein entferntes Objekt in einer anderen Programmiersprache ein Java Objekt programmiert werden muß, das dessen Funktionalität kapselt. Der Entwickler macht sich also von einer Programmiersprache — in diesem Falle von Java — abhängig, um ihren Kommunikationsmechanismus zu nutzen [Curtis 97].
CORBA bietet eine umfassende Integrationstechnologie für die Erstellung von objektorientierten, verteilten Anwendungen in einem heterogenen Netzwerk. Die Vorteile dieses offenen Standards erkauft sich der Entwickler mit dem Aufwand des Erlernens einer speziellen Schnittstellenbeschreibungssprache. Desweiteren ist die Übertragung von Parametern in entfernten Methodenaufrufen auf Objektreferenzen und Standarddatentypen wie z.B. Integer, Strings, Arrays, etc. beschränkt. An einer Erweiterung der CORBA Spezifikation zur Übergabe von Objekten als Parameter wird gearbeitet.
COM hat mit den auf den Microsoft Windows Plattformen weit verbreiteten OLE/OCX Kontrollelementen einen offenen Markt für plattformabhängige Softwarekomponententechnologie geschaffen. Diese Komponenten können unabhängig von Hersteller und verwendeter

Programmiersprache auf einem Rechner miteinander kooperieren. DCOM soll die Kooperation auf im Netzwerk verteilte Komponenten ausdehnen. Da jedoch Portierungen von DCOM auf andere Plattformen erst in Beta Versionen vorhanden sind, ist der Markt de facto auf die Intel/Windows Plattform beschränkt. Zudem erfordert die Verteilung von ActiveX Komponenten im WWW auf heterogene Plattformen entweder eine jeweils plattformspezifische Version der Komponente oder einen Browser, der 32-Bit Windows Code ausführen kann. Die beschränkten Sicherheitsmechanismen von ActiveX Komponenten im Vergleich zu Java Applets bilden im WWW bei der Ausführung unbekannten Codes ein hohes Sicherheitsrisiko für den Anwender.

8. Entwicklungen des WWW

In Kapitel 4 wurde gezeigt, daß das WWW noch nicht alle Anforderungen an verteilte Informationssysteme erfüllen kann. In diesem Kapitel werden Entwicklungen von WWW Konzepten und Technologien vorgestellt, die einige der festgestellten Defizite ausgleichen.
In Abschnitt 8.1 werden verschiedene Weiterentwicklungen und Ergänzungen von HTML beschrieben. Java ist durch ihre Plattformunabhängigkeit und die Unterstützung sicherer Codeausführung die zukunftsträchtigste Programmiersprache des WWW. Weiterentwicklungen von Java werden in Abschnitt 8.2 vorgestellt. Die Basis jedes Internetinformationssystems bildet das Internet Protokoll. Ein neues Internet Protokoll bringt u.a. neue Fähigkeiten und Möglichkeiten für das WWW (vgl. Abschnitt 8.3).

8.1 Entwicklungen und Erweiterungen von HTML

HTML ist neben HTTP einer der Grundbausteine des WWW. Neben dem Entwurf der Spezifikation von HTML 4.0 (vgl. Abschnitt 8.1.1) und der HTML Erweiterung mSQL (vgl. Abschnitt 8.1.2) wird in diesem Abschnitt die *Extensible Markup Language* (XML) (vgl. Abschnitt 8.1.3) vorgestellt.

8.1.1 HTML 4.0

HTML 4.0 ist ein Vorschlag des WWW Konsortiums (W3C) zur Standardisierung von HTML in einer neuen Version [W3C 97]. Dies ist erforderlich, um das Voranschreiten der Nutzung proprietärer HTML Konstrukte durch die Browserhersteller zu verhindern. Viele der Änderungen sind lediglich Festschreibungen von bereits von Netscape oder Microsoft in ihren Browsern implementierten proprietären Erweiterungen von HTML.
HTML 4.0 bietet u.a. folgende neue Fähigkeiten bzw. Erweiterungen:

Objektintegration: Das Schlüsselwort *Object* kennzeichnet jede Art von Objekt, das in eine HTML Seite eingefügt werden kann. Dabei kann es sich z.B. um Bilder, Audio, Video, mathematische Formeln oder Anwendungen handeln. Die spezielleren Schlüsselworte *IMG* für das Einfügen von Bildern und *Applet* für das Einbinden von Java Applets sind damit obsolet.

Unterstützung von *Style Sheets*: *Cascaded Style Sheets* erlauben die Festlegung der Darstellung (z.B. Schriftgröße und Schriftart) von mit HTML Schlüsselworten gekennzeichneten Informationen (vgl. Abschnitt 3.2).

Unterstützung von Skript Sprachen: HTML unterstützt die Einbindung von Clientskripten in beliebigen Sprachen. Dadurch wird es möglich, dynamische HTML Seiten zu gestalten, die individuell auf Benutzereingaben reagieren können, ohne dafür mit dem Server Kontakt aufzunehmen (vgl. Abschnitt 3.5.1).

Versionierungsunterstützung: Abschnitte in einem HTML Dokument können mit Hilfe der neuen Schlüsselworte *INS* und *DEL* als „eingefügt“ oder „gelöscht“ gekennzeichnet werden. In Zusammenhang mit dem Attribut *datetime*, das den Zeitpunkt des Einfügens oder Löschens eines Abschnittes kennzeichnet, können so mehrere Versionen eines Dokumentes in einer Datei gespeichert werden (vgl. Abschnitt 5.3.2).

Mit der Definition des Schlüsselwortes *Object* ist der Weg zu zusammengesetzten Dokumenten (engl. *compound documents*) bereits vorgegeben. Würde desweiteren die Möglichkeit für den Benutzer bestehen, aus diesen zusammengesetzten Dokumenten einzelne Objekte zu entnehmen, sie lokal zu editieren und dann wieder in das Dokument zurückzulegen (*drag and drop*) — dies könnte z.B. unter Microsoft Windows mit der OLE Technologie erfolgen — würde das Workflow Modell, bei dem ein zu bearbeitendes Objekt von einem zum nächsten Bearbeiter gereicht wird und jeder seine Änderungen daran vornimmt, um zur Erledigung eines Auftrages beizutragen, in greifbare Nähe rücken.

8.1.2 HTML-Erweiterung mSQL

Ein typisches Verfahren, um dynamische HTML Seiten zu erzeugen, ist die Anbindung einer Datenbank an einen WWW Server über die CGI Schnittstelle des Servers. Ein Skriptprogramm übernimmt bei einer Anfrage eines Client an den Server die Kommunikation mit der Datenbank, indem es die Verbindung zur Datenbank aufbaut, eine SQL Anfrage aus den vom Server gelieferten CGI Variblen erzeugt und an die Datenbank stellt und die von der Datenbank gelieferten Ergebnisse in Form von HTML Seiten an den Server zurückschickt, der sie dann zum Client überträgt. Dieses Vorgehen hat den Nachteil, daß bei Layoutveränderungen der erzeugten HTML Seiten das Skript geändert werden muß, d.h. Layoutänderungen erfordern Programmierkenntnisse in der verwendeten Skriptsprache.
Ein Beispiel für die Verlagerung der Datenbankaufrufe in HTML Seiten ist die WWW Schnittstelle zum Datenbankprogramm mSQL [Schmelzle, Gast 96]. Hierbei sorgt ein CGI Skript für die Verarbeitung von HTML Seiten, die neben normalem HTML Code auch spezielle mSQL Anweisungen enthalten. Das Skript nimmt als Argument eine HTML Seite und die vom Client eingegebenen Paramter, durchsucht die HTML Seite nach mSQL Anweisungen und führt sie gegebenenfalls mit den aktuellen Parametern aus. Eine Anfrage könnte z.B. lauten:

http://www.rechner.de/cgi-bin/w3-msql/namen.html?name=Bösch

Die Datei *namen.html* könnte z.B. so aussehen (nach [Schmelzle, Gast 96]):

```
<html>
<head>
<title>WWW mSQL Beispiel </title>
</head>
<body>
<! msql connect >
<! msql database beispiel >
<! msql query: "SELECT Name FROM adressen WHERE Name='$name': " ergebnis >
<ul>
<! msql print_rows ergebnis "<li> @ergebnis.0 ">
</ul>
<! msql free ergebnis>
<! msql close >
</body>
</html>
```

In diesem Beispiel werden mit den ersten beiden mSQL Kommandos die Verbindung zur Datenbank aufgebaut und danach alle Namen aus der Adreßtabelle herausgefiltert, die dem übergebenen Parameter *name* entsprechen. Die einzelnen Ergebnisse werden in einer unnumerierten Liste dargestellt. Danach wird der Speicherplatz für das Ergebnis der Anfrage wieder freigegeben und die Verbindung zur Datenbank geschlossen. Anfragen an eine Datenbank können jedes gültige mSQL Kommando enthalten, also z.B. auch: INSERT, UPDATE oder DELETE. Desweiteren können Abschnitte einer HTML Seite mit dem beliebig tief schachtelbaren *if-then-else* Konstrukt bedingt verwendet werden.
Am Beispiel von mSQL lassen sich Parallelen zwischen erweiterten HTML Dokumenten und dem Klassenkonzept in der Softwareentwicklung erkennen. Eine Instanz eines HTML/mSQL Dokumentes entsteht erst zur Laufzeit, indem die enthaltenen mSQL Anweisungen mit aktuellen Parametern versehen und ausgeführt werden. Die neue Instanz des Dokumentes wird dann mit aktuellen Daten aus der Datenbank gefüllt. Dies ist mit der Erzeugung eines neuen Objektes — das ebenfalls über Parameter initialisiert wird — als Instanz einer Klasse in der Softwareentwicklung vergleichbar.
mSQL ist nur ein Beispiel aus einer großen Menge proprietärer Erweiterungen von HTML zur Erzeugung dynamsicher HTML Seiten. Alle namhaften Datenbankhersteller bieten WWW Schnittstellen zum Zugriff auf ihre Datenbanken an. Ein ähnlicher Ansatz wird in Abschnitt 7.3.3 mit den *Active Server Pages* von Microsoft beschrieben.

8.1.3 XML

Die *Extensible Markup Language* (XML) ist eine Teilmenge von SGML [Macherius 97] und wie SGML eine Sprache zur Definition von Sprachen. Im Gegensatz zu HTML, das im Sinne von SGML eine Dokumenttypdefinition (*document type definition*, DTD) ist, erlaubt XML die Definition eigener DTDs.

Jedes XML Dokument besteht aus einer DTD und einer Instanz dieser DTD. Eine DTD ist eine formale Grammatik, die eine spezifische XML Sprache definiert. Eine DTD für einen Dokumententyp „mail“ könnte z.B. folgendermaßen aussehen:

```
<!DOCTYPE mail SYSTEM "mail.dtd">
<!ELEMENT mail (receiver, sender, date, subject, data)>
<!ELEMENT receiver (# PCDATA) >
<!ELEMENT sender (# PCDATA) >
<!ELEMENT date (# PCDATA) >
...
```

Das Layout der in der DTD definierten Schlüsselworte wird in *Style Sheets* definiert (vgl. Abschnitt 3.2). Ein Anwender verwendet diesen neuen Dokumenttyp, indem er auf die Definitionsdatei „mail.dtd“ verweist und dann die dort eingeführten Schlüsselworte wie in HTML verwendet, z.B.:

```
<!DOCTYPE mail SYSTEM "mail.dtd">
<mail>
  <receiver> "your.name@institute.edu" < /receiver>
  <sender> "my.name@institute.edu" < /sender>
  ...
</mail>
```

Mit Hilfe von XML sollen folgende Defizite von HTML ausgeglichen werden:

Erweiterbarkeit: Informationsanbieter können neue Markierungscodes und Attribute definieren. Es gibt z.B. bereits eine Anwendung von XML in Form einer *Chemical Markup Language* (CML) [Macherius 97]. CML enthält neben den HTML Schlüsselworten auch spezielle Notationsdefinitionen für Moleküle und Meßdaten. Ein CML Browser stellt die Moleküle auf der Client Seite als berechnete 3-D Grafik dar.

Strukturierung: Da HTML Tabellen (engl. *tables*) und Rahmen (engl. *frames*) nur eingeschränkte Möglichkeiten zur Strukturierung von Daten aus Datenbanken bieten, können in XML Strukturen beliebig tief geschachtelt werden.

Validierung: Sollen Informationen aus einer Datenbank auf einem WWW Server in ein lokales Informationssystem beim Client übernommen werden, bietet HTML keine Möglichkeit, die Daten auf strukturelle Gültigkeit zu überprüfen, bevor die empfangende Anwendung versucht, die Daten in das Zielsystem zu importieren. Bei XML müssen sich Client und Server auf eine gemeinsame DTD einigen und die Validierung der Daten kann bereits vor dem Import durch den XML Browser erfolgen.

XML ist nicht rückwärtskompatibel zu HTML und erfordert daher neue Browser. HTML wird durch das Aufkommen von XML nicht aus dem WWW verdrängt werden, da es bereits

weit verbreitet ist und noch keine populären XML Browser von der Funktionsvielfalt eines Netscape Navigator oder Microsoft Internet Explorer existieren. Für spezielle Anwendungsgebiete wie z.B. Verlage, Handelshäuser oder die Luft- und Automobilindustrie eignet sich XML jedoch, da so der plattformunabhängige und validierbare Datenaustausch strukturierter Informationen zwischen heterogenen Systemen über das WWW erfolgen kann.

8.2 Java

Obwohl die Programmiersprache Java unbestreitbar großen Erfolg bei Softwareentwicklern verbuchen kann, fehlen ihr diverse Eigenschaften moderner Programmiersprachen [Fischbach 96]. So ist es z.B. in Java nicht möglich, den abstrakten Datentyp einer Liste zu implementieren ohne den Typ der Listenelemente genau zu spezifizieren. Da es jedoch bei einer Liste nicht darauf ankommt, ob sie Integer oder Strings enthält, sondern allein auf die Art des Zugriffs, wird z.B. bei der Implementierung von Integer- und String-Listen ein Großteil des Codes identisch sein. In einer generischen Sprache wie TL [Matthes 93] übernimmt in so einem Fall ein Typparameter die Funktion eines Platzhalters für den Typ der Listenelemente. Dies ermöglicht eine hochgradige Wiederverwendung von Code, da nicht für jeden Typ von Listenelementen eigene abstrakte Datentypen definiert werden müssen. Ein weiterer Vorteil funktionaler Sprachen wie TL gegenüber Java sind die Funktionen höherer Ordnung. Dabei handelt es sich um Funktionen, die Funktionen als Parameter übernehmen und zurückgeben können. Durch die Unterstützung dieses Konzeptes ist ebenfalls ein erhöhtes Maß an Modularität und Codewiederverwendung möglich [Fischbach 96].
In diesem Abschnitt werden zwei Weiterentwicklungen vorgestellt, die Java um Konzepte moderner Programmiersprachen ergänzen. Pizza ist eine Entwicklung der Universität Karlsruhe, die u.a. parametrischen Polymorphismus für Java einführt (vgl. Abschnitt 8.2.1). PJava ist ein Projekt der Universität Glasgow in Kooperation mit Sun Microsystems, das orthogonale Persistenz für Java untersucht (vgl. Abschnitt 8.2.2).
Das Komponentenmodell Java Beans, bei dem es sich um eine Erweiterung des Java API handelt, wird in Abschnitt 8.2.3 vorgestellt.

8.2.1 Pizza

Pizza ist eine Erweiterung von Java, die drei Konzepte moderner Programmiersprachen in Java einbringt [Odersky, Wadler 97]:

Parametrischer Polymorphismus: Polymorphismus stellt einen generellen Ansatz zur Verfügung, der Daten oder Algorithmen beschreibt, bei denen die Struktur unabhängig vom manipulierten Elementtyp ist. Beim parametrischen Polymorphismus können Typen z.B. als Parameter von Klassen- oder Funktionsdefinitionen auftreten.

Funktionen höherer Ordnung: Funktionen höherer Ordnung — auch als Funktionen erster Klasse bezeichnet — können Funktionen genau wie Daten als Parameter bekommen, als Ergebnis zurückliefern und in Variablen oder Datenstrukturen speichern.

Algebraische Datentypen: Algebraische Datentypen werden in Pizza über Fallunterscheidungen in Klassendefinition (*class cases*) definiert. Per *pattern matching* kann der Typ einer spezifischen Instanz einer Klasse bestimmt werden.

Pizza Programme werden in Java Bytecode übersetzt. Dadurch ist der erzeugte Code auf jeder virtuellen Java Maschine (JVM) lauffähig und die Verwendung des stetig wachsenden Java API ist möglich. Die Übersetzung des erweiterten Pizza Programmcodes in Java Code erfolgt über zwei unterschiedliche Verfahren:

Heterogene Übersetzung: Bei der heterogenen Übersetzung von Pizza Code in Java Code wird für jeden benutzten Typ eine spezialisierte Kopie des Codes angelegt.

Homogene Übersetzung: Eine homogene Übersetzung benutzt eine einzelne Kopie des Codes mit einer universellen Repräsentation.

Am Beispiel des parametrischen Polymorphismus werden beide Arten der Übersetzung erläutert. Die Klasse *Pair* nimmt einen Parameter *elem*. Dieser Parameter ist ein Platzhalter für den Typ des zu bildenden Paares. Die Klasse kann z.B. ein Paar aus Zeichenketten oder Zahlen erzeugen.

```
class Pair<elem> {
   elem x; elem y;
   Pair (elem x, elem y) {this.x = x; this.y = y;}
   void swap () {elem t = x; x = y; y=t;}
}

Pair<String> p = new Pair("world", "Hello ");
Pair<int> q = new Pair(22, 64);
```

Die heterogene Übersetzung des obigen Pizza Codes in Java erfolgt so, daß für jede Instanz der Klasse *Pair* eine Kopie des Codes erzeugt wird.

```
class Pair_String {
   String x; String y;
   Pair (String x, String y) {this.x = x; this.y = y;}
   void swap () {String t = x; x = y; y=t;}
}

class Pair_int {
   int x; int y;
   Pair (int x, int y) {this.x = x; this.y = y;}
   void swap () {int t = x; x = y; y=t;}
}

Pair_String p = new Pair_String("world", "Hello ");
Pair_int q = new Pair_int(22, 64);
```

Bei der homogenen Übersetzung wird die Typvariable *elem* aus der Klasse *Pair* durch die Klasse *Object* ersetzt.

```
class Pair {
   Object x; Object y;
   Pair (Object x, Object y) {this.x = x; this.y = y;}
   void swap () {Object t = x; x = y; y=t;}
}

class Integer {
   int i;
   Integer (int i) {this.i = i;}
   int intValue() {return i;}
}

Pair p = ((Object)"world",(Object)"Hello ");
Pair q = new Pair((Object)new Integer(22), (Object)new Integer(64));
```

Die Erzeugung eines Paares aus Zeichenketten erfolgt über eine Konvertierung der Zeichenkettenobjekte in Objekte. Um ein Zahlenpaar zu erzeugen, müssen Zahlenwerte vom Basistyp *int* zuerst in ein Objekt vom Typ *Integer* konvertiert werden, bevor sie in ein Objekt vom Typ *Object* konvertiert werden können.
Im Hinblick auf die Redundanz des Codes der heterogenen Übersetzung und die Typkonvertierungen bei der homogenen Übersetzung sind die Vorteile der direkten Unterstützung von Polymorphismus sichtbar.

8.2.2 PJava

Während die Java Anwendungsprogrammierschnittstelle (API) persistente Daten über eine Schnittstelle zu Datenbanken (JDBC) unterstützt, entwickelt die Universität Glasgow eine orthogonal persistente Variante von Java: das sogenannte PJava [Neugebauer 96].
Persistenz ist die Eigenschaft von Datenstrukturen, über die Dauer von einzelnen Betriebssystemprozessen hinaus zu existieren. Orthogonal persistente Systeme sind durch folgende Merkmale charakterisierbar [Atkinson, Morrison 95]:

Unabhängigkeit der Persistenz: Programme sind unabhängig von der Langlebigkeit der in ihnen verarbeiteten Daten, d.h. der Programmierer muß sich nicht um den expliziten Übergang von flüchtigen (transienten) Daten zu dauerhaften (persistenten) Daten kümmern.

Orthogonalität der Datentypen: Datenobjekte sind unabhängig von ihrem Typ immer persistent.

Identifikation der Persistenz: Persistente Objekte sind unabhängig vom Typsystem oder von Objektbezeichnern identifizierbar. In der Regel ist Persistenz über die Erreichbarkeit eines Objektes von einer persistenten Wurzel aus definiert.

Die Vorteile orthogonal persistenter Systeme liegen u.a. in:

- der Steigerung der Produktivität in Folge der einfachen Semantik,
- der Vermeidung von Datenkonvertierungen zwischen transienten und persistenten Daten,
- der automatischen Bereitstellung referentieller Integrität für das gesamte System während seiner Lebensdauer.

Die entscheidenden Ziele für das Design von PJava sind [Atkinson et al. 96]:

- keine Sicherheitslücken durch die Benutzung der Persistenz,
- nur minimale Erweiterungen bzw. Änderungen an der Sprache Java,
- keine Beeinflussung der Performanz in Abhängigkeit von Nutzung der Persistenz,
- dem Benutzer sowohl einfache Transaktionsmodelle (ACID, verknüpfte Transaktionen) als auch universell nutzbare Transaktionsbausteine, die dem Benutzer die Erstellung eigener Modelle ermöglichen, zur Verfügung zu stellen.

Desweiteren soll in PJava die Kooperation zwischen mehreren persistenten Speichern ermöglicht werden. Dazu dient sowohl das Kopieren von Objekten zwischen Speichern sowie Methodenaufrufe über Speichergrenzen hinweg. Weitere Ziele des PJava Projektes sind u.a. ein allgemeines Modell für verteilte Anwendungen unter Berücksichtigung unzuverlässiger WANs (*Wide Area Networks*), eine entsprechende Erweiterung der Transaktionsarchitektur und die Entwicklung effizienter Algorithmen für die Speicherfreigabe auf Festplattenspeichern (engl. *disk garbage collections*).

8.2.3 Java Beans

Das Java Beans API definiert ein Softwarekomponentenmodell für Java. Das Ziel dieser Anwendungsprogrammierschnittstelle ist der Aufbau eines offenen Marktes für Java Softwarekomponenten, in dem unabhängige Softwareentwickler Komponenten erstellen und verkaufen, die vom Endbenutzer zu kompletten Applikationen zusammengesetzt werden können [Hamilton 97].
Ein Java Bean ist eine wiederverwendbare Softwarekomponente, die in einem Entwicklungswerkzeug sichtbar manipuliert werden kann. Dabei kann es sich um Benutzerschnittstellenelemente wie z.B. Schaltflächen (engl. *buttons*) oder komplexe Datenbankvisualisierungswerkzeuge handeln. Es gibt auch Java Beans, die keine Benutzerschnittstellenrepräsentation haben, wie z.B. eine Abstraktion einer Hardwareschnittstelle eines Computers oder eine Komponente einer Serveranwendung.
Ein Java Bean muß folgende Konzepte unterstützen:

Ereignisse: Ein Ereignis ist z.B. ein Mausklick, das Verschieben eines Fensters oder eine Eingabe von der Tastatur. Über Ereignisse werden Beans miteinander verknüpft.

Eigenschaften: Eigenschaften sind vergleichbar mit den Attributen einer Klasse. Eigenschaften eines Bean mit einer visuellen Benutzerschnittstellenrepräsentation sind z.B. seine Darstellungsgröße, die verwendete Schriftart oder die Hintergrundfarbe. Eigenschaften sind sowohl während der Entwicklungsphase einer Anwendung als auch zur Laufzeit nutz- und veränderbar (z.B. Änderungen von Fenstergrößen).

Introspektion: Einem Entwicklungswerkzeug muß es möglich sein zu analysieren, wie ein Bean arbeitet, über welche Attribute es verfügt, welche Methoden aufgerufen werden können, welche Ereignisse (z.B. Mausklick, Mausbewegung, Tastatureingaben) behandelt werden können, etc.

Anpaßbarkeit: Die visuelle Erscheinung und das Verhalten eines Java Bean müssen durch den Benutzer anpaßbar sein. So können z.B. die Werte von Eigenschaften geändert werden und die Methoden zur Ereignisbehandlung des Bean registriert und mit anderen Beans verknüpft werden (z.B. kann der Benutzer definieren, daß sich ein Hilfefenster öffnet, wenn eine Schaltfläche gedrückt wird).

Persistenz: Ein Bean muß es erlauben, einen in einem Entwicklungswerkzeug durch Anpassung von Eigenschaften erarbeiteten Zustand des Bean zu speichern und später wieder zu laden.

Java Beans können zum Zeitpunkt der Entwicklung einer Anwendung (*design time*) wesentlich umfangreicher sein als zur späteren Laufzeit der Anwendung (*run time*). Dies ist z.B. der Fall, wenn das Bean zu seiner Anpassung in einer Entwicklungsumgebung einen sogenannten *wizard* mitbringt, der die Anpassung der Eigenschaften des Bean über eine Reihe von Auswahldialogen vornimmt. Diese Art von Werkzeugen werden nur zum Entwicklungszeitpunkt einer Anwendung benötigt, nicht jedoch zur Laufzeit der Applikation.
Der Ansatz der Java Beans ist mit dem VBX/OCX/ActiveX Ansatz von Microsoft vergleichbar. Für diese Art von Softwarekomponenten, die nur auf den Microsoft Windows Plattformen lauffähig sind, hat sich in den letzten Jahren ein großer Markt für Softwarekomponenten entwickelt, die in Entwicklungswerkzeugen diverser Hersteller verwendet werden können und sich mit wenigen Zeilen Programmcode zu kompletten Applikationen zusammensetzen lassen.
Für die Verwendung von Java Beans Komponenten im Zusammenspiel mit anderen Komponentenmodellen wie z.B. COM bzw. DCOM (vgl. Abschnitt 7.3.3) sorgen spezielle Mechanismen, die für den Programmierer von Beans unsichtbar sind. So ist es z.B. möglich, Java Beans in OLE bzw. ActiveX Containern wie z.B. der Entwicklungsumgebung Microsoft VisualBasic zu verwenden. Auch der umgekehrte Weg ist in Form eines Migrationsassistenten verfügbar. Mit diesem Werkzeug wird aus einer ActiveX Komponente ein Implementationskelett für ein entsprechendes Java Bean generiert, das dann im Gegensatz zu der ActiveX Komponente plattformunabhängig verwendet werden kann. [1]

[1] Weitere Informationen zu diesen Werkzeugen findet der interessierte Leser im WWW unter der URL: http://splash.javasoft.com/beans/activextools.html

Entwicklungsumgebungen, die das Java Komponentenmodell unterstützen, sind bereits verfügbar.

8.3 Internet Protokoll der neuen Generation

Der Erfolg des Internet im allgemeinen und des WWW im besonderen führen das derzeit verwendete Internet Protokoll (IPv4) an seine Grenzen. Im Laufe der letzten Jahre hat sich nicht nur die Anzahl, der an das Internet angeschlossenen Rechner vervielfacht sondern auch die Anforderungen von Anwendungen an das Protokoll sind gestiegen [Comer 97]. Das Internet Protokoll der neuen Generation (IPv6) soll diesen Bedürfnissen u.a. mit folgenden Eigenschaften gerecht werden:

Erweiterter Adreßraum: Internetadressen werden im IPv6 eine Länge von 128 Bit haben, im Gegensatz zu einer Adreßlänge von 32 Bit im IPv4. Daraus ergibt sich ein Adreßraum, der für ein kontinuierliches Wachstum des Internet über mehrere Dekaden ausreicht.

Reservierung von Bandbreiten: Die Übertragung von Audio- und Videodatenströmen muß in regelmäßigen Intervallen erfolgen, um eine verzögerungsfreie Darstellung der Information auf dem Zielrechner zu ermöglichen. Das *Reservation Setup Protocol* (RSVP) setzt auf dem IP Protokoll auf und zielt auf die vom Empfänger initiierte Zuweisung von Ressourcen. Der Empfänger kann eine Bandbreite für eine Verbindung reservieren und so eine leitungsvermittelte Verbindung simulieren. Er erreicht dadurch eine höhere Qualität des Dienstes (engl. *Qualtity of Service*, QoS) für seine Datenströme. Erfordert z.B. eine Anwendung einen spezifischen QoS, wird RSVP benutzt, um diese Anforderung an jeden Vermittlungsrechner entlang des Übertragungsweges weiterzugeben.

Priorisierung von Datenströmen: Das *Internet Protocol Version 6* (IPv6) ermöglicht die Anpassung an unterschiedliche Netzbelastungen, indem es Datenströme priorisieren kann. So kann z.B. ein Videodatenstrom in mehrere Kanäle aufgeteilt werden, die eine fortschreitend höhere Bildschärfe haben. Treten Verzögerungen auf, wird die Übertragung der Kanäle mit höherer Bildschärfe ausgesetzt, damit die Bandbreitenanforderungen reduziert und so wenigstens die schnelle Übertragung von schlechter aufgelösten Bildern zum Empfänger erreicht werden kann.

Erweiterte Adressierungs- und Routingmöglichkeiten: Mit der Zunahme von Kollaborationstechniken werden Fähigkeiten des Internet Protokolls benötigt, die es erlauben, Gruppen zu bilden und z.B. jedem Mitglied einer Gruppe eine Kopie eines Datenpaketes zu schicken. In anderen Fällen werden Gruppen zur Lastverteilung genutzt. Existieren mehrere identische Kopien eines Dienstes und ein Datenpaket wird an eine solche Gruppe geschickt, wird das Paket zu der Kopie des Dienstes gelenkt, die dem Sender am nächsten ist.

9. Zusammenfassung

In dieser Arbeit werden grundlegende Konzepte und Technologien des WWW vorgestellt und untersucht. Die Bewertung dieser Konzepte und Technologien erfolgt anhand der Anforderungen von Informationsnutzern und Informationsanbietern, die sich wiederum auf die zugrundeliegende Architektur des Systems bzw. die daran beteiligten Technologien auswirken. Es wird gezeigt, daß die Anforderungen an das WWW seit 1991 permanent gewachsen sind und die Kernkonzepte wie z.B. Adressierung mittels URLs, das HTT Protokoll, HTML und CGI diese Anforderungen anspruchsvoller verteilter Informationssysteme nicht mehr erfüllen können.
Exemplarisch werden die Anforderung der Versionierung von WWW Ressourcen betrachtet, die Basiskonzepte von Versionierung erläutert und ausgewählte Realisierungen von Versionierungssystemen im WWW vorgestellt und bewertet. Eine eigene Implementierung in Form einer Anbindung des Versionierungssystems SCCS an den WWW Server Jigsaw zeigt eine weitgehende Realisierung der erarbeiteten Anforderungen und erlaubt dem WWW Benutzer via eines erweiterten URL Schemas das Abrufen versionierter WWW Ressourcen. Aus der Versionierung einzelner WWW Ressourcen ergibt sich das Problem des Konfigurationsmanagement, zu dem Konzepte und Realisierungsansätze vorgestellt werden.
Das WWW wird im Rahmen von Intranets zunehmend zu einer Basis verteilter Client/Server Anwendungen. Es existieren mit CORBA, RMI und DCOM drei konkurrierende Ansätze, die für verschiedene Anwendungsfälle prädestiniert sind. Eine vergleichende Betrachtung ergibt, daß RMI für die Erstellung von Client/Server Anwendungen, bei denen sowohl Client als auch Server von der Plattformunabhängigkeit der Programmiersprache Java profitieren sollen, eine gute Integration in die Programmiersprache und ein leicht verständliches Kommunikationsmodell bietet. Die Verwendung von CORBA ist bei der Nutzung von Java Applets als Clients für in heterogenen Programmiersprachen geschriebenen Serverprogrammen angezeigt, da das CORBA Konzept sowohl plattform- als auch programmiersprachenunabhängig ist. Werden hingegen nur Microsoft Windows Betriebssysteme eingesetzt und ist das dargestellte Sicherheitsproblem von ActiveX Komponenten nicht relevant, bietet DCOM eine weitreichende Verzahnung mit der Windows Architektur und erlaubt so mächtige Clientanwendungen, die aufgrund der Sicherheitsbeschränkungen von Java als Applets nicht möglich wären.
Neue Entwicklungen im Kontext von HTML, Java und den Internetprotokollen zeigen technologische Auswirkungen der gewachsenen Anforderungen an das WWW. An diesen Entwicklungen wird sichtbar, daß das WWW zunehmend eine neue Infrastruktur für die welt-

weite Verteilung und den Verkauf von Information, Produkten und Dienstleistungen bildet.

9.1 Kommerzielle Relevanz

War das WWW am Anfang seiner Entwicklung ein Treffpunkt für Wissenschaftler, ist es heute ein stetig wachsender Wirtschaftsfaktor. Angefangen bei Werbung und virtuellen Warenhäusern, hin zu neuen Vertriebswegen für alle Arten von Dienstleistungen und Produkten bieten sich im Kontext des WWW viele neue Möglichkeiten zu wirtschaftlicher Betätigung. Insbesondere kleinere Unternehmen, die nicht die Möglichkeit haben, große Distributoren für ihre Spezialprodukte zu finden, haben über die Infrastruktur WWW die Möglichkeit, ihre Produkte weltweit zu bewerben und zu verkaufen oder beratend tätig zu sein.
Die Betrachtung der veränderten und gewachsenen Anforderungen an das WWW und die Erarbeitung und Vorstellung neuer Konzepte und Technologien ist für den weltweiten Marktplatz WWW — nicht umsonst spricht man vom „globalen Dorf“ — wichtig, da die grundlegenden Technologien wie HTTP und HTML für diese Art der Kommunikation nicht konzipiert sind.
Information im WWW ist nicht statisch. Änderungen an bestehenden Informationsressourcen führen zu Konsistenzproblemen. Versionierung im WWW erlaubt die konsistenzerhaltende Wartung von WWW Ressourcen. Das Konzept der Versionierung ist desweiteren z.B. bei der Bearbeitung von Ressourcen durch mehrere Bearbeiter notwendig. Insbesondere durch die zunehmende Heimarbeit von z.B. Programmierern oder Grafikdesignern wird es zunehmend wichtiger, Versionskontrolle über das WWW zur Verfügung zu stellen, da viele Unternehmen gar nicht mehr über Arbeitsplätze für alle ihre Mitarbeiter verfügen. Die Unterstützung der Versionierung von HTML Seiten in dem Spezikationsvorschlag zu HTML 4.0 (vgl. Abschnitt 8.1.1) ist eine erste — wenn auch auf HTML Seiten beschränkte — Umsetzung des Versionierungskonzeptes im WWW.
Die in Kapitel 7 vorgestellten Kommunikationsmechanismen bilden die Grundlage für verteilte Anwendungen im WWW. Ein Vergleich von CORBA, RMI und DCOM gibt dem Softwareentwickler einen Überblick über die verschiedenen Konzepte und erlaubt die Auswahl einer für seine Zwecke geeigneten Technologie.

9.2 Weiterführende Arbeiten

Ein Schwerpunkt dieser Arbeit liegt auf dem Versionsmanagement von WWW Ressourcen. Die vorliegende Implementierung erlaubt bisher die Versionierung textueller Informationen (vgl. Abschnitt 5.4). In weiterführenden Arbeiten wäre zu untersuchen, inwiefern die Versionierung von Daten anderen Typs (z.B. Bild, Audio, Video, Code) für das WWW relevant ist. So ist es z.B. sinnvoll, daß vom Server zum Client übertragene Java Applets vor der Übertragung großer Codebibliotheken überprüfen, ob diese bereits beim Client vorliegen, bzw. ob die vorliegenden Versionen der Bibliotheken mit dem übertragenen Applet kompatibel sind. Im Rahmen des Tycoon Projektes an der Universität Hamburg werden zu dieser

Fragestellung bereits Betrachtungen vorgenommen [Ramme 97], deren Übertragbarkeit auf die Programmiersprache Java zu untersuchen wäre.
In Abschnitt 6.3 werden zwei Ansätze für das Konfigurationsmanagment von WWW Ressourcen vorgeschlagen. In weiterführenden praktischen Arbeiten wäre zu überprüfen, inwiefern diese Ansätze geeignet sind und in welchen Fällen das Kontextmanagement beim Server bzw. beim Client Vorteile hat. So kann es z.B. sinnvoll sein, daß, wenn ein Kontext Ressourcen auf mehreren Servern beinhaltet, das Kontextmanagement beim Client erfolgt, damit keine Replizierung der Kontextinformationen zwischen verschiedenen Servern erforderlich wird.
Ein weiterer Schwerpunkt der Arbeit liegt auf dem Vergleich der drei konkurrierenden Kommunikationsmechanismen für verteilte Anwendungen im WWW: CORBA, RMI und DCOM. Hierbei sind zwei verschiedene Ansätze erkennbar. Während RMI und CORBA von einem Java Applet als Client ausgehen, setzt Microsoft auf seine proprietäre ActiveX Technologie. Durch die Interoperabilität zwischen dem Java Komponentenmodell Java Beans (vgl. Abschnitt 8.2.3) und ActiveX, das auf DCOM basiert, bleibt zu überprüfen, inwiefern sich Microsoft mit seiner auf dem alten OLE/OCX Modell basierenden Technologie, die ursprünglich nicht für die Verteilung von Softwarekomponenten im Netz vorgesehen war, durchsetzen kann. DCOM/ActiveX hat hierbei insbesondere durch seine Plattformabhängigkeit und die unsichere Codeausführung beim Client Nachteile gegenüber den Java Beans. Andererseits gibt es aufgrund der älteren Basistechnologien von ActiveX ein gegenüber den relativ neuen Java Beans Komponenten wesentlich größeres Angebot von ActiveX Komponenten. Microsoft verfügt desweiteren durch seine weitverbreiteten Windows Betriebssysteme über eine große Macht am Markt und ist auch bereit, diese Macht gegenüber seinen Konkurrenten zu nutzen. Dies dokumentiert ein Ausspruch von Bob Muglia, Microsoft [OMG 97c]: „Microsoft unapologetically will make sure ActiveX works best on Windows."

9.3 Die Zukunft des WWW

Es ist bereits erkennbar, daß die in der Einleitung beschriebene Vision des Object Web zunehmend Realität wird. Es werden bereits heute beliebige Objekte (Texte, Bilder, Audio, Video, Dokumente in diversen Formaten) in WWW Seiten integriert und mit Hilfe von Plug-Ins, Java Applets oder ActiveX Komponenten angezeigt. Die Integration von ORB Technologie in den neuen Browser der Firma Netscape zeigt deutlich die Ambitionen, das WWW als Grundlage für verteilte Anwendungen nutzbar zu machen. Auch in der Namensgebung dieses neuen Browsers zeichnet sich der Weg des WWW zur grundlegenden Kommunikationsplattform ab: hieß der Netscape Browser früher Navigator, wird die neue Version Communicator genannt.
Werkzeuge wie Netiva erlauben auch nicht mit Programmiersprachen und Datenbanken vertrauten WWW Autoren das Erstellen verteilter Datenbankanwendungen im WWW. Dazu waren bis vor kurzem noch Kenntnisse in HTML, HTTP, CGI, Skriptsprachen, SQL und Datenbanktechnologie oder Java unumgänglich.
Die Firma Microsoft integriert in kommende Versionen seiner Windows Betriebssysteme

Browser Funktionalität. Das heißt, der Benutzer muß nicht mehr einen speziellen Browser starten, um ein WWW Dokument anzusehen, sondern er gibt — z.B. in seinem Dateimanager — die zu öffnende URL an und die entsprechende WWW Seite wird für ihn transparent aus dem Internet angefordert und auf seinem virtuellen Schreibtisch dargestellt. Damit verwischt für den Benutzer der Übergang zwischen seinem virtuellen Schreibtisch und dem Internet bis zur Unsichtbarkeit.
Insbesondere *compound documents* bieten im Zusammenhang mit Komponententechnologien eine Zukunftsperspektive für das WWW. So sollte es für den Benutzer in Zukunft möglich sein, beliebige Komponenten eines Dokumentes, wie z.B. eingebette Bilder, Grafiken, andere Dokumente, etc. per *drag and drop* in eine andere Anwendung zu übertragen, sie dort zu ändern und wieder in das zusammengesetzte WWW Dokument zurückzulegen. Workflow Modelle, bei denen mehrere Bearbeiter einen Auftrag erledigen, werden dann auch über das WWW realisierbar sein, was der zunehmenden Heimarbeit von Angestellten in vielen Branchen entgegenkommt. Unternehmen werden durch solche Technologien nicht nur flexibler bei der Globalisierung ihres Absatzmarktes sondern auch bei der Herstellung von Produkten und Dienstleistungen. Dies kann soweit führen, daß Firmen lediglich als *virtuelle Unternehmen* über eine Internetaddresse erreichbar sind und es weder Firmenräume gibt, in denen Kunden empfangen werden, noch Arbeitsräume vorhanden sind, in den die Angestellten des Unternehmens ein Produkt herstellen oder eine Dienstleistung erbringen.
Tatsache ist, daß das WWW bzw. das Internet aus der Informationsgesellschaft der Zukunft nicht mehr wegzudenken ist. Der Verlust von „echten“ Inhalten zugunsten von Layout, Werbung und Kommerz ist zwar bedauerlich, aber gleichzeitig bieten sich vielfältige Möglichkeiten für die Erschließung neuer Märkte.

Literaturverzeichnis

OMG 95: OMG. *CORBA 2.0: Architecture and Specification.* Object Management Group, 1995.

Sun Microsystems 96: Sun Microsystems. *RMI – Remote Method Invocation.* http://www.javasoft.com:80/products/jdk/1.1/docs/guide/rmi/index.html, 1996.

Open Group 97a: Open Group. *Open Group DCE Documentation.* http://www.rdg.opengroup.org/public/pubs/catalog/dz.htm, 1997.

Netiva 97b: Netiva. *Netiva The Web Database Application.* http://www.netiva.com/, 1997.

OMG 97c: OMG. *Comparing ActiveX and CORBA/IIOP.* http://www.omg.org/news/activex.html, 1997.

OMG 97d: OMG. *Security, Transactions, ... and more.* http://www.omg.org/corba/sectrans.htm, 1997.

Microsoft Corp. 97e: Microsoft Corp.. *White Paper: DCOM Technical Overview.* http://www.microsoft.com/, 1997.

Atkinson et al. 96: Atkinson, M.P., Jordan, M.J., Daynes, L., und Spence, S. *Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system.* In: *7th Int. Workshop on Persistent Object Systems*, 1996.

Atkinson, Morrison 95: Atkinson, M. und Morrison, R. *Orthogonally Persistent Object Systems.* VLDB Journal, Jg. 4, 1995, Nr. 3, S. 319–401.

Balzert 82: Balzert, H. *Die Entwicklung von Softwaresystemen.* B.I.-Wissenschaftsverlag, 1982.

Behme 97: Behme, H. *Active Server Pages: Microsofts CGI-Ersatz.* iX, Multiuser Multitasking Magazin, 1997, Nr. 7, S. 0–1.

Bichler 96: Bichler, M. *Hyperbase, World Wide Web: Basis für betriebliche Anwendungen?* iX, Multiuser Multitasking Magazin, 1996, Nr. 8, S. 44–50.

Bolinger, Bronson 95: Bolinger, D. und Bronson, T. *Applying RCS and SCCS, From Source Control to Project Control.* O'Reilly & Associates, Inc., 1995.

Bösch 95: Bösch, A. *PolitIcon: Entwurf einer Bilddatenbank für den Index zur politischen Ikonographie.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1995.

Brockschmidt 96: Brockschmidt, K. *What OLE is Really About.* http://www.microsoft.com/oledev/olecom/aboutole.htm, 1996.

CERN 96: CERN. *European Laboratory for Particle Physics.* http://www.cern.ch, 1996.

Cluts 97: Cluts, N. W. *The Microsoft Active Platform.* http://www.microsoft.com/workshop/prog/aplatfrm/platform-f.htm, 1997.

Comer 97: Comer, D.E. *Computer Networks and Internets.* Prentice-Hall International, Inc., 1997.

Cunningham, Faizi 96: Cunningham, J. und Faizi, A. *Distributed Authoring and Versioning Protocol.* http://www.ics.uci.edu/~ejw/authoring/ns_dav.html, 1996.

Curtis 97: Curtis, D. *Java, RMI and CORBA.* http://www.omg.org/news/wpjava.html, 1997.

Dalitz, Hezer 95: Dalitz, W. und Hezer, G. *Hyper-G das Internet-Informationssystem der 2. Generation.* Verlag für Digitale Technologie, 1995.

December, Ginsburg 95: December, J. und Ginsburg, M. *HTML and CGI unleashed.* Sams.net, Indianapolis, 1995.

Donnerhacke, Peter 97: Donnerhacke, L. und Peter, S. *ActiveX als Füllhorn für Langfinger.* iX, Multiuser Multitasking Magazin, 1997, Nr. 3, S. 90–95.

Feiler 91: Feiler, P.H. *Configuration Management Models in Commercial Environments.* Technical Report CMU/SEI-TR-7, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1991.

Fischbach 96: Fischbach, R. *Kalter Kaffee, Java: Programmiersprache der Zukunft?* iX, Multiuser Multitasking Magazin, 1996, Nr. 10, S. 84–89.

Gilster 95: Gilster, P. *Suchen und Finden im Internet.* Carl Hanser Verlag, München Wien, 1995.

Gundavaram 96: Gundavaram, S. *CGI Programming on the World Wide Web.* O'Reilly & Associates, Inc., 1996.

Hamann 96: Hamann, J. *Analysis of Java Database Connectiviy and its Application in a Multi–Platform, Multi–DBMS Environment.* http://www.cs.tamu.edu/people/jhamann/jdbc/, 1996.

Hamilton 97: Hamilton, G. (Hrsg.). *Java Beans Spezification, Version 1.01.* Sun Microsystems, 1997.

Hendrich 97: Hendrich, N. *Java für Fortgeschrittene.* Springer-Verlag, Berlin u.a., 1997.

Herold 94: Herold, H. *Software-Management unter Unix und MS-DOS.* Addison–Wesley, 1994.

Hicks et al. 91: Hicks, D.L., Legget, J.J., und Schnase, J.L. *Version Control in Hypertext Systems.* Technical Report TAMU HRL-91-004, Hypermedia Research Lab, Texas A & M University, 1991.

Huber 96: Huber, G. *Java Die Referenz.* Heinz Heise, 1996.

Jablonski 90: Jablonski, S. *Datenverwaltung in verteilten Systemen.* Springer-Verlag, Berlin u.a., 1990.

Kehoe 94: Kehoe, G. *Zen and the Art of the Internet. A Beginners Guide.* Prentice Hall, 1994.

Kirsch 97: Kirsch, C. *Programm Aktualisierung per Netz.* iX, Multiuser Multitasking Magazin, 1997, Nr. 8, S. 96–103.

Klute 96: Klute, R. *Puzzle in 14 Minuten, Jigsaw: der neue Server des World-Wide-Web-Konsortiums.* iX, Multiuser Multitasking Magazin, 1996, Nr. 9, S. 108–115.

Kossel 96: Kossel, A. *Hausmannskost, Wie die Softwareindustrie das Intranet schmackhaft machen will.* c't, Magazin für Computertechnik, 1996, Nr. 10, S. 298–300.

Kramer 97: Kramer, D. *Java API Overview.* http://java.sun.com:80/products/api-overview/index.html, 1997.

Kreuz, Lange 96: Kreuz, D. und Lange, T. *Offene Objektorientierte Transaktionsverarbeitung.* In: Wahlster, M. (Hrsg.). *Proceedings of the 19th European Congress Fair of Technical Communications – Online 96.* 1996.

Lassesen 95: Lassesen, K. *Using Microsoft OLE Automation Servers to Develop Solutions.* http://www.microsoft.com/oledev/oleauto/ole2soln.htm, 1995.

Lemay, Perkins 96: Lemay, L. und Perkins, C.L. *Teach Yourself JAVA in 21 Days.* Sams.net, Indianapolis, 1996.

Lockemann, Schmidt 87: Lockemann, P.C. und Schmidt, J.W. (Hrsg.). *Datenbankhandbuch.* Springer-Verlag, Berlin u.a., 1987.

Macherius 97: Macherius, I. *Revolution der Experten, XML: Professionelle Alternative zu HTML.* iX, Multiuser Multitasking Magazin, 1997, Nr. 6, S. 158–162.

Mathiske 96: Mathiske, B. *Mobilität in persistenten Objektsystemen.* Dissertation, Fachbereich Informatik, Universität Hamburg, 1996.

Matthes 93: Matthes, F. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung.* Springer-Verlag, Berlin u.a., 1993.

Maurer 96: Maurer, H. (Hrsg.). *Hyperwave — The next Generation Web Solution.* Addison–Wesley, 1996.

Meyer 90: Meyer, B. *Objektorientierte Softwareentwicklung.* Prentice-Hall International Inc., London, 1990.

Microsoft 96: Microsoft. *Microsoft Internet Explorer.* http://www.microsoft.com/ie, 1996.

Mintert 96: Mintert, S. *Rendevous, LiveConnect: Kommunikation zwischen Java und JavaScript.* iX, Multiuser Multitasking Magazin, 1996, Nr. 7, S. 110–114.

Netscape 96: Netscape. *Netscape Navigator.* http://home.netscape.com/comprod/products/navigator, 1996.

Neugebauer 96: Neugebauer, R. *Eingebrannt, Orthogonal persistentes Java.* iX, Multiuser Multitasking Magazin, 1996, Nr. 12, S. 118–125.

Nielsen 95: Nielsen, J. *Multimedia, Hypertext und Internet.* Vieweg, 1995.

Odersky, Wadler 97: Odersky, M. und Wadler, P. *Pizza into Java: Translating theory into practice.* In: *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, jan 1997.

Orfali, Harkey 97: Orfali, R. und Harkey, D. *Client/Server Programming with Java and CORBA.* Wiley & Sons, 1997.

Paradies 97: Paradies, T. *Die Wiege der Servlets.* iX, Multiuser Multitasking Magazin, 1997, Nr. 1, S. 66–73.

Ramme 97: Ramme, K. *Dynamisches Rebinden von Objekten in persistenten Objektsystemen.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

Reibold et al. 97: Reibold, H.F., Schmitz, U., und Seeger, J. *Die Rivalen: Was ActiveX von Java, JavaScript und Plug-ins unterscheidet.* iX, Multiuser Multitasking Magazin, 1997, Nr. 2, S. 108–111.

Reuter et al. 96: Reuter, J., Hänßgen, S.U., Hunt, J.J., und Tichy, W.F. *Distributed Revision Control Via the World Wide Web.* In: *Proceedings of the Sixth Internation Workshop on Software Configuration Managament*, 1996.

Richtsmeier 97: Richtsmeier, I. *Kommunizierende Informationssysteme am Beispiel autonomer Internet WebSiteProfiler: Vergleich objekt– und agentenbasierter Ansätze.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

Rieger 95: Rieger, W. *SGML fuer die Praxis.* Springer-Verlag, Berlin u.a., 1995.

Rudloff et al. 95: Rudloff, A., Matthes, F., und Schmidt, J.W. *Security as an Add-On Quality in Persistent Object Systems.* In: *Second International East/West Database Workshop, Klagenfurt, Austria*, Workshops in Computing. Springer-Verlag, Berlin u.a., 1995, S. 90–108. (Also appeared as TR FIDE/95/138).

Scheller et al. 94: Scheller, M., Boden, K.-P., Geenen, A., und Kampermann, J. *Internet: Werkzeuge und Dienste.* Springer, 1994.

Schmelzle, Gast 96: Schmelzle, O. und Gast, C. *Angebunden, Datenbankanwendungen mit dem W3-Gateway für mSQL.* iX, Multiuser Multitasking Magazin, 1996, Nr. 2, S. 158–162.

Schneider 96: Schneider, U. *Applets, schöne Applets, Java-Anwendungen selbst entwickeln.* iX, Multiuser Multitasking Magazin, 1996, Nr. 5, S. 62–68.

Schwarz 97: Schwarz, M. *"Moment, ich verbinde...", COM, SOM und CORBA — oder die Suche nach dem Software–Esperanto.* c't, Magazin für Comuputertechnik, 1997, Nr. 3, S. 256–273.

Spainhour, Quercia 96: Spainhour, S. und Quercia, V. *Webmaster in a Nutshell.* O'Reilly & Associates, Inc., 1996.

Templ 96: Templ, J. *Schwarzes Loch, Wie sicher ist Java wirklich?* iX, Multiuser Multitasking Magazin, 1996, Nr. 5, S. 70–73.

Vitali, Durand 95: Vitali, F. und Durand, D.G. *Using Versioning to Provide Collaboration on the WWW.* In: *World Wide Web Journal.* O'Reilly & Associates, Inc., 1995, S. 37–50.

Vossen 95: Vossen, G. *Verteilte Informationssysteme: Konzepte und Entwurfstechniken.* In: *GISI 95 Herausforderungen eines globalen Informationsverbundes für die Informatik.* springerPub, 1995, S. 27–34.

W3C 96a: W3C. *HTML, Hypertext Markup Language.* http://www.w3.org/pub/WWW/MarkUp/, 1996.

W3C 96b: W3C. *SGML, Standard Generalized Markup Language.* http://www.w3.org/pub/WWW/MarkUp/SGML/, 1996.

W3C 97: W3C. *HTML 4.0 Specification.* http://www.w3.org/TR/WD-html40-970708/cover.html, 1997.

Whitehead 96: Whitehead, J. *WWW Versioning Support.* http://www.ics.uci.edu/~ejw/versioning/whitehead_proposal.html, 1996.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, 25. September 1997

(Andreas Bösch)