# Universität Hamburg

**Fachbereich Informatik**


## DIPLOMARBEIT

# Ein Produktionssystem zum Online-Publishing: Analyse, Entwurf und prototypische Realisierung in Java


**Erstbetreuer:** Prof. Dr. J.W. Schmidt
**Arbeitsbereich:** Datenbanken und Informationssysteme (DBIS)

**Zweitbetreuer:** Prof. Dr. F. Vogt
**Arbeitsbereich:** Technische Informatik V -Telematik (TU-Harburg)

**Autor:**

Yervand Nahapetian



September 1997

# Inhaltsverzeichnis

**1 EINLEITUNG ..... 1-6**

1.1 ZIELE DER ARBEIT ..... 1-8

**2 ANALYSE ..... 10**

2.1 BETRIEBLICHE ANALYSE ..... 10

*2.1.1 Das Vokabular des Online-Publishing ..... 10*

*2.1.2 Das Vokabular der Geschäftsdomäne ..... 11*

*2.1.3 Akteure und Aufgaben ..... 11*

2.1.3.1 Programmierer (Online Producer) ..... 11

2.1.3.2 Administrator ..... 11

2.1.3.3 Redakteur (Content Provider) ..... 12

2.1.3.4 Online Graphiker (Online Artist) ..... 12

2.1.3.5 Projektleiter (Product Manager) ..... 12

2.1.3.6 Layouter (Art Director) ..... 12

2.1.3.7 Chef vom Dienst (CvD) ..... 12

2.1.3.8 Externer Dienstleister ..... 12

*2.1.4 Arbeitsvorgänge ..... 12*

*2.1.5 Das existierende Produktionssystem WebIza ..... 16*

2.1.5.1 Nachteile von WebIza ..... 19

*2.1.6 Produktvergleich: kommerzielle Produktionssysteme ..... 20*

2.1.6.1 Nescape Publishing System ..... 20

2.1.6.2 Microsoft Frontpage ..... 20

2.1.6.3 Netobjects Fusion ..... 20

2.1.6.4 Dynabase Publishing System ..... 20

2.2 ANALYSE DER SYSTEMANFORDERUNGEN ..... 21

*2.2.1 Hardwarevorgaben ..... 22*

*2.2.2 Softwarevorgaben ..... 22*

*2.2.3 Anwendungsfälle für die Benutzerinteraktion ..... 23*

2.2.3.1 *Erzeugen einer Resource* ..... 23

2.2.3.2 *Editieren einer Resource* ..... 24

2.2.3.3 *Verschieben einer Resource* ..... 24

2.2.3.4 *Kopieren einer Resource* ..... 25

2.2.3.5 *Löschen einer Resource* ..... 26

2.2.3.6 Umbenennen einer Resource ..... 26

2.2.3.7 Publizieren einer Resource ..... 27

2.2.3.8 Anfordern von Informationen über eine Resource ..... 27

2.2.3.9 Suchen nach einer Resource ..... 28

2.2.3.10 Anzeigen einer Resource ..... 28

2.2.3.11 Abspeichern einer Resource ..... 29

2.2.3.12 Wiederherstellen einer früheren Version der Resource ..... 29

*2.2.4 Anwendungsfälle für den Administrator ..... 30*

2.2.4.1 *Benutzer hinzufügen* ..... 31

2.2.4.2 *Benutzer entfernen* ..... 32

2.2.4.3 *Gruppe hinzufügen* ..... 32

2.2.4.4 *Gruppe entfernen* ..... 32

2.2.4.5 *Benutzer zur Gruppe hinzufügen* ..... 33

2.2.4.6 *Benutzer von der Gruppe entfernen* ..... 33

2.2.4.7 *Gruppenzugehörigkeit einer Resource ändern* ..... 34

2.2.4.8 *Übernahme alter Datenbestände* ..... 34

2.2.4.9 *Wiederherstellen einer früheren Publikation* ..... 34

*2.2.5 Das Benutzerzugangsmodell ..... 35*

*2.2.6 Konsistenzüberprüfung von Querverweisen (Link-Management) ..... 35*

*2.2.7 Unterstützung mehrerer Ausgabeformate ..... 35*

*2.2.8 Versionsverwaltung ..... 35*

**3 ENTWURF.....36**

3.1 LOGISCHER ENTWURF.....36
*3.1.1 Aufgabenteilung der Kontrollobjekte.....39*
*3.1.2 Die Prozeßstruktur des Gesamtsystems.....39*
*3.1.3 Die Ausführung zeitgebundener Aufträge (Task Scheduling).....41*
3.1.3.1 Das *Task*-Objekt.....42
*3.1.4 Die Kommunikation zwischen Client und Server.....42*
*3.1.5 Der Client.....43*
*3.1.6 Der Server.....47*
3.1.6.1 Initialisierung des Servers.....47
3.1.6.2 Der Anmeldevorgang des Clients.....49
3.1.6.3 Die Verbindungsverwaltung.....49
3.1.6.4 Der Abmeldevorgang des Clients.....51
*3.1.7 Objektverwaltung.....52*
3.1.7.1 Verwaltung des Namensraums (Namespace Management).....52
3.1.7.2 Das Datenbankschema der Resource-Objekte.....53
3.1.7.2.1 Symbolische Links.....54
3.1.7.3 Rekonstruieren einer Resource aus der Datenbank.....56
3.1.7.4 Erzeugen einer Resource in der Datenbank.....56
3.1.7.5 Löschen einer Resource in der Datenbank.....57
3.1.7.6 Datenbankoperationen zum Verschieben einer Resource.....57
3.1.7.7 Datenbankoperationen zum Kopieren einer Resource.....57
3.1.7.8 Speicherung und Verwaltung der Zielformate.....57
3.1.7.9 Qualitätssicherung.....58
*3.1.8 Objekthierarchien.....60*
3.1.8.1 Das Datenobjekt Resource.....60
3.1.8.2 Die ResourceEditor-Klasse.....63
3.1.8.3 Die *ResourceHandler*-Klasse.....64
3.1.8.4 Einführung eines neuen Resourcetyps.....65
3.1.8.5 Resourcevorlagen (Templates).....65
*3.1.9 Objektinteraktionen.....66*
3.1.9.1 Editieren einer Resource.....66
3.1.9.2 Speichern einer Resource.....67
3.1.9.3 Erzeugen einer Resource.....68
3.1.9.4 Löschen einer Resource.....69
3.1.9.5 Erstellen eines symbolischen Links.....69
3.1.9.6 Kopieren einer Resource.....69
3.1.9.7 Verschieben einer Resource.....70
*3.1.10 Zugangsmodell.....71*
3.1.10.1 Zugangskontrolle.....71
3.1.10.2 Überschreibungsschutz bei kooperativer Arbeit.....72
3.1.10.3 Workflowunterstützung.....73
3.1.10.4 Die *Group*-Klasse.....75
3.1.10.5 *Die User*-Klasse.....75
*3.1.11 Konsistenzüberprüfung von Querverweisen (Link-Management).....75*
3.1.11.1 Probleme bei der Verwaltung von Querverweisen.....75
3.1.11.2 Lösungen anderer Organisationen.....76
3.1.11.2.1 Uniform Resource Names (URN).....76
3.1.11.2.2 Persistent URLs (PURLs).....76
3.1.11.3 Der Lösungsansatz dieses Systems.....77
3.1.11.3.1 Formate und Attribute der Querverweise.....77
3.1.11.3.2 Konvertierung der Querverweise nach HTML.....79
3.1.11.3.3 Verwaltung der Querverweise in der Datenbank.....80
3.1.11.4 Operationen des Link-Managements.....81
3.1.11.4.1 Speichern von Resourcen.....81
3.1.11.4.2 Verschieben (move) von Resourcen.....81
3.1.11.4.3 Kopieren (copy) von Resourcen.....81
3.1.11.4.4 Löschen von Resourcen.....81
3.1.11.4.5 Das Publizieren („Heißschalten“) von Resourcen.....82

*3.1.12 Versionsverwaltung* ..... *84*
3.1.12.1 Probleme der Versionsverwaltung bei Hypertext-Dokumenten ..... 84
3.1.12.2 Lösungsansatz ..... 85
3.1.12.2.1 Aufzählung vorhandener Versionen und Kurzbeschreibung ..... 85
3.1.12.2.2 Wiederherstellen (Retrieve) einer Version ..... 85
3.1.12.2.3 Kreieren einer neuen Resource mit dem Inhalt einer Version ..... 86
3.1.12.2.4 Eine Publikation älteren Datums wiederherstellen ..... 86
*3.1.13 Die Behandlung von Inhalten* ..... *87*
3.1.13.1 Textdokumente ..... 87
3.1.13.1.1 Die Geschichte des Markup ..... 87
3.1.13.1.2 Struktur, Inhalt und Darstellung eines Dokuments ..... 88
3.1.13.1.2.1 Was ist die Struktur eines Dokuments ? ..... 88
3.1.13.1.2.2 Was ist der Inhalt eines Dokuments ? ..... 88
3.1.13.1.2.3 Was ist die Darstellung eines Dokuments ? ..... 88
3.1.13.1.3 Gründe für die Trennung von Struktur, Inhalt und Darstellung. ..... 88
3.1.13.1.4 Ein Plädoyer für die Benutzung von XML ..... 89
3.1.13.1.4.1 Document Type Definition (DTD) ..... 89
3.1.13.1.5 Die Darstellung eines Dokuments mit der Darstellungsbeschreibunssprache DSSSL ..... 91
3.1.13.1.6 Die Verwendung von XML und DSSSL im Produktionssystem ..... 92
3.1.13.1.6.1 Möglichkeiten der Konvertierung von XML nach HTML ..... 92
3.1.13.1.6.2 Multiple Ausgangsformate ..... 93
3.1.13.2 Bilder ..... 94
3.1.13.3 Audio ..... 95
3.1.13.4 Andere Inhaltstypen ..... 95
*3.1.14 Die Suche in und nach multimedialen Inhalten* ..... *95*
3.1.14.1 Verwendete Metadaten ..... 96
3.1.14.1.1 TITLE ..... 96
3.1.14.1.2 AUTHOR ..... 96
3.1.14.1.3 SUBJECT ..... 96
3.1.14.1.4 DESCRIPTION ..... 96
3.1.14.1.5 PUBLISHER ..... 96
3.1.14.1.6 DATE ..... 96
3.1.14.1.7 TYPE ..... 96
3.1.14.1.8 FORMAT ..... 97
3.1.14.1.9 Resource-ID ..... 97
3.1.14.1.10 RIGHTS ..... 97
*3.1.15 Die Benutzeroberfläche* ..... *98*
3.1.15.1 Der Browser ..... 98
3.1.15.1.1 Model View Controller (MVC) ..... 98
3.1.15.1.2 Elementare Bedienungsschritte ..... 99
3.1.15.2 Die Editoren ..... 103
3.1.15.2.1 Der *SGMLEditor* ..... 104
3.1.15.2.1.1 SourceComponent ..... 105
3.1.15.2.1.2 FormComponent ..... 106
3.1.15.2.2 Der ImageEditor ..... 110
3.1.15.2.3 Der *SoundEditor* ..... 110
3.1.15.2.4 Der *ImageMapEditor* ..... 110
3.1.15.2.5 *User* Editor ..... 110
3.1.15.2.6 *Group* Editor ..... 110
3.2 TECHNISCHER ENTWURF ..... 111
*3.2.1 Java Remote Method Invocation (RMI)* ..... *111*
3.2.1.1.1 Stubs und Skeletons Schicht ..... 112
3.2.1.1.2 Remote Reference Schicht ..... 112
3.2.1.1.3 Die Transportschicht ..... 112
3.2.1.1.4 Lokalisierung von verteilten Objekten ..... 113
*3.2.2 Die Datenbankzugriffe* ..... *114*
3.2.2.1 Java Database Connectivity (JDBC) ..... 114
3.2.2.1.1 Driver Manager ..... 114
3.2.2.1.2 Treiber ..... 114
**4 RESÜMEE ..... 115**
4.1 STAND DER IMPLEMENTATION ..... 116

4.2 AUSBLICK ..... 116

**5 ANHANG A: DIE KLASSEN ..... 118**

5.1.1.1 *ResourceManager* ..... 118
5.1.1.1.1.1 Erzeugen einer neuen Resource ..... 118
5.1.1.1.1.2 Löschen einer Resource ..... 118
5.1.1.1.1.3 Holen einer Resource ..... 118
5.1.1.1.1.4 Holen eines Verzeichnisses (Folder) ..... 118
5.1.1.1.1.5 Laden des Inhalts einer Resource ..... 118
5.1.1.1.1.6 Speichern des Inhalts einer Resource ..... 118
5.1.1.1.1.7 Kopieren von Resourcen ..... 119
5.1.1.2 RESOURCE ..... 120
5.1.1.3 ResourceMetadata ..... 122
5.1.1.4 ResourceEditor ..... 124
5.1.1.5 *AccessManager* ..... 124
5.1.1.6 *USER* ..... 126
5.1.1.7 *LinkManager* ..... 127

**6 ANHANG B: GESAMTDIAGRAMM ..... 127**

**7 ANHANG C: MAINSTREAM OBJECTS ..... 128**

**8 ABBILDUNGSVERZEICHNIS ..... 129**

**9 LITERATURVERZEICHNIS ..... 131**

# 1 Einleitung

Das World Wide Web (WWW) ist derzeit der wohl am meisten genutzte Dienst im Internet, ein Dienst zum Abfragen und Abholen von Informationen von einem entfernten Rechner. Man kann es somit der Client/Server[1] Architekturen zuordnen. Die Daten werden von Servern bereitgehalten und bei Anforderung durch die Clients diesen übermittelt, die dann die Daten auf dem Bildschirm darstellen. Die Daten können dabei verschiedenster Art sein, von einfachen Texten über komplette Bücher bis hin zu Bildern und Videos.

Die Entwicklung des WWW begann 1989 am CERN[2] in der Schweiz und sollte den Physikern auf der ganzen Welt die Kommunikation mittels Hypertext ermöglichen. Mit Hilfe von Hypertext werden einzelne Dokumente über (Quer-)Verweise (engl.: [hyper-] link) miteinander verbunden. Aber auch Verweise innerhalb eines Dokumentes und zwischen weltweit verteilten Dokumenten sind möglich. Ein Hyperlink kann auf multimediale Objekte jeglicher Art verweisen.

Bei Printmedien ist die Zeitspanne zwischen Dokumentenerstellung und Konsumierung durch den Leser relativ lang, da die Dokumente erst gedruckt und vertrieben werden müssen. Im Internet sind die Dokumente meistens aktueller, da man sie bereits Sekunden nach der Entstehung lesen kann. Der Leser erwartet deshalb sogar eine höhere Aktualität vom Online-Medium. Deswegen sind die Aktualisierungszyklen wesentlich kürzer als bei Printmedien.

Durch die Möglichkeit der Verbindung der Dokumente mittels Hyperlinks werden diese stark voneinander abhängig. Mit ständig wachsender Anzahl von Dokumenten und deren Kurzlebigkeit wird die Verwaltung der Querverweise immer anspruchsvoller. Während traditionelle Dokumentenverwaltungssysteme [QUAR 96, DYNA 96] mit der Verwaltung statischer, voneinander unabhängiger Dokumente beschäftigt sind, müssen Verwalter von WWW-Angeboten (Webmaster) mit einer großen Anzahl von miteinander verbundenen, sich ständig ändernden Dokumenten zurechtkommen, die zudem von verschiedenen Mitarbeitern mit unterschiedlichen Fähigkeiten und Befugnissen gemeinsam benutzt werden. Es besteht die Gefahr, daß mit der Zeit immer mehr Dokumente Querverweise enthalten, die auf mittlerweile nicht mehr vorhandene Dokumente verweisen. Da das WWW rasch an Popularität gewinnt, immer neue Websites entstehen und diese immer umfangreicher werden, wird das Problem der Verwaltung der Querverweise immer aktueller.

Die Aufgabe der Webmaster und der Mitarbeiter wird dadurch erschwert, daß in einem rasanten Tempo neue Darstellungsmöglichkeiten der Inhalte auf den Markt kommen. Um konkurrenzfähig zu bleiben, sind die Inhaltsanbieter gezwungen, diese auch einzusetzen. Zum Beispiel werden ständig neue HTML-Tags [KENN 97] definiert. Da aber bei WWW-Seiten Darstellungsform und Inhalt nicht getrennt sind, müssen jedesmal große Mengen von Dokumenten bearbeitet werden, um der neuen Darstellungsform gerecht zu werden.

Die Mitarbeiter bei einer Online-Produktion müssen technisch versiert sein, um die Dokumente zu erzeugen. Da technische und journalistische Fähigkeiten meistens nicht in einer Person vereint sind, müssen zwei Fachkräfte für die Erstellung eines Dokuments eingesetzt werden. Diese Arbeitsteilung ist nicht kostengünstig.

[1] Wechselbeziehung zwischen Dienstanforderung und Dienstbereitstellung [ORFA 96]
[2] Europäische Organisation für Kernforschung

Die erfolgreiche Verwaltung einer wachsenden, sich verändernden Website benötigt ein Verwaltungssystem, das sowohl die kooperative Arbeit als auch die Organisation der Dokumente während und nach dem Fertigungsablauf unterstützt. Die hier beschriebene Problematik bezieht sich nicht nur auf Websites, sondern auf alle hypertextbasierten elektronischen Datenhaltungssysteme.

## 1.1 Ziele der Arbeit

Das Ziel dieser Arbeit ist der Entwurf eines Systems, mit dem oben genannte Probleme gelöst werden können: Verwaltung von Querverweisen, Erleichtern der Erstellung von Dokumenten, Zugangskontrolle und Versionskontrolle.

Weiterhin soll die Eignung der Programmiersprache Java und deren Klassenbibliotheken für größere Projekte geprüft werden.

Im Rahmen der Diplomarbeit wird die Arbeitsweise der Mitarbeiter an Web-Projekten analysiert, um daraus einen Entwurf für ein Produktionssystem zu entwickeln. Dieses System soll technisch unbewanderten Menschen die Erstellung ihrer Dokumente ermöglichen. Hierbei soll nur die Erstellung einer Online-Publikation unterstützt werden. Die Diplomarbeit umfaßt nicht die Unterstützung der Print-Publikation.

Die Behandlung der Inhalte während des Online-Betriebes sind nicht Bestandteil der Diplomarbeit. Das Produktionssystem soll lediglich eine Exportfunktion zum öffentlich zugänglichen Server besitzen. Weiterhin ist die Übernahme bereits vorhandener Daten (Legacy Data) nicht Bestandteil der Diplomarbeit. Bewertbare Ergebnisse der Diplomarbeit sind die Konzepte zur Lösung oben genannter Probleme, die Analyse des laufenden Betriebes und die Datenmodellierung, wobei zur Modellierung die Methode der „Mainstream Objects“ [YOUR 96] verwendet wird. Es wird von der Modellierung mit „Mainstream Objects“ abgewichen, wo das Modell sich als unzulänglich erweist. Dies ist z.B. bei der Darstellung von Prozessen und deren Parallelität sowie der Darstellung verteilter Objekte der Fall.

Weiterhin soll eine exemplarische Implementierung des Systems vorgenommen werden, die die Grundfunktionalitäten abdeckt und Schnittstellen für die Ergänzung durch weitere Funktionen bietet. Diese dient als Prototyp, um die Anforderungen der Benutzer besser erfassen zu können. Der bei der Implementierung entstandene Source-Code ist nicht Bestandteil der Bewertung.

Das Produktionssystem soll die Dokumentenverwaltung für die Benutzer vereinfachen. Dazu werden Methoden zur Unterstützung der kooperativen Arbeit eingesetzt. Ein Benutzerzugangsmodell wird erstellt, das die Rollen der Benutzer im Produktionsablauf berücksichtigt.

Ein Schwerpunkt der Arbeit ist die Verwaltung von Querverweisen: Es soll möglich sein, Dokumente zu ändern und zu verschieben, ohne daß die Querverweise manuell geändert werden müssen. Außerdem muß ein Datenmodell erarbeitet werden, um mit Neuentwicklungen auf dem Markt im Bereich der Formate, Standards und Programmen Schritt halten zu können. Das System soll modular aufgebaut werden und klare Schnittstellen besitzen.

Ein weiterer Schwerpunkt ist die Behandlung der Inhalte. Sie sollen so aufbereitet werden, daß die Umstellung in andere Darstellungsformen und Formate reibungslos verläuft.

Diese Arbeit verfolgt den Ansatz der klassischen Programmentwicklung, aufgeteilt in Analyse, Entwurf, und Implementierung. Die Implementierung wird hier nicht beschrieben, liegt aber in Form eines Prototyps vor. Diese Arbeit weicht von einem klassischen Softwareentwurfsdokument ab, da auch der Entwicklungsprozeß beschrieben wird.

Im Kapitel 2.1 findet eine Analyse des laufenden Betriebes statt; das Vokabular der Geschäftsdomäne wird erläutert, und ein Produktvergleich existierender Produktionssysteme wird durchgeführt, um eine Daseinsberechtigung für das neu zu entwickelnde System zu finden.

Im Kapitel 2.2 findet eine Analyse der Systemanforderungen statt, die die benötigte Funktionalität des Systems und von dem Geschäftbereich vorgegebene Bedingungen erfaßt.

Kapitel 3.1 befaßt sich mit dem logischen Entwurf des Systems und beinhaltet folgende Hauptkomponenten: Prozeßübersicht, Objektverwaltung, Zugangskontrolle, Konsistenzverwaltung von Querverweisen, Versionsverwaltung, die Benutzeroberfläche und die Behandlung von Inhalten.

Kapitel 3.4 enthält die technischen Aspekte des Entwurfs und präsentiert die Objektschnittstellen.

Kapitel 4 befaßt sich mit Zukunftsperspektiven und Verbesserungsmöglichkeiten.

Diese Diplomarbeit entsteht in Zusammenarbeit mit der Gruner + Jahr Electronic Media Service GmbH. Das Resultat der Diplomarbeit wird hier bei laufenden Projekten eingesetzt. Dies ermöglicht einen Feldtest des Entwurfs. Die benötigten Hardware und Software-Komponenten werden von G+J zur Verfügung gestellt.

# 2 Analyse

## *2.1 Betriebliche Analyse*

Das System soll einem Print-Verlag ermöglichen, Produkte auf einem Online-Medium umzusetzen. Dazu muß erst einmal der derzeitige Betrieb analysiert werden.
Da im Verlagsbereich einige nicht allgemein bekannte Fachbegriffe verwendet, in der Informatik einige Wörter aus der englischen Sprache übernommen werden, deren Benutzung (Konjugation und Deklination) nicht definiert ist, sind vorweg einige Definitionen vorzunehmen.

Klassennamen, Objektnamen und Funktionen werden in dieser Arbeit *kursiv* geschrieben. Fachbegriffe aus dem Englischen werden hier, soweit es für die Verständlichkeit sinnvoll ist, übersetzt. Ansonsten werden die Begriffe in diesem Kapitel kurz erläutert und im Folgenden verwendet. Das Wort „Resource“ wird hier mit einem „s“ geschrieben, um es von dem deutschen Wort „Ressource“ zu unterscheiden. In diesem Schriftstück ist mit dem Wort „Resource“ ein Datenobjekt gemeint. Um fragwürdige Wortbildungen zu vermeiden, wird „Resource“ trotzdem wie „Ressource“ dekliniert.

### 2.1.1 Das Vokabular des Online-Publishing

**Online-Publishing:** Publizieren für ein Online-Medium z.B. Internet, BTX, etc.
**Standard Generalized Markup Language (SGML):** Eine Strukturbeschreibungssprache für Dokumente (siehe 3.1.13.1.4).
**Document Type Definition (DTD):** Die Definition der erlaubten Struktur eines SGML-Dokumentes.
**Client/Server System:** Ein System bestehend aus vernetzten Rechnern, in dem ein Rechner die Rolle des Dienstanbieters (Server), und mehrere Rechner die Rolle des Dienstanforderers (Client) besitzen.
**World Wide Web (WWW) :** Ein weltweit verteiltes System zur Bereitstellung multimedialer Inhalte.
**Multicasting** : Das Verschicken von Nachrichten an mehrere Clients.
**Firewall:** Ein System, das den Zugang von außerhalb auf das interne Netz einer Institution kontrolliert, um unbefugten Zugang zu sperren.
**Hyperlink:** Ein Querverweis innerhalb eines Hypertext-Dokumentes.
**Website:** Ein virtueller Ort, der die WWW-Seiten einer oder mehrerer Publikationen enthält und über eine Adresse erreichbar ist.
**Webmaster:** Verwalter einer Website.
**HTTP-Server:** Ein Rechner, der die Website beinhaltet.
**HTML-Tags:** Steuerzeichen im Text einer WWW-Seite, die das Layout der Seite bestimmen.
**Browser:** Ein Programm zur Navigation und Ansicht von Datenbeständen. Hier: WWW-Browser zum Ansehen von und Navigieren durch Websites.
**Browsen:** Navigieren durch Datenbestände mit Hilfe eines Browsers.
**Folder:** Verzeichnis. Hier: ein Verzeichnisobjekt.
**Java:** Eine objektorientierte Programmiersprache, die vornehmlich im Internetbereich eingesetzt wird.

**Director (von Macromedia):** Ein Programm zur Erstellung von interaktiven Multimedia-Anwendungen, die auch im Internet eingesetzt werden können.

### 2.1.2 Das Vokabular der Geschäftsdomäne

**Art Director (AD):** Graphiker, der für das Layout und die optische Konsistenz (Corporate Image CI) der gesamten Publikation zuständig ist.
**Chef vom Dienst (CvD):** Zuständiger für das Endprodukt der Publikation.
**Öffentlicher Server, „Heißer Server" (Engl. Staging Server):** Server, dessen Inhalte der Öffentlichkeit zugänglich gemacht werden.
**Produktionsserver (Engl. Production Server):** Server, auf dem produziert wird.
**Publizieren, „Heißschalten", Veröffentlichen:** Das Kopieren des Dokuments aus dem Produktionsbereich auf einen Server, der der Öffentlichkeit zugänglich ist.
**Abnehmen:** Die Genehmigung eines Dokuments durch den CvD für das Publizieren. Auch Akzeptieren (accept) genannt.
**Layout:** Die Bezeichnung für die graphische Struktur eines Dokuments.
**WYSIWYG:** what you see is what you get; Darstellung eines Layouts auf dem Bildschirm, so wie es später in der Ausgabeform erscheint.

### 2.1.3 Akteure und Aufgaben

Erster Schritt der Analyse des Betriebes ist die Feststellung, welche Akteure an diesem System beteiligt sind und welche Funktionalitäten sie benötigen:

#### 2.1.3.1 Programmierer (Online Producer)

Er ist für die technische Realisierung interaktiver Bestandteile des Projektes zuständig.

**benötigte Funktionen:**

- Definition von DTDs (siehe 3.1.13.1.4.1).
- Entwurf von interaktiven Applikationen.
- Verwaltung von Querverweisen.
- Neukompilierung des gesamten Inhalts zwecks Layoutänderung.
- Publizieren von Dokumenten.
- Einbetten und Editieren von Objekten (Java, Sound, Video, Director, usw.).

#### 2.1.3.2 Administrator

Er ist für die Betreuung des Produktionssystems zuständig.

**benötigte Funktionen:**

- Benutzerverwaltung.
- Datensicherung.
- Fehlerbehebung.
- Konsistenzüberprüfung der Produktion.

#### 2.1.3.3 Redakteur (Content Provider)

Er ist für die Inhalte des Sites zuständig.

**benötigte Funktionen:**

- Texte editieren und in Dokumente einbinden.
- Querverweise in Dokumente einfügen.
- Dokumente publizieren.

#### 2.1.3.4 Online Graphiker (Online Artist)

Er ist für die Graphiken der Produktion zuständig.

**benötigte Funktionen:**

- Bilder erstellen.
- Die Bilder in Dokumente einfügen.
- Die zugehörigen Bilder in das richtige Format konvertieren und skalieren(soll möglichst automatisch vom Produktionssystem übernommen werden).

#### 2.1.3.5 Projektleiter (Product Manager)

Er ist für das Gesamtkonzept der Produktion und deren zeitgemäße Fertigstellung zuständig.

#### 2.1.3.6 Layouter (Art Director)

Er ist für die optische Konsistenz der Produktion zuständig.

**benötigte Funktion:**

DTDs für die Standard Layouts festlegen (in Zusammenarbeit mit dem Online Producer).

#### 2.1.3.7 Chef vom Dienst (CvD)

Er ist für die Qualitätssicherung zuständig.

**benötigte Funktionen:**

- Die Abnahme von Dokumenten.
- Das Publizieren von Dokumenten.

#### 2.1.3.8 Externer Dienstleister

Er liefert Texte, Bilder, Multimedia-Anwendungen (z.B. Applets)

**benötige Funktion:**

Online-Eingabe von Texten, Bildern und anderen Inhalten zur Weiterverarbeitung durch interne Mitarbeiter.

### 2.1.4 Arbeitsvorgänge

Bei der Produktion einer Online-Publikation in einem Verlag arbeiten die Print-Redakteure Hand in Hand mit der eigens für diese Aufgabe zusammengestellten Online-Redaktion zusammen.

Die Online-Redaktion besteht aus:

1. dem Projektleiter, der für die Konzeption der Online-Publikation verantwortlich ist. Er arbeitet eng mit dem Chef vom Dienst (CvD) des Print-Mediums zusammen.

1. den Online-Producern, die für die technische Umsetzung der Entwürfe verantwortlich sind.
1. den Online-Graphikern, die für die Umsetzung und Bearbeitung der Bilder für das Online-Medium verantwortlich sind. Diese Graphiker müssen technisch bewandert sein.

Die Print-Redakteure liefern die für die Online-Publikation vorgesehenen Inhalte an die Online-Redaktion. Die Online-Redaktion bietet ihre Dienstleistungen verschiedenen Print-Redaktionen an. Es werden auch in geringem Umfang von der Online-Redaktion Artikel verfaßt, die Mitarbeiter sind aber zum großen Teil für die technische und sinnvolle Umsetzung der Inhalte für das Online-Medium verantwortlich. Es gibt keine klare Arbeitsteilung, und ein Workflow-Konzept existiert nicht.

Abbildung 2-1 gibt einen Überblick über die Geschäftsobjekte und deren Beziehungen:

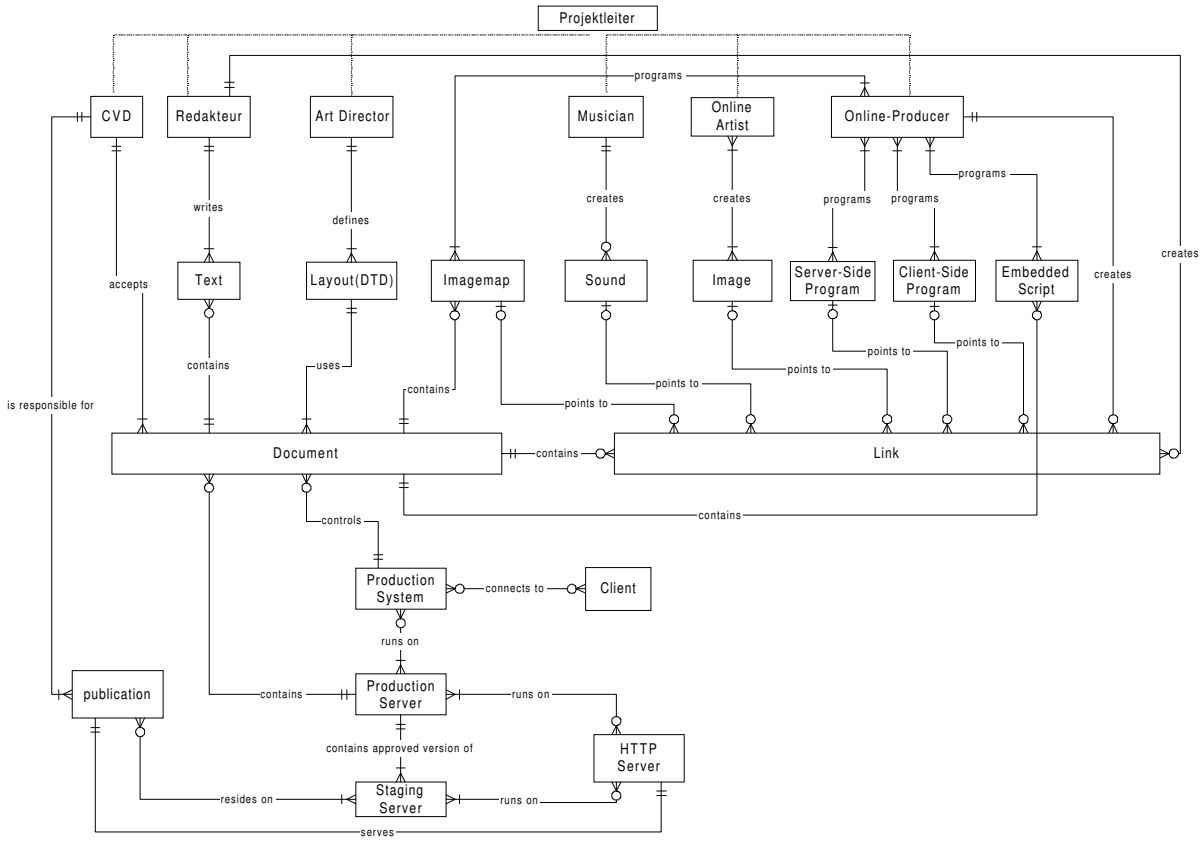



**Abbildung 2-1 Geschäftsobjekte bei der Gruner + Jahr Online Produktion**

Die Dokumente, Bilder u.ä. einer Online-Produktion, von nun an Resource genannt, durchlaufen mehrere Entwicklungsphasen. Die verschiedenen Phasen werden in Abbildung 2-2 dargestellt.

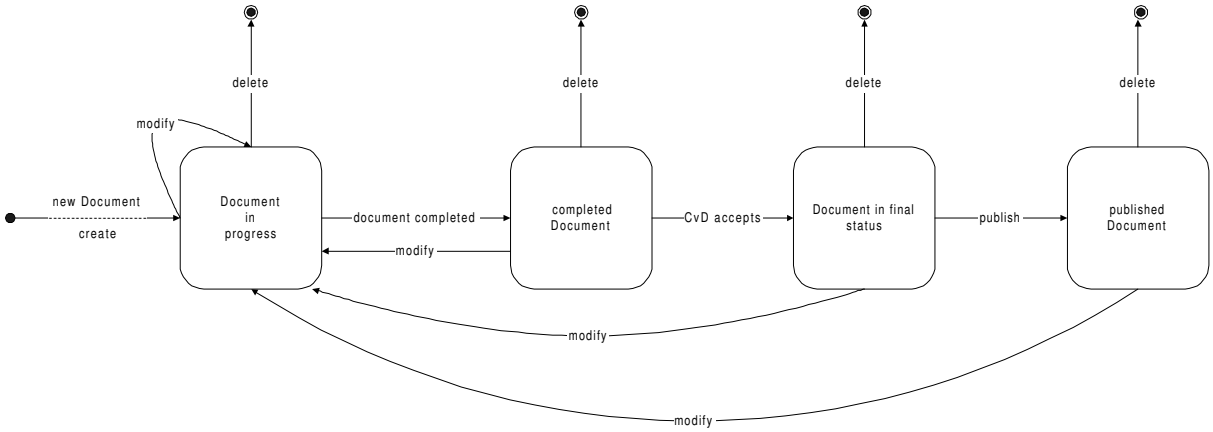



**Abbildung 2-2 Der Fertigungsablauf für Online-Dokumente**

Ein Dokument wird vom Verfasser (Redakteur, Graphiker) erstellt. Wenn dieses Dokument fertiggestellt ist, muß es vom CvD abgenommen werden. Nach der Abnahme kann das Dokument dann publiziert („heißgeschaltet“) werden. Wenn das Dokument wieder geändert wird, ist es dann erneut vom CvD abzunehmen. Dieser Vorgang soll im Produktionssystem technisch unterstützt werden.

Für die Erstellung der Online-Publikationen werden bei Gruner + Jahr zwei Server eingesetzt:
Ein Produktionsserver, auf dem produziert und getestet wird, und ein sog. öffentlicher („heißer“) Server, der öffentlich zugänglich ist. Der Produktionsserver ist hinter der Firewall vor fremden Zugriffen geschützt. Nachdem die Dokumente fertiggestellt und getestet sind, werden sie auf den öffentlichen Server, der außerhalb der Firewall liegt, transferiert. Der Grund für diese Trennung liegt in der Qualitätssicherung: Da die Dokumente auf dem öffentlichen Server sofort von der Öffentlichkeit angesehen werden können, müssen sie vorher an einem anderen Ort unter gleichen

Bedingungen getestet werden. Um eine realistische Testumgebung zu erzeugen, ist die Konfiguration des Produktionsservers identisch mit der des öffentlichen Servers.

Die Online-Producer benutzen verschiedene Betriebssysteme wie Mac, Windows, und Unix als Clients. Alle erzeugten Dokumente residieren auf dem Server; eine lokale Speicherung der Dokumente findet nicht statt. Die Dokumente werden mit Standardprogrammen zur Bild- und Textbearbeitung (Photoshop, Word u.a.) erzeugt und dann auf dem Server gespeichert. Die Mitarbeiter benutzen als Vorlage für ein neues Dokument häufig eine schon vorhandene Seite, um wiederkehrende Arbeiten nicht wiederholen zu müssen.

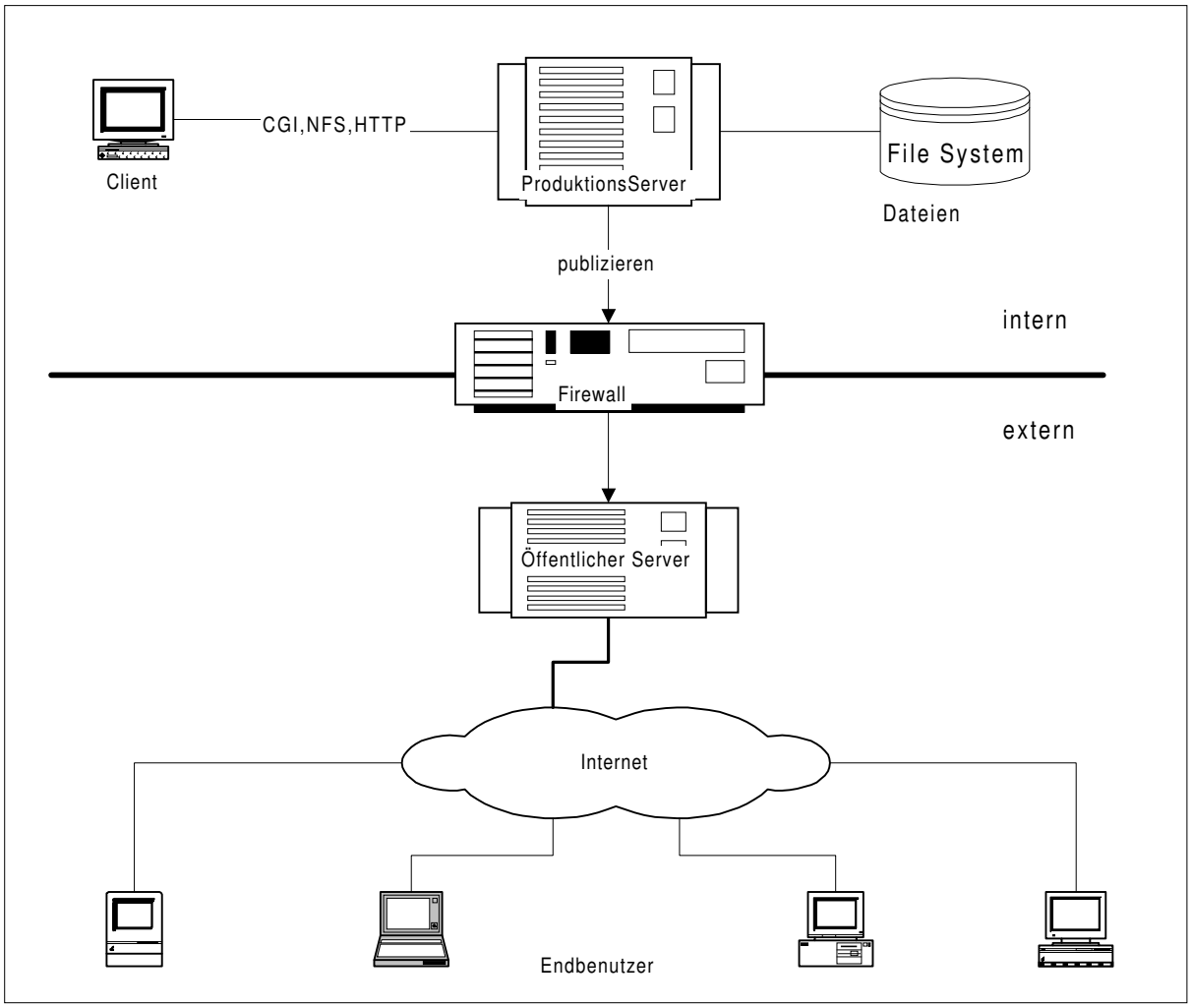



**Abbildung 2-3: Systemüberblick des aktuellen Systems**

Die Texte, die von der Print-Redaktion geliefert werden, werden über das interne Windows- NT-Netz als reine Textdateien an die Online-Redaktion weitergegeben. Bilder werden von der Print-

Version des Artikels eingescannt, obwohl sie bereits bei der Erstellung in elektronische Form vorliegen. Der Grund für diesen Medienbruch ist, daß beim Layouten der Hefte mit Bildern niedriger Auflösung gearbeitet wird, um Speicherplatz zu sparen. Nachdem das Layout fertig ist, wird es über das Gruner + Jahr Netz zur Druckzentrale geschickt, wo dann die Original-Dias direkt vor dem Drucken in hoher Qualität eingelesen werden. Da die Qualität der Bilder, die für das Layout benutzt werden, für das Online-Medium zu gering ist, werden die Bilder dann vom Heft wieder eingescannt. Die Beschaffung der Original-Dias würde einen unverhältnismäßigen Mehraufwand bedeuten, da die Druckzentrale entfernt gelegen ist und häufig die Bilder noch direkt vor dem Drucken ausgetauscht werden.

Die Kommunikation der Einzelplatzrechner mit dem Entwicklungsserver geschieht durch zwei Protokolle: Entweder über NFS[3], oder durch das nachfolgend beschriebene Produktionssystem WebIza.

### 2.1.5 Das existierende Produktionssystem WebIza

Das existierende System besteht aus der Netscape Browser- und Server-Software und einem CGI[4]-Skript. WebIza erleichtert das Editieren der Dokumente auf dem Server. Der Netscape-Browser kommuniziert durch CGI mit dem Server. Der Benutzer muß ein Paßwort eingeben, um das System benutzen zu können. Alle Benutzer haben gleiche Rechte auf das System und benutzen das gleiche Paßwort.

WebIza bietet folgende Funktionalitäten an:
1. Browsen im Filesystem des HTTP-Servers[5].
1. Hoch- und Runterladen von Dokumenten.
1. Editieren von Textdokumenten im Browserfenster.

Auf der Server-Seite wird ein Skript benutzt, das von der Redaktion selbstdefinierte Makros in den entsprechenden HTML-Code[6] umsetzt. Auf diese Weise können globale Veränderungen in den Dokumenten leichter durchgeführt werden.
Das Redaktionssystem erleichtert den Transfer der Inhalte auf den öffentlichen Server durch die Firewall.

[3] Network File System [CALL 95]
[4] Common Gateway Interface. [BREN 96]
[5] Ein Serverprogramm, das HTML-Seiten liefert. [KENN 97]
[6] Hypertext Markup Language. Formatierungssprache von WWW-Dokumenten. [KENN 97]

Die Hyperlinks verweisen auf ein CGI-Skript, das die angeforderten Funktionen ausführt und eine HTML-Seite mit dem Ergebnis erzeugt. Diese Seite wird dann zum Anzeigen wieder zum Browser geschickt. Da das HTTP-Protokoll[7] verwendet wird, handelt es sich hier um eine verbindungslose Kommunikation. Hier ein Beispiel für die Benutzeroberfläche von Webiza:

[7] Hypertext Transfer Protocol. [FIEL 97]

**Abbildung 2-4 WebIza Browser Fenster**

Zum Editieren der Seiten wird ein HTML-Textfeld verwendet. In diesem kann man, wie von einem Texteditor gewohnt, den Text editieren. Meist wird mit einem Standard Texteditor wie z.B. Word gearbeitet und das Ergebnis per Cut & Paste kopiert oder als Datei in das Textfeld geladen. Durch Betätigung der Taste „Store“ wird dann der Inhalt des Textfeldes per CGI zum Server geschickt. Das CGI-Skript konvertiert die Makros in HTML und speichert die Seite in das entsprechende Verzeichnis. Dabei werden zwei identische Verzeichnisse verwaltet: zum einen die der Sourcedokumente (mit Makro-Code) und zum zweiten die der reinen HTML-Dokumente, die im Hauptverzeichnis des HTTP-Servers liegen.

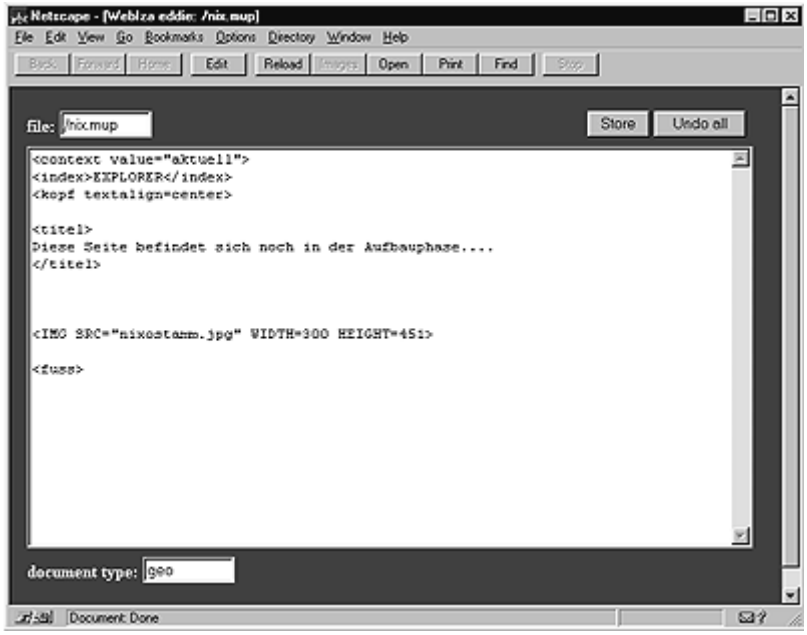



**Abbildung 2-5Das Webiza Editor-Fenster**

### 2.1.5.1 Nachteile von WebIza

Dieses System ist sehr primitiv und wird den Anforderungen einer größeren Online-Publikation nicht gerecht.

**Die Nachteile des WebIza Produktionsystems sind:**

- Alle Benutzer benutzen das gleiche Paßwort, um in das System zu gelangen. Dadurch haben alle Benutzer gleiche Rechte.
- Es existiert kein Link Management System, das die Konsistenz der Hyperlinks überprüft und wenn möglich korrigiert.
- Concurrency Control ist nicht vorhanden, d.h. es können mehrere Benutzer gleichzeitig an einem Dokument arbeiten, ohne daß auf das Problem der Überschreibung(siehe 3.1.10.2) geachtet wird.

- Es gibt keine Workflowunterstützung: Die Dokumente haben keine Zustände und können nicht entsprechend behandelt werden.
- Die Benutzeroberfläche ist für einen technisch unbedarften Anwender nicht einfach zu verwenden.

### 2.1.6 Produktvergleich: kommerzielle Produktionssysteme

Bei dem Vergleich von kommerziellen Produktionssystemen wurden folgende Produkte getestet:

- Netscape Publishing System.[NETS 96]
- Microsoft FrontPage. [MICR 96]
- Netobjects Fusion. [NETO 96]
- Dynabase Publishing System. [DYNA 96]

Die Vorgehensweise dieses Vergleichs war folgendermaßen: Sobald ein Merkmal des Produktes nicht zufriedenstellend war, wurde der Test des Produkts abgebrochen, da dieser Vergleich nur dazu dient, um festzustellen, ob die Eigenentwicklung eines Produktionssystems sinnvoll ist. Deswegen werden hier nur die negativen Eigenschaften der Systeme beschrieben. Dieser Test soll nicht als eine Bewertung der Produkte angesehen werden. Da zur Anfangszeit der Diplomarbeit diese Systeme noch in der Entwicklungsphase waren, sind diese Vergleiche nicht mehr maßgebend.

#### 2.1.6.1 Nescape Publishing System

Dieses System bietet eine komplette Lösung für Web Publishing mit Produktionunterstützung, Abrechnung für angesehene Dokumente u.a.
Nachteil dieses System ist, daß es HTML-basiert ist und damit nicht für die Ausgabe verschiedener Formate geeignet ist. Es bietet außerdem keine übergreifende Verwaltung der Querverweise zwischen Produktions- und öffentlichem Server.

#### 2.1.6.2 Microsoft Frontpage

Dieses Produkt ist für die Produktion kleiner Websites gedacht und unterstützt ebenfalls nur HTML-roduktion. Sie bietet keinerlei Workflowunterstützung an und ist zum Zeitpunkt dieser Bewertung nicht multiuserfähig. Das System ist nur für Windows erhältlich. Versionsmanagementsoftware muß zusätzlich gekauft werden.

#### 2.1.6.3 Netobjects Fusion

Dieses System besticht durch einen sehr guten WYSIWYG-Editor[8] für HTML-Seiten. Leider bietet dieses System zum Zeitpunkt des Tests keine Multiuserfähigkeit an. Es gibt keine Version für Server-Betriebsysteme wie Unix, sondern sie ist für Einzelplatzrechner gedacht.

#### 2.1.6.4 Dynabase Publishing System

[8] Ein Editor der den Inhalt so darstellt, wie er nachher auch im Ergebnis zu sehen sein wird.

Dynabase ist ein mächtiges, SGML-basiertes Produktionssystem, das sowohl Printmedien als auch Online-Medien unterstützt. Die Struktur der SGML-Dokumente ist dabei mit einem Index versehen, so daß man nach jedem definierten SGML-Tag die Datenbank durchsuchen kann.
Es wird leider nicht zwischen Produktionsserver und öffentlichem Server unterschieden. Hier wird zwar ein System angeboten, in dem nur die fertiggestellten Dokumente für die Benutzer zugreifbar sind, aber dadurch, daß alles auf einem Server stattfindet, muß der Server außerhalb der Firewall stehen und ist damit nicht geschützt.
Es gibt andere kommerzielle Lösungen für Online-Publishing Systeme, die aber alle einen oder mehrere von den folgenden aufgelisteten Mängel aufweisen:

- Es wird nicht zwischen Produktionsserver und öffentlichem Server unterschieden.
- Es wird HTML als Sprache verwendet und nicht eine übergeordnete Sprache wie SGML[9]: Dadurch wird es schwieriger, gleichzeitig für mehrere Ausgangsformate zu produzieren.
- Sie bieten kein Multiusersystem mit Zugangskontrolle an.
- Die Systeme unterstützen meist nur eine Plattform oder es muß für diese verschiedenen Plattformen neue Software gekauft werden.
- Sie sind sehr instabil gegenüber manuellen Veränderungen der Dokumente: Man kann neue Darstellungsmöglichkeiten, z.B. in HTML, nicht ausnutzen, wenn die Produktionssoftware diese Fähigkeit nicht kennt.
- Sie besitzen keine Versionsverwaltung.
- Sie sind nicht leicht erweiterbar.

Aus dieser Situation heraus entstehen die Anforderungen an das neu zu entwickelnde Produktionssystem.

## *2.2 Analyse der Systemanforderungen*

Folgende Merkmale muß das Produktionssystem aufweisen.

- Benutzerzugangskontrolle, die zwischen Benutzergruppen unterscheidet.
- Verwaltung und Konsistenzüberprüfung von Querverweisen (Link Management).
- Trennung in Produktions- und öffentlichem Server mit übergreifendem Link-Management.
- Versionsverwaltung.
- Modularer Aufbau, leicht erweiterbar.
- Zugangskontrolle.
- Plattformunabhängigkeit.
- Verwendung einer generischen Ausgangssprache, um mehrere Ausgangsformate zu ermöglichen.
- Workflowunterstützung.
- Eine anwenderfreundliche Benutzeroberfläche.

[9] Standard Generalized Markup Language (siehe 3.1.13.1.4) [GOLD 95, COLB 96, STAN 96]

Da der Verlag das Produktionssystem mit seinen vorhandenen Systemen verwenden will, entstehen folgende Vorgaben.

### 2.2.1 Hardwarevorgaben

Der Client muß auf verschiedenen Betriebssystemen laufen. Dazu gehören hauptsächlich: Windows, Mac, Unix. Diese Rechner haben meist geringe Ressourcen. Deswegen sollte die Hauptlast beim Server liegen.
Die Client Software sollte möglichst wenig Wartungsbedarf haben. Sie sollte durch wenig Konfiguration auf dem Client Rechner lauffähig sein. Die Rechner haben meistens kleine Bildschirme und müssen mit wenig Bildschirmplatz auskommen. Da verschiedene Bereiche auch verschiedene Serverplattformen[10] benutzen, muß der Server auch multiplattformfähig sein.
Es sollte möglich sein, das System auf einem Rechner zu installieren, während die Daten in einer Datenbank auf einem anderen Rechner liegen. Dadurch kann die Datensicherung zentral verwaltet werden.

### 2.2.2 Softwarevorgaben

Gruner + Jahr Electronic Media Service (EMS) hat nach einer Marktrecherche die Programmiersprache Java [CORN 96] für dieses Projekt bestimmt.
Java ist eine von Sun Microsystems entwickelte Programmiersprache zum Erstellen interaktiver und verteilter Anwendungen, die der Sprache C++ sehr ähnlich ist. Die Programme sind plattformunabhängig, d.h. sie laufen auf jedem Rechner.

Da das System den Zugriff von mehreren Benutzern auf gemeinsam benutzte Resourcen verwalten muß, eignet sich hier das Client/Server-Modell. Dabei werden alle Daten auf dem Server zentral gespeichert. Für die Verwaltung der Daten wird hier eine relationale Datenbank benutzt.

[10] Hardware-Betriebssystem-Kombination des Servers.

### 2.2.3 Anwendungsfälle für die Benutzerinteraktion

Im Folgenden werden die benötigten Hauptfunktionen des Systems definiert.

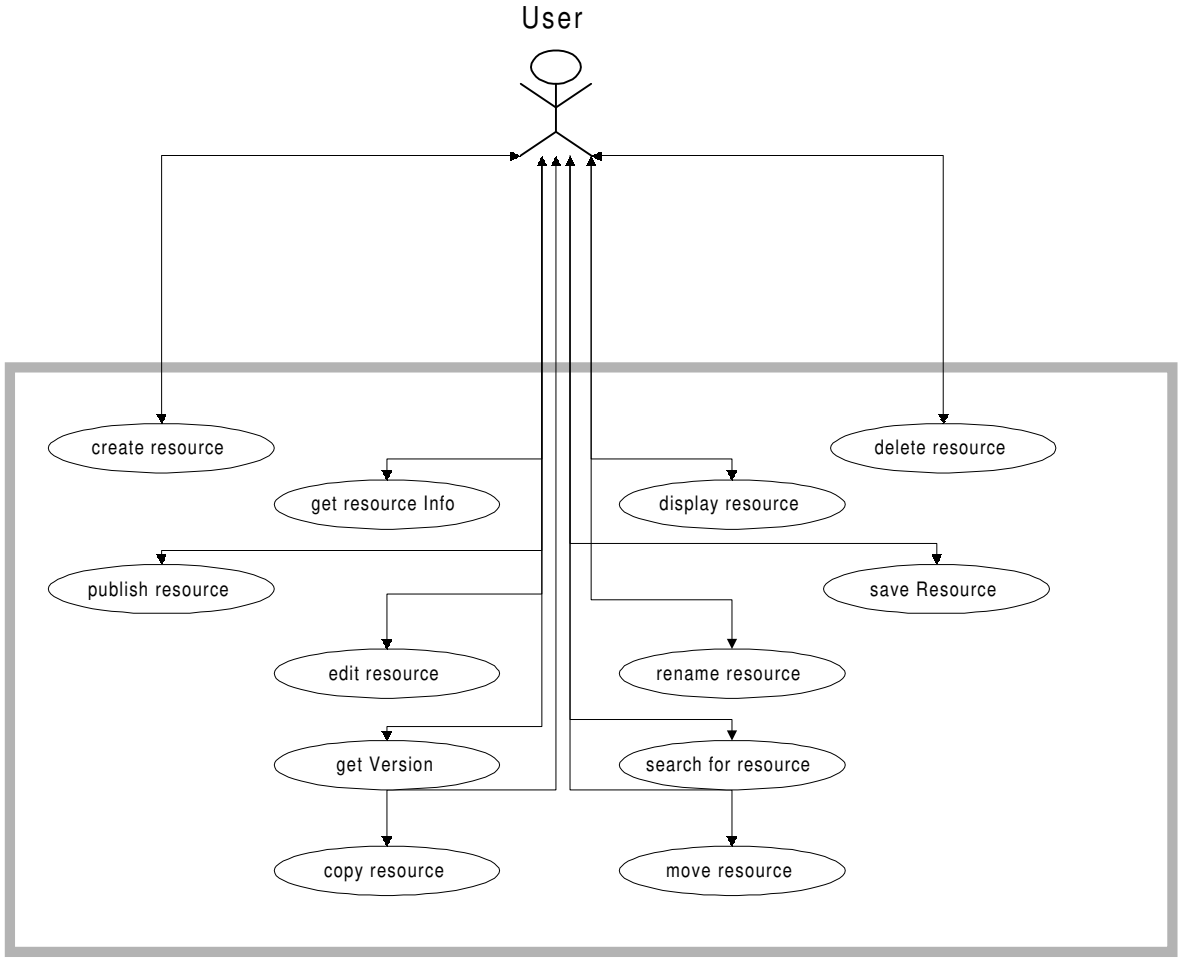



**Abbildung 2-6Anwendungsfälle für den Benutzer**

Die Handhabung dieser Funktionen wird im Folgenden detaillierter beschrieben.

#### 2.2.3.1 *Erzeugen einer Resource*

**Kurztext**
Der Benutzer will eine neue Resource erzeugen.

**Beschreibung**
- Der Benutzer muß ein Verzeichnis auswählen.
- Der Benutzer muß Schreibrechte in dem Verzeichnis besitzen.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, *SoundResource* etc.

**Standardpfad**

1. Der Benutzer wählt ein Verzeichnis aus dem Verzeichnisbaum aus.
1. Durch Betätigung der rechten Maustaste erscheint ein Popup-Menü.
1. Der Benutzer selektiert die Option „create“ im Menü.
1. Der Benutzer selektiert den Objekttyp im „create“ Untermenü.
1. Es wird nach einem Namen und der Beschreibung für die Resource gefragt.
1. Ein Editierfenster für die Resource erscheint.
1. Der Benutzer kann jetzt Daten eingeben. Dies geschieht entweder durch Drag & Drop, Cut & Paste oder durch Laden einer Datei.
1. Durch Betätigung der „save“ Taste wird die Resource gespeichert. Das Ergebnis wird in einem Preview-Fenster angezeigt. Der Benutzer kann so lange editieren und speichern, bis das gewünschte Ergebnis im Preview-Fenster angezeigt wird.

#### 2.2.3.2 *Editieren einer Resource*

**Kurztext**

Der Benutzer will eine Resource editieren.

**Beschreibung**

- Der Benutzer muß Schreibrechte für die Resource besitzen.
- Die Resource darf nicht ausgecheckt sein.

**Standardpfad**

1. Der Benutzer wählt aus dem Verzeichnisbaum die gewünschte Resource aus.
1. Durch einen Doppelklick auf die Resource wird ein Editorfenster aufgerufen.
1. Die Resource wird vom System ausgecheckt.
1. Der Benutzer kann jetzt Daten eingeben. Dies geschieht entweder durch Drag & Drop, Cut & Paste oder durch Laden einer Datei.
1. Durch Betätigung der „save“ Taste wird die Resource gespeichert. Das Ergebnis wird in einem Preview-Fenster angezeigt. Der Benutzer kann so lange editieren und speichern, bis das gewünschte Ergebnis im Preview-Fenster angezeigt wird.
1. Der Benutzer schließt den Editor.
1. Die Resource wird vom System eingecheckt.

**Wiederverwendbare Sequenzen**

- Check-in
- Check-out

#### 2.2.3.3 *Verschieben einer Resource*

**Kurztext**

Der Benutzer will eine Resource verschieben.

**Beschreibung**
- Der Benutzer muß Schreibrechte für die Resource und in dem Verzeichnis haben, in dem die Resource verschoben wird.
- Die Resource darf nicht ausgecheckt sein.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Er verschiebt die Resource durch Drag & Drop in das gewünschte Verzeichnis.
1. Es erscheint ein Fenster, im dem der Benutzer zwischen drei Tasten wählen kann: „move“ , „copy“ und „cancel“. Der Benutzer wählt „move“.
1. Die Resource wird vom System ausgecheckt.
1. Die Resource wird verschoben.
1. Die Resource wird vom System eingecheckt.

**Der Alternativpfad**
Der Benutzer betätigt den „cancel“ -Knopf. Der Vorgang wird abgebrochen.

**Wiederverwendbare Sequenzen**
- check-in
- check-out

#### 2.2.3.4 *Kopieren einer Resource*

**Kurztext**
Der Benutzer will eine Resource kopieren.

**Beschreibung**
Der Benutzer muß Leserechte für die Resource haben.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer verschiebt die Resource durch Drag & Drop in das gewünschte Verzeichnis.
1. Es erscheint ein Fenster, in dem der Benutzer zwischen drei Knöpfe wählen kann: „move“ , „copy“ und „cancel“. Der Benutzer wählt „copy“.
1. Die Resource wird vom System ausgecheckt.
1. Die Resource wird kopiert, und es findet eine Korrektur der Querverweise statt (siehe 3.1.11 ).
1. Die Resource wird vom System eingecheckt.

**Der Alternativpfad**
Der Benutzer betätigt den „cancel“ -Knopf. Der Vorgang wird abgebrochen.

**Wiederverwendbare Sequenzen**
- check-in
- check-out

#### 2.2.3.5 *Löschen einer Resource*

**Kurztext**
Der Benutzer will eine Resource löschen.

**Beschreibung**
- Der Benutzer muß Schreibrechte für die Resource haben.
- Die Resource darf nicht ausgecheckt sein.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup-Menü erscheint.
1. Der Benutzer selektiert „delete“. Wenn die Resource bereits ausgecheckt ist, bekommt der Benutzer eine Rückmeldung, daß die Resource gerade vom Benutzer B bearbeitet wird und deswegen nicht gelöscht werden kann.
1. Die Resource wird in das Papierkorb-Verzeichnis verschoben.

**Der Alternativpfad**
1. Der Benutzer kann die Resource wiederherstellen.
1. Der Benutzer klickt auf den Papierkorb und ein Popup-Menu erscheint.
1. Der Benutzer selektiert „restore“. Die Resource wird wiederhergestellt.

#### 2.2.3.6 Umbenennen einer Resource

**Kurztext**
Der Benutzer will eine Resource umbenennen.

**Beschreibung**
- Der Benutzer muß Schreibrechte für die Resource haben.
- Die Resource darf nicht ausgecheckt sein.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup-Menü erscheint.

1. Der Benutzer selektiert „Rename“. Wenn die Resource bereits ausgecheckt ist, erscheint eine Nachricht, daß die Resource nicht umbenannt werden kann, weil Benutzer B diese Resource bearbeitet.
1. Ein Eingabefeld erscheint. Der Benutzer gibt den neuen Namen der Resource ein.
1. Die Resource wird umbenannt.

**Wiederverwendbare sequenzen**
move

#### 2.2.3.7 Publizieren einer Resource

**Kurztext**
Der Benutzer will eine Resource publizieren (heißschalten).

**Beschreibung**
- Der Benutzer muß Schreibrechte für die Resource besitzen.
- Die Resource darf nicht ausgecheckt sein.
- Die Resource muß vom CvD abgenommen sein.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup- Menü erscheint.
1. Der Benutzer selektiert „Publish“. Wenn die Resource bereits ausgecheckt ist, erscheint eine Nachricht, daß die Resource nicht umbenannt werden kann, weil Benutzer B diese Resource bearbeitet. Wenn die Resource nicht vom CvD abgenommen ist, erscheint eine entsprechende Fehlermeldung.
1. Ein Kalender mit Uhrzeit erscheint. Hier kann man Datum und Uhrzeit für das Publizieren des Dokuments angeben, oder die sofortige Publikation veranlassen.
1. Die Resource wird zur genannten Uhrzeit publiziert.

**Der Alternativpfad**
Wenn die Resource vor dem Publizieren editiert wird, wird sie solange nicht publiziert, bis sie vom CvD wieder abgenommen worden ist.

#### 2.2.3.8 Anfordern von Informationen über eine Resource

**Kurztext**
Der Benutzer will Informationen über einen Resource erhalten.

**Beschreibung**

- Der Benutzer muß Leserechte für die Resource besitzen.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup-Menü erscheint.
1. Der Benutzer selektiert „Info“.
1. Ein Informationsfenster erscheint, das Informationen über Autor, Datum der letzten Änderung, Größe und Typ des Dokuments und weitere Informationen enthält.

#### 2.2.3.9 Suchen nach einer Resource

**Kurztext**
Der Benutzer will eine bestimmte Resource suchen.

**Beschreibung**
- Der Benutzer muß ein Stichwort zum Suchen eingeben.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer klickt mit der rechten Maustaste auf das Verzeichnissymbol, in dem er suchen will, und ein Popup-Menü erscheint.
1. Der Benutzer selektiert „Search“.
1. Ein Eingabefeld erscheint. Der Benutzer kann eine Stichwortsuche starten.
1. Das Wort wird bei Texten sowohl im Beschreibungsfeld als auch im Volltext gesucht. Bei Bildern, Tönen etc. wird im Beschreibungsfeld gesucht.
1. Eine Liste mit den Suchergebnissen wird zurückgeliefert.

#### 2.2.3.10 Anzeigen einer Resource

**Kurztext**
Der Benutzer will eine Resource anzeigen.

**Beschreibung**
- Der Benutzer muß Leserechte für die Resource besitzen.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zum gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup-Menü erscheint.
1. Der Benutzer selektiert „Display“.

Die Resource wird in einem entsprechenden Fenster angezeigt.

#### 2.2.3.11 Abspeichern einer Resource

**Kurztext**
Der Benutzer will eine Resource abspeichern.

**Beschreibung**
- Der Benutzer muß Schreibrechte für Resource besitzen.
- Das Dokument muß vom Benutzer schon geöffnet sein.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer betätigt die „Save"- oder „Save as" -Taste im Editor-Fenster.
1. Bei Betätigung von „Save" wird die Resource gespeichert und das Endergebnis angezeigt. Das Ergebnis wird im Defaultformat des Servers gespeichert (meistens HTML). Beim Drücken von „Save As" erscheint ein Dialog, in dem der Benutzer das Ausgabeformat und Speicherort angeben kann.
1. Der Benutzer drückt „OK".

Die Resource wird im entsprechenden Format gespeichert.

#### 2.2.3.12 Wiederherstellen einer früheren Version der Resource

**Kurztext**
Der Benutzer will eine bestimmte Version der Resource editieren.

**Beschreibung**
- Der Benutzer muß Schreibrechte für die Resource besitzen.
- Die Resource darf nicht ausgecheckt sein.
- Resourcen sind z.B. *Folder*, *TextResource*, *ImageResource*, etc.

**Standardpfad**
1. Der Benutzer browst durch den Verzeichnisbaum zu der gewünschten Resource.
1. Der Benutzer klickt mit der rechten Maustaste auf das Resourcesymbol, und ein Popup-Menü erscheint.
1. Der Benutzer selektiert „Get Version".
1. Ein Auswahlfenster erscheint, in dem die verfügbaren Versionen angezeigt werden.
1. Der Benutzer kann per Mausklick eine Version auswählen.
1. Die ausgewählte Version wird die aktuelle Version.
1. Ein Editor-Fenster wird geöffnet.
1. Der Benutzer speichert das Ergebnis durch Betätigung von „Save".

**Der Alternativpfad**
Der Benutzer speichert das Ergebnis nicht. Dann wird die Versionsinformation nicht geändert.

**Wiederverwendbare Sequenzen**
Save.

### 2.2.4 Anwendungsfälle für den Administrator

Für diese Fälle wird vorausgesetzt, das der Administrator sich mit einer Administratorkennung beim System angemeldet hat. Er bekommt dann im Browser-Fenster die Administrationssymbole zu sehen.

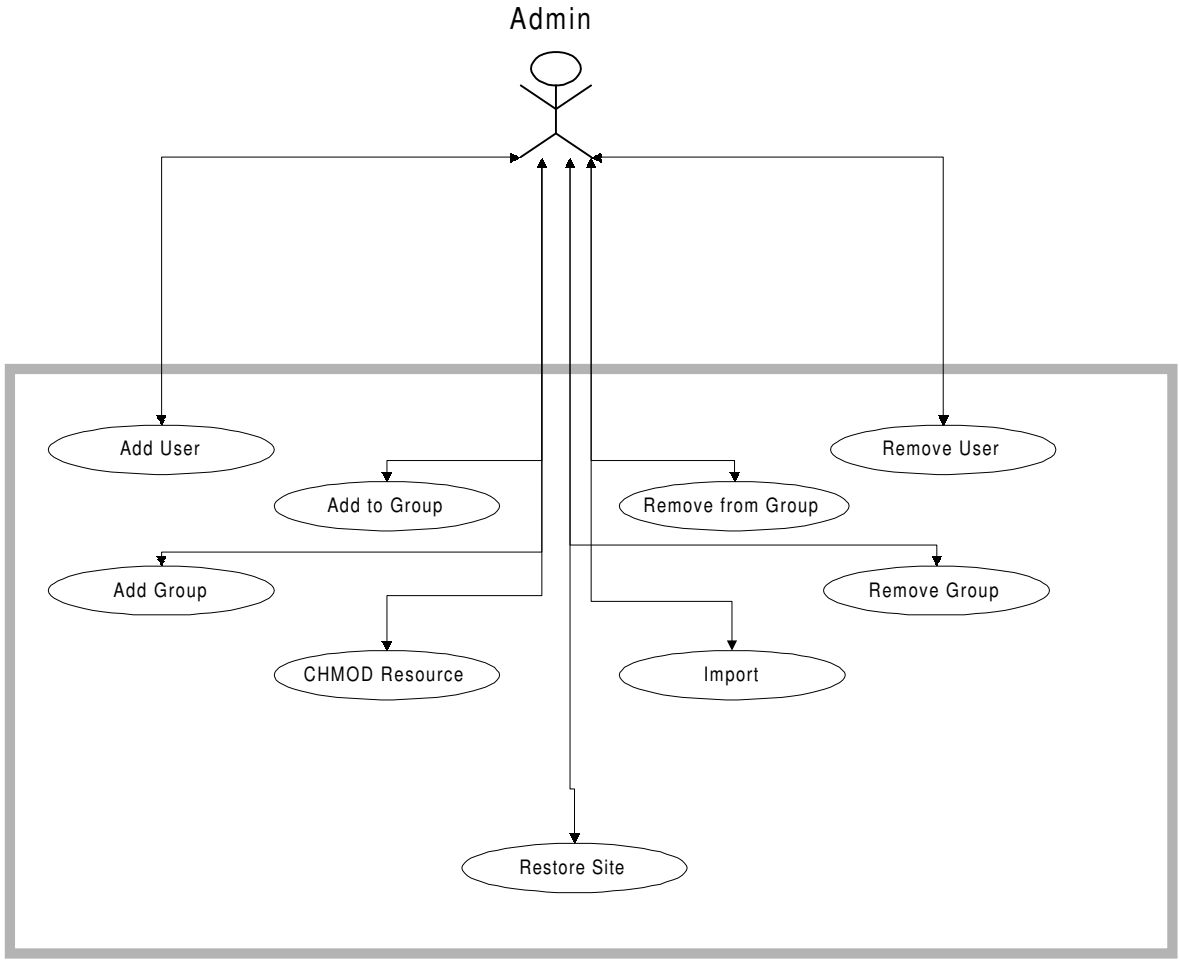



**Abbildung 2-7Anwendungfälle für den Administrator**

#### 2.2.4.1 *Benutzer hinzufügen*

**Kurztext**
Der Administrator fügt einen neuen Benutzer zum System hinzu.

**Beschreibung**
Der neue Benutzer muß zu mindestens einer Gruppe gehören.

**Standardpfad**
1. Der Administrator klickt die für den neuen Benutzer gewünschte Gruppe im Browserfenster mit der rechten Maustaste an. Es erscheint ein Popup-Menü.
1. Der Administrator wählt „Add User“. Ein Benutzer-Editor-Fenster erscheint.
1. Der Administrator kann jetzt Benutzername, Paßwort und Benutzerpräferenzen einstellen.
1. Durch Betätigung der Taste „Save“ wird der Benutzer in der entsprechenden Gruppe erzeugt.

**Der Alternativpfad**
Der Benutzer betätigt die „cancel“ -Taste. Der Vorgang wird abgebrochen.

#### 2.2.4.2 *Benutzer entfernen*

**Kurztext**
Der Administrator will einen Benutzer aus dem System entfernen.

**Beschreibung**
Der Benutzer darf nicht zur Zeit angemeldet sein.

**Standardpfad**
1. Der Aministrator doppelklickt auf eine der Gruppen, in der der Benutzer eingetragen ist. Eine Liste der Benutzer dieser Gruppe erscheint.
1. Der Administrator klickt das Symbol des gewünschten Benutzers mit der rechten Maustaste an. Es erscheint ein Popup-Menü.
1. Der Administrator wählt „Remove *User*“. Der Benutzer wird vom System entfernt.

#### 2.2.4.3 *Gruppe hinzufügen*

**Kurztext**
Der Administrator will dem System eine neue Gruppe hinzufügen.

**Beschreibung**
Die Gruppe darf nicht bereits vorhanden sein.

**Standardpfad**
1. Der Aministrator klickt mit der rechten Maustaste. Ein Popup-Menü erscheint.
1. Der Administrator wählt „Add Group“. Ein Eingabefenster erscheint, in dem der Administrator den Namen der Gruppe anzugeben hat.
1. Der Administrator klickt auf „OK“.
1. Die Gruppe wird erzeugt und wird im Objektbaum sichtbar.

#### 2.2.4.4 *Gruppe entfernen*

**Kurztext**
Der Administrator will eine Gruppe aus dem System entfernen.

**Beschreibung**
Es dürfen keine Resourcen in System vorhanden sein, die zu dieser Gruppe gehören.

**Standardpfad**
1. Der Aministrator klickt mit der rechten Maustaste auf die entsprechende Gruppe. Ein Popup-Menü erscheint.

1. Der Administrator wählt „Remove Group“. Ein Dialogfenster fragt ab, ob die Gruppe wirklich entfernt werden soll.
1. Der Administrator klickt auf „OK“.
1. Die Gruppe wird entfernt.

**Der Alternativpfad**
Der Administrator klickt auf „Cancel“. Der Vorgang wird abgebrochen.

#### 2.2.4.5 *Benutzer zur Gruppe hinzufügen*

**Kurztext**
Der Administrator will einen Benutzer einer Gruppe hinzufügen.

**Beschreibung**
Benutzer und Gruppe müssen schon vorhanden sein.

**Standardpfad**
1. Der Administrator wählt den Benutzer aus dem Objektbaum aus.
1. Per Drag & Drop kann der Administrator den Benutzer in das entsprechende Gruppenverzeichnis verschieben.
1. Es erscheint ein Fenster, in dem der Administrator zwischen zwei Tasten wählen kann: „OK“ und „Cancel“. Der Admin wählt „OK“.
1. Der Benutzer wird zu der Gruppe hinzugefügt.

**Der Alternativpfad**
Der Administrator betätigt die „Cancel“ -Taste. Der Vorgang wird abgebrochen.

#### 2.2.4.6 *Benutzer von der Gruppe entfernen*

**Kurztext**
Der Administrator will einen Benutzer von einer Gruppe entfernen.

**Beschreibung**
Benutzer und Gruppe müssen schon vorhanden sein.

**Standardpfad**
1. Der Administrator wählt den Benutzer in der entsprechenden Gruppe aus.
1. Der Administrator klickt mit der rechten Maustaste auf das Symbol des Benutzers. Ein Popup-Menü erscheint.
1. Der Administrator betätigt „Remove from Group“.
1. Der Benutzer wird von der Gruppe entfernt.

#### 2.2.4.7 *Gruppenzugehörigkeit einer Resource ändern*

**Kurztext**
Der Administrator will die Zugriffsrechte für eine Resource ändern.

**Beschreibung**
Die Resource darf nicht ausgecheckt sein.

**Standardpfad**
1. Der Administrator wählt eine Resource aus.
1. Der Administrator klickt mit der rechten Maustaste auf die Resource. Ein Popup-Menü erscheint.
1. Der Administrator wählt „CHMOD“, woraufhin Fenster erscheint, indem der Administrator die Gruppe auswählen kann, zu dem die Resource gehören soll.
1. Der Administrator klickt auf „OK“.
1. Die Resource gehört jetzt zur ausgewählten Gruppe.

**Der Alternativpfad**
Der Benutzer drückt den „cancel“ -Knopf. Der Vorgang wird abgebrochen.

#### 2.2.4.8 *Übernahme alter Datenbestände*

**Kurztext**
Der Administrator importiert eine vorhandene Website oder Teilbäume davon in das Produktionssystem.

**Beschreibung**
Es dürfen keine anderen Benutzer angemeldet sein.

**Standardpfad**
1. Der Administrator klickt mit der rechten Maustaste. Ein Popup-Menü erscheint.
1. Der Administrator wählt „Import“. Ein Fenster erscheint, in dem der Administrator den Pfad zu den importierenden Dateien auswählen kann. Hier kann er aussuchen, ob er die Dateien über das Dateisystem oder durch HTTP importiert.
1. Der Administrator klickt auf „OK“.
1. Die Dateien werden importiert.

**Der Alternativpfad**
Der Benutzer betätigt die „Cancel“-Taste. Der Vorgang wird abgebrochen.

#### 2.2.4.9 *Wiederherstellen einer früheren Publikation*

**Kurztext**

Der Administrator will den Serverzustand eines bestimmten Datums wiederherstellen.

**Beschreibung**
Es dürfen keine Benutzer angemeldet sein.

**Standardpfad**
1. Der Administrator klickt mit der rechten Maustaste. Ein Popup-Menü erscheint.
1. Der Administrator wählt „Restore“. Ein Fenster erscheint, in dem der Administrator ein Datum angeben kann.
1. Der Administrator klickt auf „OK“.
1. Der Server stellt dann den Zustand dieses Datums wieder her.

**Der Alternativpfad**
Der Administrator betätigt die „Cancel“-Taste. Der Vorgang wird abgebrochen.

### 2.2.5 Das Benutzerzugangsmodell

Bei der Analyse stellte sich heraus, daß ein Zugangsmodell basierend auf Gruppenrechte und nicht auf die Rechte eines einzelnes Benutzers von Vorteil wäre. Es sollte nämlich auf einem Produktionsserver für eine Publikation keine privaten Arbeitsbereiche geben. Falls dies trotzdem benötigt wird, kann man eine minimale Gruppe mit einem Mitglied einrichten. Hier muß darauf geachtet werden, daß die Semantik der Benutzungsrechte nicht zu komplex wird, da die Benutzer dieses Systems nicht technisch bewandert sind.

### 2.2.6 Konsistenzüberprüfung von Querverweisen (Link-Management)

Hier muß das Link-Management dem Benutzer die Arbeit abnehmen, die Gültigkeit von Querverweisen zu überprüfen, wenn das Dokument verschoben oder gelöscht wird. Diese Gültigkeit bezieht sich natürlich nur auf die technische Gültigkeit (das Vorhandensein des Objekts, auf das verwiesen wird) und nicht auf die semantische Gültigkeit (ob der Querverweis inhaltlich stimmt). Wenn möglich, sollten die Korrekturen der Querverweise automatisch stattfinden. Wenn die Korrekturen kontextabhängig sind, sollte der Benutzer davon benachrichtigt werden, welche Veränderungen er vornehmen muß. Durch das Link-Management sollte es nicht möglich sein, auf dem öffentlichen Server ungültige Querverweise zu haben.

### 2.2.7 Unterstützung mehrerer Ausgabeformate

Da die Online-Produktion meistens ihre Inhalte auf mehreren Medientypen veröffentlicht, muß es möglich sein, von den gleichen Ausgangsdokumenten automatisch verschiedene Zielformate zu erzeugen. Hier muß ein generisches Quellformat benutzt werden, um die automatische Konvertierung in die verschiedenen Formate zu ermöglichen.

### 2.2.8 Versionsverwaltung

Publikationsbedingt ist es erforderlich, eine Archivierung bereits publizierter Dokumente vorzunehmen. Bei einer Online-Publikation ist dies schwierig, weil die Inhalte sich sehr häufig ändern. Deswegen muß es möglich sein, die Online-Publikation zu einem bestimmten Zeitpunkt

wiederherzustellen. Außerdem soll es möglich sein, die letzte publizierte Version eines Dokuments wiederzuholen.

# 3 Entwurf

## *3.1 Logischer Entwurf*

Ein System, das für mehrere Benutzer konzipiert ist, sollte die von den Benutzern gemeinsam verwendeten Resourcen zentral speichern. Diese Vorgehensweise bietet mehrere Vorteile: Die Benutzer arbeiten mit denselben Daten und haben keine eigene Kopie - dadurch wird vermieden, daß verschiedene Versionen desselben Dokuments entstehen. Durch die zentrale Speicherung werden Verwaltungsaufgaben wie Zugangskontrolle, Konsistenzüberprüfung und Versionsverwaltung erleichtert. Um diese Aufgaben zu erledigen, muß ein Serverprozeß die Daten verwalten. Der Benutzer hat dann nur über den Server Zugang zu den Daten. Dadurch ergibt sich das Client/Server Modell. Die Daten werden dabei auf dem Server in einer Datenbank gespeichert und sind durch die Clientprogramme für den Benutzer verfügbar.

Der Server bedient die Anfragen der Clients. Sowohl der Server als auch der Client werden in Java implementiert, um damit die gewünschte Plattformunabhängigkeit zu garantieren .

Java ist eine von Sun Microsystems entwickelte Programmiersprache zum Erstellen interaktiver und verteilter Anwendungen, die der Sprache C++ sehr ähnlich ist. Die Programme sind plattformunabhängig, d.h. sie laufen auf jedem Rechner. Die Umwandlung von Java-Code in eine Anwendung verläuft in einem Zweiphasen-Prozeß. Das Programm wird in der ersten Stufe in Bytecode[11] kompiliert. Bei der Ausführung des Codes durch einen Interpreter (zweite Stufe) werden die Instruktionen für den speziellen Rechner umgewandelt.

Ein Applet ist eine Java Applikation, die auf einer HTML-Seite eingebettet ist und im WWW-Browser läuft. Bei Aufruf einer HTML-Seite mit eingebettetem Java-Applet erfolgt zuerst eine Übertragung des als Bytecode vorliegenden Applets vom HTTP-Server zum Browser auf den lokalen Rechner. Erst dort wird die Java-Anwendung dann mittels des Java-Interpreters in den plattformspezifischen Code transformiert und ausgeführt.

[11] Bytecodes sind Befehle, die dem Maschinen-Code sehr ähnlich sind, aber sich nicht auf eine spezielle Architektur (big-endian vs. little-endian) beziehen. Es ist nur festgelegt, was sie für eine Wirkung auf die „Virtuelle Maschine“ von Java haben. [LEMA 96]

Da der Code des Client-Applets zur Benutzungszeit vom Server geholt wird, verwenden die Benutzer immer die aktuellste Version des Clients. Damit entfällt die regelmäßige Aktualisierung von Clientprogrammen für die Benutzer.

Zur Kommunikation werden drei Protokolle eingesetzt. Remote Method Invocation (RMI) (siehe 3.2.1) [CASS 97], um zwischen Client und Server des Produktionssystems zu kommunizieren, und HTTP[12], um die Ergebnisse der Produktion im Browser anzuzeigen. Die Kommunikation zwischen dem Produktionssystem und dem öffentlichen Server erfolgt mittels FTP[13].

Abbildung 3-1 zeigt eine Übersicht der Hauptkomponenten des Systems.

[12] Hypertext Transfer Protocol [FIEL 97]
[13] File Transfer Protocol [POST 85]

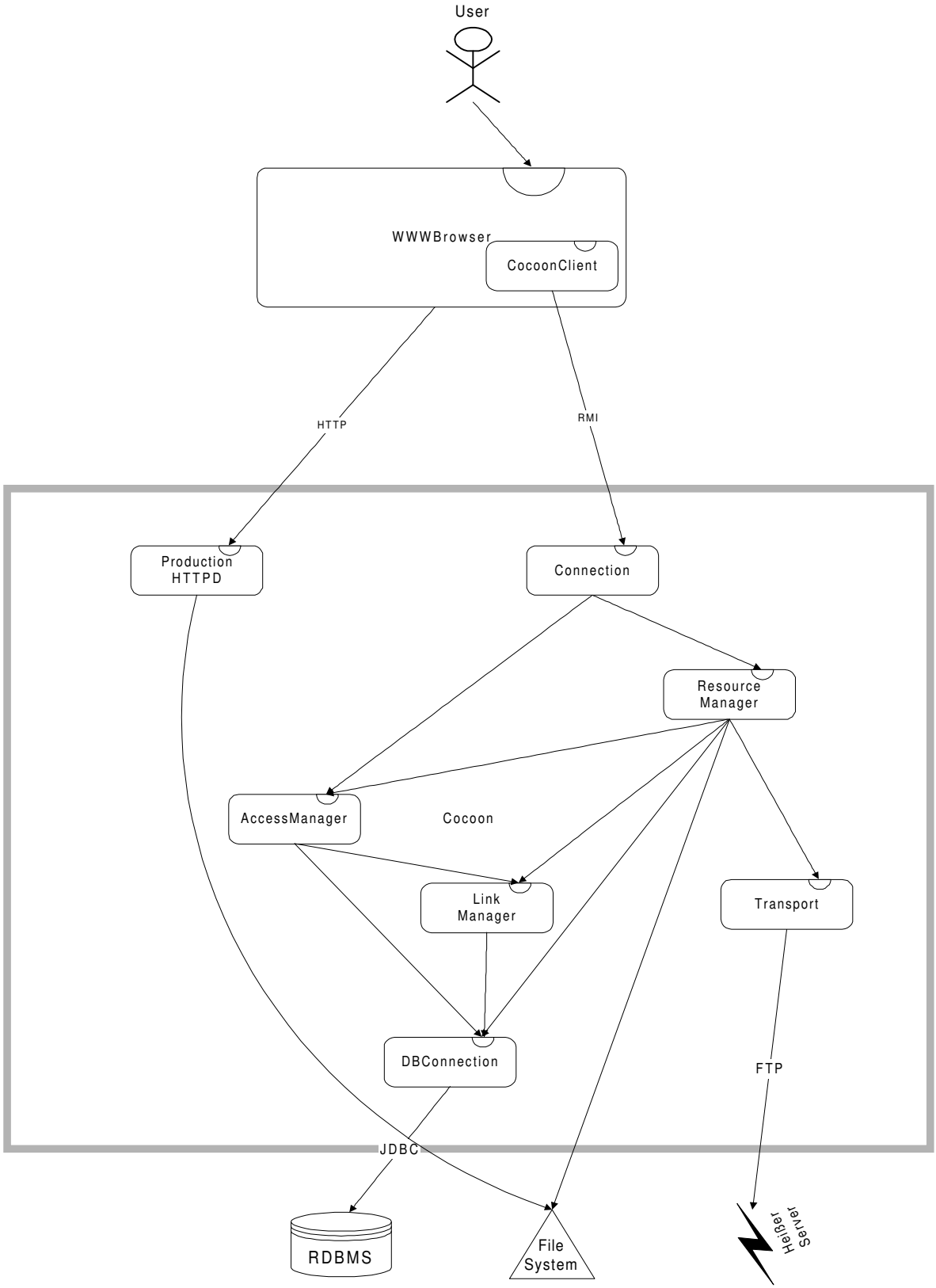



**Abbildung 3-1 Überblick der Hauptkomponenten des Cocoon-Systems**

### 3.1.1 Aufgabenteilung der Kontrollobjekte

Die Aufgaben des Systems werden verschiedenen Kontrollobjekten zugewiesen. Die Kontrollobjekte sind für die Steuerung des Systems zuständig. Sie erledigen Aufgaben wie Erzeugen von Objekten, Verwaltung von Zugangsrechten, Verbindungsaufbau und Kontrollieren von Integritätsbedingungen. Folgende Kontrollobjekte werden verwendet.

**Die *Connection*-Klasse** stellt die Verbindung zum Client dar. Alle Client-Anfragen werden durch die *Connection* an die zuständigen Klassen weitergeleitet.
**Die *AccessManager*-Klasse** ist für die Verteilung von Zugriffsrechten zuständig . Alle Funktionen müssen beim *AccessManager* nachfragen, ob eine bestimmte Operation erlaubt ist.
**Die *LinkManager*-Klasse** sorgt dafür, daß die Querverweise immer konsistent bleiben.
**Die *ResourceManager*-Klasse** ist für die Verwaltung der Objekte zuständig. Diese Klasse setzt die Objekte in ihrer Datenbankrepräsentation um. Dabei fragt der *ResourceManager* bei jeder Operation beim *AccessManager* nach, ob diese Operation zu diesem Zeitpunkt von diesen Benutzer ausgeführt werden darf. Bei Funktionen, die eine Konsistenzüberprüfung der Querverweise benötigen, wird nach dem Ausführen der Operation die entsprechende Funktion beim *LinkManager* ausgeführt. Zum Beispiel führt der *ResourceManager* bei einer Kopieroperation seine eigene Kopierfunktion aus (*ResourceManager.copyResource()*) und danach führt er die Kopierfunktion des *LinkManager*s (*LinkManager.copyResource()*) aus.
**Die *DBConnection*-Klasse** stellt eine Datenbanksession dar. Sie ist die Schnittstelle zwischen dem System und der Datenbank und beinhaltet alle Abfragen.
**Die *Transport*-Klasse** ist für die Benutzung anderer Protokolle zum Datentransport zuständig.
**Die *MulticastMessenger*-Klasse** verteilt die eingehenden Nachrichten an alle Clients.
**Die *Scheduler*-Klasse** führt in regelmäßigen Abständen Aufgaben aus, die zu einer bestimmten Zeit ausgeführt werden müssen.

Diese Klassen sind die Hauptklassen des Servers und enthalten seine Funktionalität.
Sie werden später detaillierter erläutert.

### 3.1.2 Die Prozeßstruktur des Gesamtsystems

Es gibt einen Serverprozeß, den *CocoonServer*, der auf Anmeldungen von Clients wartet und für jeden Client ein eigenes *Connection*-Objekt erzeugt, das in einem eigenen Thread[14] läuft.
Er ist für die Laufzeit des Servers persistent und ist als entferntes Objekt[15] für die Client Applets erreichbar. Das Client-Objekt verbindet sich mit dem *CocoonServer*. Wenn das Client-Objekt sich richtig authentifiziert hat, erzeugt der *CocoonServer* ein *Connection*-Objekt und übergibt eine Referenz an den Client. Danach ist der *CocoonServer* nicht mehr für den Client zuständig und wartet auf weitere Anmeldungen.

Die Clients besitzen jeweils zwei Threads. Das *Messenger*-Objekt, das in einem eigenen Thread läuft, nimmt Nachrichten des Servers entgegen. Diese Nachrichten (Events) werden in einer Queue gespeichert, bis sie ausgeführt werden können. Der Hauptthread bearbeitet die Benutzerinteraktionen.

[14] Ein leichtgewichtiger Prozeß. [CORN 96]
[15] Ein Objekt, das von einem entfernten Rechner erreichbar ist. [ORFA 95]

Nachrichten, die für alle Clients bestimmt sind, werden von den *Connection*-Objekten zum *MulticastMessenger* geschickt. Der *MulticastMessenger* ist ein Objekt, das für das Multicasting von Nachrichten an die Clients zuständig ist. Er speichert die Events in einem Queue und verschickt sie dann an alle *Messenger*-Objekte. Er läuft ebenfalls in einem eigenen Thread, der die Events in der Queue abarbeitet.

Die Nachrichten, die auf einen bestimmten Client bezogen sind, werden direkt vom *Connection*-Objekt an das *Messenger*-Objekt des betreffenden Clients geschickt.

Wenn sich der Zustand einer Resource geändert hat, werden alle Clients mit einem Event darüber benachrichtigt. Auf diese Weise stellt man sicher, daß die Kopien der Resource-Objekte immer den aktuellen Zustand der Resource widerspiegeln.

Da eine *race condition* auftreten kann, wenn mehrere *Connection*-Objekte gleichzeitig auf den *MulticastMessenger* zugreifen, wird die Funktion zum Entgegennehmen der Events im *MulticastMessenger* als *synchronized* [CORN 96] definiert. Das bedeutet, daß die Ausführung der Funktion durch einen Thread abgeschlossen sein muß, bevor ein anderer Thread diese Funktion ausführen kann (Mutual Exclusion). Solche Funktionen heißen *Monitors*.

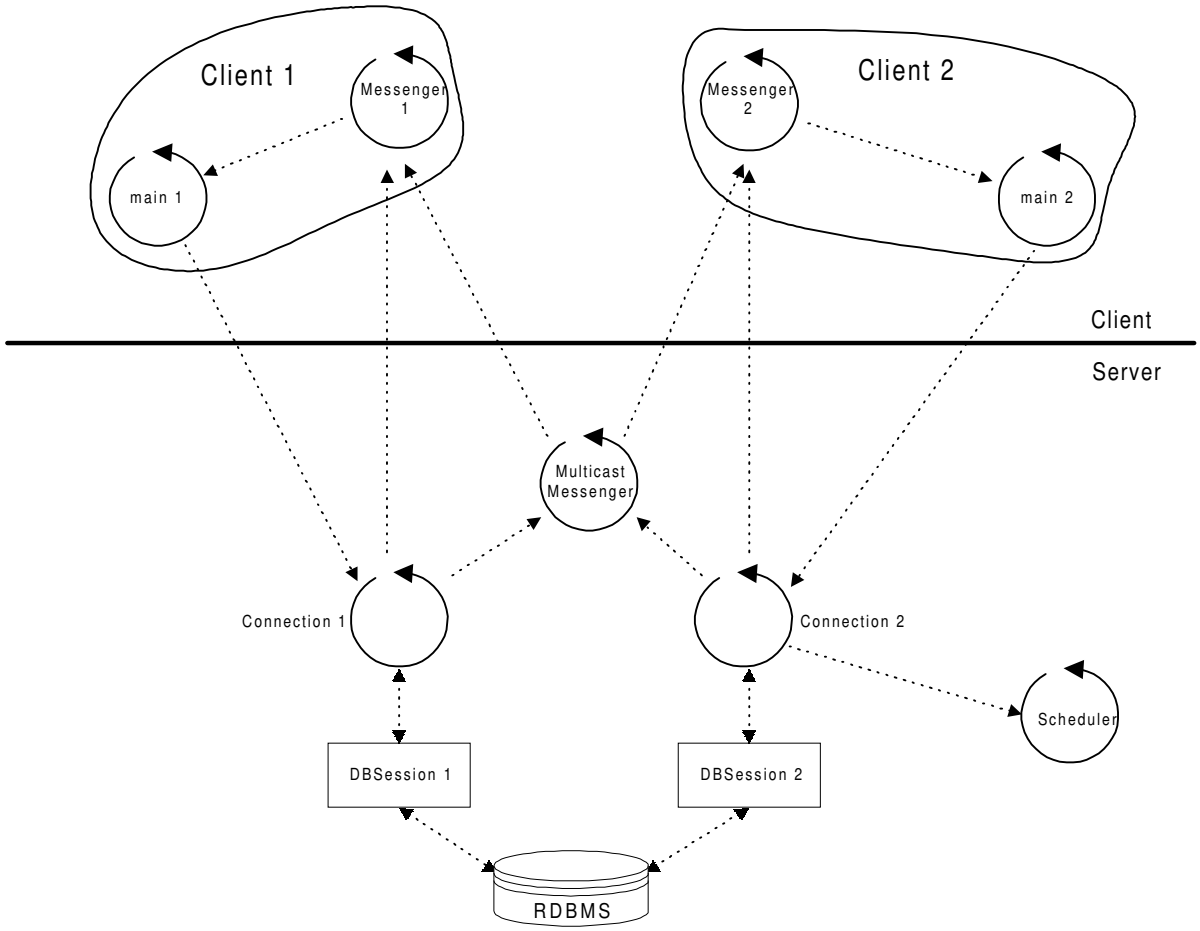



**Abbildung 3-2 Prozeßstruktur vom Gesamtsystem**

Für jedes *Connection*-Objekt, also für jeden verbundenen Client, wird genau eine Datenbanksession gestartet, über die alle Datenbankoperationen des Benutzers ausgeführt werden. Das Lösen der Nebenläufigkeitsprobleme, die durch Datenzugriffe mehrere Benutzer zustande kommen, wird durch Verwendung von einer Datenbanksession pro Client auf das Datenbank Management Systems verlagert.

Die Transaktionsverwaltung wird implizit von den Serverfunktionen ausgeführt, indem bei jeder Anfrage zuerst die betreffenden Daten für andere Benutzer gesperrt werden. Danach wird überprüft, ob die Operation vollständig durchgeführt werden kann. Wenn dies nicht der Fall ist, werden die Daten wieder freigegeben. Dabei wird zwischen Operationen unterschieden, die nur einen Lesezugriff (z.B. Kopieren von Objekten) oder auch einen Schreibzugriff benötigen (z.B. Verschieben von Objekten). Bei ersterem werden die Daten nicht gesperrt.

Schreibzugriffe auf die Daten werden erst ausgeführt, wenn die Überprüfung der Datenbankdaten ergibt, daß die Operation sicher durchgeführt werden kann. Die Operation wird in einer einzigen Datenbanktransaktion ausgeführt, auch wenn sie mehrere Schritte benötigt. Da alle Operationen Datenbankoperationen sind, ist hierdurch die Verantwortung der Transaktionssicherheit auf das DBMS verschoben worden.

Die Verwaltung einer Datenbanksession für den Benutzer ist die Aufgabe der *DBConnection*-Klasse. Für jedes *Connection*-Objekt gibt es ein *DBConnection*-Objekt. Es stellt die Verbindung zur Datenbank her und beinhaltet alle SQL-Abfragen, die vom Server benutzt werden. Durch dieses Objekt ist für jeden verbundenen Benutzer eine Datenbanksession aktiv.

Dadurch, daß der datenbankspezifische Code in einer Klasse eingekapselt ist, können Anpassungen an andere Datenbankstrukturen leichter durchgeführt werden.
Die SQL-Befehle werden in einer Konfigurationsdatei (sql.ini) gespeichert, so daß man bei Änderungen in den Datenbanktabelle nicht den Source-Code editieren muß.

### 3.1.3 Die Ausführung zeitgebundener Aufträge (Task Scheduling)

Der *Scheduler* ist dafür verantwortlich, Aufgaben, die zu einem vorher vereinbarten Zeitpunkt ausgeführt werden sollen, in eine Liste aufzunehmen und sie zum vorgegebenen Zeitpunkt auszuführen[16]. Dieser läuft in einem eigenen Thread, der in bestimmten Intervallen in der Datenbanktabelle (*Task*s-Tabelle) nachprüft, ob Aufgaben zum Ausführen vorliegen. Die vorliegenden Aufgaben sind als *Task*-Objekte in der Tabelle gespeichert.

Für den *Scheduler* sind die auszuführenden *Task*s transparent. Er ruft nur die Funktion *execute()* beim *Task*-Objekt auf. Dadurch ist die Erweiterbarkeit durch verschiedene *Task*s gewährleistet. Kann der *Task* nicht ausgeführt werden, so bleibt er in der Tabelle, und es wird versucht, ihn beim nächsten Zeitpunkt auszuführen.

[16] Dieser Vorgange wird im allgemeinen Sprachgebrauch Terminierung genannt (laut Duden), da aber dieses Wort in der Informatik anders belegt ist, wird auf dessen Gebrauch verzichtet.

**Funktionen:**
Um *race conditions* durch nebenläufige Zugriffe auf diese Funktionen zu vermeiden, werden die Funktionen als *synchonized* definiert. Damit fungieren sie als Monitore.

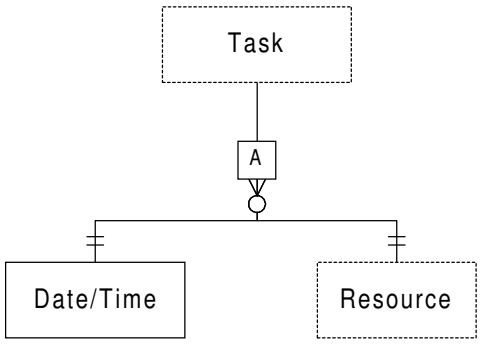
*public* synchronized *void addTask(Task Task) throws AuthorizeException*
Fügt dem *Scheduler* einen *Task* hinzu.

*public* synchronized *void removeTask(Resource r, String Task_name)*
Entfernt einen *Task* aus dem *Scheduler*.

*private void doTask*s*()*
Prüft nach, welche *Task*s zu diesem Zeitpunkt ausgeführt werden sollen. Existieren *Task*s, deren Ausführungszeit in der Vergangenheit oder zum jetzigem Zeitpunkt liegt, dann werden die *execute()* Funktionen dieser *Task*-Objekte ausgeführt. Diese Funktion wird in regelmäßigen Abständen vom Thread des *Scheduler*-Objekts ausgeführt.

#### 3.1.3.1 Das *Task*-Objekt

Das ***Task***-Objekt repräsentiert eine auszuführende Aufgabe des ***Scheduler***s. Dieser ***Task*** wird auf Resourcen ausgeführt. Unterklassen von ***Task*** führen verschiedene Aufgaben aus. Ein ***Task***-Objekt beinhaltet genau ein Resource-Objekt und ein Date/Time-Objekt, das die Zeit der Ausführung beinhaltet.




**Funktionen:**
*execute()*
Diese Funktion führt die Aufgabe dieses *Task*-Objekts auf eine Resource aus.

### 3.1.4 Die Kommunikation zwischen Client und Server

Die Kommunikation zwischen Client und Server geschieht über Event-Objekte. Das *Connection*-Objekt ist für das *CocoonClient*-Objekt als entferntes Objekt erreichbar. Die Event-Objekte werden zwischen *CocoonClient* und *Connection*-Objekt hin- und hergeschickt und beinhalten die Befehle, die ausgeführt werden sollen. Diese Objekte sind Funktionsobjekte, die den auszuführenden Befehl in sich beinhalten.

*CocoonEvent* ist die abstrakte Oberklasse der Events. Sie definiert folgende Funktionen:

*public abstract execute()*

Diese abstrakte Funktion muß von allen Unterklassen implementiert werden. Sie beinhaltet die Funktionalität dieses Events.

Zu jedem Event gibt es eine Unterklasse von *CocoonEvent*. Der Vorteil dieser Methode ist, daß man für die Einführung neuer Funktionen nicht den Client Code bearbeiten muß, sondern die neue Funktionalität in der Unterklasse von *CocoonEvent* implementiert. Der Client führt dann nur die Befehle aus.

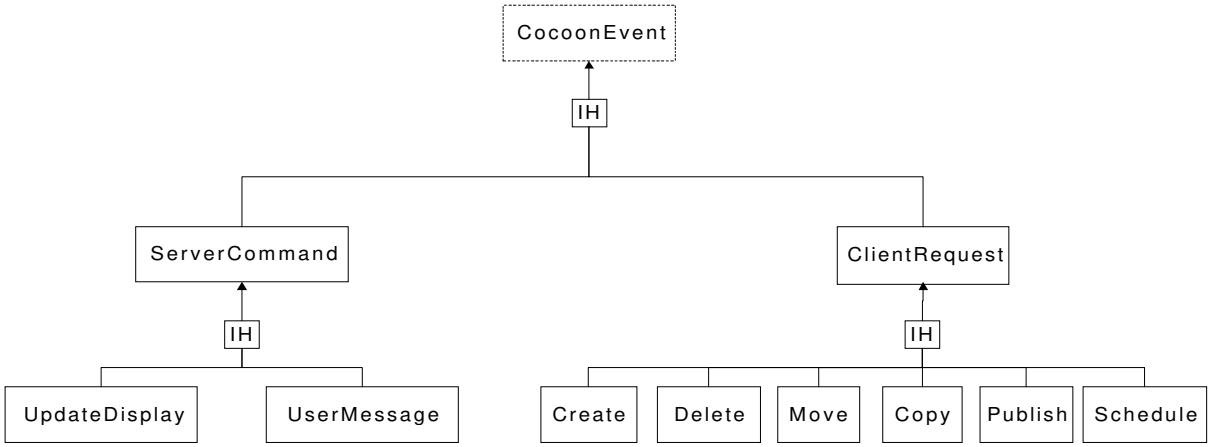



**Abbildung 3-3 Vererbungshierarchie der Event-Objekte**

### 3.1.5 Der Client

Der Client verwendet einen javafähigen WWW-Browser zur Ausführung des Client-Applets[17]. Javafähig bedeutet, daß der Browser einen Java-Interpreter beinhaltet. Der Browser wird zur Ausführung eines Client Applets benutzt.

Das System unterstützt hauptsächlich die Produktion von WWW-Angeboten mit einer Exportfunktion zu anderen Formaten. Die Ergebnisse der Produktion werden durch denselben Browser, indem das Client-Applet läuft, vom HTTP-Server geholt und angezeigt. Das Client-Objekt, *CocoonClient* genannt, ist ein schlanker Client, der nur zur Darstellung der Informationen des Servers fähig ist. Die Resourcen werden darin dargestellt. und die Operationen des Benutzers werden an die entsprechende Resource weitergeleitet. Danach wird meist der Befehl direkt an dem Server weitergeleitet, der dann die Operation ausführt. Die Benutzeroberfläche des Clients wird im 3.1.15 erklärt.

[17] Ein Applet ist ein Java-Programm, das innerhalb eines Browsers läuft. [CASS 97]

Der Benutzer verbindet sich durch den WWW-Browser mit dem HTTP-Server und fordert eine Seite an, in der das Client-Applet eingebettet ist. Die Seite enthält Parameter für das Client-Applet, z.B. den Namen des *CocoonServer*-Objekts und den Namen der Client-Hauptklasse.

Der Browser liest den Namen der Client-Klasse ein, holt sie von dem HTTP-Server und startet das Applet. Der Client liest beim Start den Namen des *CocoonServer*-Objekts ein und holt vom Registry[18] mit Hilfe des Objektnamens eine Referenz auf das entfernte *CocoonServer*-Objekt. Das Applet verbindet sich dann mittels RMI mit dem durch die Referenz angegebenen *CocoonServer*-Objekt.

Es gibt für jede Produktion ein eigenes *CocoonServer*-Objekt und damit ein eigenes Serversystem. Für verschiedene Server gibt es jeweils eigene Seiten, die die Parameter des jeweiligen Servers beinhalten. Diese werden zusammen mit dem Client-Code auf einem zentralen HTTP-Server gespeichert. Für die verschiedenen Produktionen gibt es jeweils einen eigenen HTTP-Server. Weiterhin existiert eine Registry, die die Namen aller *CocoonServer*-Objekte verwaltet.

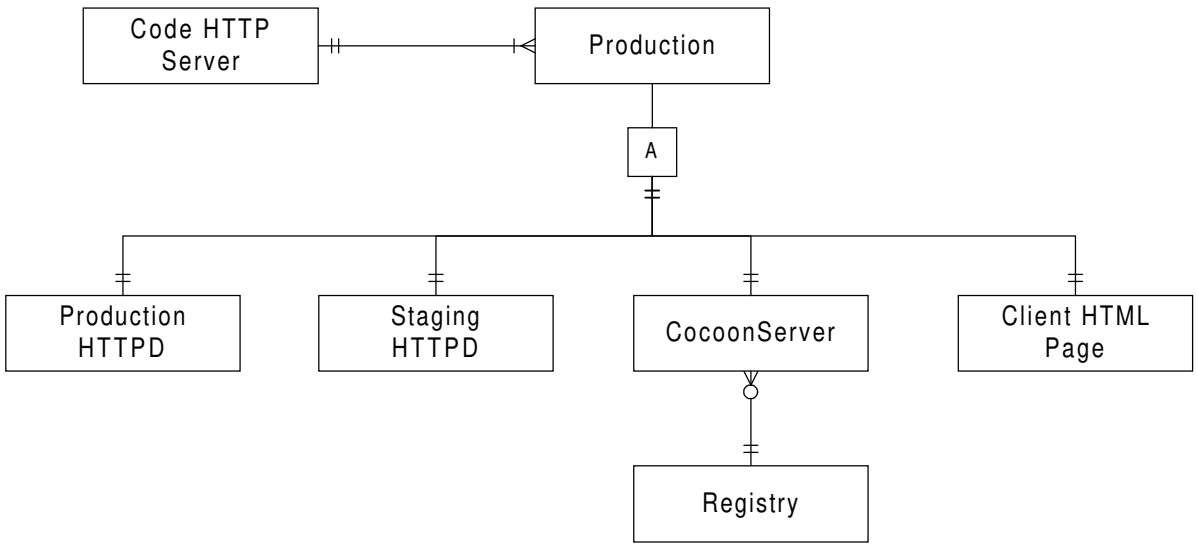



**Abbildung 3-4 Beziehungen der HTTP-Server**

Beim Start des Client-Applets wird der Benutzer nach seinem Namen und Paßwort gefragt.
Das Applet erzeugt dann ein eigenes entferntes Objekt (Messenger), und gibt dieses, den Namen und das Paßwort an den Server weiter (register). Das Messenger-Objekt wird beim Server (als Stub[19]) gespeichert und dazu benutzt, Nachrichten an den Client zu schicken. Der Client bekommt vom Server einen Stub zu einem Connection-Objekt, mit dem er in Zukunft kommuniziert. Der Client holt danach vom Server das Hauptverzeichnis des Produktionssystems und stellt es dar. Der Hauptverzeichnis ist ein Folder-Objekt, das eine Unterklasse von Resource ist.

[18] Die Registry bietet eine Namensdienst an, bei dem man durch Angabe des Namens eines entfernten Objekts eine Objektreferenz erhalten kann. (siehe 3.2.1)
[19] Ein Stub ist ein Stellvertreter eines entfernten Objekts, das die entfernt zugänglichen Methoden und Daten dem benutzenden Programm bereitstellt. [CASS 97]

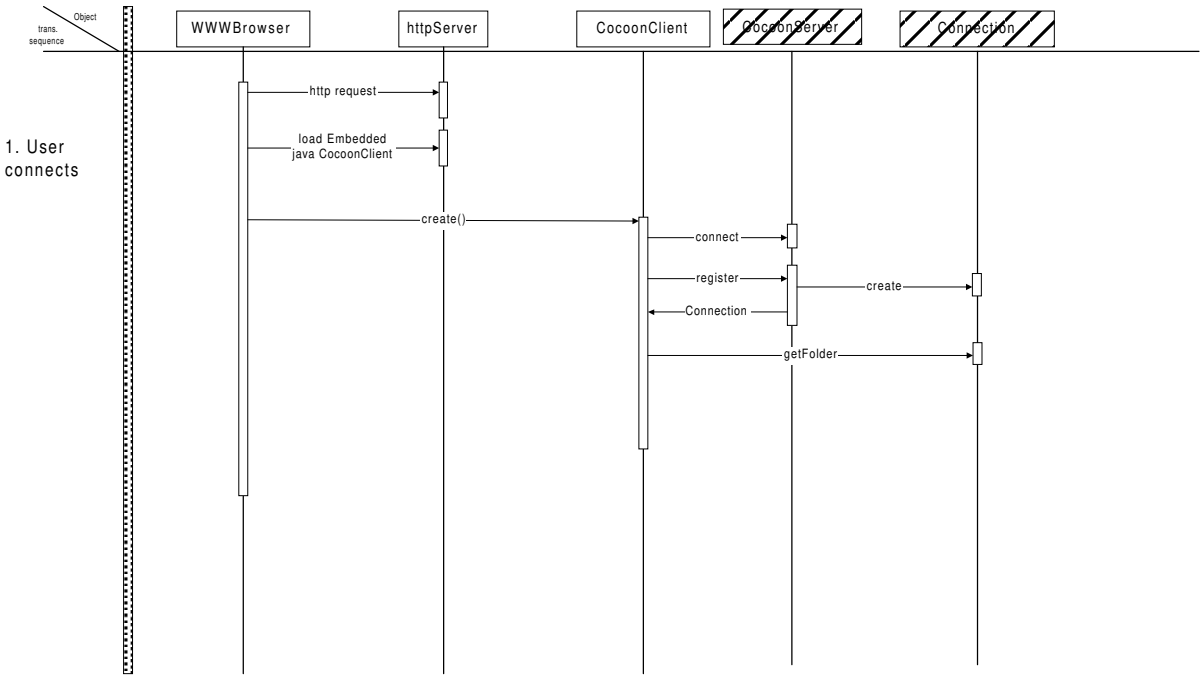



**Abbildung 3-5 Die Client-Sicht des Anmeldevorgangs**

Von nun an geht die gesamte Client/Server Kommunikation über die Resource. Der Client leitet nur noch die Eingaben des Benutzers an die entsprechende Resource weiter. Der Client ist nur für das Anzeigen von Verzeichnisbäumen, Resourcen und Nachrichten des Servers zuständig. Ein typischer Verlauf der Funktionsaufrufe wäre z.B.: Der Client gibt einen Befehl an die Resource weiter. Die Resource wiederum schickt ein Event-Objekt zum *Connection*-Objekt. Das *Connection*-Objekt führt die execute()-Funktion des Event-Objekts aus. Das Event-Objekt führt eine Funktion im *ResourceManager* oder *AccessManager* aus und gibt ein Objekt zur Resource zurück.

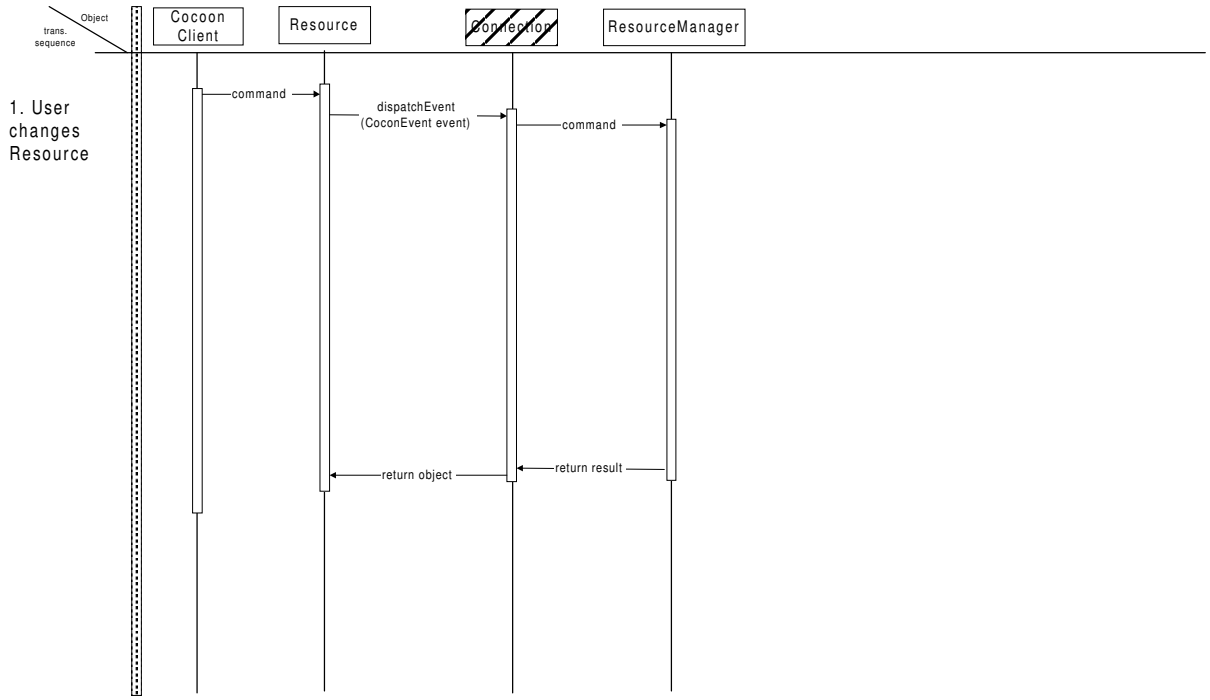



**Abbildung 3-6 Die Kommunikation zwischen Client und Server**

### 3.1.6 Der Server

Der Server besteht aus einer Datenbank, einem entfernten Objekt (*CocoonServer*) und den Kontrollobjekten. Im Folgenden wird der Verbindungsauf- und -abbau aus Sicht des Servers erklärt.

#### 3.1.6.1 Initialisierung des Servers

Beim Starten des Servers (*CocoonServer*) wird eine Initialisierungsdatei gelesen, die die Einstellungen des Servers beinhaltet. Diese Datei enthält u.a. den entfernten Namen des *CocoonServer*-Objekts, die vorhandenen Resourcetypen, die Namen der HTTP-Server (Produktionsserver und öffentlicher Server) und den Benutzernamen des Servers für den Zugriff auf die Datenbank. Der Server meldet sich bei der Registry mit seinen entfernten Namen als entferntes Objekt an.

Der Server checkt beim Start alle Resourcen ein, damit es keine Resourcen gibt, die nach einem Ausfall des Servers ausgecheckt sind. Damit sind nach einem Neustart des Servers die Resourcen wieder verfügbar. Der Server erzeugt ein *Connection*-Objekt, ein *AccessManager*-Objekt, ein *DBConnection*-Objekt, einen *MulticastMessenger* und einen *Scheduler*. Der *CocoonServer* hat damit ein eigenes *Connection*-Objekt und ist selber ein Client des Systems.

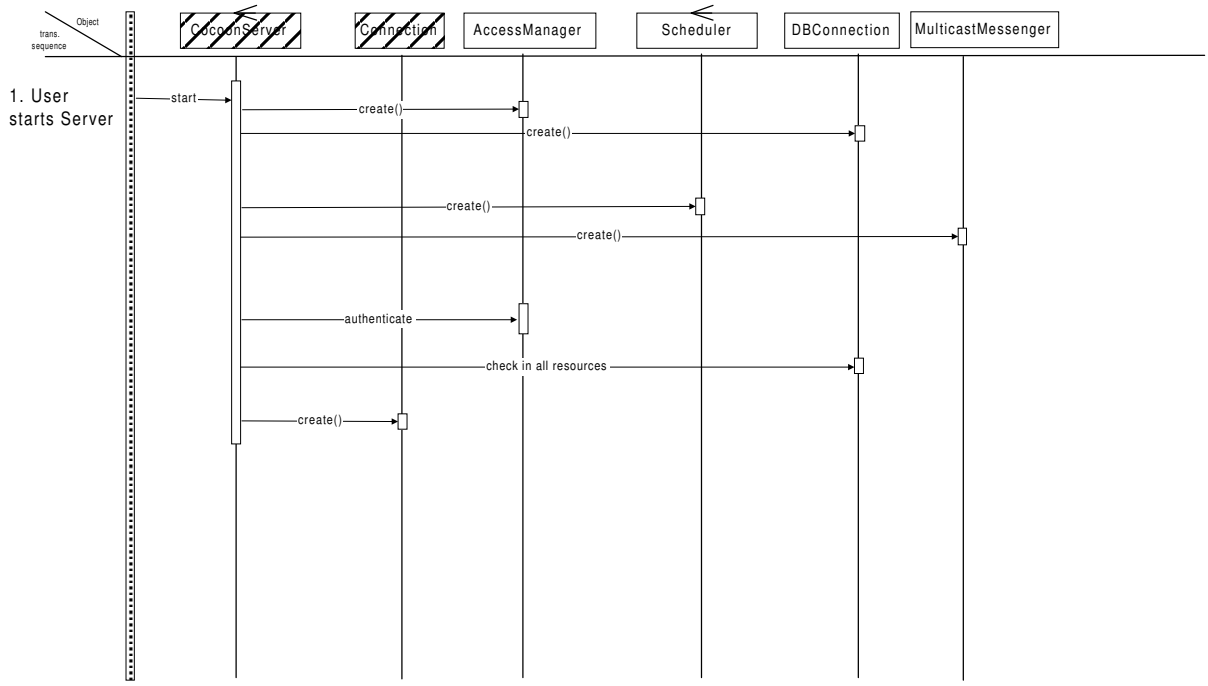



**Abbildung 3-7 Objektinteraktionen beim Initialisieren des Servers**

Nach dem Initialisieren wartet der CocoonServer auf Anmeldungen der Clients.

### 3.1.6.2 Der Anmeldevorgang des Clients

Beim Verbinden des Clients mit dem Server kommuniziert der Client zuerst mit dem *CocoonServer*-Objekt. Nach der Angabe des Benutzernamens und des Paßworts überprüft der *CocoonServer* die Identität des Benutzers. Wird die Kennung bestätigt, so wird auf der Serverseite ein *Connection*-Objekt erzeugt. Das *Connection*-Objekt erzeugt die Objekte, die benötigt werden (siehe Abbildung 3-8) und meldet den Client beim *MulticastMessenger* an (Er gibt dem *MulticastMessenger* eine Referenz auf das *Messenger*-Objekt). Eine Referenz auf das *Connection*-Objekt wird an den Client weitergegeben. Von nun an kommuniziert der Client nur noch mit dem *Connection*-Objekt.

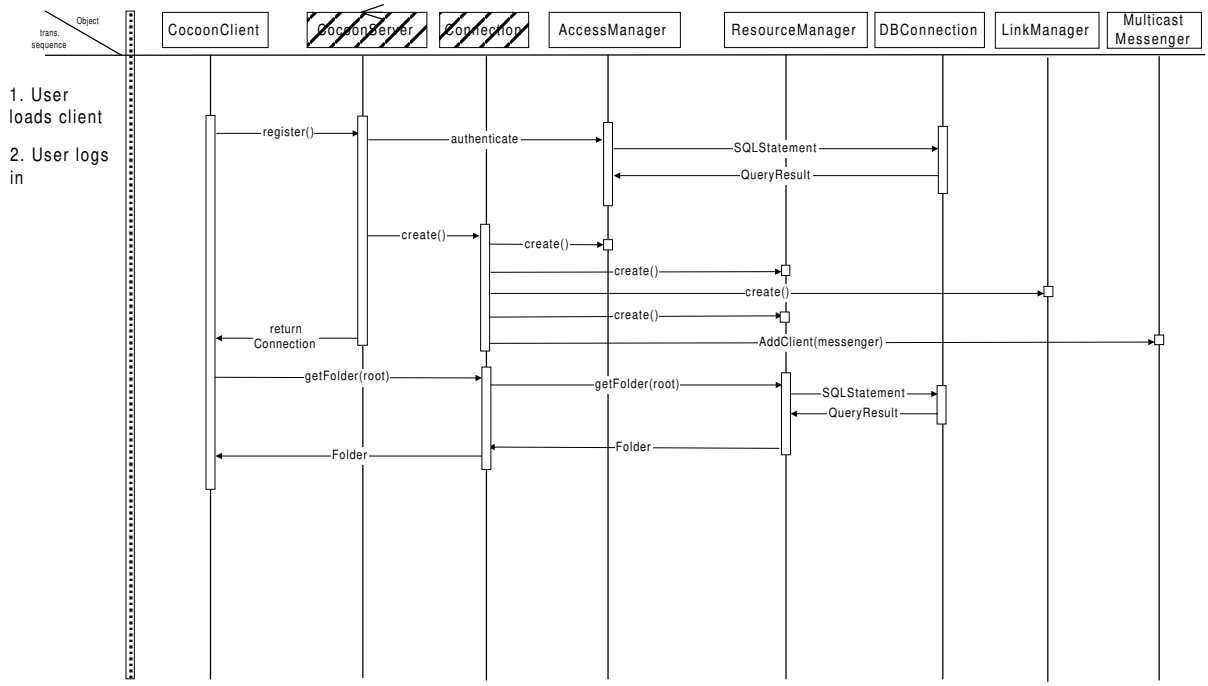



**Abbildung 3-8 Das Anmelden eines Clients auf dem Server**

### 3.1.6.3 Die Verbindungsverwaltung

Das *Connection*-Objekt verwaltet die Verbindung mit einem Client. Wenn der Client die Verbindung abbricht, sorgt das *Connection*-Objekt für die Freigabe der von dem Benutzer verwendeten Ressourcen. Es gibt genau ein *Connection*-Objekt pro Client. Es existiert, solange die Verbindung zum Client besteht. Es beinhaltet ein Referenz-Objekt zum Client (*Messenger*) und kann dadurch Nachrichten an seinen Client verschicken. Beim Verbindungsaufbau sorgt es für die Zuweisung der benötigten Ressourcen, und beim Verbindungsabbruch sorgt es für die saubere

Beendigung der Verbindung. *Connection* nimmt Events des Clients entgegen und führt sie aus. Es liefert das Ergebnis und eventuelle Exceptions[20] zum Client zurück.

Das *Connection*-Objekt verschickt Events, die nur für seinen Client bestimmt sind, zum *Messenger*-Objekt des Clients. Das *Messenger*-Objekt ist die Repräsentation des Clients auf dem Server. Es wird benutzt, um Nachrichten des Servers an den Client zu schicken und um festzustellen, ob der Client noch verbunden ist. Events für alle Clients schickt es zum *MulticastMessenger*.

Die Aktionen eines Benutzers werden in genau einem Thread ausgeführt. Der Benutzer kann nicht mehrere parallele Aktionen auf dem Server auslösen. Durch diese Entscheidung stellt man sicher, daß die Aktionen eines Benutzers nicht miteinander kollidieren.

Es gibt pro Server einen *MulticastMessenger*. Pro *Connection* gibt es einen *AccessManager*, eine *DBConnection*, ein *Messenger*- und ein *ResourceManager*-Objekt.
Abbildung 3-9 gibt einen Überblick der Beziehungen.

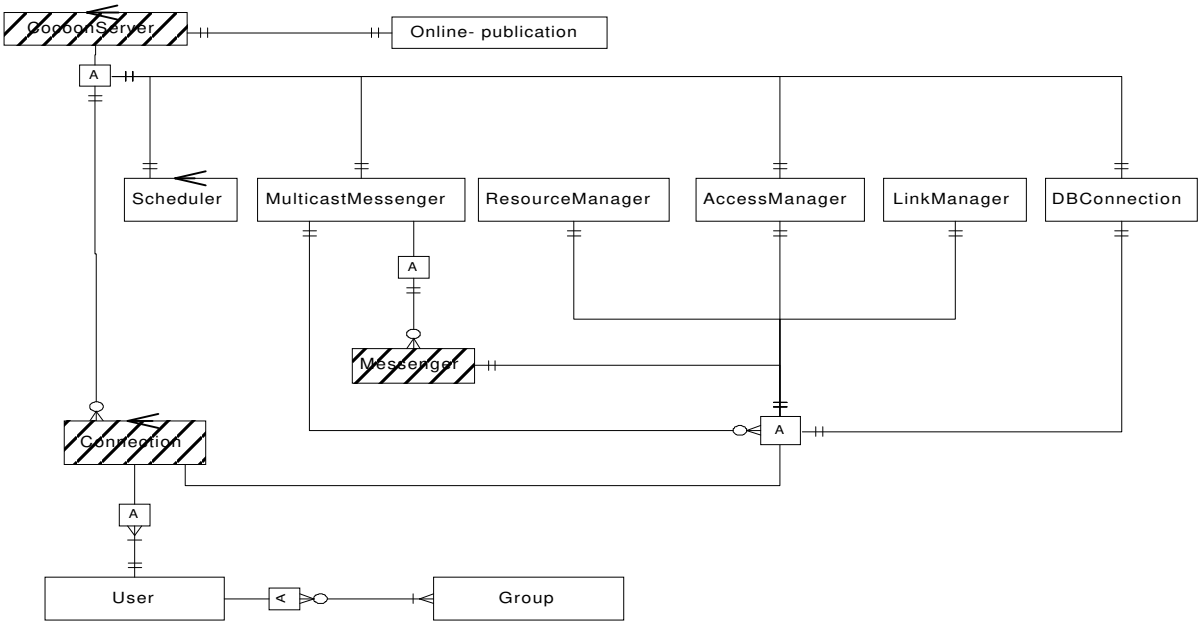



**Abbildung 3-9 Beziehungen der Kontrollobjekte.**

[20] Ein Exception in Java ist ein Objekt, das erzeugt wird, wenn ein Fehler in einer Funktion auftritt, um diesen Fehler den aufrufenden Funktionen zu melden. [LEMA 96]

#### 3.1.6.4 Der Abmeldevorgang des Clients

Beim Abmelden werden das *Connection*-Objekt und alle damit verbundenen Objekte vernichtet. Das *Connection*-Objekt gibt Anweisungen an die anderen benutzerspezifischen Objekte, ihre Ressourcen freizugeben und danach sich selbst zu vernichten. Danach vernichtet sich auch das *Connection*-Objekt selbst.

Der *AccessManager* sorgt dafür, daß alle von dem Benutzer ausgecheckten Resourcen wieder eingecheckt werden, um sie für andere Benutzer zugänglich zu machen. Der *MulticastMessenger* wird davon benachrichtigt, daß der Client nicht mehr existiert. Der *MulticastMessenger* entfernt die Referenz zu diesem Client aus seiner Liste, damit er nicht versucht, Nachrichten an den nun nicht mehr vorhandenen Client zu senden. Der ResourceManager gibt alle Verbindungen (Handles) mit Dateien im Dateisystem wieder frei. Das *DBConnection*-Objekt beendet die Datenbanksession des Benutzers.

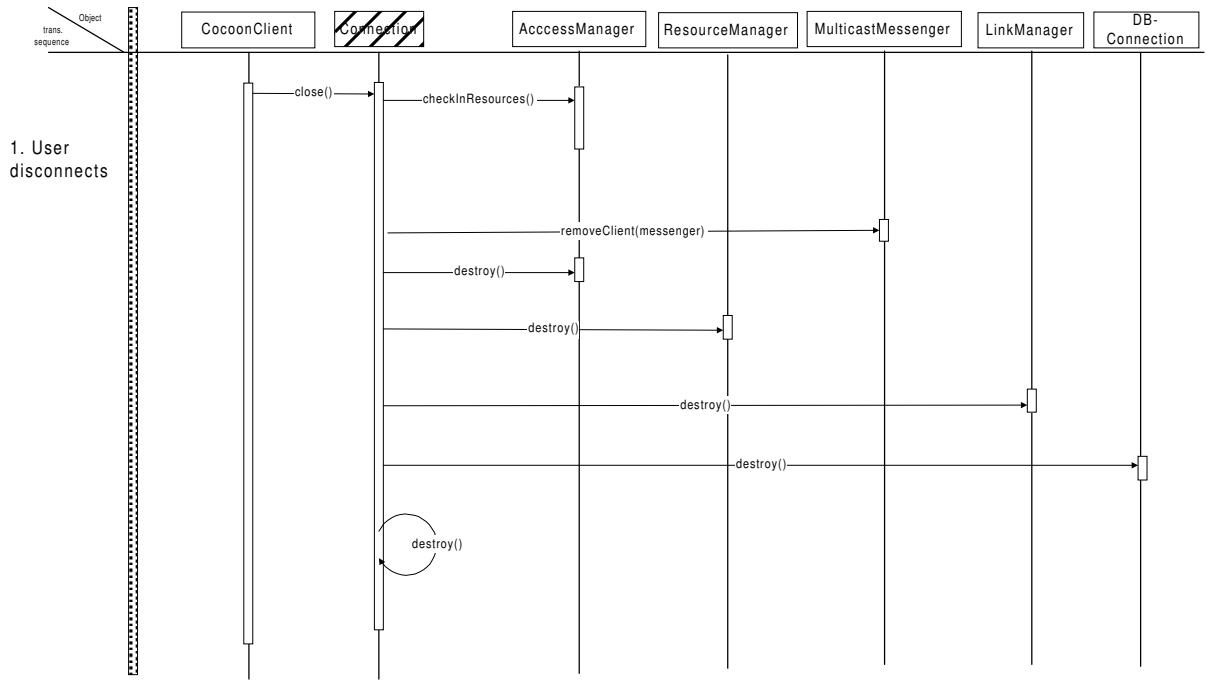



**Abbildung 3-10 Objektinteraktionen beim Abmelden des Clients**

### 3.1.7 Objektverwaltung

In einer objektorientierten Programmiersprache werden die Daten und Funktionen als Objekte definiert. In diesem System wird die objektorientierte Sprache Java verwendet. Da die Daten aber in diesem System in einer relationalen Datenbank gespeichert werden, muß eine entsprechende Datenbankrepräsentation (Schema) der zu speichernden Objekte definiert werden. Dabei werden in der Datenbank nur die Daten der Objekte gespeichert, und nicht die zu ihnen gehörigen Funktionen.

Bei der Speicherung der Objekte in der Datenbank werden die Daten der Objekte in entsprechenden Tabellen gespeichert.

Wenn ein Objekt aus der Datenbank rekonstruiert werden soll, werden aus den verschiedenen Tabellen, die die Daten des Objekts beinhalten, die Daten des gesuchten Objekts zusammengestellt. Danach wird ein leeres Objekt der entsprechenden Klasse instanziiert und mit diesen Daten „aufgefüllt“. Damit eine eindeutige Zuordnung der Daten zu einer bestimmten Klasse stattfinden kann, muß der Klassenname ebenfalls in einer der Tabellen gespeichert worden sein. Damit kann man ein Objekt eindeutig wiederherstellen.

Im Folgenden werden die Datenbankrepräsentationen der von diesem System benötigten Objekte erläutert.

#### 3.1.7.1 Verwaltung des Namensraums (Namespace Management)

Die Daten der Resource-Objekte in der Datenbank müssen eindeutig gekennzeichnet sein, damit die Objekte mit Hilfe dieser Kennzeichnung rekonstruiert werden können.

Jede Resource erhält daher eine eindeutige Identifikationsnummer, die als Primärschlüssel dient. Da die Benutzer eine Navigationsmöglichkeit benötigen und sich nicht Identifikationsnummern merken wollen, muß hier eine virtuelle Namensgebung eingeführt werden. Die Resourcen erhalten somit benutzerfreundliche Namen, die hierarchisch aufgebaut werden können (wie in einem Dateisystem). Diese Namen werden dann den Resource-IDs eindeutig zugeordnet.

Für den hierarchischen Aufbau des Namensraums wird das Konzept einer Verzeichnisstruktur übernommen. Für die Benutzer verhält sich das System dann so, als ob es die Dateien in einer Verzeichnisstruktur speichert. Die Benutzer können Verzeichnisse anlegen, Dateien darin speichern, und andere von Dateisystemen gewohnte Funktionen durchführen.

Diese „virtuelle“ Verzeichnisstruktur wird in der Datenbank durch eine Knotentabelle (Resources) und eine Kantentabelle (Vertices) modelliert. In der Knotentabelle wird eine Kante von A nach B dargestellt, indem in eine Spalte die Resource-ID von A und in eine andere die von B gespeichert wird. Dabei ist die Kante gerichtet.

Die Knotentabelle beinhaltet u.a. die Daten der Resource-Objekte, die für ihre Identifikation notwendig sind. Wenn eine Resource in einem Verzeichnis liegt, wird diese durch eine Kante vom

Verzeichnisknoten zum Resourceknoten dargestellt. Dadurch können die Resourcen abgefragt werden, die in einem Verzeichnis liegen. Durch geeignete Datenbankabfragen kann man somit durch die virtuelle Verzeichnisstruktur navigieren.

Durch die Darstellung der Verzeichnisstruktur durch Knoten- und Kantentabellen ist der virtuelle Namensraum nicht auf eine Verzeichnisstruktur beschränkt. Da man durch diese Tabellen einen beliebigen Graphen darstellen kann und nicht nur einen Baum, kann hier in späteren Versionen des Systems eine andere Darstellung (z.B. ein semantisches Netz) verwendet werden.

Der Vorteil dieser „virtuellen Navigation“ ist, daß eine intuitive Schnittstelle für den Benutzer angeboten werden kann und gleichzeitig das darunterliegende System effizient arbeitet. Die Benutzer können durch die virtuelle Namensgebung Dateinamen und Verzeichnisse verwenden, als ob sie mit einem Dateisystem arbeiten. Sie kommen mit den ID-Nummern nicht in Kontakt. Da aber das System die Daten durch ID-Nummern referenziert, können diese Daten schneller wiedergefunden werden.Wenn z.B. der Benutzer eine Resource vom Verzeichnis A nach B verschiebt, muß vom System keine Verschiebeoperation ausgeführt werden. Es muß lediglich in der Datenbanktabelle, die die Kanten beinhaltet, die Kante vom Verzeichnisknoten zur Resource geändert werden. Dadurch befindet sich die Resource in einem anderen virtuellen Verzeichnis. Die Referenzierung der Resourcen mit Resource-ID hat auch weitere wichtige Vorteile, die in 3.1.11 erläutert werden.

#### 3.1.7.2 Das Datenbankschema der Resource-Objekte

Multimediale Daten besitzen, zusätzlich zu ihrem Inhalt, Informationen über den Inhalt, z.B. Autor, Erstellungsdatum, Thema u.a. Diese Informationen über die Information (Metadata) können benutzt werden, um nach Resourcen zu suchen, bei denen keine Volltextsuche möglich ist (z.B. Bilddateien). Durch die Suche in Metadaten kann gegenüber der Volltextsuche die Zahl der unerwünschten Ergebnisse bei einer Suche reduziert werden. Zum Beispiel kann man in einem Datenfeld „Author“ nach einem bestimmten Autor suchen. Das Ergebnis einer solchen Suche ist ergiebiger, als wenn man eine Volltextsuche nach „Autor UND Name“ startet, da das Wort „Autor“ auch im Textzusammenhang erscheinen kann.

Da man die Metadaten in Zusammenhang mit dem Inhalt bringen und bei Operationen als eine Einheit mit dem Inhalt behandeln muß, ist es sinnvoll, die Metadaten und den Inhalt als Einheit zu speichern. Das Problem bei dieser Methode ist, daß man innerhalb einer Binärdatei Metadaten nur speichern kann, wenn man das Format verändert. Dieses System soll aber formatunabhängig sein. Metadaten, die außerhalb der Resource gespeichert werden, haben den Vorteil, daß sie auch für Datentypen eingesetzt werden können, die z.B. keinen Text beinhalten (Bilder).

Als Kompromiß zwischen beiden Ideen beinhalten die Resource-Objekte sowohl den Inhalt als auch die Metadaten. Die Resourcen werden dabei in einer Datenbank abgelegt. Bei der Ablage in der Datenbank werden die Metadaten und die Inhalte der Resourcen in verschiedenen Tabellen gespeichert. Durch die getrennte Speicherung der Daten und der Metadaten in der Datenbank kann leichter nach Kriterien in den Metadaten gesucht werden. Wenn die Datenbankrepräsentation

einer Resource in ein Resource-Objekt umgewandelt wird, beinhaltet das Objekt beide Datengruppen und kann somit als eine Einheit behandelt werden.

Die Hauptdaten der Resourcen werden in einer Tabelle namens „Versions“ gespeichert. Hier kann jede Version eines Inhaltes durch die Angabe von Data-ID und Versionsnummer abgefragt werden.

Die Metadaten der Resourcen werden in der Tabelle „Resources“ gespeichert. Diese Tabelle dient auch zur Darstellung der Knoten des Baumes, der die Verzeichnisstruktur der gesamten Publikation wiedergibt. (Für eine Beschreibung der Metadaten siehe 3.1.14.1)

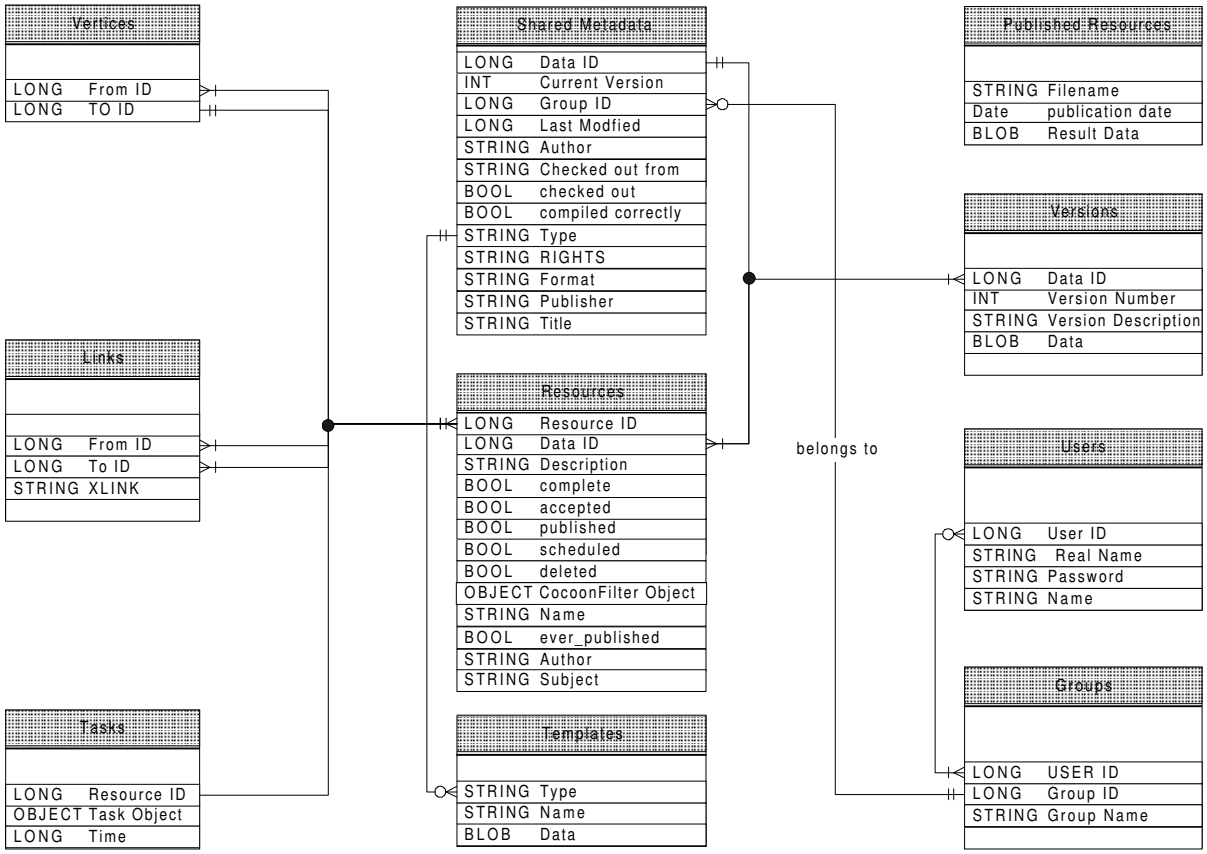



**Abbildung 3-11 Ein Überblick aller Datenbanktabellen des Cocoon-Systems**

#### 3.1.7.2.1 Symbolische Links

Zusätzlich zum Modell des Dateisystems wird hier das Konzept der symbolischen Links eingeführt. Symbolische Links sind Objekte, die vom System als getrennte Entitäten behandelt werden, aber dieselben Daten verwenden. Ein Beispiel dafür sind die symbolischen Links (ab hier Symlink genannt) des UNIX Dateisystems [ABRA 97]. Dort ist ein Symlink z.B. als Datei oder Verzeichnis dargestellt, der aber in Wirklichkeit nur ein Verweis auf eine andere Datei oder Verzeichnis ist. In diesem System wird ein Symlink durch ein Resource-Objekt dargestellt, dessen

Datenfeld dieselben Daten enthält wie eine andere Resource. Um diese Daten identisch zu halten, zeigen beide Resourcen auf dieselben Daten in der „Versions“-Tabelle.

Wenn zwei Resourcen dieselben Daten benutzen, haben sie Metadaten, die sie gemeinsam benutzen und andere Metadaten, die unterschiedlich sind. Zum Beispiel muß die Zustandsvariable „checked out“ für beide zu jedem Zeitpunkt identisch sein, da man verhindern will, daß zwei Benutzer gleichzeitig zwei Symlinks mit denselben Daten bearbeiten. Da es aber sinnvoll sein kann, Symlinks verschiedene Namen zu geben, ist der Attribut „Name“ nicht Teil der gemeinsamen Metadaten.

Für die Modellierung dieses Konzepts wird eine weitere Tabelle „shared metadata“ angelegt, die die gemeinsamen Metadaten der Symlinks beinhaltet. Da die Symlinks eine gemeinsame Data-ID haben, um auf dieselben Daten zu zeigen, wird die Data-ID als Primärschlüssel für „shared metadata“ benutzt. Die nicht gemeinsamen Metadaten werden hingegen in der Tabelle „Resources“ gespeichert (Siehe Abbildung 3-12).

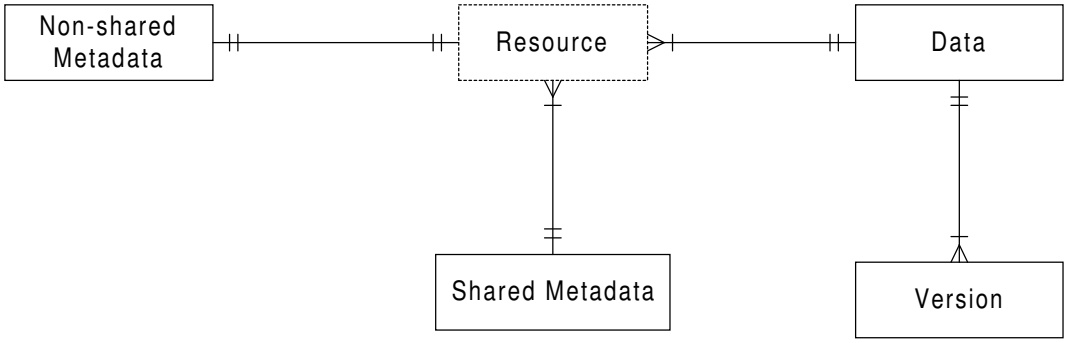



**Abbildung 3-12 Beziehungen der Daten und Metadaten einer Resource**

Das Konzept eines Symlinks wird in diesem System noch weitergeführt, so daß die Symlinks zwar dieselben Daten verwenden, aber unterschiedliche Ausgabedaten erzeugen können. Die Ausgabe kann z.B. für eine *ImageResource* Bilder in verschiedenen Größen und Datenformaten erzeugen.

In den nicht-gemeinsamen Metadaten der Resource gibt es ein Feld, das ein Funktionsobjekt *ResourceFilter* beinhaltet, welches die Quelldaten in die Zieldaten umwandelt. Da die Resource-Objekte verschiedene Funktionsobjekte beinhalten können, können sie aus denselben Quelldaten unterschiedliche Ausgabedaten erzeugen.

Der Vorteil dabei ist, daß, wenn die Quelldaten verändert werden, auch beide Versionen der Zieldatei verändert werden. Der *ResourceHandler* fordert den Inhalt der Resource mit der Funktion *getResultData()* beim Speichern der Resource an (s.u.). Bevor die Daten von der Resource ausgegeben werden, wandelt der *ResourceFilter* diese in ihr Zielformat um. Durch die Möglichkeit, in den Resourcen jeweils unterschiedliche Parameter der Funktionsobjekte zu

definieren, können bestimmte Resourcen individuell behandelt werden. Wenn keine bestimmten Parameter eingestellt werden, werden die Defaultwerte angenommen. Für die verschiedenen Resourcetypen gibt es jeweils unterschiedliche Unterklassen von *ResourceFilter*.

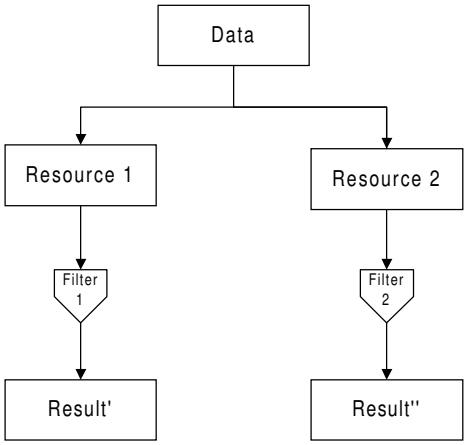



**Abbildung 3-13 Gemeinsame Daten von Symlinks werden durch unterschiedliche Funktionsobjekte verarbeitet und liefern unterschiedliche Ergebnisse.**

Symbolische Links des Unix Systems sind ein Sonderfall dieses Konzepts. In diesem Fall erzeugen beide Symlinks (Resourcen) die gleiche Ausgabe.

#### 3.1.7.3 Rekonstruieren einer Resource aus der Datenbank

Beim Rekonstruieren eines Resource-Objekts aus der Datenbank werden durch die Resource-ID die nicht-gemeinsamen Metadaten geholt. Sie beinhalten unter anderem die Data-ID. Über die Data-ID werden die gemeinsamen Metadaten geholt. Sie beinhalten wiederum u.a. die aktuelle Versionsnummer. Mit der Data-ID und der aktuellen Versionsnummer können dann aus der Tabelle „Versions“ die Daten geholt werden. Wenn eine Resource rekonstruiert wird, beinhaltet sie zunächst **nur** die Metadaten. **Erst wenn der Inhalt benötigt wird, holt das System die entsprechende Version des Inhalts aus der Datenbank.**

#### 3.1.7.4 Erzeugen einer Resource in der Datenbank

Beim Erzeugen einer Resource wird zuerst überprüft, ob sich bereits eine Resource mit dem gleichen Namen im gleichen Verzeichnis befindet. Ist dies nicht der Fall, wird dieser Resource eine eindeutige Resource-ID zugewiesen, und die entsprechenden Tabellen werden um die Daten der Resource ergänzt.

Eine Resource ohne Symlinks besteht aus einem Eintrag in der Resource-Tabelle, einem Eintrag in der „shared metadata“ Tabelle und einem in der Versionstabelle.
Zwei Resourcen mit denselben Daten (Symlinks) bestehen aus jeweils einem Eintrag in der „Resources“ Tabelle, einem einzigen, gemeinsamen Eintrag in der „shared metadata“ Tabelle und den gemeinsamen Daten in der “Versions“ Tabelle.

#### 3.1.7.5 Löschen einer Resource in der Datenbank

Wenn der Benutzer eine Resource löscht, wird sie nicht aus der Datenbank entfernt, sondern es wird nur ein „delete“-Flag[21] in der entsprechenden Tabellenzeile gesetzt.
Damit ist die Resource für den Benutzer nicht mehr vorhanden. Da der „delete“-Flag zu den nicht gemeinsamen Metadaten gehört, kann ein Symlink gelöscht werden, ohne den anderen zu beeinflussen. Die andere Resource zeigt immer noch auf die Daten, und da die Daten nicht gelöscht werden, sind sie immer noch für die Resource vorhanden.

Die Resource kann durch Löschen der „delete“-Flag wiederhergestellt werden. Wenn der Benutzer versucht, eine Resource mit dem gleichen Namen zu erzeugen, wird die Resource wiederhergestellt und die neue Resource als neue Version dieser Resource definiert. Eine neue Resource mit dem gleichen Namen aber verschiedenen Typs kann man nicht erstellen.

Es gib ein besonderes Verzeichnis: den „Mülleimer“, der die gelöschten Resourcen in ihrer Verzeichnisstruktur anzeigt. Diese Resourcen können wiederhergestellt werden. Nur der Administrator kann die Resourcen aus der Datenbank entfernen. So stellt man sicher, daß die Versionsverwaltung konsistent bleibt.

#### 3.1.7.6 Datenbankoperationen zum Verschieben einer Resource

Beim Verschieben einer Resource wird in der Datenbanktabelle, die die Kanten beinhaltet, die Kante vom Verzeichnisknoten zur Resource auf den neuen Verzeichnisknoten umgestellt. Dadurch befindet sich die Resource in einem anderen virtuellen Verzeichnis.

#### 3.1.7.7 Datenbankoperationen zum Kopieren einer Resource

Beim Kopieren einer Resource wird die Resource rekonstruiert und seine Daten unter einem neuen Namen und Resource-ID gespeichert.

#### 3.1.7.8 Speicherung und Verwaltung der Zielformate

Da in diesem System für eine Website produziert wird und die meisten HTTP-Server ihre Seiten nicht direkt aus der Datenbank lesen können, müssen die Zielformate der Resourcen als Dateien im Verzeichnis eines HTTP-Servers ausgegeben werden.

Das System löst sich von der klassischen Struktur eines Dateisystems. Die Dateien, die für das WWW erzeugt werden, werden alle in einem Verzeichnis gespeichert. Sie erhalten alle einen Dateinamen, der der Identifikationsnummer (Resource-ID) des Resource-Objekts entspricht. Als Extension werden die üblichen Extensionen wie „.html“, „.jpg“, etc. verwendet. Die Extensionen werden vom HTTP-Server und WWW-Browser benötigt, um den Inhaltstyp der Datei zu erkennen. Wenn z.B. eine HTML-Datei aus einer Resource mit der Resource-ID „2307“ erzeugt wird, erhält sie den Dateinamen „2307.html“. Die Vorteile dieser Methode werden in 3.1.11 erläutert.

[21] Mit Flag ist hier ein Eintrag in einen Tabellenspalte gemeint, die boolesche Werte beinhaltet.

Zwischen den Resource-Objekten in der Datenbank und den Dateien im Produktionsserver herrscht eine 1:1 Master/Slave Beziehung. Wenn eine Resource erzeugt wird, wird auch eine Datei erzeugt, und wenn die Resource gelöscht wird, wird auch die Datei gelöscht. Wenn eine Resource verschoben wird, wird die Kante vom Verzeichnisknoten zur Resource geändert. Im Dateisystem des Produktionsservers ändert sich nichts, da die Dateien alle in einem Verzeichnis stehen und durch die Resource-ID identifiziert werden. Bei Symlinks gibt es pro Symlink eine HTML-Datei.

### 3.1.7.9 Qualitätssicherung

Um zu gewährleisten, daß nur die Dokumente öffentlich zugänglich sind, die auch fertiggestellt und abgenommen sind, verwendet das System zwei HTTP-Server: Einen zum Testen und Produzieren (Produktionsserver), den anderen zum Publizieren (öffentlicher Server).

Der öffentliche Server befindet sich außerhalb der Firewall und ist öffentlich zugänglich. Der Produktionsserver befindet sich innerhalb der Firewall und ist durch diese vor dem öffentlichen Zugriff geschützt. Eine Resource wird erst auf den öffentlichen Server transferiert (publiziert), wenn sie fertiggestellt und abgenommen ist (siehe 3.1.10.3). Beim Publizieren einer Resource wird der Inhalt ihrer aktuellen Version per FTP in das Dateisystem des öffentlichen Servers geschrieben.

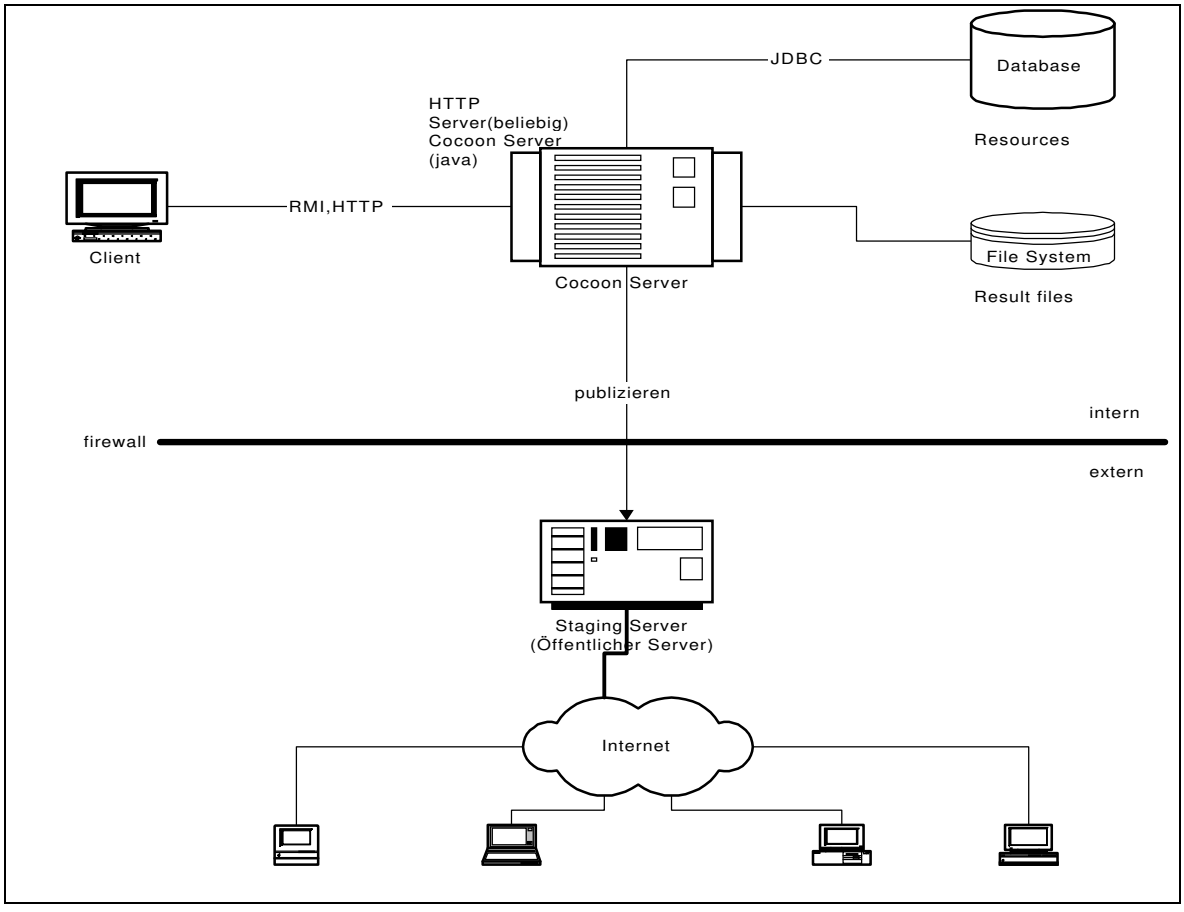


**Abbildung 3-14 Überblick des gesamten Systems**

Die Dateien werden auf dem öffentlichen Server in der gleichen Struktur gespeichert wie auf dem Produktionsserver. Die Server haben die gleichen Einstellungen, damit die Testumgebung mit der öffentlichen Umgebung identisch ist.

Zwischen den drei Systemen Datenbank, Dateisystem des Produktionsservers und Dateisystem des öffentlichen Servers herrscht eine Master/Slave Beziehung. Die Datenbankobjekte sind dabei die Master-Objekte. Aktionen, die auf den Master-Objekten ausgeführt werden, beeinflussen auch die Slave-Objekte. (Siehe Abbildung 3-15).

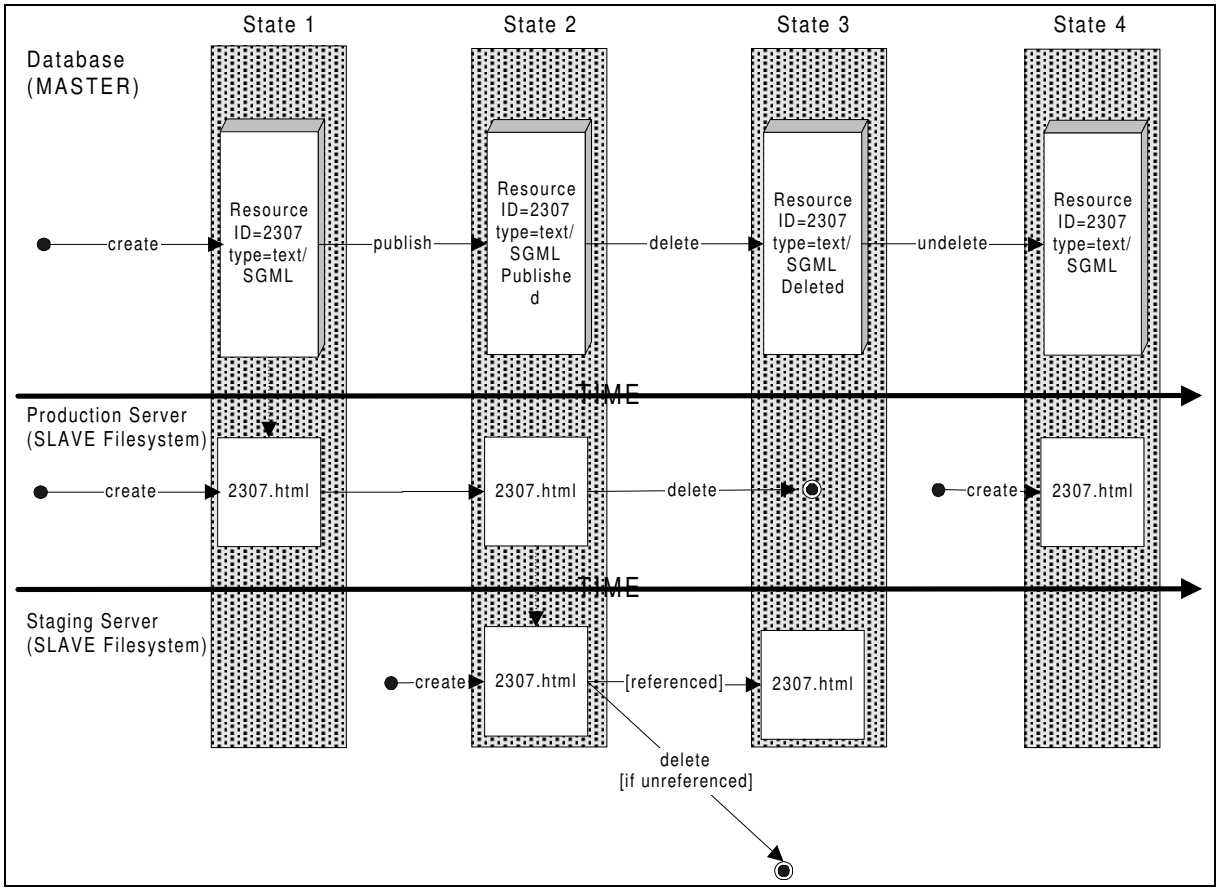



**Abbildung 3-15 Beziehung der Datenbankobjekte mit den Dateien im Dateisystem**

Die Beziehungen der Daten im Produktionssystem zu denen auf den öffentlichen Server werden in 3.1.11 näher erläutert.

Die Aufgaben der Objektverwaltung werden vom *ResourceManager* ausgeführt.
Diese Klasse ist dafür zuständig, beim Rekonstruieren die Resourcen aus der Datenbank in Objekte umzuwandeln und beim Speichern die Objekte in ihre Datenbankrepräsentation umzusetzen. Vor jeder Operation fragt der *ResourceManager* beim *AccessManager* nach, ob die Operation zulässig ist. Die Funktionen des *ResourceManagers* werden vom Resource-Objekt

aufgerufen. Ein Resource-Objekt schickt einen Event an sein *Connection*-Objekt, das den Event in einen entsprechenden Funktionsaufruf des *ResourceManager* verwandelt.

### 3.1.8 Objekthierarchien

#### 3.1.8.1 Das Datenobjekt Resource

Resource ist die abstrakte Oberklasse aller Datenobjekte und beinhaltet die Funktionalität, die für jedes Datenobjekt benötigt wird. Zum Beispiel enthält sie die Funktionen zum Lesen und Speichern der Resource.

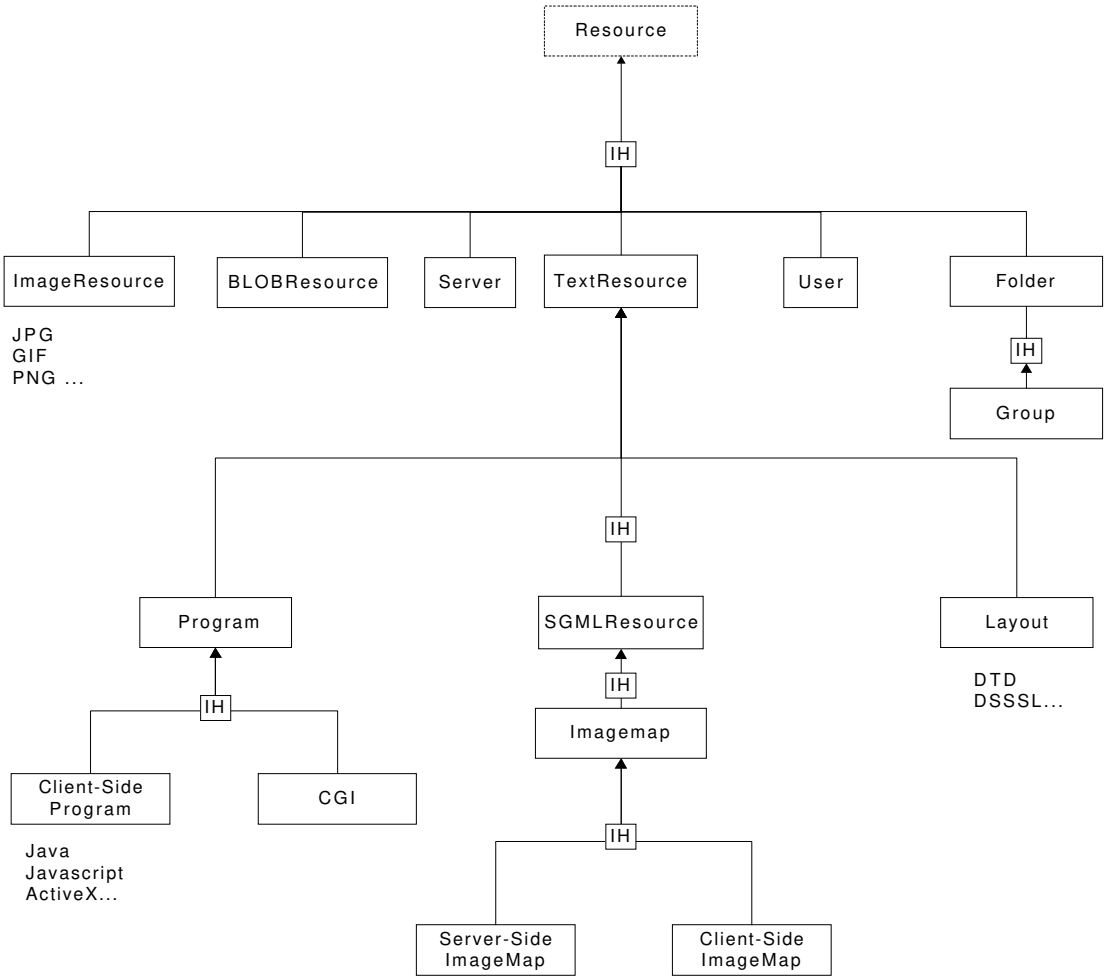



**Abbildung 3-16 Vererbungshierarchie der Resource-Unterklassen.**

Funktionen zur Manipulation einer Resource sind in der Resource-Klasse definiert. Wenn eine Funktion in der Resource-Klasse aufgerufen wird, ruft diese die entsprechende Funktion auf dem Server auf. Die Unterklassen von Resource können durch Überladen der benötigten Funktionen die Funktionen erweitern. Die Löschfunktion einer *Folder*-Unterklasse von Resource z.B. gibt den Löschbefehl zuerst an alle Resourcen weiter, die sich in dem *Folder* befinden, und führt dann die eigene Löschfunktion aus

Das Resource-Objekt steht in Relation zu den in Abbildung 3-17 gezeigten Objekten.

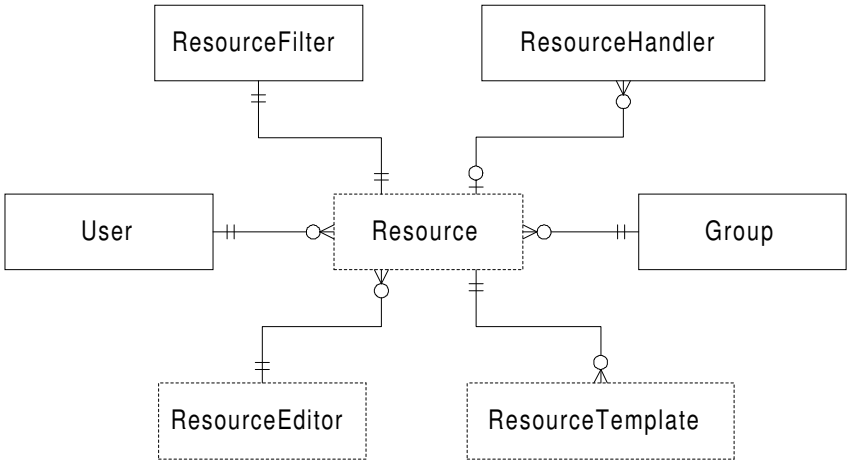



**Abbildung 3-17 Relationen der Resource mit seinen bearbeitenden Objekten**

Im Folgenden werden die Aufgaben und Hierarchien dieser Klassen erläutert.

### 3.1.8.2 Die ResourceEditor-Klasse

Zu jeder Resource gibt es einen entsprechenden Editor. Ein Editor muß eine Unterklasse von der abstrakten Oberklasse *ResourceEditor* sein. *ResourceEditor* ist eine Oberklasse, die die grundsätzlichen Funktionen zur Bearbeitung einer Resource beinhaltet. Der Editor kommuniziert nur mit der Resource. Die Resource ruft den Editor auf und gibt sich selber als Parameter an den Editor weiter. Die Funktionsweise des Editors wird in 3.1.15 genauer erklärt.

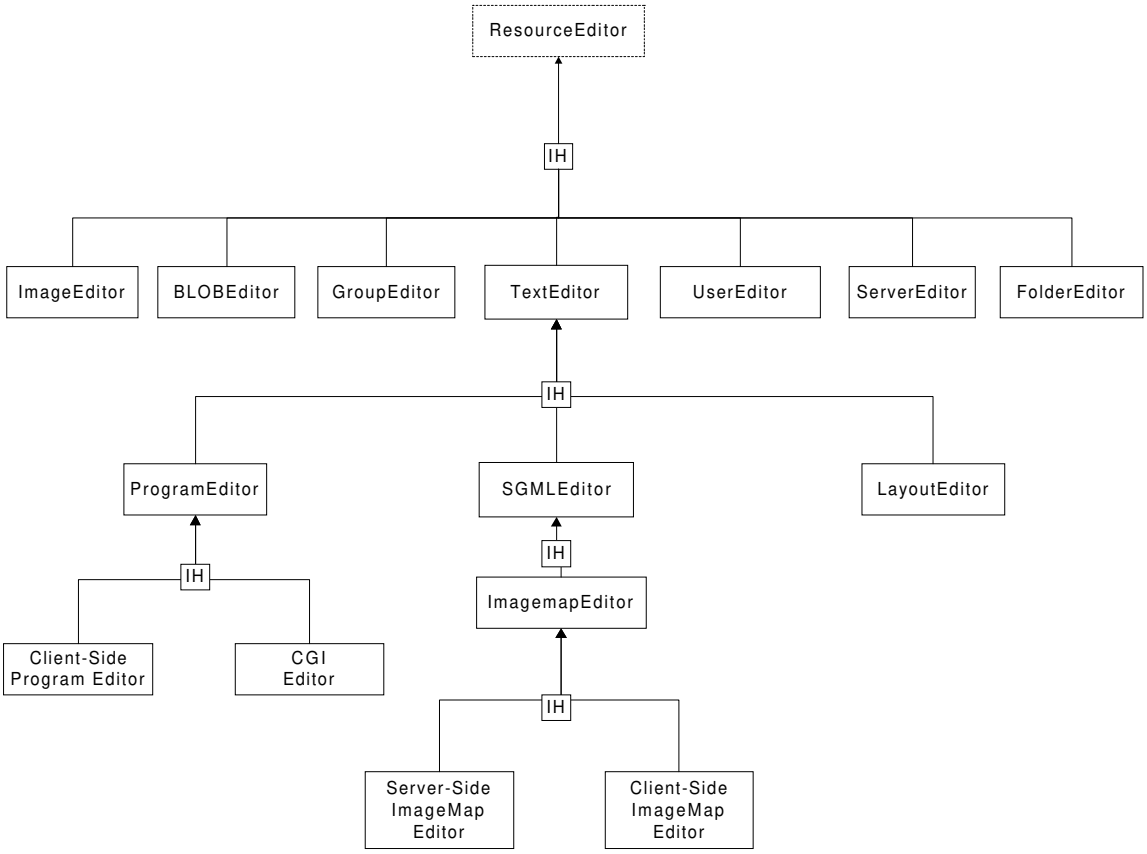



**Abbildung 3-18 Vererbungshierarchie der *ResourceEditor*-Klassen**

### 3.1.8.3 Die *ResourceHandler*-Klasse

Um Besonderheiten der Resource beim Abspeichern zu berücksichtigen, gibt es für jeden Resourcetyp einen entsprechenden *ResourceHandler*. Dieser *ResourceHandler* übernimmt Aufgaben, die beim Speichern einer Resource anfallen. Zum Beispiel wird das Zielformat einer Resource nach der Konvertierung in das Dateisystem des Produktionsservers geschrieben.

Jeder *Handler* muß eine Unterklasse von *ResourceHandler* sein. Für jeden Resourcetyp ist ein *ResourceHandler* zu definieren. Beim Ausführen des *ResourceHandler*s fordert er von der Resource die Zieldaten an. Die Resource führt dazu das *ResourceFilter*-Objekt auf die Daten aus, das seinerseits einen den Resourcetypen entsprechenden Konverter aufruft (z.B. für *SGMLResource* einen SGML-Konverter [in diesem Fall DSSSLMaschine[22]] oder für eine *ImageResource* ein Werkzeug zur Konvertierung von Bildern). Das Ergebnis der Konvertierung wird vom *ResourceHandler* dann gespeichert. Bei einer neuen Resource wird ein *ResourceFilter* mit Standardwerten eingesetzt, dessen Parameter durch den *ResourceEditor* geändert werden können.

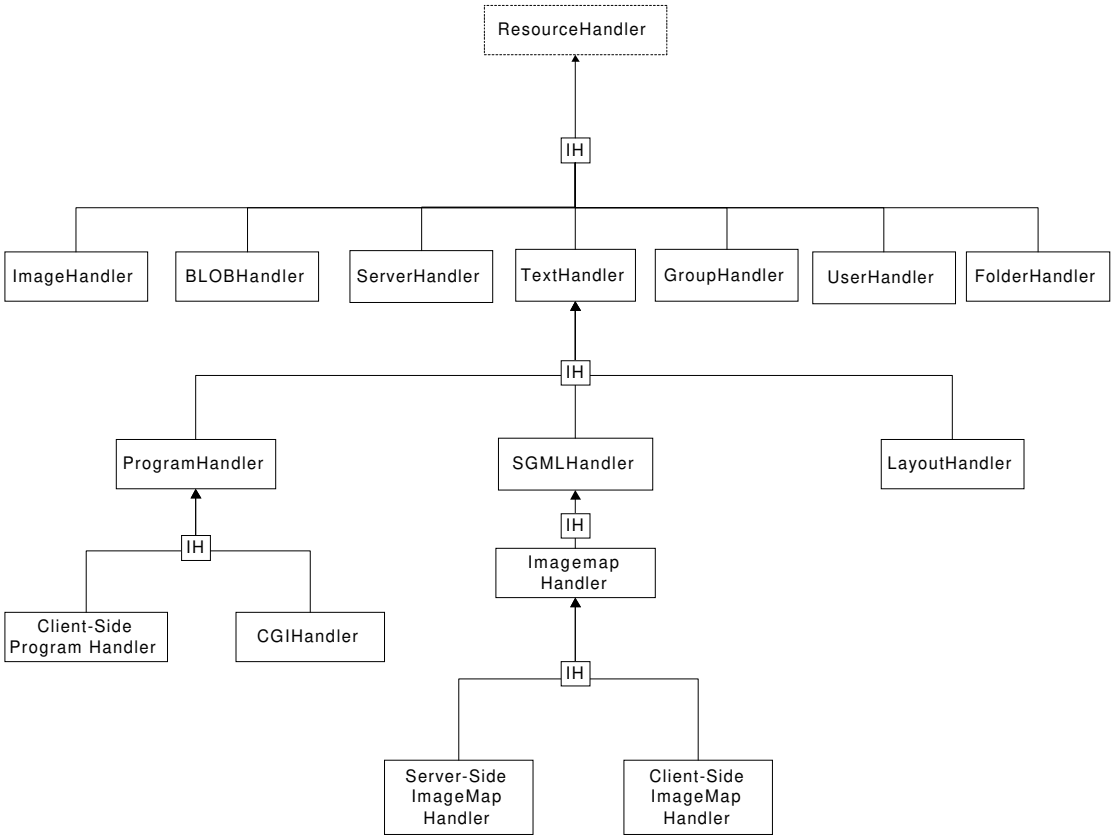



**Abbildung 3-19 Vererbungshierarchie der *ResourceHandler***

[22] Eine DSSSLMaschine ist ein Konvertierungswerkzeug, das die SGML-Dokumente mit Hilfe der in einer DSSSL-Datei definierten Transformation in eine Darstellungs- oder Ausgabeform umwandelt.[COVE 97]

### 3.1.8.4 Einführung eines neuen Resourcetyps

Um einen neuen Resourcetyp einzuführen, muß man vier Klassen definieren: eine Unterklasse von Resource, eine Unterklasse von *ResourceFilter,* eine entsprechende Unterklasse von *ResourceEditor* und eine Unterklasse von *ResourceHandler*. Danach muß die Tabelle der verfügbaren Resourcetypen um den neuen Typ erweitert werden. Daraufhin kann die Resource bereits verwendet werden, ohne den Server neu starten zu müssen. Der Menüpunkt „Create“ wird dann im Untermenü automatisch um den neuen Resourcetyp erweitert.

### 3.1.8.5 Resourcevorlagen (Templates)

Die Resourcen verfügen über Vorlagen, sogenannte Templates, die als Dokumentenvorlagen für die Inhalte der Resource dienen. Zum Beispiel besitzen in einer elektronischen Zeitschrift alle Artikel eines bestimmten Typs das gleiche Layout. Hier kann die Resource aus der Vorlage erzeugt und dann mit Inhalt angereichert werden. Es können zu jedem Resourcetyp mehrere Templates definiert werden.

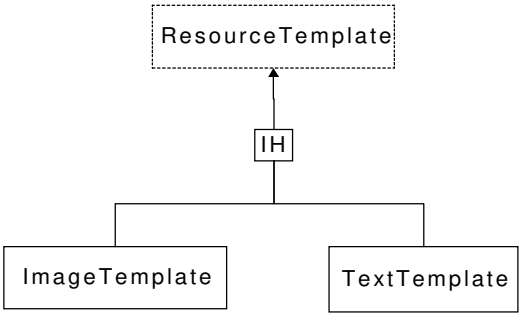



**Abbildung 3-20 Hierarchie der Template-Objekte**

### 3.1.9 Objektinteraktionen

#### 3.1.9.1 Editieren einer Resource

Beim Editieren einer Resource gibt der *CocoonClient* den Edit-Befehl an die zu bearbeitende Resource weiter. Die Resource schickt eine Nachricht an das *Connection*-Objekt, um ihre Daten vom Server zu holen. Die Resource ist bis zu diesem Zeitpunkt ein inhaltloses Objekt auf der Clientseite, das lediglich die Metadaten beinhaltet. Erst wenn der eigentliche Inhalt benötigt wird, wird er vom Server geholt. So wird die Netzwerklast reduziert. Das *Connection* Objekt fordert wiederum vom *ResourceManager* die Daten an. Wenn der *AccessManager* diesen Zugriff zuläßt, wird die Resource ausgecheckt und es werden die Daten zur Resource, die auf der Client-Seite ist, geschickt.
Die Resource instanziiert einen dem Resourcetypen entsprechenden Editor und gibt sich selber als Parameter an den Editor weiter. Danach übernimmt der Editor die Bearbeitung der Resource.

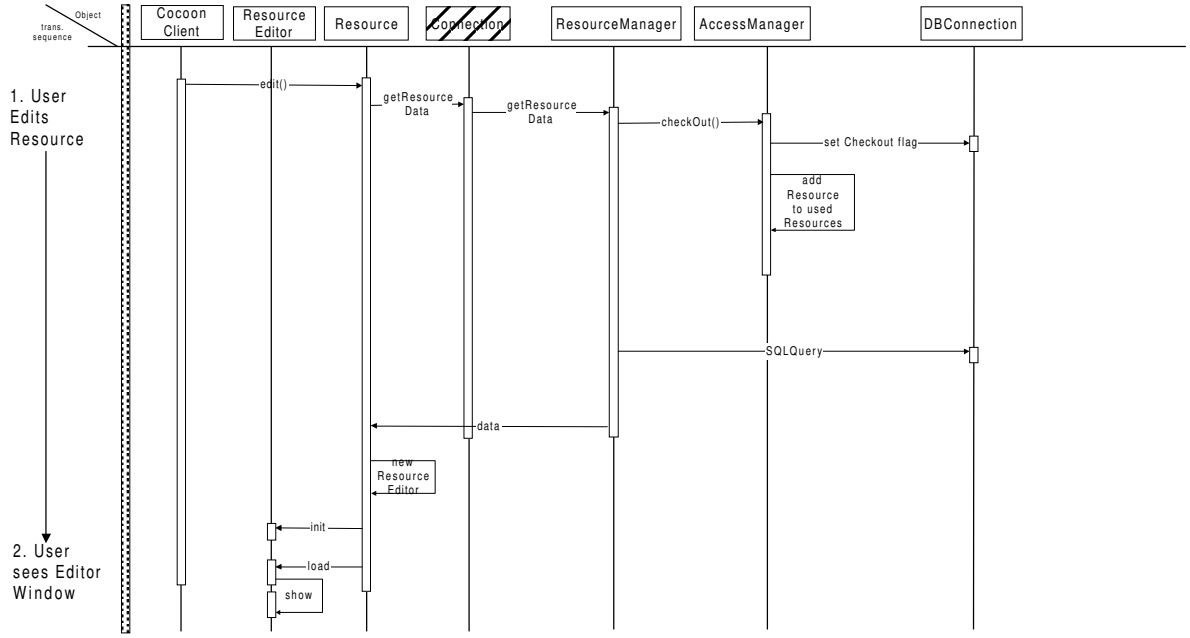



**Abbildung 3-21 Objektinteraktionen zum Editieren einer Resource**

### 3.1.9.2 Speichern einer Resource

Beim Abspeichern einer Resource sendet der *ResourceEditor* eine Nachricht an die Resource, daß sie sich selber speichern soll. Daraufhin schickt die Resource einen Speicherbefehl an die *Connection,* wobei sie sich selbst als Parameter mitgibt. Danach gibt das *Connection*-Objekt die Nachricht an den *ResourceManager* weiter.

Die Daten der Resource werden nur in der Datenbank gespeichert, wenn der *AccessManager* es zuläßt. Danach erzeugt der Server ein dem Resourcetyp entsprechendes *ResourceHandler*-Objekt; der Name der *ResourceHandler*-Unterklasse ist in der Resource gespeichert. Mit dem *handle()* Befehl wird dann der *ResourceHandler* angewiesen, die resourcespezifischen Speicheroperationen auszuführen.

In Abbildung 3-22 ist folgendes Beispiel dargestellt: Der Inhalt der Resource wird nach Querverweisen durchsucht und das Ergebnis vom *LinkManager* in der Link-Tabelle gespeichert. Der *ResourceHandler* fordert danach die Zieldaten von der Resource an und speichert das Ergebnis im Dateisystem.

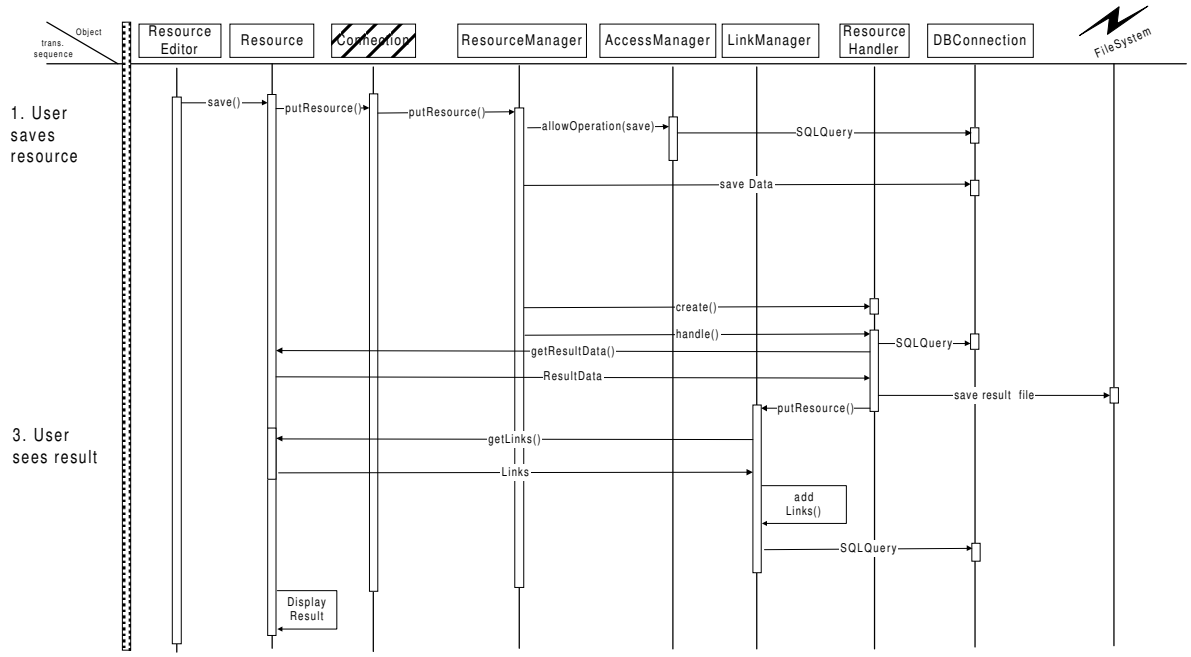



**Abbildung 3-22 Objektinteraktionen zum Speichern einer Resource**

### 3.1.9.3 Erzeugen einer Resource

Beim Erzeugen einer Resource wird eine neue Resource vom Server angefordert. Wenn der *AccessManager* es zuläßt, wird ein Resource-Objekt auf dem Server erzeugt, in der Datenbank gespeichert und zum Client zurückgeschickt. Der Client führt dann die Editierprozedur durch.(siehe 3.1.9.1).

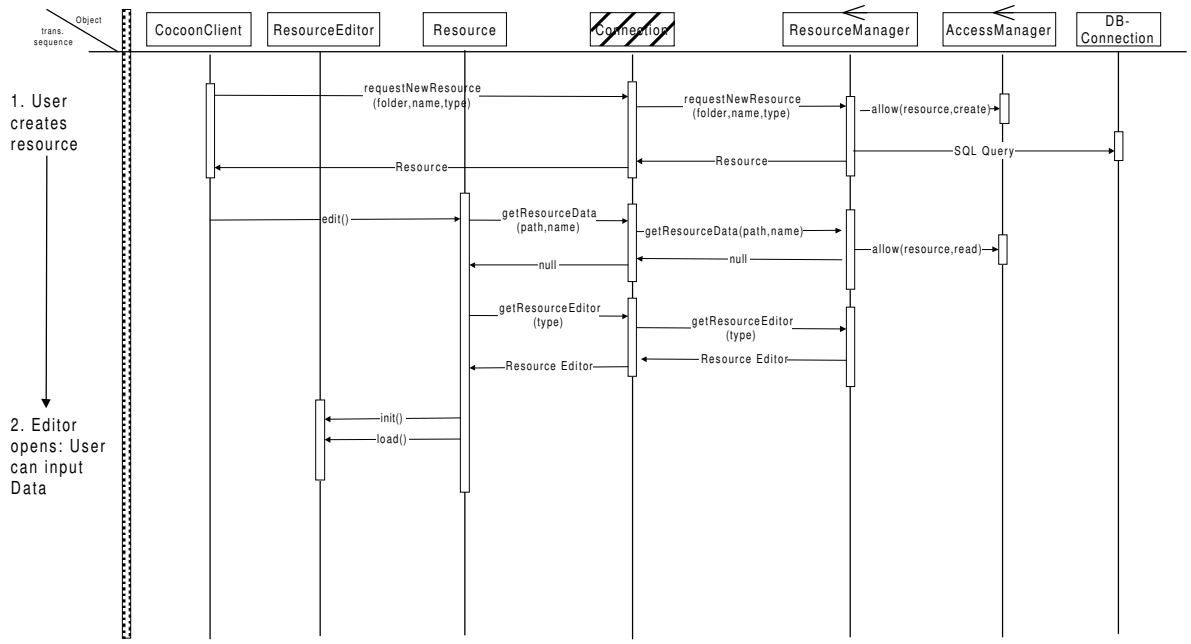



**Abbildung 3-23 Objektinteraktionen beim Erzeugen einer Resource**

### 3.1.9.4 Löschen einer Resource

Der Resource wird eine Nachricht geschickt, daß sie sich löschen soll. Daraufhin leiter die Resource die Nachricht zum Server weiter. Der Server setzt in der Datenbank einen Flag, der die Resource als gelöscht markiert. Das macht sie für den Benutzer unsichtbar. Die Ausgabedatei, die im Pfad des Produktionsservers liegt, wird gelöscht. Die gelöschte Resource taucht im Verzeichnis des „Mülleimers“ auf und kann bei Bedarf wiederhergestellt werden. Bei symbolischen Links sind die gemeinsam benutzen Daten für die nicht gelöschte Resource weiterhin verfügbar.

### 3.1.9.5 Erstellen eines symbolischen Links

Beim Erstellen eines Symbolischen Links wird eine neue Resource erzeugt, die auf den gleichen Inhalt zeigt, wie die ursprüngliche Resource.

### 3.1.9.6 Kopieren einer Resource

Beim Kopieren einer Resource wird das Resource-Objekt geklont[23] und Name und Pfad der Kopie geändert. Danach wird die neue Resource im System gespeichert. Beim Kopieren von Verzeichnissen (*Folder*) geschieht dies mit jedem Objekt, angefangen von den *Folder*-Objekten. Dabei werden auch die Links geändert.(siehe 3.1.11)

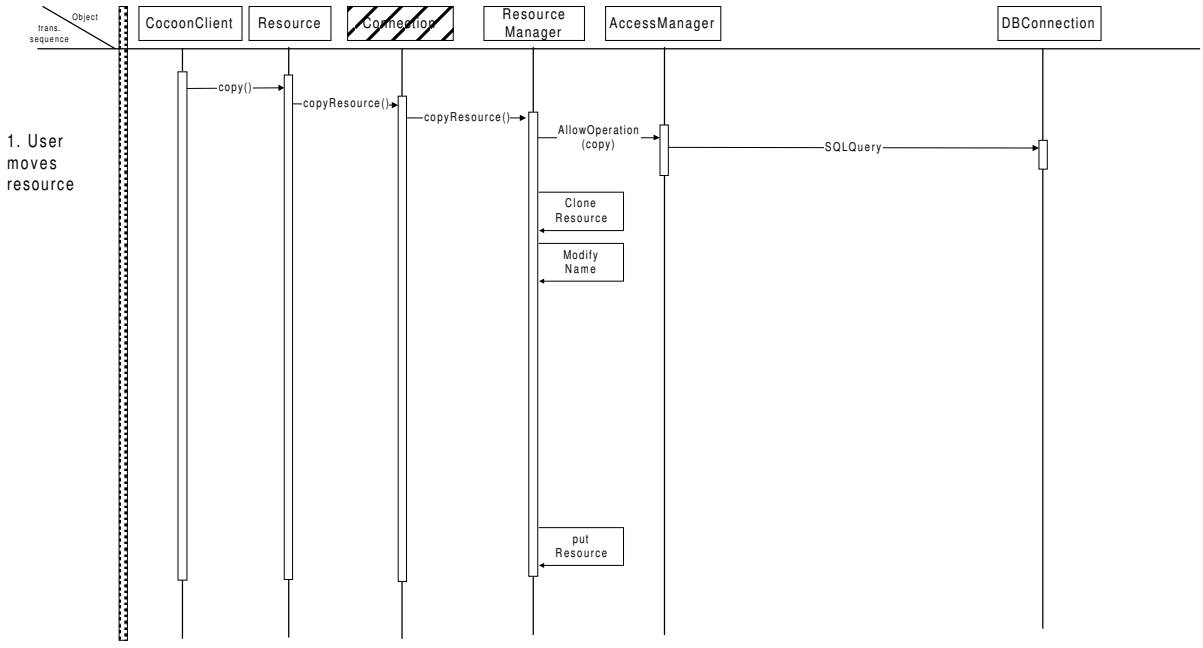



**Abbildung 3-24 Objektinteraktionen für das Kopieren einer Resource**

[23] Klonen heißt hier, daß eine Kopie des Objekts mit dem gleichen Inhalt erzeugt wird.

### 3.1.9.7 Verschieben einer Resource

Beim Verschieben einer Resource gibt die Resource einen Verschiebebefehl, der sie selbst als Parameter enthält, an das *Connection*-Objekt weiter. Das *Connection*-Objekt leitet die Nachricht an den *ResourceManager* weiter. Wenn der *AccessManager* es zuläßt, wird in der Kantentabelle (Vertices) die Kante, die vom Verzeichnisknoten zur Resource führt, umgeleitet, so daß sie vom neuen Verzeichnisknoten zur Resource führt.

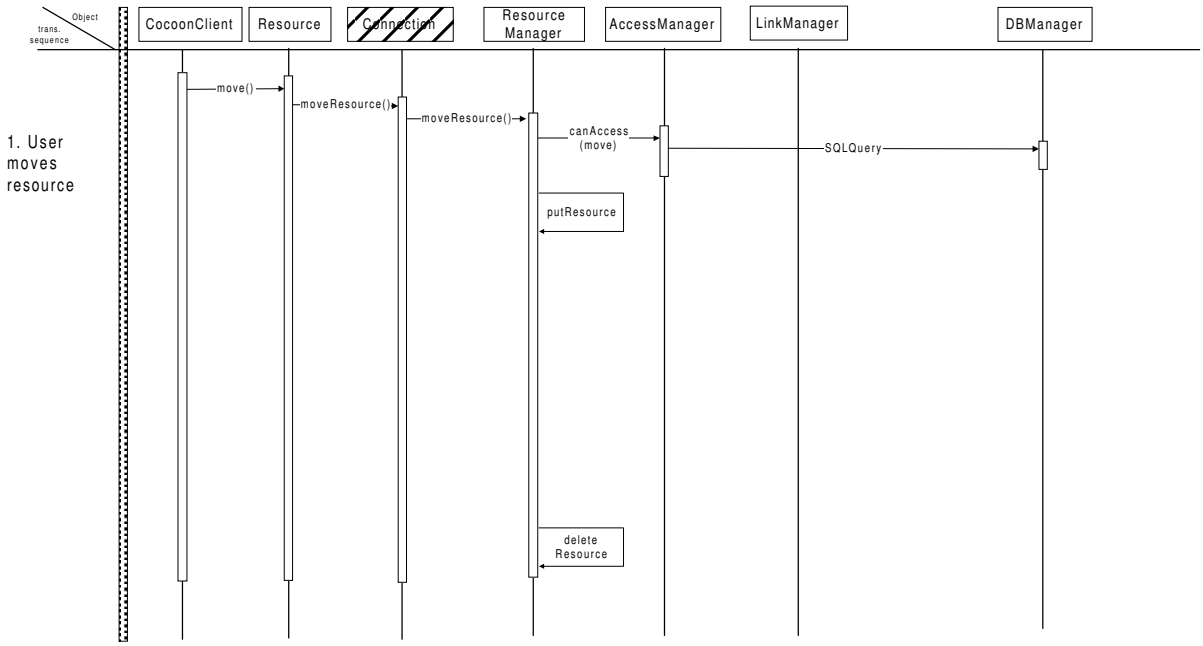



**Abbildung 3-25 Objektinteraktionen für das Verschieben einer Resource**

### 3.1.10 Zugangsmodell

In einem System, das von mehreren Benutzern verwendet wird, muß es die Möglichkeit geben, Dokumente nur für befugte Benutzer zugänglich zu machen. Die Benutzer haben verschiedene Rollen in der Produktion; daher ist es sinnvoll, ihnen rollenbedingte Zugangsrechte zu gewähren.

#### 3.1.10.1 Zugangskontrolle

Dieses System verwendet ein auf Gruppenrechte basierendes Zugangsmodell. Da die Zugangsrechte sich auf eine einzige Publikation beziehen, müssen Gruppen nicht zur Unterscheidung verschiedener Publikationen eingesetzt werden. Damit wird es möglich, jeder Gruppe jeweils eine Rolle zuzuordnen. Zum Beispiel gehören die Graphiker dann zur Gruppe „Graphiker“ und dürfen etwa keine Programmtexte (Source-Code) editieren.

An dieser Stelle muß die Semantik der Zugriffsrechte definiert werden. Sie muß einfach gehalten sein, da die Benutzer technisch kaum bewandert sind und man verhindern möchte, daß durch die Veränderung der Zugangsrechte durch unkundige Benutzer unerwünschte Nebenwirkungen auftreten (z.B. daß sie sich die Zugangsrechte für die eigenen Dokumente wegnehmen).

Ein Benutzer kann zu mehreren Gruppen gehören. Wenn ein Benutzer zur selben Gruppe gehört wie eine Resource, hat er volle Zugangsrechte zu dieser. Ansonsten hat er nur Lesezugriff.

Jede Resource gehört zu genau einer Gruppe. Eine Resource gehört zu der Gruppe, zu der auch das Verzeichnis gehört, in dem sie erzeugt, verschoben, oder kopiert wird. Die Gruppenzugehörigkeit einer Resource kann nachträglich geändert werden.

Ein Verzeichnis, zu dem ein Benutzer Zugang hat, kann auch Resourcen beinhalten, auf die er nicht zugreifen darf, weil sie Gruppen angehören, zu denen er keine Zugangsrechte besitzt. Versucht ein Benutzer, ein solches Verzeichnis zu verschieben oder zu löschen, passiert folgendes: Beim Ausführen dieser Operation überprüft das *Folder*-Objekt zunächst, ob die Operation an allen in dem *Folder* befindlichen Resourcen ausgeführt werden kann. Ist dies der Fall, wird erst dann die Operation ausgeführt. Das bedeutet, daß ein Benutzer Schreibrechte auf alle Resourcen innerhalb eines Verzeichnisses und dessen Unterverzeichnisse haben muß, um diese verschieben oder löschen zu können.

**Es gibt zwei Gruppen, die eine besondere Funktion haben:** Benutzer der Gruppe *cvd* haben das zusätzliche Recht, Dokumente abzunehmen. Benutzer der Gruppe *admin* haben Administrationsrechte zum Erzeugen und Löschen von Gruppen und Benutzern, Importieren von ganzen Serverinhalten und zum Ändern der Servereinstellungen.

Das Löschen von Benutzern ist nur möglich, wenn die Benutzer nicht angemeldet sind. Das Löschen von Gruppen ist nur möglich, wenn es keine Resourcen gibt, die zu dieser Gruppe gehören und keine Benutzer, die diese Gruppe als einzige Gruppe haben. Damit verhindert man,

daß nach Löschen einer Gruppe die Dokumente „verwaisen“ und für alle Benutzer unzugänglich werden.

Die Beziehungen der Gruppen, Benutzer und Resource werden im Abbildung 3-26 dargestellt.

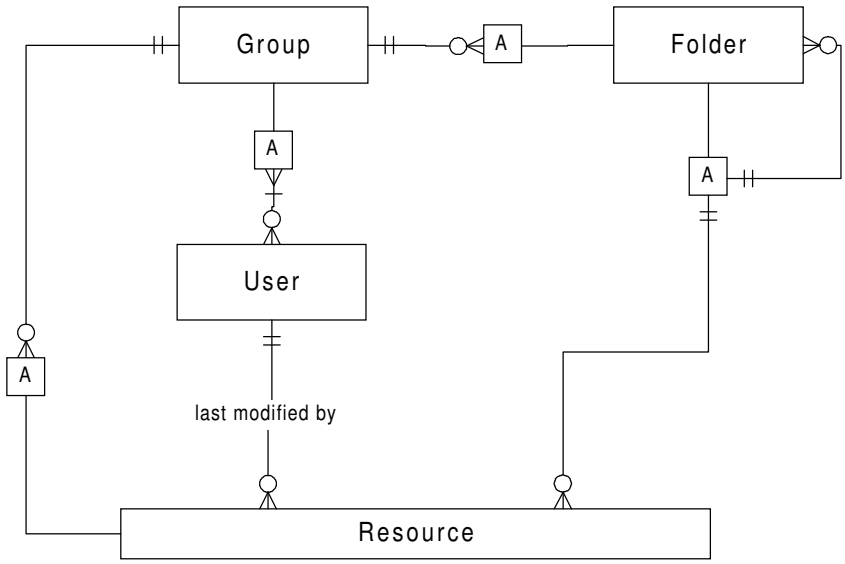



**Abbildung 3-26 Objektrelationen für Gruppen- und Benutzer-Objekte**

### 3.1.10.2 Überschreibungsschutz bei kooperativer Arbeit

Durch die gemeinsame Benutzung von Dokumenten entstehen Probleme, wenn Benutzer versuchen, gleichzeitig eine Resource zu bearbeiten. Wenn z.B. beide Benutzer dieselbe Resource zum Editieren geöffnet haben, und ein Benutzer seine veränderte Version abspeichert gehen diese Änderungen verloren, wenn der zweite Benutzer seine Version speichert. Dieses Problem wird allgemein das Überschreibungsproblem (engl. overwrite problem) genannt. Um solche Probleme zu vermeiden, muß eine Absprache zwischen beiden Benutzern stattfinden.

Es gibt viele Ansätze, um dieses Problem zu lösen. Die am weit verbreitetsten sind [WHIT 97]:

- **Zwischenmenschliche Absprachen:** Diese einfache Absprachemethode setzt voraus, daß die Benutzer durch verbale Absprachen verhindern, die Arbeit des anderen zu überschreiben. Der eine Benutzer wird darüber informiert, daß der andere die Bearbeitung der Resource abgeschlossen hat.
- **Reservierung:** Diese technische Unterstützung der Kollaboration versetzt die Resource in einen Bearbeitungszustand, während sie editiert wird. Versucht ein anderer Benutzer, dieselbe Resource zu editieren, warnt ihn das System, daß die Resource bereits bearbeitet wird. Der Benutzer kann die Resource aber trotzdem editieren. Er trägt die Verantwortung in diesem Fall, denn das System informiert ihn nur.
- **Strenger Ausschluß (Strict Locking):** Mit dieser Methode wird die Resource, die editiert wird, vom System für alle anderen Benutzer zum Schreiben gesperrt.

Der folgende Ansatz übernimmt das Konzept des strengen Ausschlusses, da die Benutzer eventuell über verschiedene Zeitzonen verteilt sind, was zwischenmenschliche Absprachen erschwert. Nur durch strengen Ausschluß kann die Konsistenz der Resourcen bewahrt werden. Dabei wird im Client optisch sichtbar, welche Resourcen gerade bearbeitet werden; der Benutzer kann zusätzlich nachsehen, von wem. Wird die Resource wieder freigegeben, ist diese Zustandsänderung sofort beim Client des Benutzers sichtbar (siehe 3.1.15).

Ob eine Resource gerade bearbeitet wird oder nicht, wird durch ein Flag in der Datenbanktabelle dargestellt. Bei Symlinks gehört der Flag zu den gemeinsamen Daten, damit der Inhalt nicht von zwei verschiedenen Links aus gleichzeitig bearbeitet wird. Das bedeutet, daß, wenn ein Symlink bearbeitet wird, alle Symlinks, die dieselben Daten benutzen, sich auch im Zustand der Bearbeitung befinden.

Um eine Resource bearbeiten zu können, muß sie vom System vorher ausgecheckt werden, wie in einer Bibliothek, wo ein Buch am Schalter ausgeliehen (ausgecheckt) wird. Ist eine Resource ausgecheckt, so ist sie nur für den Benutzer zugänglich, der sie ausgecheckt hat. Nach der Bearbeitung einer Resource wird sie wieder eingecheckt (das Buch wird zurückgegeben) und ist dann für andere Benutzer wieder verfügbar.

Eine Resource kann nur von dem Benutzer eingecheckt werden, von dem sie auch ausgecheckt worden ist.

Das *Connection*-Objekt verwaltet eine Liste der von seinem Client ausgecheckten Resourcen. Bei einem Verbindungsabbruch checkt das *Connection*-Objekt die entsprechenden Resourcen wieder ein, um sie für die Verwendung durch andere Benutzer wieder freizugeben.

#### 3.1.10.3 Workflowunterstützung

Eine Resource durchläuft einige Entwicklungsphasen, bis sie für den Endbenutzer bereitgestellt wird. In diesen verschiedenen Phasen sind nur bestimmte Operationen mit der Resource erlaubt. Diese Phasen werden als Zustände der Resource definiert. Die Resource erlaubt oder verweigert, je nach ihrem Zustand, die Ausführung einer Operation.

Eine Resource durchläuft folgenden typischen Fertigungsablauf: Wenn ein Benutzer eine Resource erstellt, befindet sie sich im Zusand „¬completed“. Er bearbeitet die Resource solange, bis er mit dem Ergebnis zufrieden ist. Die Resource befindet sich dann im Zustand „*completed*“. Sie muß danach beim Chef vom Dienst (CvD) zum Akzeptieren vorgelegt werden. Nachdem der CvD die Resource akzeptiert (abgenommen) hat, befindet sich die Resource im Zustand „accepted“ und kann publiziert werden. Nach einer Änderung der Resource befindet sie sich wieder im Zustand „¬completed“ und muß die oben genannten Entwicklungsphase in derselben Reihenfolge wieder durchlaufen. Die Resource befindet sich im Zustand „in Bearbeitung“ wenn sie nicht im Zustand „*completed*“ oder „*accepted*“ ist. Alle Zustände werden durch Flags in der Datenbank und boolesche Variablen in den Resource-Objekten dargestellt.

Nur im „accepted“ Zustand ist es möglich, die Resource zu publizieren. Man möchte damit verhindern, daß Resourcen publiziert werden, die noch nicht einen vorzeigbaren Zustand erreicht haben. Eine Resource kann nicht vom CvD als „*accepted*“ markiert werden, wenn der Autor sie noch nicht als „*completed*“ markiert hat. Diese Bedingung verhindert, daß der CvD ein noch nicht fertiges Dokument zum Publizieren freigibt. Der CVD hat die Möglichkeit, eine Liste der noch zu akzeptierenden Resourcen erstellen zu lassen. Dadurch kann er in Erfahrung bringen, welche Resourcen noch zu akzeptieren sind.

Nachdem die Resourcen als „accepted“ markiert sind, können sie entweder manuell oder automatisch (zu einen vorher festgelegten Zeitpunkt) publiziert werden.

Die Zustände und die zustandsabhängigen Funktionen sind im folgenden Objektlebenszyklusdiagramm dargestellt.

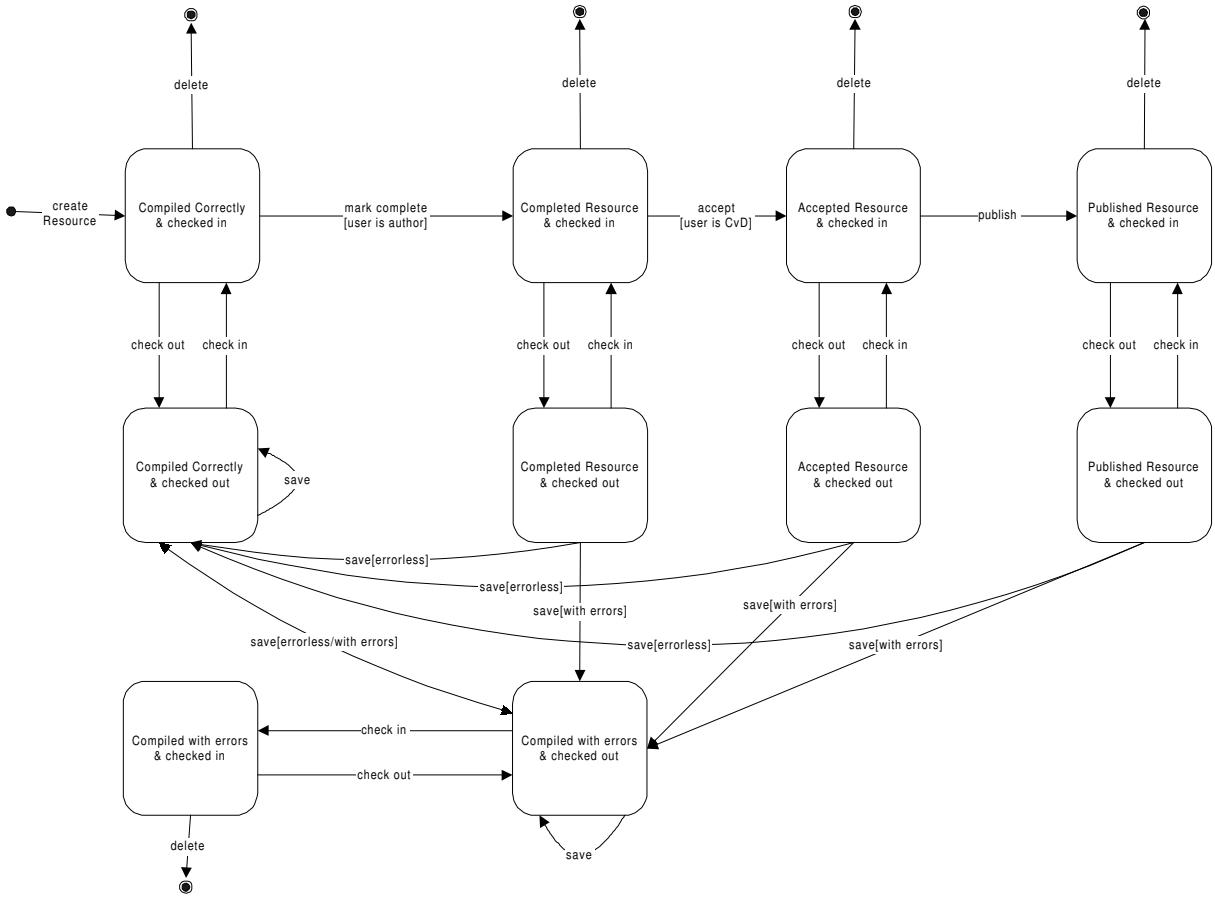



**Abbildung 3-27 Zustände einer Resource und die erlaubten Operationen**

Die in diesem Kapitel beschriebenen Aufgaben werden vom *AccessManager* übernommen. Bei jeder Funktion, die vom *ResourceManager* ausgeführt wird, wird der *AccessManager* gefragt, ob die Funktion mit dem angegebenen Benutzer und der angegebenen Resource zulässig ist. Die Funktion wird erst ausgeführt, wenn der *AccessManager* es zuläßt. Es gibt je einen *AccessManager* pro angemeldeten Benutzer. Beim Anmeldevorgang wird der Benutzer im

*AccessManager* gespeichert. Diese *AccessManager*-Instanz ist dann für alle Operationen dieses Benutzers zuständig. Der *AccessManager* bietet für den Administrator zusätzliche Funktionen zur Verwaltung der Benutzer und Gruppen an (siehe 2.2.4).

#### 3.1.10.4 Die *Group*-Klasse

Die *Group*-Klasse stellt ein Gruppe im System dar. Sie ist eine Unterklasse von *Folder*. Diese Klasse wird vom *ResourceManager* aus der Tabelle „*Group*s“ erzeugt. Sie beinhaltet die Benutzerobjekte und Funktionen zur Selbstmanipulation.

#### 3.1.10.5 *Die User*-Klasse

Die *User-Klasse* stellt einen Benutzer im System dar. Sie beinhaltet alle Gruppen, in denen der Benutzer Mitglied ist. Sie ist eine Unterklasse von Resource und beinhaltet Funktionen zur Selbstmanipulation.

### 3.1.11 Konsistenzüberprüfung von Querverweisen (Link-Management)

Eine fundamentales Merkmal von Hypertext ist die Möglichkeit, Querverweise auf andere Dokumente (Objekte) zu erstellen. In HTML können Querverweise auf eine andere Stelle im selben Dokument, auf ein lokales Dokument oder auf ein Dokument irgendwo im Internet verweisen.

#### 3.1.11.1 Probleme bei der Verwaltung von Querverweisen

HTML bietet nur eine primitive Verweismethodik an und kann nur unidirektionale Querverweise erzeugen. Das bedeutet, daß, wenn Dokument A auf Dokument B verweist, Dokument B keine Informationen über den Verweis enthält.

**Abbildung 3-28 Ein unidirektionaler Verweis von A nach B**

Die Lage des Dokumentes bei einem HTML-Link wird über den Uniform Resource Locator[24] (URL) beschrieben. Ein URL beinhaltet das Protokoll, welches genutzt werden soll (HTTP, FTP, u.ä.), die IP-Adresse oder Namen des Servers, die Pfadangaben und den Dateinamen. Ein URL sieht dann wie folgt aus:

*Protokollname://IP-Addresse/Pfad/zu/meiner/Datei.*

[24] Ein URL wird zur eindeutigen Kennzeichnung einer Ressource im Internet verwendet.[BERN 94]

Ein Verweis auf einer HTML-Seite z.B. auf den GEO Server, sieht dann wie folgt aus:

*http://www.geo.de/Pfad/zu/meiner/Datei.html*

Da der URL für das Beschreiben eines Querverweises das Dateisystem benutzt und eine IP-Adresse angegeben werden muß, entsteht ein Problem, wenn Dokumente verschoben werden oder die IP-Adresse geändert wird: Da die Verweise auf die veränderten Dokumente sich auf die Position des Dokuments im Verzeichnisbaum bei der spezifischen IP-Adresse beziehen, werden alle Verweise auf die alte Position des verschobenen Dokuments ungültig. Eine andere Problematik tritt auf, wenn Seite A auf Seite B zeigt und Seite B gelöscht wird. Es existiert kein Mechanismus, damit Seite A eine Benachrichtigung über das Löschen der Seite B erhält, und deswegen verweist dann Seite A immer noch auf eine nun nicht mehr vorhandene Seite B.

#### 3.1.11.2 Lösungen anderer Organisationen

Für diese Probleme gibt es verschiedene Lösungsansätze:

##### 3.1.11.2.1 Uniform Resource Names (URN)

Die von der Internet Engineering Task Force (IETF) [BERL 94] vorgeschlagene Lösung basiert darauf, daß anstatt von URLs die Objekte einen global eindeutigen Identifikationsnamen erhalten. Dabei wird die Einrichtung einer zentralen Instanz, die die Anmeldung von Objekten im Netz entgegennimmt, erforderlich, ähnlich dem DNS[25], der die Internet-Adressen in IP-Nummern übersetzt. Jedes neue Objekt im Netz muß dann von dieser Instanz eine Identifikationsnamen erhalten.

Auf dieses Objekt kann dann immer über den Identifikationsnamen zugegriffen werden, ganz gleich, wo es sich im Netz befindet. Das Objekt behält den Namen, solange es existiert. Beim Zugriff auf das Objekt wird durch den Nameserver der gegenwärtige Ort des Objektes ermittelt. Mit dieser Ortsangabe kann das Objekt dann geholt werden. Diese Idee stammt aus dem Bibliothekswesen und entspricht einer ISBN-Nummer. Diese Lösung setzt voraus, daß alle existierenden DNS-Server auf das neue System umgestellt werden und daß alle auf das Internet basierenden Systeme und Dokumente sich auf diese Struktur umstellen.

##### 3.1.11.2.2 Persistent URLs (PURLs)

Da diese Entscheidungen sehr langwierig sein können, bietet das OCLC[26] eine Zwischenlösung an, bei der die URLs wenigstens innerhalb einer Website persistent sind. Durch die lokale Verwendung eines (veränderten) DNS-Servers und eines (veränderten) WWW-Servers, der die ID-Namen der Objekte über diesen DNS-Server auflöst, können die lokalen Objekte einer Website immer mit einem eindeutigen Identifikationsnamen arbeiten. Damit die sogenannten Persistent URLs (PURLs) von handelsüblichen Web-Browsern verwendet werden können, sind sie zum URL-Format kompatibel. Der Unterschied der PURLs zum URL-Format liegt darin, daß die Angabe des Pfads und Dateinamens einen für die Website eindeutigen Identifikationsnamen darstellt und nicht, wie bei URLs, die Stellung der Datei im Dateisystem beschreibt.

[25] Domain Name Service. [WETT 97]
[26] Online Computer Library Center. [WEIB 96]

Wenn ein Verweis verfolgt und das Dokument von diesem Webserver abgefragt wird, löst der Server den Identifikationsnamen mit Hilfe des speziellen DNS-Servers auf. Der DNS Server liefert die Position der Datei im Dateisystem des Servers zurück. Danach kann der Server das angeforderte Dokument ausgeben. Wenn das Dokument verschoben wird, stellt man die neue Position der Datei im DNS-Server ein. Damit ist sie weiterhin über denselben Identifikationsnamen erreichbar.

#### 3.1.11.3 Der Lösungsansatz dieses Systems

Obwohl der Ansatz des OCLC seine Vorteile hat, wird das Problem des Löschens der Dokumente damit nicht gelöst. Außerdem kann man nicht mehr die üblichen Web-Server verwenden, sondern muß auf einen von dem OCLC veränderten Webserver und DNS-Server zurückgreifen. Diese Systeme sind allerdings noch nicht auf Sicherheit und Zuverlässigkeit erprobt, und die veränderten Web-Server bieten nicht alle Fähigkeiten eines modernen Web-Servers an. Entscheidet man sich für diese Lösung, wäre man nicht mehr produktunabhängig.

Um die Probleme der Ortsabhängigkeit der Verweise zu lösen, wird in diesem System die Idee der eindeutigen Identifikationsnamen der OCLC übernommen. Hier referenzieren die Verweise allerdings auf eindeutige, ortsunabhängige ID-Nummern (nicht Namen) für die Resourcen. Durch die Benutzung von ID-Nummern im Gegensatz zu Namen ist eine Performancesteigerung beim Verwalten der Verweise in der Datenbank zu erwarten. Es findet hier im Gegensatz zur OCLC-Methode keine Namensauflösung statt.

Die Dateien werden im Verzeichnis des Web-Servers alle in einem Verzeichnis ohne jegliche Unterverzeichnisse in einer flachen Struktur gespeichert und erhalten als Dateinamen ihre Resource-ID (siehe 3.1.7). Der Verweis auf eine Resource mittels der Resource-ID ist damit gleichzeitig der Verweis auf den Dateinamen. Diese Methode hat zwei Vorteile gegenüber PURLs: Zum einen ist die Performance besser, da keine zusätzliche Namensauflösung beim Liefern eines Dokuments durch den WWW-Server stattfinden muß. Zum anderen müssen Web-Server und DNS-Server nicht verändert werden, da wie üblich Dateinamen verwendet werden. Dadurch muß beim Verschieben der Dokumente nichts mehr korrigiert werden, da eine Resource immer über die eindeutige Resource-ID referenziert wird.

##### 3.1.11.3.1 Formate und Attribute der Querverweise

Um Querverweise zu erstellen, gibt es eine Vielzahl von Standards, die alle auf SGML basieren. Die wichtigsten dieser Standards sind:

1. Hypertext Markup Language (HTML) [KENN 97]
1. HyTime [ISOR 97]
1. Text Encoding Intitiatives Guidelines (TEI P3) [TEIN 97]
1. Extended Markup Language (XML) ][W3CX 97]

Das von HTML verwendete URL-Format ist die am weitesten verbreitete Methode, um in Online-Medien Querverweise zu erstellen. Die Standards von HyTime und TEI sind sehr viel mächtiger, aber setzen voraus, daß die Dokumente im Endprodukt auch SGML-Dokumente sind. Da das Produktionssystem SGML lediglich als Quellformat verwendet, das dann in andere Formate für Online-Medien umgewandelt wird, können die Vorteile dieser Standards nicht ausgenutzt werden. Das Linkformat von XML wäre hier die beste Alternative, da es für Online-Medien konzipiert ist. Obwohl es noch in der Spezifikationsphase ist, verwendet dieses System einige Elemente der momentanen Definition eines XML-Links.

Zuständig für den Entwurf eines Standardmechanismus hinsichtlich Hyperlinks in XML ist die Arbeitsgruppe W3C-SGML-ERB (ERB = Editorial Review Board), die auch an XML insgesamt arbeitet. Als Vorbilder dienen das Linkmodell der Text Encoding Initiative (TEI), einer der wichtigsten SGML-Sprachen im akademischen Bereich, der Norm Hytime (ISO 10744) und der URL des WWW.

Jeder Link in XML besitzt einen oder mehrere Lokatoren, ein Verhalten und optional einen Namen. Im einfachsten Fall handelt es sich bei einem Lokator um einen URL, der eine Ressource adressiert. Wenn der Lokator auf ein anderes XML-Dokument zeigt, kann durch Anhängen eines Xlink an einen URL innerhalb der Baumstruktur des referenzierten Dokuments weiter navigiert werden.
XLinks sind eine spezielle Abfragesprache, deren Fähigkeiten der Standard Document Query Language des DSSSL[27]-Standards ISO 10179 nicht unähnlich sind (siehe 3.1.13.1.5). Die Dokumentelemente des Ergebnisses dieser Anfrage werden als durch den Lokator referenzierte Ressourcen angesehen.
Ein Beispiel: Der Lokator

*http://www.geo.de/wissen/heft1#CHILD(artikel,3)(seite,2)*

wählt die zweite Seite aus dem dritten Artikel des ersten GEO-Heftes (Heft1) aus. Das funktioniert natürlich nur, wenn das ganze Heft als einzige Dokumentinstanz gespeichert ist und die Tags <artikel> und <seite> verwendet werden.

Das Verhalten bestimmt, wie ein Link ausgelöst wird, entweder automatisch beim Antreffen (AUTO) oder erst durch Benutzerwunsch (USER). Eine andere Achse des Verhaltens legt fest, was mit der referenzierten Ressource geschehen soll.

Das Ergebnis der Verfolgung des Links kann entweder in die aktuelle Ressource eingebettet werden (EMBED), sie ersetzen (REPLACE) oder eine neue Ressource erzeugen (NEW), etwa in einem eigenen Fenster. Ein Link mit dem Verhalten AUTO und EMBED entspricht in seiner Funktion also weitgehend dem <IMG> -Tag von HTML, der ein Bild in das Dokument einfügt.

Eine XML-Anwendung erkennt als Anker eines Hyperlinks dienende Tags daran, daß ein Attribut namens XML-LINK in ihrer Attributliste vorhanden ist. Es typisiert den Link als einfach (SIMPLE), erweitert (EXTENDED) oder Linkgruppe (GROUP). Der einfache Link erlaubt die genannten Mechanismen zur Kontrolle seines Verhaltens, kann aber nur einen Lokator enthalten.

[27] Document Style Semantics and Specification Language. [COVE 97]

Er entspricht damit in etwa dem <A> von HTML. Ein erweiterter Link kann beliebig viele Lokatoren enthalten, etwa um Varianten einer Ressource in verschiedenen Sprachen anzubieten. Die Linkgruppe betrachtet die Links mehrerer Dokumente gleichzeitig, um zum Beispiel semantische Netze oder bidirektionale Links auszudrücken.

Dieses System verwendet nur Links vom Typ „SIMPLE“. Andere werden nicht unterstützt. Das Verhalten der Links (AUTO, USER) und der Verfolgung (EMBED, REPLACE, NEW) und die Xlinks werden ebenfalls verwendet.

Da die Links dieses Systems auf Resource-IDs verweisen sollen, müssen die Lokatoren auf diese Nummer verweisen. Ein lokaler Link sieht dann wie folgt aus:

*<A XML-LINK="SIMPLE" HREF="1098476#CHILD(artikel,3)(seite,2)" ACUATE=“USER“ SHOW=“REPLACE“ >Heft 1 des GEO Magazins</A>*

Dieser Link verweist auf die zweite Seite aus dem dritten Artikel des Dokumentes mit der Resource-ID 1098476. Durch SHOW=“REPLACE“ wird beim Verfolgen dieses Links die vorherige Seite durch die neue ersetzt. Der Link wird nicht automatisch ausgeführt, weil ACTUATE=“USER“ gesetzt ist.

Ein Verweis auf Daten im Internet (externer Link) sieht so aus:

*<A XML-LINK="SIMPLE" HREF="http://www.netscape.com/" ACUATE=“USER“ SHOW=“REPLACE“ >Homepage von Netscape</A>*

Bei externen Links sind Xlinks und das Attribut „EMBED“ nicht erlaubt. Diese können erst angewendet werden, wenn alle Dokument im Internet als XML-Dokumente gespeichert werden.

#### 3.1.11.3.2 Konvertierung der Querverweise nach HTML

Beim Konvertieren der XML-Dokumente in HTML-Dateien werden die Resource-IDs der lokalen Links in die Dateinamen der Zieldateien aufgelöst: Die Resource-ID 1098476 wird in den Dateinamen „1098476.html“ umgewandelt. Wenn ACUATE=“AUTO“ gesetzt ist, wird der Link in der HTML-Datei in einen Client-Pull[28] Mechanismus des HTML-Browsers umgewandelt. REPLACE und NEW werden durch Attribute der HTML-Tags realisiert. Wenn EMBED bei Texten verwendet wird, dann wird der Text, auf den referenziert wird, während der Konvertierung in das Hauptdokument eingefügt und zusammen mit diesem als ein Dokument gespeichert.

Wenn der Resourcetyp, auf den verwiesen wird, ein Bild ist und EMBED gesetzt ist, wird ein HTML-IMG-Tag verwendet. Werden mit EMBED andere Binärdateien referenziert, wird der OBJECT-Tag von HTML verwendet.

[28] Ein Mechanismus in WWW-Browsern, mit dem nach Aufruf einer Seite automatisch mehrere Seiten hintereinander aufgerufen werden.

#### 3.1.11.3.3 Verwaltung der Querverweise in der Datenbank

Die Querverweise werden in einer Tabelle in der Datenbank gespeichert. Diese Tabellen werden Webs [STAN 96] genannt (Dieser Name wird sich wahrscheinlich ändern, da dies auch die Abkürzung für das World Wide Web ist). Nur die Querverweise der aktuellen Version des Dokuments werden in der Link-Tabelle gespeichert. Die Verweise älterer Versionen werden nicht berücksichtigt.(siehe 3.1.12).

Beim Link-Management können natürlich nur Verweise innerhalb des Produktionssytems berücksichtigt werden. Verweise auf Resourcen im Netz dagegen müssen durch periodisches Überprüfen kontrolliert werden, da das System keine Benachrichtigung erhält, wenn ein Dokument auf einem anderen Server gelöscht oder verschoben wird.

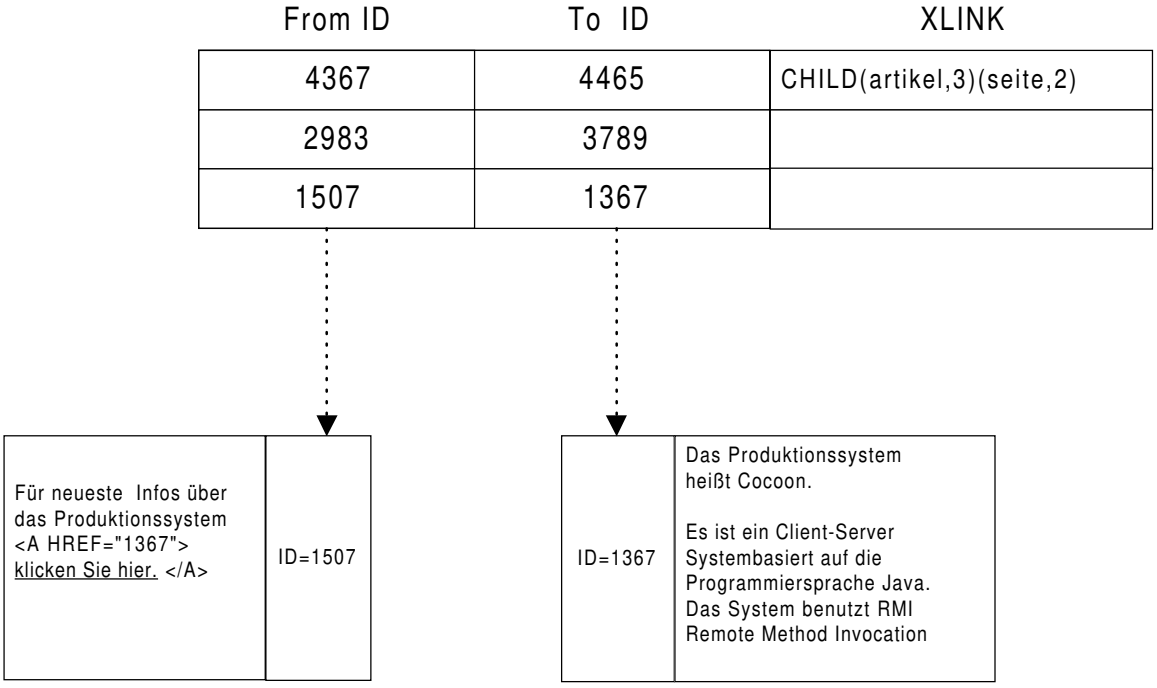



**Abbildung 3-29 Speicherung der Links in der Datenbank**

Die Link-Tabelle enthält folgende Felder :

1. Die Resource-ID des Dokuments, das den Verweis beinhaltet.
1. Die Resource-ID des Dokuments, auf das der Verweis zeigt.
1. Ein Feld, das die Xlinks als String speichert.

Durch diese Tabelle kann man die Querverweise in beide Richtungen verfolgen: Man kann sowohl alle Dokumente auflisten, auf die ein bestimmtes Dokument verweist, als auch alle Dokumente, die auf ein bestimmtes Dokument verweisen.

Durch das XLINK-Feld kann man überprüfen, ob die Dokumentenstruktur, auf die verwiesen wird, noch existiert.

Im Folgenden werden die Funktionen des Link-Managements erläutert. Diese Funktionen sind Aufgabe der Klasse *LinkManager*. Die *LinkManager*-Klasse übernimmt alle Aufgaben, die mit dem Verwalten der Querverweise zusammenhängen. Für die Datenbankzugriffe verwendet sie die Dienste des *DBSession*-Objekts.

Folgende Hauptfunktionen werden vom *LinkManager* implementiert.

### 3.1.11.4 Operationen des Link-Managements

Beim Verändern einer Resource muß das Link-Management aktiv mitwirken.

#### 3.1.11.4.1 Speichern von Resourcen

Beim Speichern einer Resource werden alle Querverweise, die die Resource enthält, in der Datenbanktabelle gespeichert, und der Benutzer wird über ungültige Links informiert. Wenn ein Link über Xlink auf eine Stelle in der Dokumentenstruktur verweist, wird dieser Link ungültig, wenn sich die Dokumentenstruktur ändert. In diesem Fall wird der Benutzer über den ungültigen Xlink informiert.

Eine strengere Verweisüberprüfung findet erst beim Publizieren statt. So kann man gewährleisten, daß der öffentliche Server keine ungültigen Verweise enthält, und gleichzeitig wird auf dem Produktionsserver die Arbeit nicht durch zu strenges Link-Management behindert; zum Beispiel könnte man bei zu strenger Verweisüberprüfung keine Verweise auf noch nicht existierende Dokumente erstellen, was sehr hinderlich wäre.

#### 3.1.11.4.2 Verschieben (move) von Resourcen

Da hier ortsunabhängige Resource-IDs benutzt werden, müssen die Dokumente, die einen Verweis auf die verschobene Resource haben, nicht verändert werden.

#### 3.1.11.4.3 Kopieren (copy) von Resourcen

Beim Kopieren einzelner Resourcen ist seitens der Querverweise nichts zu berücksichtigen. Beim Kopieren von Verzeichnissen hingegen muß aber sichergestellt werden, daß die Verweise innerhalb des Verzeichnisses und seine Unterverzeichnisse auf die kopierten Resourcen verweisen und nicht mehr auf die Originalresourcen.

#### 3.1.11.4.4 Löschen von Resourcen

Beim Löschen einer Resource werden Verweise auf diese ungültig. Der Benutzer erhält vom System die Meldung, daß der Verweis in einigen Dokumenten ungültig ist und zeigt ein Dokument als Beispiel an. Der Benutzer muß dann den Querverweis in diesem Beispieldokument korrigieren. Dieselbe Korrektur wird dann vom System bei allen anderen Dokumenten durchgeführt, die einen Querverweis auf das gelöschte Dokument hatten. Die Korrektur des Beispieldokuments kann nicht automatisch vorgenommen werden, da diese Änderung semantischer Natur ist.

Der Benutzer kann nur dann eine Resource löschen, wenn er Änderungsrechte auf alle Resourcen hat, die auf die zu löschende Resource verweisen. Wenn Resource A auf Resource B verweist und Resource B gelöscht werden soll, bedeutet dies implizit, daß A auch editiert wird. Daher muß der Benutzer die Rechte für diesen Vorgang besitzen. Beim Verschieben von Seite B dagegen muß A nicht editiert werden. Deswegen entfällt diese Einschränkung für das Verschieben der Resource B.

Die Konsistenz der Verweise auf dem öffentlichen Server werden durch den Löschvorgang nicht beeinträchtigt, da das entsprechende Dokument auf dem öffentlichen Server erst gelöscht wird, wenn alle Resourcen, die darauf verweisen, korrigiert und publiziert worden sind. Das bedeutet für den öffentlichen Server, daß Dokumente nur gelöscht werden, wenn keine Dokumente auf dem öffentlichen Server auf sie verweisen.(siehe 3.1.11.4.5)

#### 3.1.11.4.5 Das Publizieren („Heißschalten“) von Resourcen

Um die Konsistenz der Verweise auf dem öffentlichen Server gewährleisten zu können, muß die Verweisüberprüfung beim Publizieren streng gehandhabt werden. Hier wird das Konzept einer *Collection* definiert: Die *Collection* einer Resource ist die Resource selber und alle Resourcen, auf die sie verweist.

Folgende Schritte werden beim Publizieren durchlaufen:

**I. Überprüfen, ob die Resource publiziert werden kann**

Damit eine Resource publiziert werden kann, müssen folgende Bedingungen erfüllt sein: Die *Collection* der Resource muß als „completed“ und „accepted“ markiert sein. Außerdem muß sie ohne Fehler konvertiert sein.

Wenn eine Resource aus der *Collection* in der aktuellen Version bereits publiziert ist , wird sie nicht noch einmal publiziert.

Wenn eine Resource aus der *Collection* nicht publizierbar ist, müssen folgende Fälle berücksichtigt werden:

1. **Die Resource existiert in einer älteren Version auf dem öffentlichen Server:** In diesem Fall wird eine Warnung ausgegeben, daß die aktuelle Version nicht publiziert wurde. Der Publiziervorgang der *Collection* wird fortgesetzt, da durch diesen Vorgang keine ungültigen Querverweise entstehen.
1. **Die Resource existiert nicht auf dem öffentlichen Server:** Der gesamte Publikationsvorgang wird abgebrochen.

**II. Resource publizieren**

Beim Publiziervorgang passiert folgendes:

A) Die Resourcen aus der *Collection* werden zum öffentlichen Server transferiert. Diese Funktion arbeitet rekursiv und verfolgt in der Linktabelle alle Verweise, die von dem zu publizierenden Dokument ausgehen und erstellt eine Liste der Dokumente, auf die die

Resource direkt oder indirekt verweist (Elemente der *Collection*). Die Funktion enthält als Parameter den Verweis, den sie zuletzt bearbeitet hat. Damit verhindert man, daß, wenn zwei Dokumente einander gegenseitig referenzieren, die Funktion in eine unendliche Schleife gerät. Gleichzeitig wird eine Liste der von dieser Funktion bereits verfolgten Verweise erstellt. Wenn die Funktion in einem Dokument auf einen Verweis trifft, der bereits verfolgt wurde, wird dieser nicht nochmal verfolgt. Damit verhindert man, daß diese Funktion bei (indirekten) Zyklen in der Verweisstruktur eine unendliche Schleife bildet. Wenn alle Resourcen auf ihre Publikationstauglichkeit überprüft worden sind, wird die Rekursion nachgeklappert und die Resourcen, die keine Verweise besitzen, werden zuerst publiziert. Dadurch wird die Anzahl der ungültigen Links auf dem öffentlichen Server zur Zeit der Publikation minimiert. Die publizierten Ausgabedateien werden mit ihren Dateinamen und Datum der Publikation in der Tabelle „published resources“ gespeichert. (siehe 3.1.12).

A) Wenn Resourcen auf dem internen Server gelöscht worden sind, dann wird überprüft, ob sie auch auf dem öffentlichen Server gelöscht werden dürfen. Dafür müssen folgende Bedingungen erfüllt sein:

   a) Alle Links auf die gelöschte Resource müssen entfernt worden sein.
   b) Diese Resourcen müssen publiziert worden sein, da die Referenzen auch auf dem öffentlichen Server nicht existieren dürfen.

A) Die gelöschten Resourcen werden auf dem öffentlichen Server gelöscht, wenn die Überprüfung in B) es zuläßt.

An dieser Stelle könnte man auch die externen Querverweise überprüfen, aber da das Internet für solche umfangreiche Überprüfungen nicht zu jeder Tageszeit schnell genug ist, ist es nicht praktikabel. Die Überprüfung der externen Querverweise geschieht periodisch für den gesamten Inhalt. Dies geschieht durch ein bereits vorhandenes Werkzeug (Spider), das periodisch alle Seiten der Website durchsucht und alle ungültigen Links dem Administrator meldet.

### 3.1.12 Versionsverwaltung

Viele Dokumente existieren nicht nur in einer Form. Sie werden wieder editiert, erweitert oder sogar aufgeteilt und in andere Dokumente eingebaut.

#### 3.1.12.1 Probleme der Versionsverwaltung bei Hypertext-Dokumenten

Durch diese Vorgänge entstehen mehrere Probleme, etwa: Worauf soll ein Querverweis zeigen, nachdem sich das Originaldokument geändert hat ?

Betrachten wir zur Erläuterung folgenden Fall:
Nehmen wir an, daß Herr Adam einen Artikel schreibt und publiziert. Einige Zeit später liest Frau Meyer diesen Artikel, findet einen Fehler darin und schreibt eine Kritik darüber, die auf den Artikel verweist. Herr Adam liest daraufhin die Kritik und korrigiert seinen Artikel. So weit so gut.

Aber was passiert, wenn ein Dritter zu einem späteren Zeitpunkt die Kritik liest ? Er wird dann dem Querverweis zum Artikel von Herrn Adam folgen. Soll er jetzt die alte oder die neue (korrigierte) Version des Artikels zu Lesen bekommen ?

Wenn er die alte Version bekommt, entsteht der (richtige) Eindruck, daß Frau Meyer einen Fehler im Artikel gefunden hat, aber er bekommt nicht die Information, daß dieser Fehler korrigiert worden ist. Wenn der Leser andererseits nur den korrigierten Artikel zu sehen bekommt, entsteht der (falsche) Eindruck, daß Frau Meyer nicht existierende Fehler im Artikel bemängelt hat.

Douglas Engelbarts „Augment“ System [ENGE 84] löste dieses Problem schon in den 60er Jahren. Jedesmal, wenn ein Dokument erstellt wurde, wurde eine Kopie dieser Version des Dokuments permanent gespeichert. Die Verweise auf dieses Dokument blieben dadurch auch semantisch immer gültig. Wenn ein Verweis verfolgt wurde, bekam der Benutzer einen Hinweis, daß eine neuere Version des Dokuments existiert.

Da solche Systeme nur angewendet werden können, wenn das gesamte System (bestehend aus Browser, Server und Produktionssystem) Verweisversionen unterstützt, kann dieses Problem bei der Betrachtung von Dokumenten mit WWW-Browsern nicht gelöst werden, da sie dies (zur Zeit) nicht unterstützen.

Ein anderes Problem entsteht, wenn alte Versionen einer Resource wiederhergestellt werden.
Für die Verwaltung der Querverweise in dieser Resource gibt es verschiedene Möglichkeiten:

A) Die Dokumente, auf die dieses Dokument verweist, werden mit der gleichen Versionsnummer wiederhergestellt. Hier entsteht das Problem, daß der Benutzer, der die Version des Dokuments A wiederherstellt, eventuell nicht die Rechte für die Dokumente hat, auf die Dokument A verweist und sie damit nicht wiederherstellen kann.
   Es werden auch gelöschte Dokumente, auf die das Dokument verweist, wiederhergestellt. Das bedeutet, daß Dokumente überhaupt nicht gelöscht werden dürfen, sondern in einer Sonderablage gespeichert werden.

A) Man führt Versionsverwaltung für Links ein. Wenn z.B. eine neue Version von Dokument B entsteht, muß die alte Version von Dokument A, die auf B verweist, korrigiert werden, so daß sie auf die alte Version von B verweist.
A) Man berücksichtigt nur Links der aktuellen Versionen der Dokumente und überläßt die Überprüfung der semantischen Integrität der Links dem Benutzer. Nur die technische Integrität wird durch das System garantiert.

Nach näherer Betrachtung erscheint Möglichkeit C) als die vernünftigste. Der Grund dafür ist der folgende: Ein Benutzer sollte wissen, was für Folgen seine ausgeführten Operationen auf das System haben. Es ist aber nahezu unmöglich, bei Möglichkeiten A) und B) den Überblick zu behalten, wenn eine einzige Operation (Wiederherstellen einer Version eines Dokumentes) einen kaskadierenden Vorgang zur Folge hat.

### 3.1.12.2 Lösungsansatz

Dieses System verwendet deshalb folgende Vorgehensweise:
Alle Querverweise zeigen immer auf die aktuelle Version einer Resource. Es werden nur die Querverweise, die in der aktuellen Version des Dokumentes existieren, im Link-Management berücksichtigt. Beim Editieren und Abspeichern einer Resource wird der Inhalt unter der Versionsnummer *aktuelle_version +1* gespeichert. Beim erneuten Abspeichern wird immer die gleiche Versionsnummer benutzt. Eine Version wird erst eingefroren, wenn sie publiziert wird. Nach dem Publizieren wird die aktuelle Versionsnummer um eins erhöht. Dadurch ist die Version festgehalten, denn beim nächsten Abspeichern bekommt der Inhalt eine neue Versionsnummer.

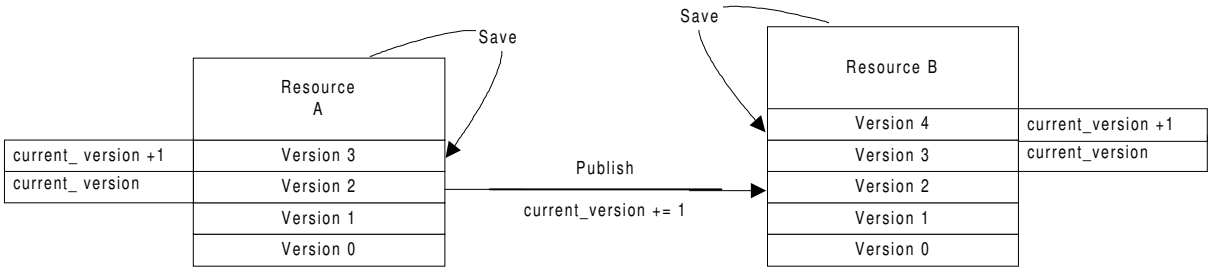



**Abbildung 3-30Mechanismen der Versionsverwaltung**

Folgende Operationen der Versionen sind vorhanden:

#### 3.1.12.2.1 Aufzählung vorhandener Versionen und Kurzbeschreibung

Hier erhält man eine Liste der vorhandenen Versionen.

#### 3.1.12.2.2 Wiederherstellen (Retrieve) einer Version

Beim Wiederherstellen einer alten Version wird diese zur aktuellen Version. Dabei werden Querverweise auf Dokumente, die gelöscht worden sind, verfolgt und die entsprechenden

Resourcen wiederhergestellt. Dies geschieht dadurch, daß deren „delete“ Flag gelöscht wird. Verweise, die auf nicht gelöschte Resourcen zeigen, verweisen auf die aktuelle Version der Resource. In der Linktabelle werden die Verweise der aktuellen Version durch die der wiederhergestellten Version überschrieben (wie beim Speichern einer Resource).

Es ist möglich, eine Version durch Datumsangabe wiederherzustellen. Findet das System keine Version , die diesem Datum entspricht, wird die Version wiederhergestellt, die diesem am nächsten kommt.

Beim Wiederherstellen einer Version eines Verzeichnisobjektes (*Folder*) werden alle Resourcen innerhalb des Folders auf dieselbe Versionsnummer wiederhergestellt.

#### 3.1.12.2.3 Kreieren einer neuen Resource mit dem Inhalt einer Version

Mit dieser Funktion kann man aus einer Version des Dokuments eine neue Resource erzeugen.

#### 3.1.12.2.4 Eine Publikation älteren Datums wiederherstellen

Da die Ausgabedateien der Publikation und deren Publikationsdatum in einer Tabelle gespeichert sind, kann man mit einer Abfrage die Dateien, die zu dem gewünschten Zeitpunkt öffentlich zugänglich waren, wiederherstellen. Da keine Verzeichnisstruktur verwendet wird, muß sie hier auch nicht in der Tabelle gespeichert werden.

## 3.1.13 Die Behandlung von Inhalten

### 3.1.13.1 Textdokumente

Gesellschaften, Firmen und Einzelpersonen haben jeweils unterschiedliche Informationsbedürfnisse und brauchen deshalb auf sie zugeschnittene Informationsstrukturen. Durch Informationsaustausch mit anderen können sie aber profitieren. Zum Beispiel hat ein Medizinstudent, der Krebsforschung in Indien betreibt, Interesse daran, die neuesten Forschungsergebnisse aus der ganzen Welt abzurufen.

Um einen solchen Informationsaustausch auf dem elektronischen Wege zu ermöglichen, müssen die Dokumentenformate untereinander kompatibel sein. Es existieren sehr viele verschiedene Plattformen, auf denen verschiedene Betriebsysteme laufen: Unix, Windows, Macintosh. Es muß also eine Lösung gefunden werden, um den Austausch zwischen diesen Systemen zu ermöglichen.

#### 3.1.13.1.1 Die Geschichte des Markup

Früher wurden Dokumente mit Steuerzeichen versehen, die ein bestimmtes Programm oder Hardware abarbeitete, um das Dokument gemäß der Steuerzeichen darzustellen. Der Nachteil dieser Methode ist, daß man diese Dokumente nur mit dem spezifischen Programm, für das die Dokumente geschrieben sind, auch wieder lesen und bearbeiten kann. Entwickler dieser Systeme haben damals nicht an den Austausch der Dokumente zwischen verschiedenen Systemen gedacht, da es keine heterogenen Netzwerke gab und der Austausch nicht notwendig war. Außerdem wollten die Firmen die Benutzer, die Daten austauschen wollten, von der Software ihrer Firma abhängig machen (Produktbindung).

Da die Benutzer nach einer Weile den Wunsch entwickelten, zwischen verschiedenen Programmen Daten auszutauschen, entstanden unzählige Konvertierungsprogramme, die das eine proprietäre Format in das andere konvertierten. Diese Programme waren aber fehleranfällig, und das Ergebnis der Konvertierung war -gelinde gesagt- nie hundertprozentig zufriedenstellend.

Der nächste Entwicklungsschritt war, die Dokumente mit Steuerzeichen zu versehen, die im Gegensatz zu den früheren Lösungen keine Aussführungsanweisungen enthielten, sondern die gestalterischen Eigenschaften der Textabschnitte beschrieben. Zum Beispiel wurde ein Textabschnitt, der als *kursiv* dargestellt werden sollte, durch zwei Steuerzeichen definiert, die diesen Text umschlossen. Diese Zeichen beinhalteten keine Befehle für das Programm, wie es den Text kursiv darstellen sollte, sondern definierten lediglich den Text als kursiv. Dem Programm wurde überlassen, wie es den Text kursiv darstellt. Da diese Steuerzeichen keine Steueranweisungen mehr beinhalteten, wurden sie Tags (Markierungen) genannt. Der Prozeß der Markierung der Textabschnitte mit den Tags heißt Markup (Markierung). Die Textabschnitte, die durch ein Tag-Paar markiert sind, werden (Struktur-) Elemente des Dokuments genannt.

Hypertext Markup Language (HTML) verwendet, wie man aus dem Namen entnehmen kann, die soeben beschriebene Methode zur Markierung ihrer Dokumente. Damit können Browser, die das

Dokument darstellen sollen, dieses interpretieren und die Elemente des Dokuments in eine rechner- und softwarespezifische Darstellung umsetzen.
Der Nachteil von HTML ist, daß der Markup nicht nur die Struktur eines Dokuments darstellt, sondern eine Mischung aus Struktur und Darstellung. Zum Beispiel kann man mit einem Tag eine Tabelle definieren (Strukturelement) und mit einem anderen eine Schrift fett markieren (Darstellung). Dadurch sind die logischen Elemente, wie z.B. Überschriften, Fußnoten usw., nicht immer durch den Markup als solche markiert, sondern nur als fett oder kursiv gekennzeichnet. Warum dies von Nachteil ist, wird im folgenden Absatz erklärt.

#### 3.1.13.1.2 Struktur, Inhalt und Darstellung eines Dokuments

Ein Dokument besteht aus drei logischen Einheiten: Struktur, Inhalt und Darstellung.

##### *3.1.13.1.2.1 Was ist die Struktur eines Dokuments ?*

Die Struktur eines Dokuments definiert seine logische Organisation. Sie definiert die Zusammenhänge der Strukturelemente untereinander (das Layout der Dokumente). Die Dokumentenstruktur für eine Hausmitteilung kann zum Beispiel aus folgenden Elementen bestehen: Einem Titel, einem Adressfeld, einem Absenderfeld, einem Datumsfeld, der Anrede sowie der eigentlichen Nachricht. Diese Elemente haben Beziehungen zueinander: Der Titel steht nach dem Adressfeld; ein Absenderfeld besteht aus einem Namensfeld, gefolgt von einem Straßenfeld, gefolgt von einer Postleitzahl. Diese Beziehungen sind die Struktur des Dokuments.

##### *3.1.13.1.2.2 Was ist der Inhalt eines Dokuments ?*

Der Inhalt eines Dokuments ist die reine Information, die durch dieses Dokument vermittelt wird. Die Texte und Bilder, die z.B. die Konstruktionsschritte eines Saxophons beschreiben, sind der Inhalt des Dokuments.

##### *3.1.13.1.2.3 Was ist die Darstellung eines Dokuments ?*

Die Darstellung eines Dokuments ist die visuelle Unterscheidung der Strukturelemente. Ein Strukturelement „Überschrift“ ist für den Leser des Dokuments nur als eine Überschrift erkennbar, wenn sie z.B. mit einer großen Schrift und als fett dargestellt wird.

#### 3.1.13.1.3 Gründe für die Trennung von Struktur, Inhalt und Darstellung.

In jeder Art der Kommunikation gibt es zwei Ebenen der Informationsvermittlung:

1. Das, was wir als Inhalt bezeichnen.
1. Eine subtilere Information, *über* diesen Inhalt.

Letztere Information, wie z.B. fettgedruckter Text oder das Schreien während einer Unterhaltung, ist die Darstellung der Struktur. Der Mensch macht beim Lesen eines Dokuments Annahmen über die Struktur, die auf der Darstellung und dem Kontext basieren. Wird ein Textabschnitt fett dargestellt, kann der Leser mittels dieses optischen Hinweises und des Kontextes auf die Funktion dieses Abschnittes schließen. Der Abschnitt kann -je nach Kontext- eine Überschrift oder eine Betonung im Text darstellen. Soll aber ein Rechner diese Annahmen vornehmen, bedarf es der künstlichen Intelligenz.

In einem System, das zwischen Struktur und Darstellung unterscheidet, kann man eigene Strukturelemente definieren, um die Struktur des Dokuments wiederzugeben. Zum Beispiel kann man einen Tag „<Artikelüberschrift>“ definieren, der genau das beschreibt, was er ist, nämlich eine

Artikelüberschrift. Will man die Artikelüberschrift in einem System definieren, das nicht zwischen Struktur und Darstellung unterscheidet, dann kann man den Inhalt der Artikelüberschrift lediglich als fettgedruckt markieren.

Wenn man bei einem Dokument eine bestimmte Funktion ausführen will, die z.B. nur Artikelüberschriften betrifft, so scheitern Systeme, die nicht zwischen Darstellung und Struktur unterscheiden: Das System kann nicht zwischen Artikelüberschriften und den anderen fettgedruckten Strukturelementen differenzieren.

Damit sind die Vorteile der Trennung klar: In unterscheidenden Systemen kann man -da die Artikelüberschriften eindeutig gekennzeichnet sind- diese heraussuchen und ändern.

In einem System, das zwischen Struktur und Darstellung unterscheidet, bringt es weitere Vorteile, wenn man auch den Inhalt von der Darstellung trennt. Aufgrund dieser Trennung muß zu jeder Klasse von Strukturelementen eine Darstellungsart definiert werden.

Das wird anhand eines Beispiels erläutert: Für die Klasse der Strukturelemente „Artikelüberschrift“ definiert man, daß sie fett und schwarz dargestellt wird. Dadurch werden **alle** Artikelüberschriften im System fett und schwarz dargestellt. Will man bei allen Artikelüberschriften die Farbe ändern, muß man lediglich die Definition der Darstellung einer Artikelüberschrift ändern. Die Artikelüberschriften werden danach bei Ansicht der Dokumente in der neuen Farbe erscheinen.

#### 3.1.13.1.4 Ein Plädoyer für die Benutzung von XML[29]

Aus den oben genannten Gründen ist eine klare Trennung von Struktur, Inhalt und Darstellung notwendig. Diese höhere Abstraktionsstufe erleichtert wesentlich die Bearbeitung der Dokumente. Mit der Norm ISO 8879 (SGML) [GOLD 95] steht eine Architektur zur Verfügung, mit der Dokumente für beliebige Medien aufbereitet werden können, ohne die Struktur der enthaltenen Daten zu verlieren. HTML ist eine Anwendung von SGML. SGML hat eine komplizierte Syntax und Funktionalitäten, die teilweise nur in technischen Dokumenten benötigt werden. An dieser Stelle setzt die Extensible Markup Language (XML) an.

XML ist eine Teilmenge von SGML, die von einer Arbeitsgruppe des W3C (des WWW-Normengremiums) im November 1996 vorgestellt wurde. Ohne daß die Idee des strukturierten Markups von SGML aufgegeben wurde, wurden alle selten verwendeten Eigenschaften entfernt. XML bleibt aufwärtskompatibel zu SGML. Das von SGML nach XML übernommene Kernkonzept des strukturierten Markup ist einfach und leistungsfähig: Jedes Dokument wird in die Teile Inhalt, Struktur und Layout getrennt. Jedes XML-Dokument besteht aus den Teilen Definition (DTD) und Instanz.

##### *3.1.13.1.4.1 Document Type Definition (DTD)*

Eine DTD ist eine formale Grammatik, die eine bestimmte XML-Sprache definiert. In ihr werden die Namen der in Dokumenteninstanzen erlaubten Strukturelemente (Tags) und ihr Inhaltsmodell, d.h. ihre mögliche Schachtelung definiert. Sie macht keine Aussage über die Bedeutung der Tags.

[29] Dieses Kapitel lehnt sich an der Darstellung von Ingo Macherius in „IX Magazin für professionelle Informationstechnik“ Ausgabe Juni 1997, Verlag Hans Heise.

Man kann die DTD als einen Konstruktionsplan für korrekt strukturierte Dokumente betrachten. Sie definiert z.B., daß es ein Adresselement gibt und daß dieses Adresselement ein Namenselement beinhalten muß, gefolgt von einem Postleitzahlelement.

DTDs definieren einen besonderen Tag, den Dokument-Typ. Er trägt denselben Namen wie die definierte Sprache selbst und dient als Startsymbol der formalen Grammatik, die alle gültigen Dokumenteninstanzen dieses Typs erzeugen kann.

*<!DOCTYPE geo SYSTEM „http://www.geo.de/cocoon/dtds/artikel.dtd“>*
*<!ELEMENT geo (...)>*

Es muß ein <!ELEMENT> geben, das denselben Namen wie der <!DOCTYPE> hat und es muß dem Wurzelknoten in der Baumdarstellung des Dokumententyps entsprechen. Die Terminale der Grammatik sind die Namen aller in der DTD vereinbarten Elemente und der reservierte Name #PCDATA (parseable character data- gültige Zeichen eines Alphabets).
Ein Inhaltsmodell ist die Verknüpfung der Terminale mit den Operatoren aus der unteren Tabelle. Die reservierten Namen EMPTY und ANY kennzeichnen ein leeres bzw. ein beliebiges Inhaltsmodell.

| Zeichen | Bedeutung |
|---|---|
| , | Notwendige Elemente in fester Reihenfolge |
\| | 1 aus n -Auswahl |
| ? | Optionale Elemente |
| + | Reflexive Hülle (1...n Wiederholungen) |
| * | Reflexiv-Transitive Hülle (0...n Wiederholungen) |
| () | Bildung von Modellgruppen durch Klammerung |

Für jeden Tag können erlaubte Attribute vereinbart werden, mit denen sich die Funktion des Tags in der Dokumenteninstanz weiter spezialisieren läßt. Jedem Attribut ist ein Typ zugeordnet, wobei die möglichen Typen gegenüber SGML wiederum etwas eingeschränkt sind. Mit den Attributtypen ID und IDREF ist bereits ein einfacher Hyperlink-Mechanismus innerhalb von XML-Dokumenten vorhanden. Attribute können entweder optional, obligatorisch oder fest sein.

Die DTD wird von Spezialisten für bestimmte Publikationen erstellt. Sie spielt für XML eine vergleichbare Rolle wie ein Relationenschema im Datenbankbereich, ist aber ungleich flexibler. Eine DTD ist Teil jedes gültigen Dokuments, auch wenn im Normalfall viele Dokumenteninstanzen dieselbe Definition verwenden werden.

Die in der DTD formalisierte Sprache sollte sich an der Struktur der im Dokument enthaltenen Daten orientieren. Dokumente, die genug gemeinsame Eigenschaften haben, benutzen eine DTD. Zum Beispiel würde ein Dokument vom Typ „Artikel“ für alle Instanzen die gleiche DTD benutzen, wenn die Gliederung der Artikel ähnlich ist.

Jedes Dokument, dessen Inhalt mit den Tags einer bestimmten DTD ausgezeichnet ist, wird als deren Instanz bezeichnet. Es wird zwischen wohlgeformten (well formed) und gültigen (valid) Dokumenteninstanzen unterschieden. Erstere enthalten Tags, die den grundlegenden Regeln von

XML entsprechen, letzere zusätzlich die DTD, zu der sie gültig sind. Gültige Dokumente dürfen keines der Inhaltsmodelle ihrer DTD verletzen. Damit nicht in jedem gültigen Dokument die DTD angegeben werden muß, kann sie auch von einer externen Stelle eingebunden werden. Zum Beispiel durch

*<!DOCTYPE GEO SYSTEM "http://www.geo.de/dtds/artikel.dtd">*

am Beginn des Dokuments. Aus einem wohlgeformten Dokument kann so ein gültiges werden, sofern es die Regeln der DTD erfüllt.

Die Syntax von Tags in wohlgeformten Instanzen unterscheidet sich ein wenig von SGML und damit von HTML. Es ist zwingend vorgeschrieben, jeden geöffneten Tag explizit zu schließen. Tags ohne Inhalt wie <IMG> in HTML erhalten eine neue Syntax. Statt <IMG> muß in XML <IMG/> geschrieben werden. Diese beiden Vereinbarungen ermöglichen es, wohlgeformte XML-Dokumente zu parsen, ohne die DTD zu kennen. Im Gegensatz zu SGML kann für einen solchen Parser sogar eine lex/yacc Grammatik angegeben werden. Gültige Dokumente erfordern zusätzlich die Überprüfung der Inhaltsmodelle bezüglich der DTD.

In der DTD vereinbarte Tags werden sozusagen als benannte Klammern verwendet, um den Inhalt in Elemente zu gliedern. Ein Starttag (<TAG>) kennzeichnet den Beginn und ein Endtag </TAG> das Ende eines Elements. Durch die Schachtelung von Elementen ergibt sich eine Baumstruktur . Dessen innere Knoten drücken die Struktur aus, die äußeren enthalten den eigentlichen Inhalt und Elemente mit leerem Inhaltsmodell. Dieser Baum ist die eigentliche Arbeitsgrundlage von XML-Software. XML-Dateien stellen eigentlich nur eine durch Traversierung des Baums („depth first search“) gebildete Form dar.

#### 3.1.13.1.5 Die Darstellung eines Dokuments mit der Darstellungsbeschreibunssprache DSSSL

XML befaßt sich nicht mit der Darstellung eines Dokuments. Bei der Ansicht eines XML-Dokuments überprüft der Parser, ob das Dokument wohlgeformt ist. Danach wird eine Datei zur Beschreibung der Darstellung eingelesen. Diese Datei beschreibt zum Beispiel, daß ein Element „Artikelüberschrift“ in der Schriftart „Arial“, Größe 12pkt. in schwarz dargestellt werden soll. Durch die Beschreibungen in dieser Datei kann der Browser das Dokument darstellen. Im Falle von HTML ist die DTD die von den Benutzern und Inhaltsanbietern unveränderbare HTML-DTD, und die Darstellung ist (noch) dem Browser überlassen. Die Darstellung des Dokuments kann-je nach System- verschiedene Formen annehmen.

Für die Definition der Darstellung muß eine Definitionssprache gewählt werden. Damit diese Sprache von allen Browsern interpretiert werden kann, ist es sinnvoll, einen Standard zu verwenden, der von allen Browsern verstanden wird.

DSSSL (Document Style Semantics and Specification Language) ist ein junger Standard (ISO 10179) für die Transformation und Darstellungsformatierung auf eine plattform- und herstellerunabhängige Art.

Syntax und Kern von DSSSL stammen aus der IEEE-Norm Scheme, einem Lisp-Dialekt.Entsprechend ist die Sprache turingvollständig. Grundlage der Formatierung in DSSSL sind Flow Object (FO) genannte Konstrukte, die für verschiedene Grundkonzepte im Dokumentenlayout stehen, etwa Absätze, Linien, Tabellen Seiten etc.

Jedes FO kann durch Zuweisung von Charakteristika weiter spezifiziert werden. Eine DSSSL-Maschine baut aus einer XML-Instanz und einem Style Sheet (eine in DSSSL definierte Darstellungsdatei) zunächst einen Baum aus FOs auf, den sogenannten Flow Object Tree.

Die FOT-Darstellung ist eine vom Ausgabemedium unabhängige Sicht des Layouts einer Instanz. Mit verschiedenen Treibern (Backends) wird der FOT dann für ein konkretes Medium endgültig formatiert. Das vollständige Formatierungsmodell FOT ist zu leistungsfähig, um es hier vorzustellen. Im Vergleich mit bekannter DTP-Software oder TeX ergeben sich aber eher Verbesserungen als Einschränkungen.

Der zweite für XML wichtige Teil von DSSSL ist die Standard Document Query Language (SDQL). Sie erlaubt Anfragen und Navigation in der Baumdarstellung des Dokuments, in der DSSSL-Sprache Grove genannt. So sind die weiter oben beschriebenen XLinks einfach als SDQL-Anfragen zu formulieren. Häufig werden in Style Sheets durch SDQL-Anfragen auch die Teile festgelegt, auf die ein bestimmtes Layout angewendet werden soll.

Zusätzlich zum SDQL gibt es u.a. die *Transformation Language* in DSSSL: Sie beschreibt die Transformation eines Dokuments von einer DTD zu einem anderen. Damit kann man z.B. ein Dokument, für das man selbst eine DTD definiert hat, in ein HTML-Dokument (ein Dokument, das der HTML-DTD gehorcht) umwandeln.

#### 3.1.13.1.6 Die Verwendung von XML und DSSSL im Produktionssystem

In Zukunft werden WWW-Browser, XML-Dokumente lesen können wenn sie mit einem Stylesheet versehen sind. Da die Browser, die dieses unterstützen, noch nicht von allen Benutzern verwendet werden und die Dokumente auch für andere elektronische Medien aufbereitet werden müssen, müssen die Dokumente von XML in die gewünschten Formate konvertiert werden.

##### *3.1.13.1.6.1 Möglichkeiten der Konvertierung von XML nach HTML*

Es gibt zwei Möglichkeiten, um die Dokumente nach HTML zu konvertieren: Man kann die Dokumente in dem Moment, wo sie vom Benutzer angefordert werden, dynamisch („on the fly“) konvertieren. Vorteile dieser Methode sind, daß man für verschiedene HTML-Browser, die bekanntlich verschiedene HTML-Dialekte verwenden, den entsprechenden HTML-Code aus den Quelldaten erzeugen kann. Außerdem ergeben sich Vorteile bei der Suche in der Struktur der Dokumente (siehe 3.1.14). Nachteil dieser Methode ist, daß sie sehr viel Rechenleistung verbraucht, was sich mit steigender Anzahl der Leser potenziert.([COLB 96] Seite 351)

Man kann auch die Dokumente vorher schon in HTML übersetzen und für den Leser bereithalten („batch conversion“). Damit entfallen die hohen Rechenkosten für jedes angeforderte Dokument. Allerdings ist man dann gezwungen, zwei Kopien der Daten zu verwalten, denn die XML-

Resourcen und die HTML-Dateien müssen getrennt gespeichert werden. Weiterhin müssen alle XML-Dokumente neu nach HTML übersetzt werden, wenn eine Konversionsdatei geändert wird. Ein dritter Nachteil ist, daß man nicht mehr unterschiedliche HTML-Derivate an verschiedene Clients verschicken kann.

Da der Nachteil der hohen Rechenkosten bei der dynamischen Methode ziemlich groß ist, wird in diesem System die Methode der „batch Conversion“ angewandt.

Das Produktionssystem verwendet XML für die Textdokumente. Die Konvertierung der Dokumente findet durch eine DSSSLMaschine statt: die Transformationen werden in der von DSSSL definierten *Transformation Language* geschrieben. Es findet für die Benutzung der Dokumente durch einen WWW-Server eine Umwandlung in HTML statt. Es existieren bereits funktional vollständige DSSSLMaschinen, die in Java programmiert sind. Diese können in das System integriert werden, da sie die gleiche Programmiersprache verwenden.

#### *3.1.13.1.6.2 Multiple Ausgangsformate*

Durch die Verwendung von XML ergeben sich noch weitere Vorteile. Man kann aus den gleichen Ausgangsdaten verschiedene Ausgabeformate erzeugen. Die DSSSL-Maschine bekommt dafür für jedes Format eine andere Transformationsdatei und setzt Inhalt und Struktur der Dokumente in das entsprechende Format um. Dies ist nur möglich durch die Trennung von Inhalt und Struktur des Dokuments, da der Konverter nur so erkennen kann, wie die Struktur aufgebaut ist.

Beim Abspeichern eines XML-Dokuments wird die Doctype aus dem Dokument extrahiert. Die DSSSLMaschine verwendet dann die DTD zum Validieren des Dokumentes. Dem Benutzer wird gemeldet, wenn die Struktur des XML-Dokuments keine nach der DTD gültige Struktur darstellt. Ist das Dokument fehlerfrei, so wird sie von der DSSSLMaschine unter Benutzung der Transformationen in der Konversionsdatei in das gewünschte Format konvertiert. (siehe Abb. 3-31).

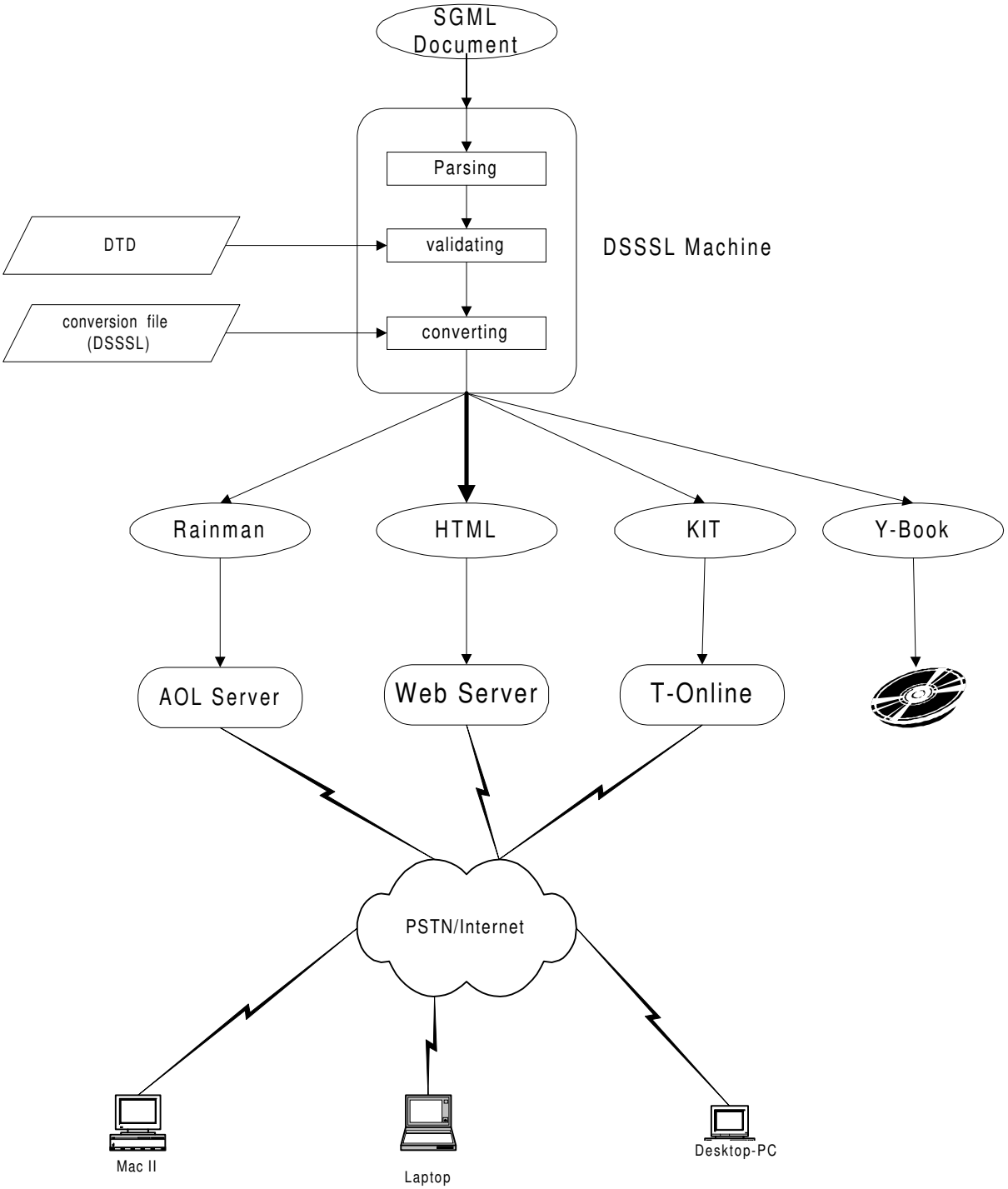



**Abbildung 3-31 Der Konvertierungsvorgang der XML-Dokument in die Zielformate**

### 3.1.13.2 Bilder

Bilder werden für die Benutzung in Online-Publikationen meistens komprimiert oder verkleinert. Da man bei der Wiederverwendung und -bearbeitung der Bilder keinen Qualitätsverlust hinnehmen will, müssen die Bilder vom System in einem verlustfreiem Quellformat gespeichert werden. Die

Bilder werden dann beim Erzeugen des Ausgabeformats in eine vom Benutzer angegebene Größe und Format konvertiert.

Für das verlustfreie Quellformat wird das PNG(Portable Network Graphics) [BOUT 97] Format verwendet. Die typischen Ausgangsformate sind GIF [DCSE 90] und JPEG [HAMI 92]. Hier kann mit dem *ResourceFilter*-Objekt definiert werden, welches Ausgabeformat erzeugt werden soll. Sollen von den gleichen Quelldaten zwei verschiedene Ausgabedateien erzeugt werden, dann werden zwei Symlinks definiert, jeweils mit verschiedenen *ResourceFilter*-Objekten.

#### 3.1.13.3 Audio

Für Audiodaten wird das Wave Format [WAVE 94] mit 44 khz. Abtastrate bei einer Auflösung von 16 Bit verwendet. Die Daten werden für die Zieldatei in bandbreitensparendere Formate umgewandelt (au, realaudio [REAL 97]).

#### 3.1.13.4 Andere Inhaltstypen

Andere Inhaltstypen werden vom System ohne interpretiert zu werden gespeichert und geladen. Hier können in Zukunft für deren Bearbeitung *ResourceHandler*, *ResourceFilter* und Editoren hinzugefügt werden. Wenn kein spezieller *ResourceHandler* oder Editor existiert, werden die generischen *ResourceHandler*, *ResourceFilter* und Editoren angewandt.

### 3.1.14 Die Suche in und nach multimedialen Inhalten

Das Finden von relevanten Informationen im Internet wird täglich problematischer, da die Anzahl der Resourcen im Internet rasant anwachsen. Gegenwärte Indizierungsmechanismen wurden schnell entwickelt, um die Nachfrage nach Recherchewerkzeugen zu stillen. Eine Volltextindizierung ist zwar nützlich, aber ein schwacher Ersatz für leistungsfähigere Arten der Beschreibung von Ressourcen.

Bei der Suche nach multimedialen Inhalten entstehen einige Probleme: Bei Volltextsuchen erhält man zusätzlich zur gesuchten Information mehrere Seiten mit irrelevantem Inhalt. Der Grund dafür liegt in der Art der Suche. Es wird im gesamten Text des Dokuments nach dem angegeben Wort gesucht. Der inhaltliche Kontext des Wortes wird nicht berücksichtigt.

Die Suche nach Binärdateien, die keinen Text enthalten (z.B. Bilder), gestaltet sich noch schwieriger, da ein Suchprogramm nicht einfach nach Eigenschaften des Bildes suchen kann. Die Suche beschränkt sich dann meistens auf den Dateinamen des Bildes. Derzeit werden Systeme entwickelt, bei denen man grobe Umrisse eines Bildes beschreiben kann [FORS 97]. Mit dieser Beschreibung kann dann das Programm durch statistische Analysen ähnliche Bilder heraussuchen (query by example). Diese Art der Suche braucht aber sehr viel Rechenleistung. Außerdem müßte man das Problem ebenfalls für Audio- und Videodateien lösen.

Um die Suche nach solchen Inhalten trotzdem zu ermöglichen, werden in diesem System zu jeder Resource Metadaten definiert. Sie beinhalten u.a. Informationen wie Typ, Name, Autor und eine Kurzbeschreibung der Resource. In diesen Metadaten kann dann gesucht werden.

Ein Gruppe von Bibliothekaren, Forschern aus dem Bereich digitaler Bibliotheken und Textformatierungsspezialisten traf sich im März 1995, um die Probleme der Suche nach Ressourcen anzusprechen. Die Aktivitäten dieser Gruppe nennt sich Dublin Core Metadata Workshop Series. [WEIB 97]

Diese Gruppe definierte u.a. eine gemeinsame Menge an Metadaten (Dublin Core Elements), die eine Ressource haben sollte. Die Ziele dieser Definition sind:

- Vereinfachung der Pflege und Instandhaltung der Daten
- Internationale Verwendbarkeit
- Allgemein verständliche Semantik
- Erweiterbarkeit
- Interoperation von verschiedenen Indizierungssystemen

#### 3.1.14.1 Verwendete Metadaten

Folgene Metadaten aus den Dublin Core Elements werden in diesem Produktionssystem als Metadaten gespeichert:

##### 3.1.14.1.1 TITLE

Der Titel, den der AUTHOR der Resource gegeben hat.

##### 3.1.14.1.2 AUTHOR

Derjenige, der das Urheberrecht des Dokuments besitzt.

##### 3.1.14.1.3 SUBJECT

Ein thematische Einordung der Resource. Die Einhaltung einer Namenskonvention und Verwendung von formalen Klassifikationsschemata wird empfolen, aber nicht vom System erzwungen.

##### 3.1.14.1.4 DESCRIPTION

Eine verbale Kurzbeschreibung des Dokuments (engl. Abstract).

##### 3.1.14.1.5 PUBLISHER

Herausgeber der Resource.

##### 3.1.14.1.6 DATE

Das Datum der Publikation der Resource in ihrer gegenwärtige Version. Hier wird das Format YYYY-MM-DD-HH:MM. verwendet, z.B. 1997-09-11-10:43. Dieses Format wird dem jeweiligen Datumsformat der verwendeten Datenbank angepaßt.

##### 3.1.14.1.7 TYPE

Hier weicht das Produktionssystem von den Dublin Core Elements ab. Das System verwendet TYPE, um anzugeben, welche Objektklasse durch die Daten in der Datenbank dargestellt wird. Sie enthält in Klartext den Namen der entsprechenden Klasse.

#### 3.1.14.1.8 FORMAT

Beschreibt das Datenformat des Inhalts der Resource. Eine *ImageResource* kann z.B. ein Bild im GIF-, JPG- oder PNG-Format beinhalten.

#### 3.1.14.1.9 Resource-ID

Beinhaltet eine eindeutige Objektindentifikationsnummer.

#### 3.1.14.1.10 RIGHTS

Ein urheberrechtlicher Hinweis (Copyright) für die Resource.
Bei der Suche nach einem Stichwort wird bei den Textdokumenten im Text und in der Beschreibung gesucht, wobei die Dokumente, die das gesuchte Wort bereits in der Beschreibung beinhalten eine höhere Relevanz zugewiesen bekommen. Bei Binärdateien hingegen werden nur die Metadaten durchsucht.

Der Benutzer kann nach jeglichen in den Metadaten gespeicherten Informationen suchen. Im Eingabefenster für die Suche erscheinen alle durchsuchbaren Metadaten. Nachdem der Benutzer das zu durchsuchende Metadatum angegeben hat, kann er durch die Eingabe eines oder mehreres Suchworte und durch die Formulierung eines regulären Ausdrucks [ABRA 97] die Suche starten.

### 3.1.15 Die Benutzeroberfläche

In diesem Kapitel werden Mechanismen für die Benutzerinteraktion und exemplarisch einige Darstellungsformen für die Benutzeroberfläche beschrieben. Die endgültige Erscheinungsform der Benutzeroberfläche wird hier aber offen gelassen, da verschiedene Benutzer unterschiedliche Wünsche haben können. Die Darstellungsformen werden den logischen Entwurf des Systems nicht beeinflussen.

#### 3.1.15.1 Der Browser

Um den Browser zu entwerfen, muß ein Modell der Interaktion zwischen dem Benutzer und der Objekte gewählt werden. [YOUR 96]

##### 3.1.15.1.1 Model View Controller (MVC)

Die Grundidee des MVC [YOUR 96] besteht in der Trennung der zum Problembereich gehörenden Objektklassen von den Objektklassen, die für die Anzeige von Informationen und die Interaktion mit dem Anwender (Interface-Klassen) bestimmt sind. Ein Interface-Objekt (View) benötigt direktes Wissen über die von ihm dargestellten Entity-Objekte (Model). Andererseits kann ein Entity-Objekt durch verschiedene Interface-Objekte dargestellt werden, über die es keine direkten Informationen besitzen sollte.

Eine Zustandsänderung in einem Entity-Objekt resultiert aus dem Empfang einer Nachricht. Um diese Änderung in allen verwandten Interface-Objekten widerzuspiegeln, verwendet das MVC einen speziellen Verbreitungsmechanismus.

Dieser beruht darauf, daß jedes Entity-Objekt eine Liste abhängiger Objekte besitzt, in der alle verwandten Interface-Objekte registriert sind. Ein Interface-Objekt registriert sich bei seiner Erstellung selbst bei seinen Entity-Objekten. Bei einer Zustandsänderung des Entity-Objektes kann jetzt eine Nachricht an alle „interessierten“ Interface-Objekte geschickt werden.

Der Vorteil dieses Modells ist, daß man das Interface jederzeit verändern kann, ohne die Entity-Objekte verändern zu müssen.

In diesem System wird ein ähnlicher Mechanismus wie beim MVC verwendet. Die Resource-Objekte werden im Client durch ResourceView-Objekte repräsentiert. Die Verzeichnisstruktur ist ein View des Dateisystems im Client. Wenn der Client eine Aktion auf eines der Interface-Objekte ausführt, wird diese Aktion zum Entity-Objekt des Interface-Objektes geschickt, das wiederum die tatsächliche Operation ausführt. Wenn zum Beispiel eine Resource gelöscht werden soll, klickt der Benutzer auf die View der Resource (die hier als Symbol dargestellt wird). Ein Popup-Menü erscheint, und der Benutzer wählt die Option „delete“. Diese Funktion wird vom Interface-Objekt zur Resource weitergeleitet, die dann den Löschvorgang ausführt. Der Server meldet dann über einen Event, daß die Resource gelöscht worden ist. Das Interface-Objekt löscht sich dann selbst.

#### 3.1.15.1.2 Elementare Bedienungsschritte

Um das System benutzen zu können, muß ein java-fähiger WWW-Browser wie Netscape oder Hotjava gestartet werden. Mit Hilfe des Browsers gelangt man zur Website des Produktionssystems. Hier kann man die zu bearbeitende Website aussuchen. (siehe Bild).

**Abbildung 3-32 Die Einstiegsseite zu den Produktionsystemen verschiedener Websites**

Danach erscheint ein Fenster mit zwei Frames[30] :Im rechten werden die Ergebnisse der bearbeiteten Seiten angezeigt. Im linken Frame läuft das Produktionssystem. Hier muß der Benutzer jetzt Name und Paßwort eingeben.

[30] Abschnitte einer Fensteraufteilung , die unabhängig voneinander geändert werden können.

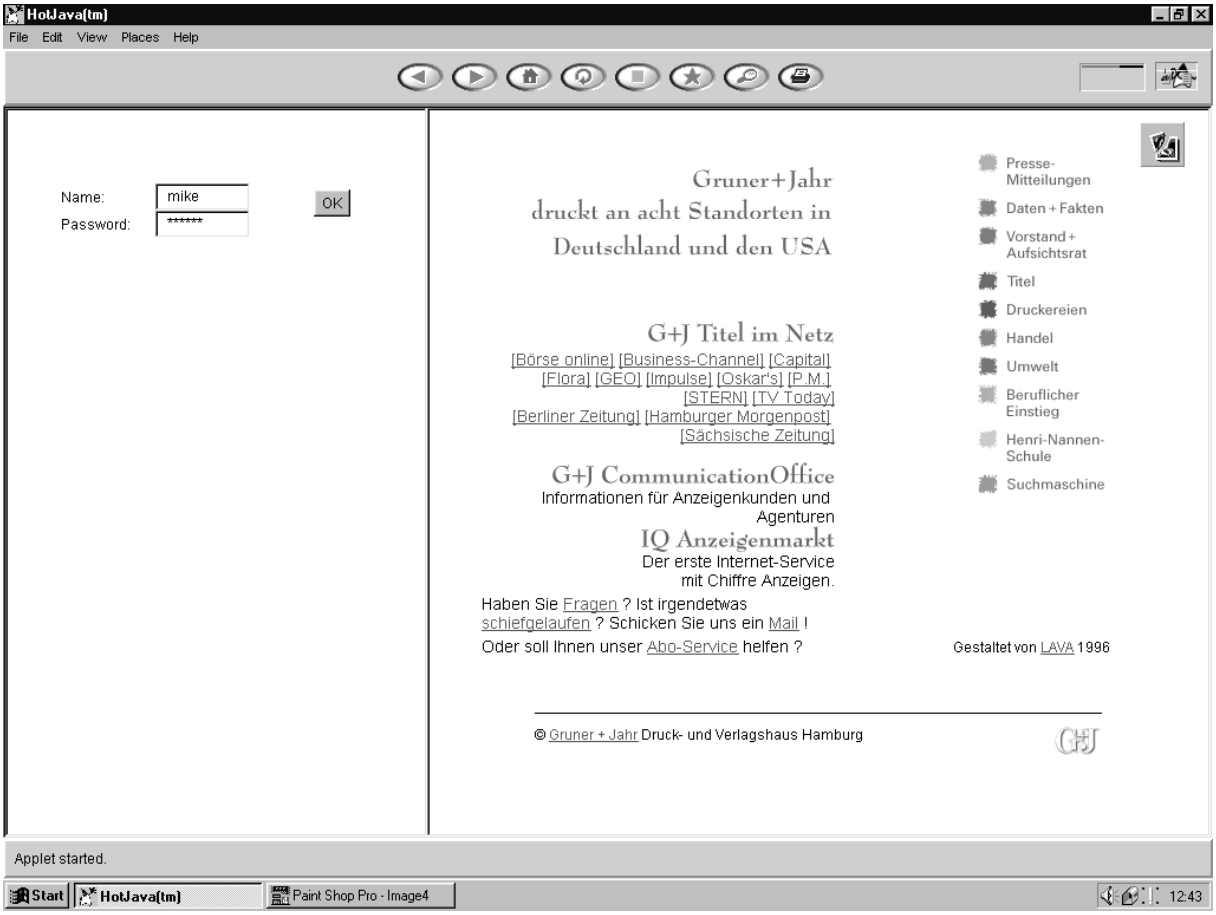



**Abbildung 3-33 Fenster mit Dialogfeld zum Anmelden**

Nach der korrekten Eingabe des Benutzernamens und Paßworts erscheint im linken Frame eine hierarchische Darstellung der Serverstruktur (1.). Hier kann man durch Doppelklicken auf die Verzeichnissymbole in der Verzeichnisstruktur des Servers navigieren.

Mit einem Doppelklick auf Resourcen -außer *Folder*(Verzeichnisse)- wird ein entsprechender Editor aufgerufen (siehe 3.1.15.2) Dabei kann der Benutzer nur in Teilen des Servers navigieren, für die er Zugangsrechte hat. Wenn man auf eine Resource (5.) mit der rechten Maustaste klickt, erhält man ein Popup-Menü (2.), das die für die angeklickte Resource verfügbaren Funktionen anbietet.

Diese Funktionen hängen davon ab, welche Rechte der Benutzer hat und in welchem Zustand sich die Resource befindet. Nicht zugängliche Funktionen tauchen im Popup-Menü nicht auf. Das Ergebnisfenster (4.) wird zur Anzeige der fertiggestellten Seite verwendet.

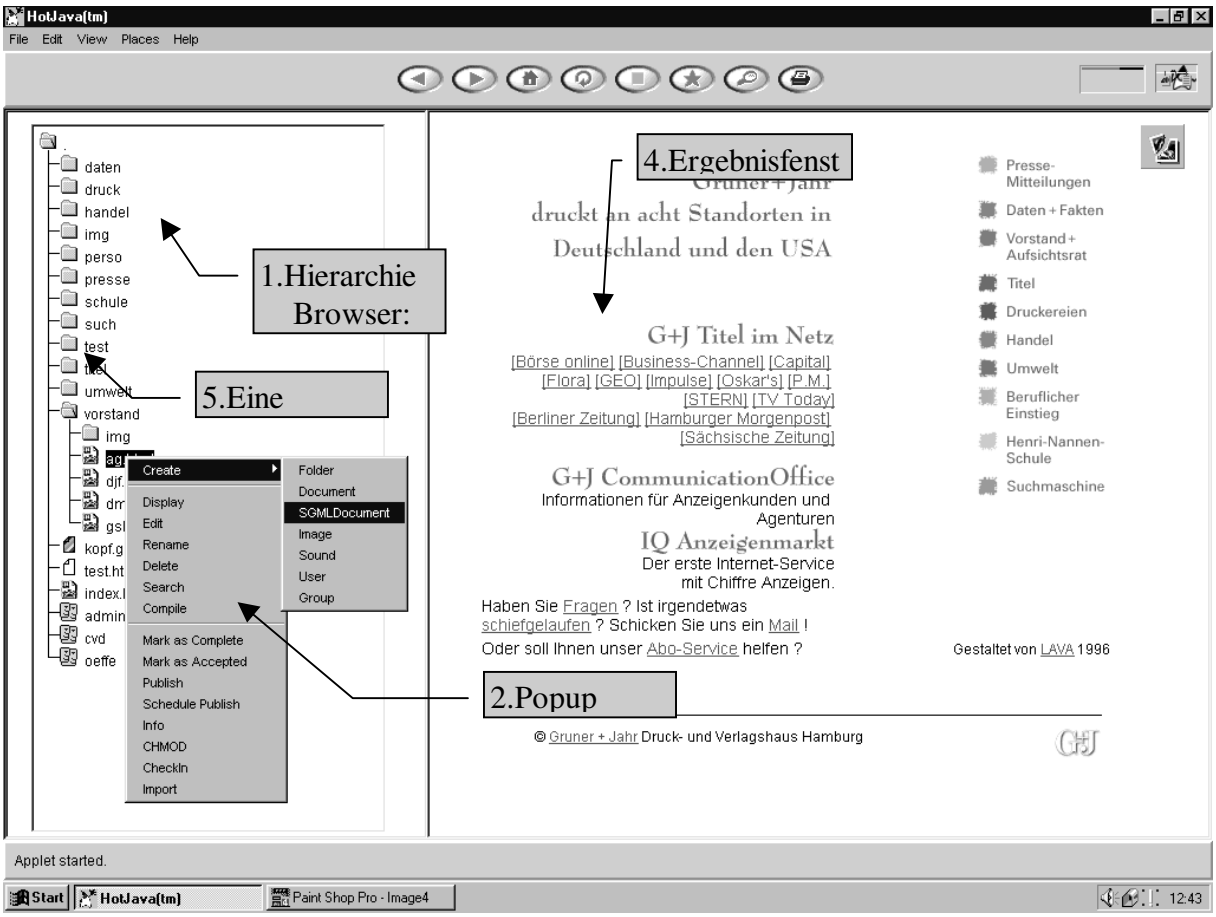


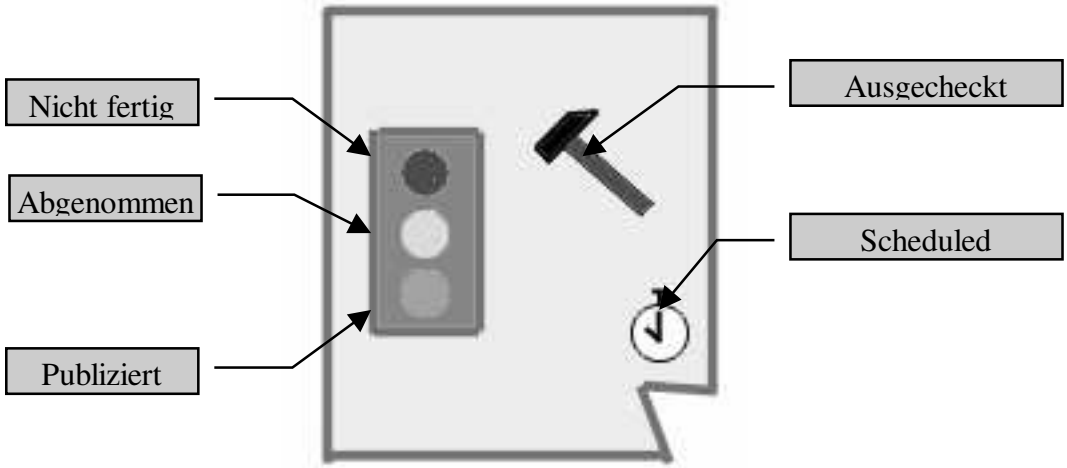

**Abbildung 3-34 Das Bedienungsfenster**

Das Kopieren und Verschieben von Resourcen geschieht durch Drag & Drop im Verzeichnisbaum. Beim Ausführen dieser Funktion wird der Benutzer gefragt, ob die Resource kopiert oder verschoben werden soll.

Dies gilt auch für *User*- und *Group*-Resourcen. Beim Kopieren eines Benutzers in eine Gruppe wird dieser Benutzer Mitglied der Gruppe. Beim Verschieben wird der Benutzer von einer Gruppe entfernt und der neuen Gruppe hinzugefügt.

Da der Verzeichnisbaum (1.) eine View der Resourcehierarchie darstellt, kann man zwischen verschiedenen Views hin- und herschalten. Zum Beispiel kann man statt der Verzeichnishierarchie die Beziehungen der Resourcen, die durch die Querverweise bedingt sind, darstellen.

Resourcen werden durch Symbole (Icons) dargestellt. Für jeden Resourcetyp gibt es jeweils ein anderes Symbol. Die für den Benutzer relevanten Zustände der Resource werden durch zusätzliche Symbole auf dem Resourcesymbol dargestellt. Über einen Notify-Mechanismus wird sichergestellt, daß die Zustände der Resource immer auf dem aktuellen Stand sind.




### 3.1.15.2 Die Editoren

Alle Editoren haben folgende gemeinsame Funktionen :

- Laden und Speichern auf das lokale System.
- Laden und Speichern im Produktionssystem.
- Cut & Paste von anderen Programmen.
- Einbetten von *CocoonComponent*-Objekten.
- Laden von Schablonen (Templates).
- Umschalten zwischen verschiedenen Komponenten.

Auf dem Server liegen *Template*-Dateien, die per Knopfdruck in den Editor geladen werden können. Sie werden gebraucht, um wiederkehrende Dokumente nicht immer manuell neu erzeugen müssen. Die Schablonenknöpfe werden dynamisch erzeugt. Je nachdem, wieviele Schablonen vorhanden sind, werden entsprechend viele Knöpfe mit der Bezeichnung der Schablonen erzeugt.

Das Einbetten von Komponenten ermöglicht es, zwar nur einen Editor pro Resourcetypen zu definieren, aber verschiedene Sichten (Views) auf die Daten anzubieten. (siehe dazu 3.1.15.2.1). Alle Komponenten sind Unterklassen von *CocoonComponent*. Dadurch können die Komponenten leichter eingebettet werden. *CocoonComponent* hält sich an den Java Beans Standard[31]. Dadurch können hier Komponenten von Drittanbietern eingebettet werden. Hier wird ein ähnlicher Mechanismus wie beim Model View Controller (MVC) angewandt. Dabei ist die Resource das Model, während der ResourceEditor den Controller darstellt. Die Unterklassen von *CocoonComponent* dienen dabei als Sichten(Views) des Inhalts. Sie müssen zwei Funktionen implementieren: *getData*() und *setData*(). Der Benutzer kann zwischen den verschiedenen verfügbaren Views umschalten. Die Knöpfe zum Umschalten zwischen den Views

[31] Java Beans ist ein von Sun Microsystems entwickelter Standard für das Einsetzen von Komponenten in Java. [CASS 97]

(*CocoonComponents*) werden dynamisch erzeugt, je nach Anzahl verfügbarer Komponenten für einen Resourcetyp. Wenn die Bearbeitung des Inhalts abgeschlossen ist, betätigt der Benutzer im Editor die Taste „save“. Der Editor holt den Inhalt von der Komponente mit *getData()* und speichert sie in die Resource. Danach weist es die Resource an, sich auf dem Server zu speichern.

#### 3.1.15.2.1 Der *SGMLEditor*

Der *SGMLEditor* hat folgende zusätzliche Funktionen:

- Überprüfen von Links. Per Knopfdruck können interne und externe Links auf Richtigkeit überprüft werden.
- Suchen und Ersetzen im Text

Der *SGMLEditor* besitzt folgende *CocoonComponents:*

#### *3.1.15.2.1.1 SourceComponent*

Diese *CocoonComponent* zeigt den Source des SGML-Dokuments an. Hier kann wie in einem üblichen Texteditor editiert werden.

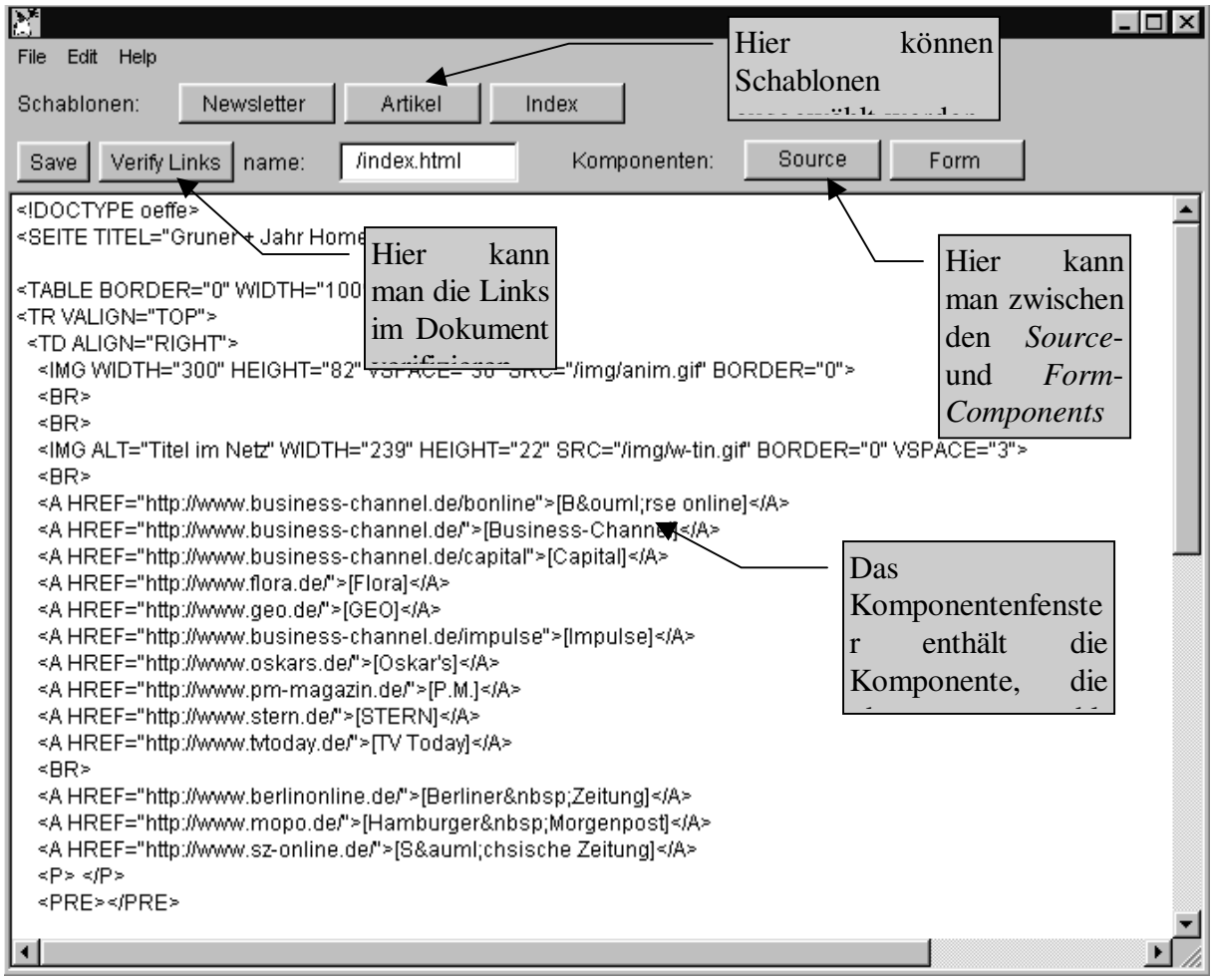



**Abbildung 3-35 Die SourceComponent**

*3.1.15.2.1.2 FormComponent*

Die Idee der *FormComponent* beruht darauf, daß die Layouts von Seiten einer Publikation von Graphikern erstellt werden. Am Layout darf ein Redakteur nichts ändern. Er ist nur für den Inhalt zuständig. Hier werden Formatvorlagen (Templates) definiert, dessen Inhalt von Redakteuren aufgefüllt werden können. Damit die Redakteure nur die Teile des Dokumentes bearbeiten können, die sich mit dem Inhalt befassen, werden in den Templates spezielle Tags definiert, die die Editierbarkeit eines Strukturelementes (Tag) festlegen. Der *FormComponent* parst das *SGMLTemplate* und erzeugt für jeden Tag und jedes Attribut, das mit „Edit-“ anfängt, ein entsprechendes Editorfeld. Diese Felder werden selbst auch als *Components* definiert.

Beispiel eines Templates:

```
<!DOCTYPE OEFFE>

<SEITE TITEL="Neues Produktionssystem">
<Context Rubrik="Home"/>

<A LABEL="HTML-Seite auf die das Bild zeigt"
EDIT-HREF="">

<IMG EDIT-SRC="" LABEL="Bild im Kopf der Seite" EDIT-WIDTH=""
EDIT-HEIGHT="" EDIT-BORDER=""/></A>

<H1 EDIT-TEXTFIELD="Artikelüberschrift"></H1>

<A EDIT-TEXTAREA="Artikel"></A>

</SEITE>
```

Der *FormEditor* stellt dann eine Seite zusammen, die nur aus den editierbaren Elementen des Dokuments besteht. Diese Felder haben jeweils eine Beschreibung und eine oder mehrere Eingabemöglichkeiten (siehe Abbildung 2-1). Die Eingabe eines Querverweises erfolgt über ein Auswahlfenster, das ähnlich dem Hierarchie-Browser aufgebaut ist. Der Benutzer navigiert zu der entsprechenden Resource, auf die er verweisen will, und das System trägt den Namen der Resource in das Feld ein.

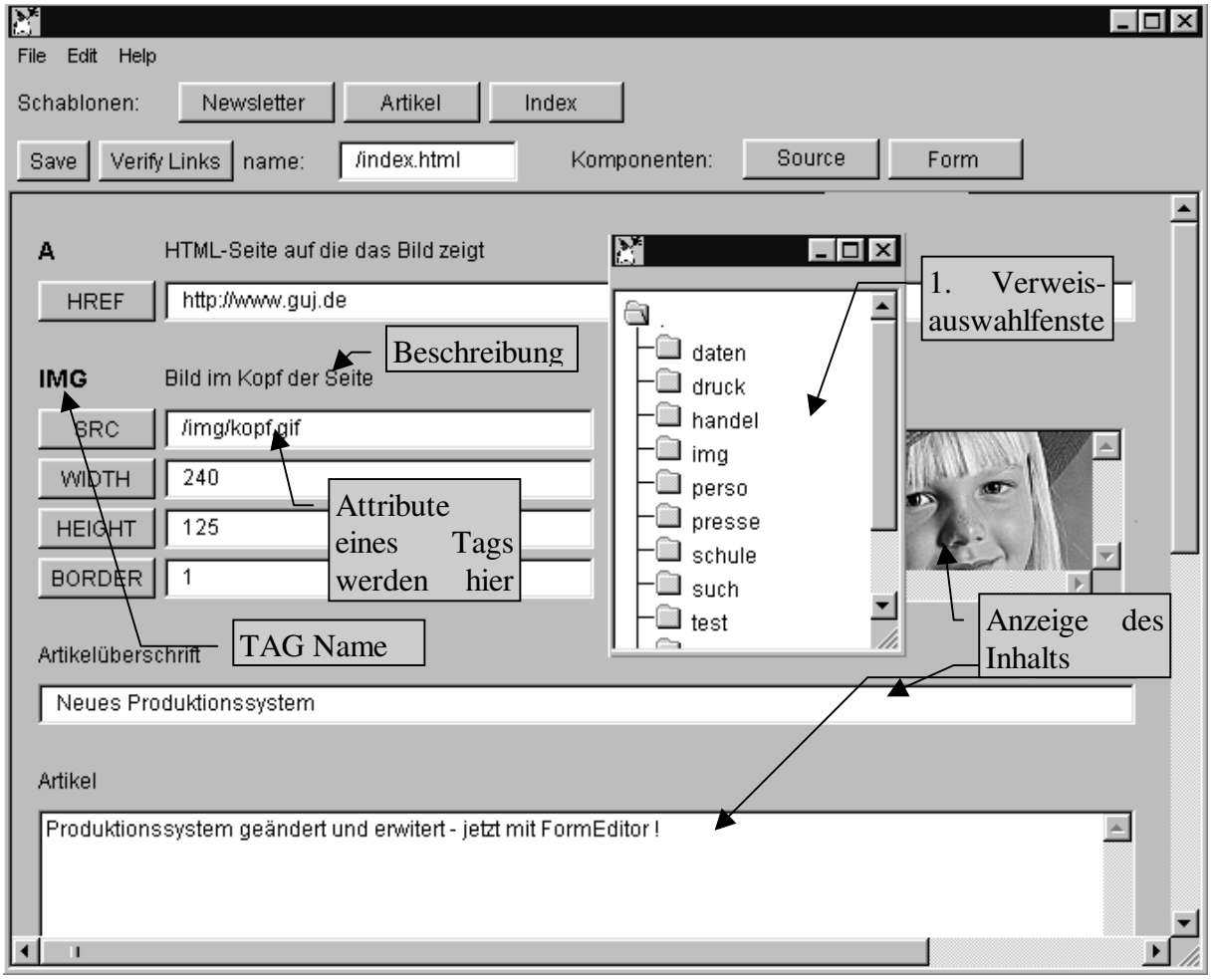



**Abbildung 3-36 Die Benutzerschnittstelle des FormComponents**

Die Resource löst beim Speichern die Pfadnamen in Resource-IDs auf.
Das Ergebnis, das in der Resource gespeichert wird, sieht folgendermaßen aus:

```
<!DOCTYPE OEFFE>

<SEITE TITEL="Neues Produktionssystem">
<Context Rubrik="Home"/>

<A EDIT-LABEL="HTML-Seite auf die das Bild zeigt"
EDIT-HREF="http://www.guj.de">

<IMG EDIT-SRC="5830.gif" EDIT-LABEL="Bild im Kopf der Seite"
EDIT-WIDTH="240" EDIT-HEIGHT="125" EDIT-BORDER="1"/></A>

<H1                    EDIT-TEXTFIELD="Artikelüberschrift">Neues
Produktionssystem</H1>
<A   EDIT-TEXTAREA="Artikel">Produktionssystem   geändert   und
&uuml;berarbeitet !<BR/>
Jetzt mit FormEditor !<BR/><BR/></A>
```

```
</SEITE>
```

Die Editfelder werden behalten, damit beim erneuten Editieren die Editierfelder existieren.

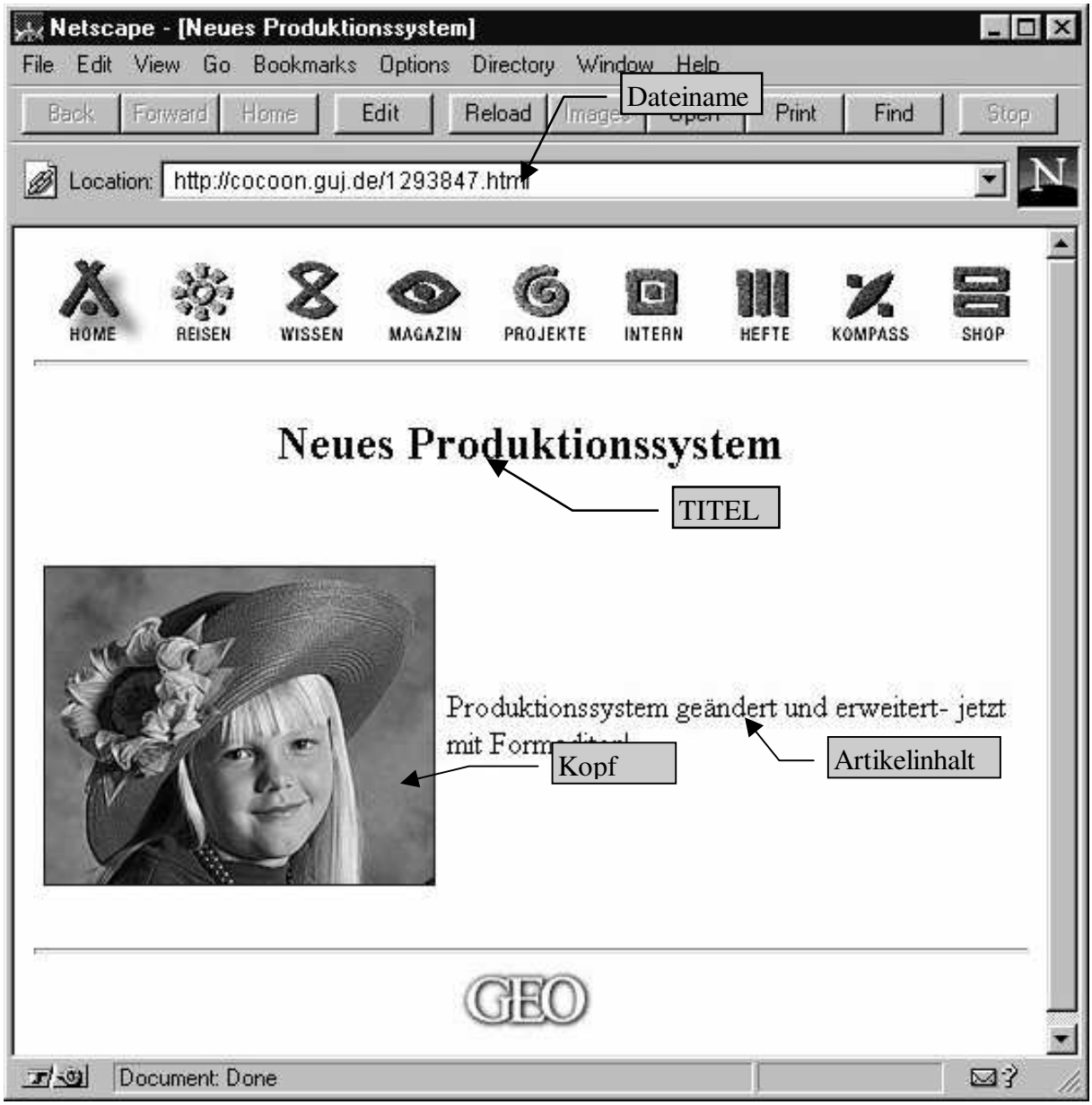
Das Ergebnis des Parsens der Datei wird in einem Parsebaum gespeichert. Die Tags des Dokuments werden als Tag-Objekte im Baum gespeichert. Die Anzeigefelder (Views) des Editors sind mit den Tag-Objekten verbunden. Beim Ändern der Daten eines Tags wird die Änderung direkt in das Tag-Objekt geschrieben. Wenn der *CocoonEditor* vom *FormComponent* die Daten zurückverlangt (mit der Funktion *getData*()), wird der Parsebaum traversiert und eine flache Struktur (Dokument) wieder erzeugt. Damit stehen die Ergebnisse der Änderungen an den richtigen Stellen, und die Daten können in der Resource gespeichert werden.

Nachdem die Resource gespeichert worden ist, wandelt sie die DSSSLMaschine in HTML um und ergänzt sie um Navigationsleisten und andere benötigte Tags.

Das Ergebnis sieht im Source dann so aus:

```
<HTML>

     <HEAD>
          <TITLE>Neues Produktionssystem</TITLE>
     </HEAD>

     <BODY BGCOLOR="ffffff">
          <Center>
               <IMG SRC="1584.gif">
               <BR><HR><BR>
               <H2>Neues Produktionssystem</H2>
               <TABLE>
               <TR>
                    <TD>
                         <A  HREF="3982.html"><IMG  SRC="5830.gif"
                         WIDTH="240" HEIGHT="125" BORDER="1"></A>
                    </TD>
                    <TD>
                         Produktionssystem      geändert      und
                    erweitert- jetzt mit Formeditor !
                    </TD>
               </TR>
               </TABLE>
               <BR><HR>
               <IMG SRC="8456.gif">

          </Center>

     <BODY>
</HTML>
```

Wenn das Ergebnis in einem HTML-Browser angezeigt wird, sieht es wie folgt aus:




**Ansicht des Dokuments im Netscape-Fenster**

Durch diesen Ansatz ist es möglich, die Änderung des Layouts durch den Redakteur zu verhindern. Gleichzeitig wird durch dieses Data-Hiding-Konzept der Redakteur nicht mit überflüssigen Optionen und Quellcode überfordert. Die Benutzung eines WYSIWYG-Editors wäre hier nicht von Vorteil, weil der Redakteur dann das Layout ändern könnte.

Es ist aber vorstellbar, für Websites die im Layout nicht so restriktiv sind wie ein Verlag, einen WYSIWYG-Editor zu implementieren, der als Grundlage der Darstellung des Inhalts die DSSSL-Definition verwendet. Dieses wird hier nicht weiter ausgeführt.

#### 3.1.15.2.2 Der ImageEditor

Der *Imageeditor* ist für einfache Änderungen an Bildern gedacht, wie zum Beispiel Größenänderung. Diese und ähnliche Funktionen werden durch Drittanbieter implementiert.
Für andere Funktionen werden kommerzielle Programme verwendet und das Ergebnis importiert. Hier können, wie bei den andere Editoren auch, weitere *CocoonComponents* definiert werden, die eine leistungsfähigere Editierung ermöglichen.

#### 3.1.15.2.3 Der *SoundEditor*

Der *Soundeditor* ist für die Bearbeitung von Klängen gedacht, wie zum Beispiel Längenänderung.

#### 3.1.15.2.4 Der *ImageMapEditor*

Der *ImageMapEditor* hat die Funktion, ImageMaps zu editieren. Er beinhaltet Funktionen zum Laden von Bildern und die graphische Erstellung von anklickbaren Bereichen. Nach Definition aller anklickbaren Bereiche wird der Code für einen *Image*map automatisch erzeugt.

#### 3.1.15.2.5 *User* Editor

Hier kann man Paßwort und Gruppenzugehörigkeiten des Benutzers ändern. Die Gruppenzugeörigkeit kann aber auch per Drag & Drop im Hierarchiefenster bewerkstelligt werden.

#### 3.1.15.2.6 *Group* Editor

Der GroupEditor kann zum Verändern der Gruppenzugehörigkeit von Benutzern verwendet werden, was auch im Hierarchiefenster per Drag & Drop möglich ist.

## *3.2 Technischer Entwurf*

Dieses Kapitel beschreibt einige technische Aspekte des Entwurfs sowie Techniken, die zur Realisierung des Systems verwendet wurden.

### 3.2.1 Java Remote Method Invocation (RMI)

RMI ist das Objektmodell für verteilte Objekte in der Programmiersprache Java. Es beinhaltet die Semantik des Java Objektmodells. Der Vorteil von RMI gegenüber anderen verteilten Systemen (z.B. CORBA) ist seine nahtlose Integration in das Java Objektmodell. Zum Beispiel unterstützt CORBA ein heterogene, mehrsprachige Umgebung und muß deswegen ein sprachneutrales Objektmodell besitzen. RMI hingegen unterstützt eine Java Umgebung und kann im Java-Objektmodell integriert werden. Dadurch können z.B. die Garbage-Collection Mechanismen von Java auch auf verteilte Objekte angewandt werden.

RMI kann als ein Drei-Schichten-Modell dargestellt werden. Jede Schicht bietet der darüber liegenden Schicht seine Dienste an.

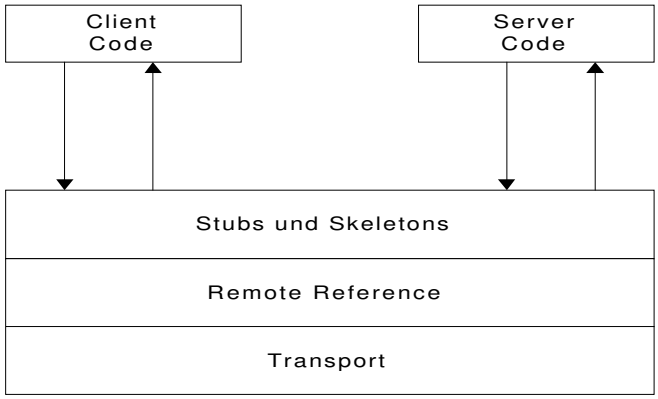



**Abbildung 3-37 Schichten des RMI Modells**

Wenn ein Client eine Remote-Methode aufruft, fließt der Aufruf durch die Schichten, um in der Transportschicht als Datenstrom weitergereicht zu werden. Auf der Serverseite durchlaufen die Daten die Schichten in umgekehrter Reihenfolge, wobei sie wieder rekonstruiert werden.

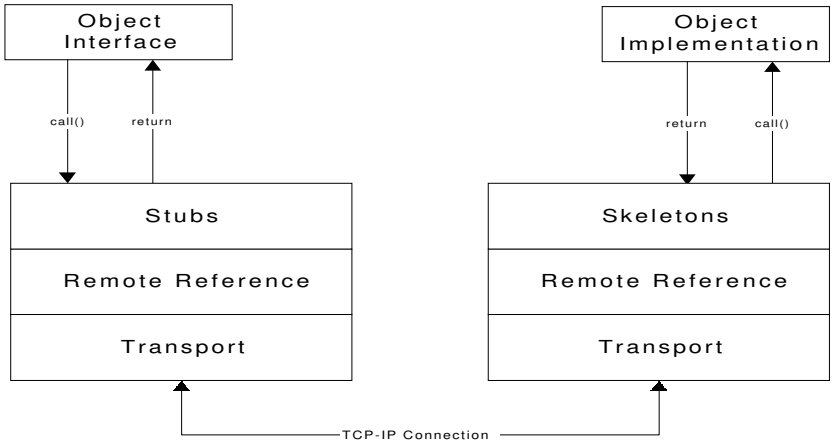



**Abbildung 3-38Kommunikation der entfernten Objekte**

Jede Schicht hat seine eigenen Aufgaben, wie nachfolgend beschrieben wird:

#### 3.2.1.1.1 Stubs und Skeletons Schicht

Stubs und Skeletons erfüllen komplementäre Aufgaben. Sie bilden die Schnittstelle zwischen den RMI-Diensten und dem Code, der die Funktionen aufruft und implementiert.
Sie sind verantwortlich für das Marshalling der Argumente und Unmarschalling der Rückgabewerte. Sie benutzen dazu den Object Serialization Service, der Objekte in Streamrepräsentationen umwandelt und umgekehrt. Die Serialisation erlaubt die transparente Bewegung von Objekten zwischen Prozessen. Die Remote Reference Schicht versorgt die Stubs und Skeletons mit den Streams. Stubs stellen die Schnittstelle zu den verteilt zugänglichen Funktionen eines Objekts dar. Skeletons stellen die Schnittstelle zu der Implementation eben genannter Funktionen dar.

#### 3.2.1.1.2 Remote Reference Schicht

Die Remote Reference Schicht bietet eine konsistente Schnittstelle für die Stubs und Skeletons an. Um dies zu erreichen, muß diese Schicht die Unterschiede in den Protokollimplementierungen der Server für die oberen Schichten maskieren.

#### 3.2.1.1.3 Die Transportschicht

Diese Schicht befaßt sich mit den Funktionen auf Verbindungsebene und stellt eine Verbindung zwischen den Java Virtual Machines her, die den Client und den Host Code ausführen.
Hier werden Object Identifier zur Bezeichnung der Objekte verwendet. Für die Bezeichnung der virtuellen Maschinen werden *endpoints* benutzt. Dieses Protokoll nennt sich *wire protocol,* worauf hier nicht näher eingegangen wird.

Der Aufruf einer Funktion eines entferntes Objekt hat die gleiche Syntax wie der Funktionsaufruf bei einem normalen Objekt.

**Das RMI-Modell unterscheidet sich vom Java Objektmodell in folgenden Punkten:**

- Clients von entfernten Objekten arbeiten mit *Remote-Interfaces,* nicht direkt mit den Implementationsklassen.
- Nicht verteilte Argumente und Ergebnisse eines Funktionsaufrufes werden als Kopie (*pass by copy*) weitergegeben, da Objektreferenzen nur innerhalb einer virtuellen Maschine sinnvoll sind.
- Ein entferntes Objekt wird als Referenz weitergegeben (*pass by reference*). Hier wird ein stub (verteiltes Interface) als Ersatz für das Objekt weitergegeben.

#### 3.2.1.1.4 Lokalisierung von verteilten Objekten

Um Methoden eines Objekts aufrufen zu können, benötigt man eine Objektreferenz. Für diesen Zweck hat das RMI System *Registries* vorgesehen. Objekte in einem Serverprozeß haben vom Entwickler vergebene Namen, die als Strings dargestellt sind. Registries fungieren als Name-Server, die die Zuordnung zwischen den Objekten auf einem Host und deren Namen verwalten.

Eine Registry kann alle Namensbindungen für einen Host verwalten oder es können mehrere Registries gleichzeitig tätig sein: eine für jeden Serverprozeß.

Damit ein Client eine Funktion auf einem verteilten Objekt ausführen kann, braucht er zuerst eine Referenz auf das Objekt. Der RMI Naming Service bietet Funktionen für lookup, bind, unbind und rebind an. Die Namen werden hier im URL-Format angegeben.

Durch Angabe des Namens mit lookup erhält man dann eine Referenz auf das Objekt. Ein URL für diesen Zweck kann folgendermaßen aussehen:

***„rmi://mein.server.de:7867/meinObjekt“***

Bei diesem System wird eine Registry für alle *CocoonServer* benutzt. Die *CocoonServer* sind mit ihrem Namen durch die Registry als entfernte Objekte erreichbar. Die Resourcen können durch ihre Referenz auf die *Connection*-Objekte die Funktionen des *Connection*-Objektes direkt ausführen.

Zur Speicherung der *ResourceFilter*-Objekte in der Datenbank wird der Object Serialization Service verwendet. Die *ResourceFilter* werden in ihrer Streamrepräsentation umgewandelt und in der Datenbank als BLOBs[32] gespeichert.

[32] Binary Large Objects.

### 3.2.2 Die Datenbankzugriffe

#### 3.2.2.1 Java Database Connectivity (JDBC)

JDBC [NORT 97] ist ein call-level SQL[33] API, das eine Schnittstelle zu einer großen Auswahl von relationalen Datenbanken bietet. Es ist konzeptionell der ODBC[34] API von Microsoft ähnlich.

JDBC bietet einen Kern an, auf dem höhere APIs und Tools entwickelt werden können.Die Hauptfunktion von JDBC ist die Ausführung von SQL-Abfragen und das Zurückliefern der Ergebnisse. Es definiert Klassen zur Darstellung von Datenbankverbindungen, SQL Statements, Result Sets und Metadata.

JDBC führt das Konzept eines *Prepared Statement* ein. Ein *Prepared Statement* ist ein präkompilierter SQL-Befehl. In diesem System wird bei der ersten Ausführung eines Befehls dieser als *Prepared Statement* auf der Datenbank erzeugt. Dieses Statement wird dann bei wiederholter Benutzung dieses Befehls wiederverwendet, so daß eine Performancesteigerung zu erwarten ist.

##### 3.2.2.1.1 Driver Manager

Der Driver Manager kontrolliert die Interaktion zwischen der Applikation und des benutzten Treibers. Er unterstützt mehrere Treiber, die mit verschiedenen DBMS verbunden sind.

##### 3.2.2.1.2 Treiber

Treiber sind die Schnittstellen zwischen der Applikation und der herstellerspezifischen DBMS. Sie bieten die nötige Transparenz, um die Applikation datenbankunabhängig zu programmieren.

[33] Structured Query Language
[34] Open Database Connectivity

## 4 Resümee

Die vorliegende Arbeit enthält die Entwicklung eines Systems, das die Produktion von Online-Medien unterstützt. Hauptsächlich werden WWW-Inhalte produziert; der Grund ist die wachsende Beliebtheit des WWW als Online-Medium.
Bei der Bearbeitung von Hypertext-Dokumenten entstehen Probleme durch Ungültigwerden von Querverweisen. Ein Beitrag zur Arbeitserleichterung liegt in der Verwaltung der Querverweise.

Ein zweites Problem entsteht, wenn bereits vorhandene Dokumente ein neues Layout bekommen sollen. Außerdem ist es schwierig, aus einem Dokument mehrere verschiedene Zielformate mit dem gleichen Inhalt zu erzeugen. Die Arbeit löst diese Probleme durch die Verwendung von XML als Quellformat.

Die Dokumente durchlaufen während der Produktion verschiedene Bearbeitungszustände. Je nach Zustand des Dokuments sind nur bestimmte Operationen sinnvoll. Daher läßt das System zustandsbedingt Operationen zu bzw. verbietet sie.

Vorteilhaft war der Einsatz der Objektmodellierungstechnik „Mainstream Objects“, da dadurch ein klares Objektmodell entstand und man anhand der Modelle den Entwurf auf funktionale Vollständigkeit überprüfen konnte. Dadurch konnten bereits in der Entwurfssphase solche Fehlentscheidungen vermieden werden, deren Auswirkungen sonst erst in der Implementierungsphase deutlich geworden wären. Darüber hinaus vermittelte es den Programmiern eine präzise und übersichtliche Darstellung des Entwurfs. Allerdings mußte der Autor diese Technik zunächst erläutern, da den Programmierern des Systems „Mainstream Objects“ nicht geläufig war.

Es hat sich im Laufe der Programmierarbeiten gezeigt, daß die Programmiersprache Java und deren Klassenbibliotheken sich sehr gut als Basis für die Entwicklung eignen. Zur Zeit der Programmierung standen allerdings die vom System verwendeten Kommunikationsmethoden (RMI und JDBC) noch in der Entwicklungsphase. Dadurch wurde die Entwicklung erschwert, denn man hatte mit Fehlern in der Implemenation dieser Kommunikationsklassen zu kämpfen. Außerdem waren damals nur dürftige Dokumentationen dieser Klassen erhältlich.

Es hat sich gezeigt, daß es sinnvoll ist, den Entwurf mit der prototypischen Realisierung zu koppeln, da dadurch die in der Entwicklungsphase nicht erkannten Probleme sich im Prototyp zeigten. Außerdem konnte der Prototyp dazu dienen herauszufinden, ob die Benutzer mit dem System umgehen können und ob die Funktionen und die Benutzeroberfläche ihren Wünschen entsprechen. Es wurde eng mit den Benutzern des Systems zusammengearbeitet, und deren Wünsche flossen in den Entwurf mit ein. Es fand ein evolutionärer Prozeß statt, bestehend aus den Phasen Entwurf, Erstellung eines Prototyps und Rücksprache mit den Benutzern.

Die Akzeptanz des Systems konnte noch nicht evaluiert werden, da es sich derzeit noch in der Einführungsphase befindet.

## 4.1 Stand der Implementation

Die prototypische Realisierung des Systems ist in weiten Teilen abgeschlossen, und das System wird inzwischen bei Gruner + Jahr Electronic Media Service für die Produktion mehrerer Websites verwendet.

Noch nicht implementiert sind die Funktionen zum Versionsmanagement sowie die Suchfunktionen. Zur Zeit wird nur für das WWW produziert, es ist aber vorgesehen, die Inhalte auch für AOL[35] zu verwenden. Durch die tatsächliche Benutzung des Systems sind neue Anforderungen an dieses entstanden, die sich aber ohne größere Veränderungen des Modells realisieren lassen.

Das System besteht zur Zeit aus 20.000 Zeilen Code, die in 125 Klassen verteilt sind. Hierin enthalten sind aber schon die Implementationen der Benutzeroberfläche und der Editoren für das System. Außerdem enthalten sind die Klassen zum Parsen und Konvertieren von XML-Dokumenten.

## 4.2 Ausblick

Bei objektorientierten Systemen, die eine relationale Datenbank verwenden, entsteht immer das Problem, daß die Objekte auf Tabellen in der Datenbank abgebildet werden müssen. Tabellen repräsentieren keine Klassen, da sie Vererbung nicht ermöglichen, nur Attribute speichern und nur Attributtypen aus einer vordefinierten Menge zulassen. „Ein Objekt in einer relationalen Datenbank zu speichern ist, als ob man einen Wagen vollständig auseinandermontiert, um ihn zu parken“ (C’T Magazin für Computer Technik, Ausgabe Mai 1997, Seite 294).

Die Abbildung der Datenobjekte in einer relationalen Datenbank hat sich als sehr mühselig erwiesen. Die meiste Implementationszeit wurde für diese Abbildung aufgewendet. Durch den Einsatz einer objektorientierten Datenbank könnte man den Code des Produktionssystems drastisch reduzieren und damit mögliche Fehlerquellen eliminieren. Dieses würde außerdem die Entwicklung um einiges beschleunigen.

Leider gab es zur Anfangszeit der Entwicklung keine objektorientierten Datenbanken, die sich nahtlos in die Programmiersprache Java integrieren ließen (Persistent Object Store für Java). Deswegen wurde der Einsatz einer solchen Datenbank zurückgestellt.

Weiterhin wäre es wünschenswert, die Dokumentenstruktur eines SGML-Dokumentes so in der Datenbank zu repräsentieren, daß nach Strukturelementen gesucht und mit ihnen Operationen ausgeführt werden können. Dies bedeutet, daß der Strukturbaum, der nach dem Parsen des Dokument erzeugt wird, in der Datenbank gespeichert wird, statt des flachen Dokuments. So könnte man etwa eine Suche starten, bei der man nach allen Artikelüberschriften sucht, die z.B. das Wort „Theater“ beinhalten. Es würde dann eine Ergebnisseite zurückgeliefert werden, auf der

[35] Ein kommerzieller Online -Dienst, der ein eigenes Netz aufgebaut hat. Er verwendet nicht das Internet zur Kommunikation. [AOLN 97]

sich alle Namen der Dokumente und der Inhalt der jeweiligen Strukturelemente „Kurzbeschreibung“ befinden.

Wenn man Datenbankoperationen auf die Struktur eines Dokuments ausführen kann, können globale Veränderungen der Dokumenteninhalte innerhalb der Datenbank stattfinden, ohne daß die Dokumente einzeln aus der Datenbank ausgelesen, nach dem zu verändernden Element geparst und in die Datenbank zurückgeschrieben werden müssen. Erst durch diese Art der Speicherung könnte man die Möglichkeiten von SGML voll ausschöpfen. Allerdings setzt das voraus, daß man eine objektorientierte Datenbank verwendet, denn relationale Datenbanken lassen nur Attributtypen aus einer vordefinierten Menge zu und erschweren dadurch die Abbildung sich ständig verändernder Dokumentenstrukturen in ein Relationenschema.

Da Microsoft und Netscape sich in den letzten Tagen über die Verwendung von XML in ihren Browsern geeinigt haben[36], könnte der Schritt der Konvertierung von XML nach HTML bald überflüssig werden. Man müßte dann nur noch die XML-Seiten ausliefern und eine Darstellungsdefinition (in diesem Fall CSS[37], eine Untermenge von DSSSL) mitliefern.

Leider ist es utopisch anzunehmen, daß die Browser der verschiedenen Hersteller in Zukunft tatsächlich die gleichen Verhaltensweisen bei der Darstellung eines Dokuments aufweisen werden und daß keine Balkanisierung[38] von HTML bzw. XML stattfindet. Daher ist es eher unwahrscheinlich, daß der Konversionsschritt völlig entfallen wird.

[36] Laut PC-WELT NEWS 4.9.97. (http://www.pcwelt.de/)
[37] Cascading Style Sheets. [HAKO 97]
[38] Die Erweiterung der Sprachen mit proprietären Eigenschaften.

# 5 ANHANG A: Die Klassen

Hier werden einige der Klassenschnittstellen dargestellt.

### 5.1.1.1 *ResourceManager*

#### *5.1.1.1.1.1 Erzeugen einer neuen Resource*

public Resource requestNewResource(*Folder Folder*, String name,
String description,
String type) throws AuthorizeException
Erzeugt eine neue Resource auf dem Server und liefert sie an den Client zurück.

#### *5.1.1.1.1.2 Löschen einer Resource*

public void deleteResource(Resource resource) throws AuthorizeException
Beim Ausführen dieser Funktion wird das Feld „deleted“ in der Tabelle „Resources“ gesetzt. Damit ist die Resource für den Benutzer nicht mehr sichtbar.

#### *5.1.1.1.1.3 Holen einer Resource*

public Resource getResource(Long Resource_ID)
Liefert die Resource mit der angegebenen Resource-ID zurück.

#### *5.1.1.1.1.4 Holen eines Verzeichnisses (*Folder*)*

public *Folder* get*Folder*Contents(Long Resource_ID) throws AuthorizeException
Liefert ein Folder mit den Resourcen, die es beinhaltet, zurück.

#### *5.1.1.1.1.5 Laden des Inhalts einer Resource*

public byte[] getResourceData(Resource r) throws AuthorizeException
Liefert die Daten der aktuellen Version einer Resource zurück.

*public byte[] getResourceData(Resource r, Long Version)*
Liefert die Daten der angegeben Version der Resource zurück.

#### *5.1.1.1.1.6 Speichern des Inhalts einer Resource*

public void putResource(Resource resource) throws AuthorizeException
Speichert die Daten der Resource in der Datenbank ab.

*public Template getTemplate(String name) throws AuthorizeException*
Liefert aus der Tabelle „Templates“ den gewünschten Template zurück.

*5.1.1.1.1.7 Kopieren von Resourcen*

public void copyResource(Resource r,
*Folder Folder*,
String name,
String description) throws RemoteException

Beim Kopieren einer Resource wird die Resource im gegebenen Verzeichnis mit requestNewResource und putResource erzeugt.

public void moveResource(Resource r,
*Folder Folder*,
String to_name,
String description) throws AuthorizeException

Die Resource wird in dem angegeben Verzeichnis(Folder) mit neuem Namen und neuer Beschreibung gespeichert.

Import

*public synchronized void Import(*Folder Folder*,*
*String path,*
*String name)*

Importiert Dateien aus dem gegebenen Verzeichnis in den gegebenen *Folder*. Die Einordnung der Resourcetypen erfolgt über die Extension des Dateinamens. Die Zuordnung der Extensionen zu Resourcetypen ist in einer Tabelle gespeichert. Wenn eine unbekannte Extension auftaucht, wird der Benutzer gefragt, mit welchem Resourctypen sie importiert werden soll. Die Antwort wird in der Tabelle gespeichert, und bei der nächsten Begegnung mit der Extension erzeugt das System die richtigen Resourcetypen.

*public ResourceMetadata getInfo(Resource resource)*

Liefert einen kompletten Satz der Metadaten der Resource zurück. Diese wird aus den Tabellen „Resources“ und „Shared Metadata“ zusammengestellt.

*public Vector getCompleteResources()*

Liefert alle Resourcen zurück, die als completed aber nicht als accepted markiert sind. Dient dazu, dem CvD eine Liste von Resourcen zu erzeugen, die er noch bearbeiten muß.

*public Hashtable getDataTypes()*

Liefert die im Moment verfügbaren Resourcetypen. Dient dazu, beim create Befehl den Benutzer eine Auswahlliste der Typen zu erzeugen.

*public void markComplete(Resource resource)*
Markiert die Resource als complete. In der Tabelle „Resources“ wird das Feld completed gesetzt.

*public void markAccepted(Resource resource) throws RemoteException*
Markiert die Resource als akzeptiert. In der Tabelle „Resources“ wird das Feld accepted gesetzt.

*public void checkPublishable(Resource resource,*
*String caller,*
*boolean mark_schedule) throws AuthorizeException*
Überprüft, ob eine Resource publizierbar ist.(siehe 3.1.11)

*public void publishResource(Resource resource,*
*String caller,*
*boolean* Folder*_oriented) throws AuthorizeException*
Publiziert eine Resource und alle, die damit zusammenhängen. (siehe 3.1.11)

*public void schedule(Resource resource, Task Task, Date date)*
Die Resource wird für das angegebene *Task* zum angegebenen *Date* im *Scheduler* eingetragen.

#### 5.1.1.2 RESOURCE

**Variables**

*public Metadata metadata*
Enthält die Metadaten der Resource.

*private byte data[]*
Enthält den Inhalt der Resource.

*private ResourceFilter resource_filter.*
Enthält das *ResourceFilter* Objekt zum Konvertieren der Resource.

+

**Methods**

*public void edit()*
Öffnet einen Editor für diese Resource. Jede Resource Unterklasse weiß, welche CocoonEditor-Unterklasse sie instanziieren muß. Die Resource erzeugt eine Instanz des Editors, startet sie und gibt sich selber an den Editor weiter.

*public void setData(byte data[])*
Speichert den gegebenen Inhalt in die Resource.

*public byte[] getData()*
Liefert den Inhalt der Resource zurück.

*public byte[] getResultData()*
Liefert den konvertierten Inhalt der Resource.

*protected void loadData() throws RemoteException*
Lädt den Inhalt der Resource vom Server.

*public byte[] loadData(Long Version)*
Holt vom Server eine ältere Version des Inhalts wieder.

*public void updateMetaData(ResourceMetaData rmd)*
Aktualisiert die Metadaten der Resource.

*public Vector getLinks(boolean local)*
Liefert eine Liste der Querverweise, die die der Resource beinhaltet. Bei Datentypen, die keine Querverweise besitzen, liefert diese Funktion eine leere Liste zurück.

*public void replaceLinks(long from, long to)*
Ersetzt alle Querverweise, die den Resource_ID „ from“ haben, zum Wert von „to“.

*public void delete()*
Die Resource löscht sich selber.

*public void move(Folder Folder, String name, String description) throws RemoteException*
Verschiebt die Resource mit dem gegebenen Pfad, neuer Name und neue Beschreibung.

*public void copy(Folder) throws RemoteException*
Kopiert die Resource.

*public Template getTemplate(String template_name)*
Holt den gegebenen Template vom Server.

*public void checkIn()*
Checkt die Resource wieder ein. Löscht den Inhalt der Resource, da er nicht mehr verwendet wird (und werden darf).

*public void save()*
Speichert die Resource ab.

*public String[] getFunctions()*
Gibt die verfügbaren Funktionen der Resource zurück. Dieses wird für den Aufbau des Popup Menus verwendet.

*public void markAccepted()*
Markiert die Resource als abgenommen.

*public void markComplete()*
Markiert die Resource als fertig.

*public void publish()*
Publiziert die Resource.

*public void schedule(Task Task, Date date)*
Die Resource wird für das angegebene *Task* zum angegebenen *Date* im *Scheduler* eingetragen.

*public void CHMOD(String Group)*
Ändert die Gruppenzugehörigkeit der Resource.

*public Enumeration listTemplates()*
Liefert eine Liste von den vorhandenen Templates. Dient zur Creation der Template-Optionen im Editor.

#### 5.1.1.3 ResourceMetadata

Dieses ist ein Datenobjekt, das die Metadaten einer Resource enthält. Sie wird benutzt, um die Felder eines Resourceobjektes mit den Daten aus der Datenbank zu aktualisieren.

**Variablen**

*public long Resource_ID*
Enthält die eindeutige Resource-ID zum Identifizieren der Resource.

*public long Data_ID*
Enthält die eindeutige Data-ID zum Identifizieren der Daten dieser Resource.

*public String Format*
Enthält das Format der Resource.

*public int current_version*
Enthält die aktuelle Versionsnummer.

*public String name*

Enthält den Namen der Resource.

*public String Title*
Enthält den Titel der Resource.

*public String Group*
Enthält den Gruppennamen der Resource.

*public String description*
Enthält eine Kurzbeschreibung der Resource. Dient zur Stichwortsuche.

*public String subject*
Enthält die thematische Einordnung der Resource.

*public Date last_modified*
Enthält das Datum der letzten Änderung der Resource.

*public String author*
Enthält den Namen des Autors der Resource.

*public String publisher*
Enthält den Namen des Verlegers.

*public String Rights*
Enthält einen urheberrechtlichen Hinweis.

*public String is_checked_out_from*
Enthält den Benutzernamen desjenigen, der die Resource gerade bearbeitet.

*public boolean is_checked_out*
Beschreibt, ob die Resource ausgecheckt ist oder nicht.

*public boolean compiled_correctly*
Beschreibt ob die Resource ohne Fehler kompiliert ist oder nicht.

*public boolean completed*
Beschreibt, ob die Resource fertig ist, oder nicht.

*public boolean accepted*
Beschreibt, ob die Resource abgenommen ist oder nicht.

*public boolean published*
Beschreibt, ob die Resource in der jetzigen Version heißgeschaltet ist oder nicht.

*public boolean scheduled*

Beschreibt, ob die Resource für eine Aufgabe im *Scheduler* gemeldet ist oder nicht.

#### 5.1.1.4 ResourceEditor

**Variables**

*Resource resource*
Die Resource,die von diesem Editor bearbeitet wird.

*protected String local_dir*
Enthält den Pfad auf dem lokalen Rechner, der zuletzt benutzt wurde.

*protected String local_file*
Enthält den Dateinamen auf dem lokalen Rechner, der zuletzt benutzt wurde.

*private Vector windows*
Enthält einen Vector mit Referenzen zu allen Fenstern, die von diesem Editor kreiiert wurden, damit man sie auch vernichten kann, wenn der Editor geschlossen wird.

**Methoden**

*public abstract void init()*
Baut das User-Interface für den Editor auf.

public void reg(Frame f)
Registriert ein Fenster mit dem Editor.

*public void unreg(Frame f)*
Entfernt ein Fenster aus der Registrierung.

public void close()
Diese Funktion vernichtet alle Views, die es erzeugt hat, checkt die Resource ein und vernichtet sich selbst.

*protected void save_local()*
Speichert den Inhalt der Resource auf das lokale System.

*protected void load_local()*
Lädt Daten aus einer lokalen Datei in den Editor ein.

*protected void save()*
Speichert das Ergebnis der Änderung in der Resource und schickt eine Nachricht an die Resource, daß sie sich selber speichern soll.

#### 5.1.1.5 *AccessManager*

*public void allowOperation(Resource r, String operation) throws AuthorizeException*
Diese Funktion ist die wichtigste Aufgabe des *AccessManager*s. Vor jeder Operation wird der *AccessManager* gefragt, ob sie zulässig ist. Ist die Operation auf die Resource nicht zulässig, dann wird ein Authorize Exception geworfen.

*public String[] getFunctions(Resource r)*
Liefert eine Liste von Funktionen, die zu diesem Zeitpunkt auf die gegebene Resource von diesem Benutzer ausgeführt werden kann. Diese Funktion wird vom Resource Objekt aufgerufen.

*public void checkOutResource(Resource r) throws AuthorizeException*
Checkt eine Resource aus.

*public void checkInResource(Resource r) throws AuthorizeException*
Checkt eine Resource wieder ein.

*public User getCurrentUser()*
Liefert das *User*-Objekt, das diese Instanz des *AccessManager*s benutzt.

*public boolean authenticate(String User, String password)*
Liefert true, wenn der Benutzer Zugang zum System haben darf, ansonsten false.

*public void chMod(String Group, Resource resource) throws AuthorizeException*
Ändert die Gruppenzugehörigkeit einer Resource. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public boolean exists(Resource r)*
Prüft nach, ob die angegebene Resource bereits in der Datenbank existiert.

*public void checkInResources()*
Gibt alle Resourcen, die von diesem Benutzer bearbeitet wurden, wieder frei. Diese Funktion wird beim Verlassen des Systems durch den Benutzer ausgelöst.

*public Vector UserList() throws AuthorizeException*
Liefert einen Vector mit allen vorhandenen *User* Objekten zurück. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public Vector GroupList() throws AuthorizeException*
Liefert einen Vector mit allen vorhandenen *Group* Objekten zurück. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public Group getGroup(String Group) throws AuthorizeException*
Liefert ein *Group* Objekt, das alle seine Mitglieder (*User* Objekte) beinhaltet. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public User getUser(String User) throws AuthorizeException*

Liefert ein *User* Objekt für den gegebenen Namen zurück. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public Resource addUser(String User, String passwd, String Group) throws AuthorizeException*

Fügt dem System einen neuen Benutzer hinzu. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public void removeUser(String User) throws AuthorizeException*

Entfernt Benutzer vom System. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist, oder wenn der Benutzer eingeloggt ist.

*public String[] getFunctions(Resource resource)*

Liefert die Funktionen, die ein Benutzer auf eine bestimmte Resource ausführen darf. Wird zum Kreieren des Funktionenmenüs benutzt.

*public void addToGroup(String User, String Group) throws AuthorizeException*

Fügt den gegebenen Benutzer der gegebenen Gruppe hinzu. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

*public void removeFromGroup(String User, String Group) throws AuthorizeException*

Entfernt den Benutzer von der Gruppe. Löst einen AuthorizeException aus, wenn der Benutzer kein Administrator ist.

#### 5.1.1.6 *USER*

**Variablen**

*public String name*

Enthält den Benutzernamen.

*private String password*

Enthält eine verschlüsselte Version des Paßworts.

*private String realname*

Beinhaltet den wahren Namen des Benutzers.

*private Vector Groups[]*

Beinhaltet die Gruppen, in denen der Benutzer Mitglied ist.

*Public Hashtable preferences*

Beinhaltet die Präferenzen des Benutzers.

**Funktionen:**

*public void removeFromGroup*(String *Group*)

Entfernt den Benutzer von der Gruppe.

*public String getRealName()*
Liefert den wahren Namen des Benutzers wieder.

*public Vector getGroups()*
Liefert die Gruppen zurück, in denen der Benutzer Mitglied ist.

*public boolean isInGroup(String Group_name)*
Prüft, ob der Benutzer in der angegebenen Gruppe ist.

*public Hashtable getPreferences()*
Liefert die Präferenzen des Benutzers zurück.

#### 5.1.1.7 *LinkManager*

*addLink (Long from_ID, Long To_ID)*
Fügt einen Querverweis der Datenbank hinzu.

*removeLinks (Long From_ID)*
Entfernt den Verweis aus der Datenbank.

*public boolean exists(Long Resource_ID)*
Liefert true, wenn das (lokale) Resource existiert.

*public Vector linksFrom(Long Resource_ID)*
Liefert einen *Vector*, der alle Resourcen beinhaltet, auf den diese Resource zeigt.

*public Vector linksTo(Long Resource_ID)*
Liefert einen *Vector*, der alle Resourcen beinhaltet, die auf diese Resource zeigen.

## 6 Anhang B: Gesamtdiagramm

Im Folgenden eine Übersicht des Gesamtsystems:

## 7 Anhang C: Mainstream Objects

Der Autor hat folgende Ergänzungen zum Modell der „Mainstream Objects“ vorgenommen:

Ein Thread wird mit folgendem neu eingeführten Symbol dargestellt:

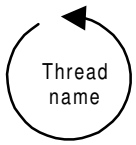



**Abbildung 7-1 Die Darstellung eines Threads**

Ein entferntes Objekt wird mit gestrichelten Hintergrund dargestellt:

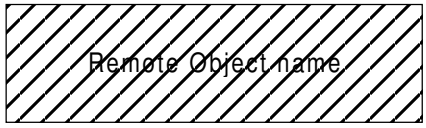



**Abbildung 7-2 Darstellung eines entfernten Objekts**

Zur Unterstützung des Vertändnisses des Lesers wird hier eine kurze Einführung von „Mainstream Objects“ aus der Diplomarbeit von Nico Johanisson „Eine Umgebung für mobile Agenten: Agentenbasierte verteilte Datenbanken am Beispiel der Kopplung autonomer ‘Internet Web Site Profiler’“, Fachbereich Informatik der Universität Hamburg. April 1997, eingefügt.

## 8 Abbildungsverzeichnis

Abbildung 2-1 Geschäftsobjekte bei der Gruner + Jahr Online Produktion ..... 13
Abbildung 2-2 Der Fertigungsablauf für Online-Dokumente ..... 14
Abbildung 2-3: Systemüberblick des aktuellen Systems ..... 15
Abbildung 2-4 WebIza Browser Fenster ..... 18
Abbildung 2-5Das Webiza Editor-Fenster ..... 19
Abbildung 2-6Anwendungsfälle für den Benutzer ..... 23
Abbildung 2-7Anwendungfälle für den Administrator ..... 31
Abbildung 3-1 Überblick der Hauptkomponenten des Cocoon-Systems ..... 38
Abbildung 3-2 Prozeßstruktur vom Gesamtsystem ..... 40
Abbildung 3-3 Vererbungshierarchie der Event-Objekte ..... 43
Abbildung 3-4 Beziehungen der HTTP-Server ..... 44
Abbildung 3-5 Die Client-Sicht des Anmeldevorgangs ..... 45
Abbildung 3-6 Die Kommunikation zwischen Client und Server ..... 47
Abbildung 3-7 Objektinteraktionen beim Initialisieren des Servers ..... 48
Abbildung 3-8 Das Anmelden eines Clients auf dem Server ..... 49
Abbildung 3-9 Beziehungen der Kontrollobjekte. ..... 50
Abbildung 3-10 Objektinteraktionen beim Abmelden des Clients ..... 51
Abbildung 3-11 Ein Überblick aller Datenbanktabellen des Cocoon-Systems ..... 54
Abbildung 3-12 Beziehungen der Daten und Metadaten einer Resource ..... 55
Abbildung 3-13 Gemeinsame Daten von Symlinks werden durch unterschiedliche Funktionsobjekte verarbeitet und liefern unterschiedliche Ergebnisse ..... 56
Abbildung 3-14 Überblick des gesamten Systems ..... 59
Abbildung 3-15 Beziehung der Datenbankobjekte mit den Dateien im Dateisystem ..... 59
Abbildung 3-16 Vererbungshierarchie der Resource-Unterklassen. ..... 61
Abbildung 3-17 Relationen der Resource mit seinen bearbeitenden Objekten ..... 62
Abbildung 3-18 Vererbungshierarchie der *ResourceEditor*-Klassen ..... 63
Abbildung 3-19 Vererbungshierarchie der *ResourceHandler* ..... 64
Abbildung 3-20 Hierarchie der Template-Objekte ..... 65
Abbildung 3-21 Objektinteraktionen zum Editieren einer Resource ..... 66
Abbildung 3-22 Objektinteraktionen zum Speichern einer Resource ..... 67
Abbildung 3-23 Objektinteraktionen beim Erzeugen einer Resource ..... 68
Abbildung 3-24 Objektinteraktionen für das Kopieren einer Resource ..... 69
Abbildung 3-25 Objektinteraktionen für das Verschieben einer Resource ..... 70
Abbildung 3-26 Objektrelationen für Gruppen- und Benutzer-Objekte ..... 72
Abbildung 3-27 Zustände einer Resource und die erlaubten Operationen ..... 74
Abbildung 3-28 Ein unidirektionaler Verweis von A nach B ..... 75
Abbildung 3-29 Speicherung der Links in der Datenbank ..... 80
Abbildung 3-30Mechanismen der Versionsverwaltung ..... 85
Abbildung 3-31 Der Konvertierungsvorgang der XML-Dokument in die Zielformate ..... 94
Abbildung 3-32 Die Einstiegsseite zu den Produktionsystemen verschiedener Websites ..... 100
Abbildung 3-33 Fenster mit Dialogfeld zum Anmelden ..... 101
Abbildung 3-34 Das Bedienungsfenster ..... 102
Abbildung 3-35 Die SourceComponent ..... 105

Abbildung 3-36 Die Benutzerschnittstelle des FormComponents ..... 107
Abbildung 3-37 Schichten des RMI Modells..... 111
Abbildung 3-38Kommunikation der entfernten Objekte ..... 112
Abbildung 7-1 Die Darstellung eines Threads ..... 128
Abbildung 7-2 Darstellung eines entfernten Objekts..... 128

## 9 Literaturverzeichnis

[ABRA 97] Abrahams, Paul W.; Larson, Bruce R. ; UNIX for the impatient. 1997. 0201419793 Addison Wesley

[AOLN 97] America Online. Online Dienst. 1997.http://www.aol.com

[BERL 94] Berners-Lee, Tim ; 1994. Uniform Resource Identifiers (URI) Internet Draft, work-in-progress of the Internet Engineering Task Force (IETF).WWW-Consortium http://www.w3.org/Addressing/URL/URI_Overview.html

[BERN 94] Berners-Lee, T. ; Masinter, L. ; McCahill., M. ; 1994 Uniform Resource Locators (URL). RFC 1738, CERN. ftp://ds.internic.net/rfc/rfc1738.txt

[BOUT 97] Boutell, T. et al. ; 1997. PNG (Portable Network Graphics) Specification. RFC 2083, NEXOR http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2083.txt

[BREN 96] Brenner, Steven E. ; Aoki, Edwin ; Introduction to CGI/Perl. 1996. 1558514783 M&T Books 115 West 18th Street New York, New York 10011

[CALL 95] Callaghan, B. ; Pawlowski, B. ; Staubach, P. ; 1995. NFS: Network File System Protocol Specification. RFC, 1813 Sun Microsystems, Inc.http://src.doc.ic.ac.uk/computing/internet/rfc/rfc1813.txt

[CASS 97] Cassady-Dorian, Luke ; 1997; Industrial strength Java. 1562056344 New Riders Publishing.

[COLB 96] Colby, Martin ; Jackson, David S. ; 1996 Using SGML Special Edition. 0789704145 QUE.

[CORN 96] Cornell, Gary; Horstmann, Cay S.; Core Java. 1996. 0135968917 Prentic Hall SunSoft Press.

[COVE 97] Cover, Robin ; 1997. DSSSL (Document Style Semantics and Specification Language). ISO/IEC 10179:1996. Summer Institute of Linguistics (http://www.sil.org/sgml/related.html

[DCSE 90] Department of Computer Science. The University of Edinburgh 1990. GIF Graphic Format (c) CompuServe Incorporated Columbus, Ohio http://www.dcs.ed.ac.uk/~mxr/gfx/2d/GIF89a.txt

[DYNA 96] Inso Corporation 1996. Inso Dynamic Publishing Solutions. Dynabase Web Management System. http://www.inso.com/frames/consumer/db/index.htm.

[ENGE 84] Engelbart 1984; Engelbart & Lethman 1988 ; Augment. Bootstrap Institute. http://www2.bootstrap.org/library.htm

[FIEL 97] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Berners-Lee, T. ; 1997. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068. U.C. Irvine, Irvine CA. http://www.ics.uci.edu/pub/ietf/http/rfc2068.txt

[GOLD 95] Charles F. Goldfarb; The SGML Handbook first published 1995. 0198537379 Oxford University Press, Walton Street, Oxford.

[HAKO 97] Lie, Hakon; Bos, Bert ; 1997. Cascading style sheets : Designing for the Web. 020141998X Addison-Wesley Long Hi Ed.

[HAMI 92] Hamilton, Eric ; 1992. JPEG Graphic Format. C-Cube Microsystems Department of Computer Science. The University of Edinburgh. http://www.dcs.ed.ac.uk/~mxr/gfx/2d/JPEG.txt

[ISOR 97] International Standars Organization. 1997 HyTime Second Edition. ISO 10744 (1992) http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html.

[KENN 97] Kennedy; Musciano; Using HTML : The definitive Guide. 2nd ed 1997. 1565922352 OReilly & Associates, Inc.

[LEMA 96] Lemay, Laura ; Perkins, Charles L. ; Teach yourself JAVA in 21 Days 1575210304 Howard W Sams (Sams.net)

[MICR 96] Microsoft Corporation 1996. FrontPage: A Web Publishing Solution. http://www.microsoft.com/frontpage/learn.htm.

[NETO 96] NetObjects Company 1996. NetObects Fusion. http://www.netobjects.com/html/winproducts.html

[NETS 96] Netscape Corporation 1996. Netscape Publishing System White Paper. http://www.netscape.com/comprod/products/iapps/capps/pubsys_white_paper.html .

[NORT 97] North, Ken ; The complete guide to Java database programming JDBC, ODBC & SQL 1997. 0079132863 McGraw-Hill USA.

[ORFA 95] Orfali, R., Harkey, D., Edwards, J. The Essential Distributed Objects Survival Guide 1995. 0471129933 Wiley.

[ORFA 96] Orfali, R., Harkey, D., Edwards, J. ; The Essential Client/Server Survival Guide 2nd ed. 1996. 0471153257 Wiley.

[POST 85] Postel, J. ; Reynolds, J. ; 1985. File Transfer Protocol (FTP), RFC 959. ftp://ds.internic.net/rfc/rfc959.txt

[QUAR 96] Quark Publishing System. 1996. QPS: The Basics Quark, Inc. P.O. Box 480446 Denver, CO 80248-0446 http://www.quark.com/qps005.htm

[REAL 97] Realaudio Audio Format. http://www.realaudio.com/

[STAN 96] Stanek, William R. ; HTML, CGI, SGML, VRML, Java, Web Publishing Unleashed, w. CD-ROM 1996. 1575210517 SAMS NET H.

[TEIN 97] Text Encoding Initiative (TEI) 1997. http://www.uic.edu/orgs/tei/index.html.

[WAVE 94] Waveform Audio File Format (WAVE) http://www.neurophys.wisc.edu/www/aud/riff-format.txt

[WEIB 96] Weibel, Stuart ; Jul, Erik ; 1996. Persisten Uniform Resource Locators (PURLs). Online Computer Library Center (OCLC) http://www.oclc.org/oclc/research/publications/review96/purl.htm

[WEIB 97] Weibel, S. ; Kunze, J. ; Lagoze, C. 1997. Dublin Core Metadata for Simple Resource Description. Internet Draft, work-in-progress. ftp://ds.internic.net/internet-drafts/draft-kunze-dc-01.txt

[WETT 97] Wettern, Joern; Hawkins, Nancy; Inside TCP/IP. Reissued 3rd ed 1997. 1562057146 New Riders Publishing.

[WHIT 97] Whitehead, E. James ; 1997. World Wide Web Distributed Authoring. and Versioning (WebDAV): An Introduction. From StandardView Vol. 5, No.1 March 1997.

[W3CX 97] W3C Architecture domain. 1997. Extensible Markup Language (XML) http://www.w3.org/XML/

[YOUR 96] Yourdon, E., Whitehead, K., Thomann, J., Oppel, K., Nevermann, P.; Mainsteam Objects. Deutsche Ausgabe 1996. 3827295173 Prentice Hall Verlag Haar bei München.

Ich erkläre hiermit, die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen benutzt zu haben.

Hamburg, den 15. September 1997 _____

(Yervand Nahapetian)