Content Management Systeme

Diplomarbeit an der TU Budapest
und der TU Dresden
Mai 2001

Fiala Zoltán

Betreuer:

Prof. Dr. Csopaki Gyula, *Budapesti Műszaki Egyetem*
Prof. Dr. Klaus Meißner, *Technische Universität Dresden*
Ronny Franke, *Multimedia Software GmbH, Dresden*

Aufgabenstellung

Content Management Systeme

Die Menge der gesammelten und im Web veröffentlichten Informationen vervielfacht sich in immer kleineren Abständen. Die Organisation, Strukturierung, Bündelung und Aktualisierung von Informationen, sowie der Transport der richtigen Inhalte an den richtigen Ort ist heute eine der größten Herausforderungen an E-Commerce-Lösungen.

Web Content Management Systeme (WCMS) ermöglichen die strukturierte und automatisierte Erstellung, Aufbereitung, Verwaltung, Änderung und Aktualisierung von Informationen im Inter- und Intranet. Kennzeichen von *WCM*-Systemen sind die datenbankgestützte Echtzeit-Publikation und Verwaltung von Web-Inhalten durch mehrere Personen sowohl aus dem Entwickler- als auch aus dem Redaktions- und Verwaltungsbereich.

Im Rahmen dieser Diplomarbeit soll auf die allgemeine Struktur, Architektur und Funktionalität der heute am meisten verwendeten *WCM*-Lösungen durch eine vergleichende Analyse eingegangen werden. Im Mittelpunkt der Untersuchungen steht die Evaluation preiswerter Systeme und Lösungsmöglichkeiten, die eine günstige Alternative gegenüber den heute am breitesten verwendeten kostaufwändigen kommerziellen Systemen bieten.

Zur Veranschaulichung der Forschungsergebnisse soll ein kleines *WCM*-Beispielsystem in einer preisgünstigen Umgebung (*Linux*, *Apache Web-Server*, *PHP4*, *MySQL*) realisiert werden, die die wesentlichsten Funktionalitäten einer *WCM*-Lösung, so wie *Trennung von Inhalt und Layout*, *Staging*, *Storage*, *Versionierung*, *Verwaltung und Aktualisierung von Assets* etc. darstellt. Es soll dabei eine Evaluierung der verschiedenen zur Verfügung stehenden preisgünstigen Lösungsmöglichkeiten erfolgen.

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen erstellt zu haben.

Dresden, den 11.05.2001 Zoltán Fiala

Inhaltsverzeichnis

Kapitel 1

Einführung

Wir leben heute im sog. Informationszeitalter. Die rasante Entwicklung der Telekommunikation bzw. der Datenverarbeitung hat in den vergangenen Jahren dazu geführt, daß die digitale Kommunikation nicht nur einer der stärksten Wirtschaftszweige, sondern auch einer der bestimmendsten Faktoren unseres alltäglichen Lebens geworden ist.

Verglichen mit früheren Zeiten, als ein allgemeiner Mangel an schnell zugänglichen Informationen geherrscht hat, haben wir es heute mit einem „Überschuß" von Inhalten zu tun. Vor allem die schnelle Entwicklung des Internet hat dazu beigetragen, daß jedem von uns praktisch unendliche digitale Informationsquellen zur Verfügung stehen. Die entscheidende Frage unserer Zeit ist, wie jemand die für sich relevanten Informationen schnell und effizient heraussuchen und bearbeiten kann.

Endverbraucher suchen heute nach Anbietern, die zeitnahe, hilfreiche Informationen aufbereiten und auf eine komfortable, verständliche Art und Weise bereitstellen können. Es genießen jene Anbieter einen besonderen Vorteil, die sich um die Strukturierung, die Qualität und Aktualität ihrer Inhalte kümmern.

Der Grundgedanke von *Content Management (CM)* ist die systematische, strukturierte Aufbereitung, Bearbeitung und Veröffentlichung der verschiedensten digitalen Inhalte. *Content Management Systeme* ermöglichen, daß die richtigen Inhalte zur richtigen Zeit am richtigen Ort ankommen und führen dadurch zu einem signifikanten Produktivitätsschub in Unternehmen.

In dieser Arbeit werden zunächst *Content Management Systeme* allgemein beschrieben und klassifiziert.
Dann wird ein Vergleich der heute existierenden *CM*-Werkzeuge vorgenommen, wobei der Schwerpunkt auf die Untersuchung preiswerter Lösungsmöglichkeiten gelegt wird.
Anschließend wird ein *CMS* in einer preiswerten Umgebung konzipiert und anhand eines Prototypen realisiert.

Kapitel 2

Content Management Systeme

In diesem Kapitel werden die wichtigsten Eigenschaften und Kennzeichen von *Content Management Systemen* (*CMS*) diskutiert. Es wird auf ihre allgemeine Struktur, Architektur und Funktionalität durch eine vergleichende Analyse eingegangen.
Zunächst werden die wichtigsten Begriffe definiert, dann die wesentlichsten Funktionen von *CMS*-Werkzeugen zusammengefaßt. Schließlich werden Kriterien zu ihrer Bewertung aufgestellt.

2.1 Definitionen

Content Management ist eine neue Disziplin in der Informatik, die sich mit der automatisierten Verwaltung von digitalen Informationen beschäftigt. Vor der genauen Untersuchung von *Content Management Systemen* ist es nötig, die wichtigsten Begriffe zu definieren.

Definition 2.1.1 Content *ist Information gespeichert auf einem Datenträger.*

Definition 2.1.2 Content Management *umfaßt alle geschäftlichen und technischen Prozesse der Aufbereitung (Capturing), der Abfrage (Retrieval), der Verwaltung (Maintenance) und der Veröffentlichung (Publication) von Content.*

Die beiden Definitionen sind etwas zu allgemein, um mit ihnen weiterarbeiten zu können. Außerdem ist es wichtig zu betonen, daß *Content Management* heute in erster Linie in der Welt des *World Wide Web* Anwendung findet. Wenn man heute über ein *CMS* redet, meint man damit meistens ein *Web Content Management System (WCMS)*.
In dieser Arbeit wird die Diskussion von *Web Content Management Systeme*n in den Mittelpunkt gestellt, des weiteren werden auch allgemeine *CM*-Funktionen erwähnt. Die folgenden Definitionen versuchen die Grundbegriffe der in dieser Arbeit diskutierten Technologien etwas genauer zu erfassen:

Definition 2.1.3 *Unter* Web Content *(Web Inhalt) versteht man im* WWW *die Gesamtheit der Inhalte in einem Web-Auftritt. Inhalte sind:*

- *die Web-Seiten selbst, sowie ihre funktionellen Elemente: Texte, Grafiken, Videos, Sound, etc.*
- *die im Hintergrund der Web-Seiten laufenden Programme, Skripte, Datenbankprozeduren und alle logischen Informationen, die die dynamische Funktionsweise von Web-Auftritten ermöglichen,*
- *die hinter den Web-Seiten stehenden Datenbanken,*
- *Informationsunterlagen (Dateien), auf die über eine Web-Seite zugegriffen werden kann.*

Definition 2.1.4 *Unter* Assets *versteht man die oben genannten Inhaltskomponenten, wie Texte, Grafiken, Images, Sounds, Animationen, Videos, Skripte, etc.*

Definition 2.1.5 Web Content Management *ist die systematische Verwaltung und Administration der genannten Web-Inhalten. Die wesentlichen Aufgaben von* Content Management *sind:*

- Planung *(Design)*
- Erstellen *und* Editieren *(Authoring)*
- Gestaltung *(Conversion)*
- Speicherung *(Storage)*
- Publikation *(Publishing)*
- Installation *und der* Transport *(Deployment and Staging)*
- Verwaltung *und* Aktualisierung *(Maintenance and Update)*
- Archivierung *(Archival)*
- Testen *und die* Analyse *(Reporting and Analysis)*

dieser Inhalte.

Definition 2.1.6 *Ein* Content Management System *ist eine komplexe Softwareplattform für die Unterstützung dieser Aufgaben. Mit der Hilfe von* Content Management Systemen *lassen sich Web-Auftritte während ihrer gesamten Lebensdauer zentral verwalten. Ein wichtiges Kennzeichen von* CM-*Systemen ist die Verwaltung von Web-Inhalten durch mehrere Personen sowohl aus dem Entwickler- als auch aus dem Redaktions- und Verwaltungsbereich.*

Definition 2.1.7 *Der Begriff* Content-Life-Cycle *beschreibt den Lebenszyklus von Inhalten auf einer Web-Seite. Der theoretische Ansatz teilt die Informationsverarbeitung in folgende Abschnitte - siehe auch Abbildung 2.1 auf Seite 10.*

Analyse und Planung: *In dieser Phase werden die Inhalte und das Aussehen eines Auftrittes geplant. Es werden die Anforderungen der Kunden bzw. die zur Verfügung stehenden Techniken analysiert und die logische, die inhaltliche und die gestalterische Struktur der Präsentation entworfen.*

Erstellung: *In dieser Phase erstellen die Autoren, Designer und Grafiker alle digitalen Assets, die die Inhalte der Web-Site darstellen. Diese Dokumente sind Texte, Grafiken, Images, Animationen, Skripte, etc.*

Kontrolle und Freigabe: *In der nächsten Phase werden die erstellten Assets auf semantische, syntaktische und gestalterische Korrektheit überprüft. Ist das Ergebnis der Überprüfung zufriedenstellend, so können sie weitergereicht werden, ansonsten kommen sie wieder in die Erstellungsphase zurück.*

Publikation: *Die freigegebenen, genehmigten Inhalte werden im Inter-, Intra- oder oder Extranet veröffentlicht. Hier kommt es auch zur Frage von Caching - siehe 2.2.8.*

Aktualisierung: *Die Inhalte und Dienste eines Auftrittes müssen ständig aktualisiert werden. Dies kann oft automatisch erfolgen, in einigen Fällen ist jedoch eine „manuelle" Überarbeitung der Inhalte nötig. So erfolgt eine „Rückkoppelung" zur Kontrollphase. Die aktualisierten Inhalte sollen dann natürlich wieder überprüft und freigegeben werden.*

Bei der Aktualisierung der Assets ist es meistens erforderlich, die alten Assets als Version zu speichern – siehe Archivierung.

Archivierung: *Die alten Inhalte sollen archiviert und versioniert werden, um Inhalte wiederverwenden und Backups erstellen zu können. Es soll ferner die effiziente Suche unter den gespeicherten Inhalten ermöglicht werden.*

2.2 Kennzeichen und Komponenten von *WCM*-Systemen

Im Abschnitt 2.1 wurde der Begriff *Content Management* definiert, des weiteren wurden seine wichtigsten Aufgaben dargestellt. Dieser Abschnitt wird der ausführlichen Besprechung der wesentlichsten Kennzeichen und Komponenten von *Content Management Systemen* gewidmet.

2.2.1 Trennung von Inhalt und Layout

Die „Sprache" des *World Wide Web* ist *HTML*, d.h. die veröffentlichten Inhalte erscheinen im Internet als *HTML*-Seiten. Die klassische Methode der Publikation von *Web-Content* ist das Erstellen von statischen *HTML*-Seiten. Mit den heute zur Verfügung stehenden Editoren ist diese Aufgabe ziemlich einfach.

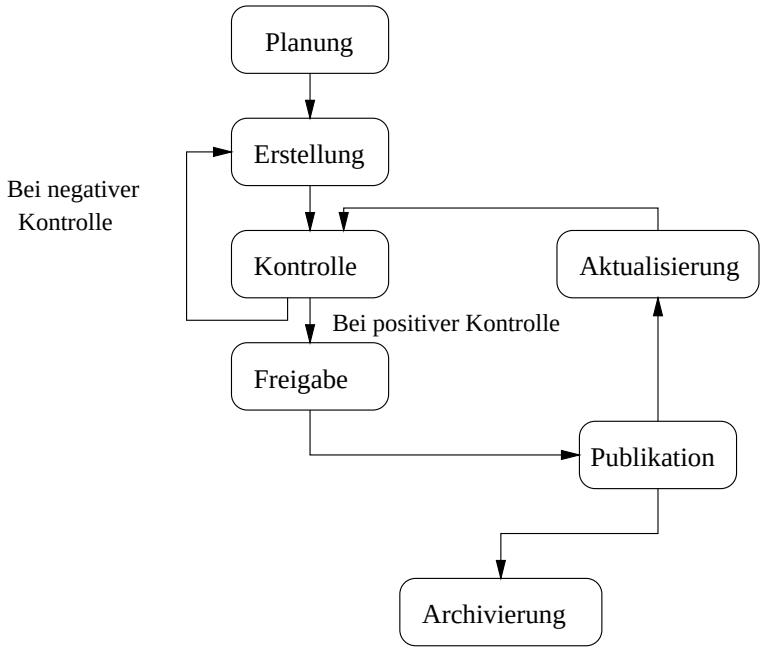

Abbildung 2.1: Content-Life-Cycle

Der Zuwachs der zu veröffentlichenden Informationen und der Bedarf, die Web-Auftritte aktuell und konsistent zu halten, zeigen jedoch die Nachteile dieses Verfahrens. Die Handhabung der Informationsmengen auf großen Sites ist nach der klassischen Methode über statische *HTML*-Seiten nicht mehr effizient möglich.
Betrachten wir die Situation, daß die Inhalte einer Site verändert werden. In diesem Fall sollen die statischen *HTML*-Seiten einzeln überarbeitet und aktualisiert werden. (Die Veränderung des Aussehens der Seite hat natürlich die gleichen Folgen, d.h. es ist eine sehr enge Zusammenarbeit von Redakteuren und Layout-Designern nötig.) Noch komplizierter ist es, wenn die veränderten Informationen auf mehrere Seiten eine Auswirkung haben, dann sollen diese alle einzeln aktualisiert werden. So ist es eine ziemlich arbeits- und kostenaufwendige Aufgabe, Web-Auftritte konsistent zu halten. Bei großen Web-Auftritten, die ihr Layout und ihre Inhalte täglich oder noch öfter verändern, fallen nicht selten 90 % der Kosten für die Pflege der Web-Seiten an.

Grundprinzip von *Content Management Systemen* ist die *Trennung von Layout und Inhalt*, wodurch erreicht wird, daß die Redakteure sich nicht mehr um die Darstellung oder die Einbindung ihrer Beiträge in das Gesamtangebot kümmern müssen. Dies erfolgt mit dem Einsatz von *Templates*.
Ein *Template* ist im Grunde genommen eine *dynamische Layout-Vorlage* für eine Web-Seite, sie enthält Informationen über ihr Aussehen und die Quelle ihrer Inhalte. Die eigentlichen Inhalte befinden sich meistens in einer Datenbank oder im Dateisystem des *CMS*. Beim Abruf der Seite wird der dem Web-Browser zurückgeschickte *HTML*-Code auf der Server-Seite anhand des Template und der aktuellen Inhalte in der Datenbank erzeugt.
Die Aktualisierung der Inhalte der Seite bedeutet in diesem Fall nur die Veränderung der ent-

sprechenden Einträge in der Datenbank. Dieses Verfahren ist vor allem dann sehr effizient, wenn die neuen Inhalte in mehreren Dokumenten erscheinen sollen.
Die Abbildung 2.2 zeigt das Prinzip der *Trennung von Content und Design*. Der systeminterne *TemplateProcessor* erstellt die Seiten dynamisch anhand der Layout-Vorgaben und der Datenbankinhalte.

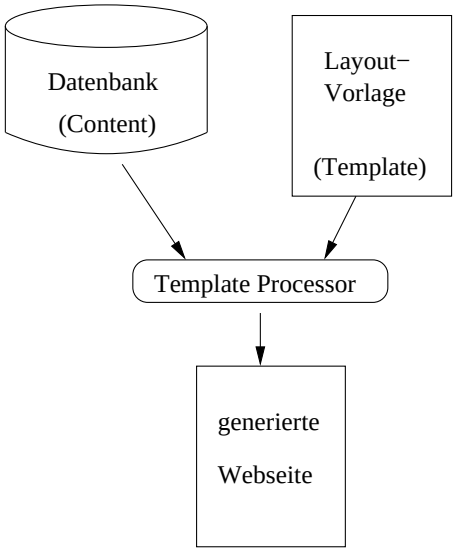

Abbildung 2.2: Trennung von Inhalt und Layout mit Templates

Das Template enthält Code, der an den *TemplateProcessor* übergeben und dynamisch verarbeitet wird. Dieser interpretiert den Code, holt die entsprechenden Einträge aus der Datenbank und baut die zu veröffentlichende Web-Seite zusammen.

2.2.2 Assetmanagement

Unter *Assetmanagement* versteht man die systematische Speicherung, Strukturierung, Verfolgung und Kontrolle der in einem *CMS* verwalteten Inhalte. Assetmanagement beschäftigt sich u.a. mit der Darstellung, der Adressierung und der Archivierung der Assets.

Darstellung der Informationsstruktur - logische und physikalische Adressierung

Beim Abruf einer Web-Seite greifen Web-Server standardmäßig auf das Dateisystem zu und suchen dort nach den Inhalten, seien es einfache statische *HTML*-Seiten, dynamische *CGI*-Skripte oder multimediale Elemente. Alle Ressourcen haben im Dateisystem einen *physikalischen Namen*, der sie eindeutig identifiziert. Je nach Konfiguration des Servers können sie sich in beliebigen Verzeichnissen des Dateisystems befinden.
Die Trennung von Inhalt und Layout - siehe 2.2 - erfordert in den meisten Fällen die Verwendung von Datenbanken, die alle Inhalte systematisch und strukturiert speichern und verwalten können. Datenbanken können die verschiedensten Arten von Inhalten enthalten, es ist nicht selten der Fall, daß auch ganze *HTML*-Seiten oder *CGI*-Skripte samt ihrer früheren

Versionen in der Datenbank abgelegt werden. Die Datenbank adressiert die einzelnen Assets physikalisch nicht mehr mit Pfadnamen, sondern mit numerischen Identifikatoren - *ID*s.

Um eine Web-Seite effektiv verwalten zu können, ist es nötig, ihre Inhalte auch semantisch zu strukturieren. Die Inhalte sollen eine leicht überschaubare Struktur bekommen, sie sollen durch *logische Namen* referenziert und untereinander verknüpft werden können. Solche bevorzugten Strukturen sind z.B. *Baumgrafen*, die die hierarchische Adressierung von Content ermöglichen.

Eine wichtige Eigenschaft von *Content Management Systemen* ist die *Trennung der physikalischen und der logischen Adressierung der Inhalte*. Die Nutzer des Systems können dadurch in logischen, übersichtlichen Strukturen denken, mit der physikalischen Speicherung der Inhalte sollen sie sich nicht beschäftigen. Um z.B. Assets in den Datenbanken adressieren zu können, verwendet man statt numerischer Identifikatoren einfache Namen. Es ist die Aufgabe des *CMS*, diese logischen Namen auf die physikalischen Adressen abzubilden. So kann es z.B. sein, daß Inhalte, die sich logisch als Knoten eines hierarchischen Baumgrafen referenzieren lassen, in Wirklichkeit in einer einfachen Struktur, z.B. in einer „flachen" Datenbanktabelle gespeichert sind.
Das *CMS* implementiert also ein Modul zur Übersetzung der logischen Namen in die physikalischen Adressen. Die Abbildung 2.3 zeigt die Funktionsweise dieses Moduls.

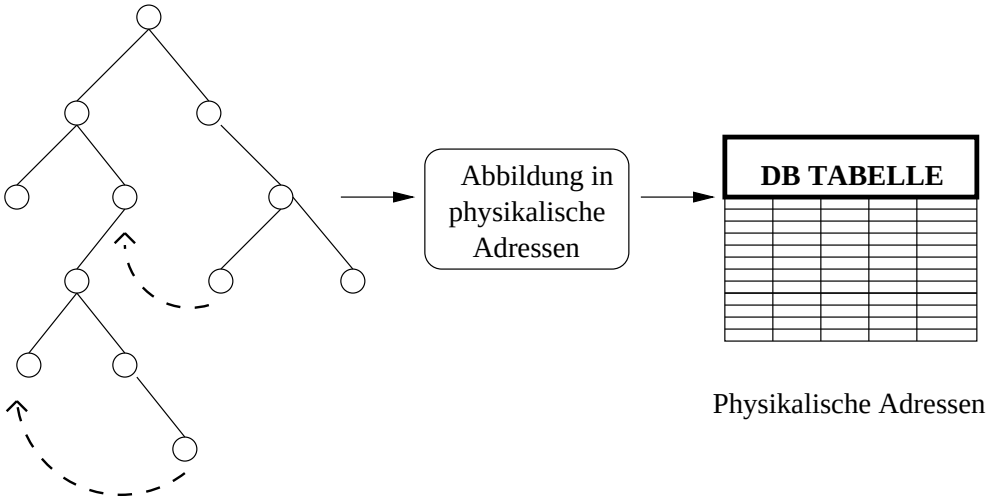

Abbildung 2.3: Trennung der logischen und der physikalischen Adressierung

Die logische Adressierung wird in der Praxis so erreicht, daß die eigentlichen Inhalte mit sog. *Meta-Informationen* versehen werden, die über ihre Eigenschaften und die Beziehungen zwischen ihnen Auskunft geben.

Definition 2.2.1 *Ein* Meta-Datum *ist zusätzliche logische Information zur Beschreibung von Content.* Meta-Data *besteht im allgemeinen aus* Schlüssel/Werte-*Paaren, wobei die Werte meistens textuelle Informationen sind. In einigen Fällen können jedoch auch Bilder oder Audio-Daten Meta-Data Werte sein.*

Versionierung

Web-Seiten werden ständig weiterentwickelt und aktualisiert, sie wechseln ihre Inhalte und Layouts schnell. Die Inhalte ändern und erweitern sich, immer wieder werden alte Informationen gelöscht oder archiviert bzw. neue Informationen hinzugefügt. In vielen Situationen ist es erwünscht, die Veränderungen einer Web-Site nachvollziehen, ihre „Vorgeschichte" rekonstruieren oder Statistiken über ihre Entwicklung erstellen zu können. Diese Aufgabe erfüllt die sog. *Versionierung*.
Moderne *Content Management Systeme* ermöglichen deshalb, die Inhalte einer Web-Site bzw. ihre Darstellung zu einem gegebenen Zeitpunkt als *Version* „einzufrieren". Die gespeicherten Versionen können zu jeder Zeit wieder rekonstruiert, bearbeitet und wiederverwendet werden. Dies ist in vielen Fällen eine sehr komplexe Aufgabe. Es müssen alle Komponenten gesichert werden: Datenbankeinträge, Layout-Vorgaben, Skriptprogramme, alle statischen und dynamischen Informationen, die zur Gestaltung der Site zum gegebenen Zeitpunkt beigetragen haben.

2.2.3 Im- und Exportschnittstellen für Content

Die Inhalte eines *CMS* können aus verschiedenen Quellen stammen und die verschiedensten Formate haben. Wie schon im Abschnitt 2.2 erwähnt, müssen sie in ein „webgerechtes" Format (*HTML*) gebracht werden, damit sie im Internet publiziert werden. Andererseits ist es auch eine wichtige Anforderung an das System, daß die Autoren und die Redakteure sich nicht um die technische Umsetzung der Veröffentlichung der Inhalte kümmern müssen, sondern herkömmliche Autorenprogramme verwenden können. Das *Content Management System* soll Ex- bzw. Importschnittstellen implementieren, mit deren Hilfe unterschiedliche „gängige" Dokumentenformate in das System importiert bzw. aus ihm exportiert werden können.
Die Importschnittstelle muß folgende Aufgaben erfüllen können:

- Sie soll verschiedene digitale Dokumentenformate importieren und umwandeln können. Die gängigsten, zu unterstützenden Formate sind:
  - *Text*-, *HTML*- und *Rich-Text*-Dateien (*MS Frontpage, MS Office, etc.*)
  - *Microsoft Word* und *Excel* Dateien (*MS Office*)
  - *Adobes Portable Document Format* (*Adobe Acrobat*)
- Informationen aus Datenbanken sollen einfach eingebunden werden können.
- Der Import bereits bestehender Web-Seiten in das System soll ermöglicht werden.

Der Import von veschiedenen Formaten ins *CMS* erfolgt mit sogenannten *Filtern* - siehe Abbildung 2.4 auf Seite 14. Die Dokumente aus einer externen Anwendung werden von einem *Filter* in ein Format gebracht, das im *CMS* eingebunden, indexiert und publiziert werden kann.

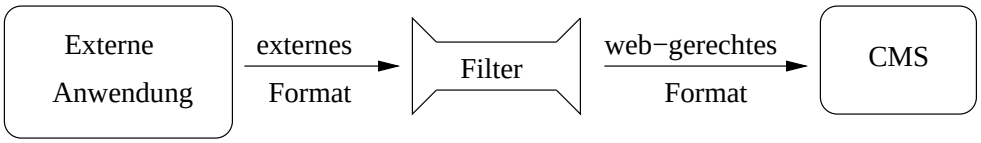

Abbildung 2.4: Importfilter

2.2.4 Personalisierung

Die im Inter- oder Intranet veröffentlichten Web-Seiten werden von vielen Nutzern besucht. Häufig ist es wichtig, bestimmte Inhalte nur für bestimmte Besucher zugänglich zu machen und vor anderen zu „verstecken", das Aussehen der Site an die Eigenschaften des aktuellen Nutzers anzupassen, sog. *Nutzerprofile* zu erstellen.
Eine andere Aufgabe der Personalisierung ist die Anpassung der gelieferten Inhalte an die zur Verfügung stehende Bandbreite bzw. die Endgeräte der Nutzer. Diese können zur Kommunikation die verschiedensten Technologien verwenden: mobile Telefone, analoge Telefonverbindungen, *ISDN*-Anschlüsse, *set top boxes*, *ATM*-Netzwerke etc. Es ist die Verantwortung des Inhalt-Anbieters, die Inhalte an die verschiedenen Schnittstellen anzupassen. Auf dieses Thema wird im Abschnitt 2.3.3 eingegangen.

Die *Personalisierung* einer Web-Site erfordert die dynamische Generierung der Seiten beim Abruf durch einen Nutzer. Der Nutzer wird identifiziert, das *CMS* bestimmt seine Rechte und alle mit ihm verbundenen Informationen, anhand deren die Seiten erstellt werden. Das Nutzerverhalten wird während der Sitzung - *Session* - beobachtet und verwaltet.

Personalisierung im *WWW* durch Sessionverwaltung

Um die Tätigkeiten der Nutzer in *CM*-Systemen oder i.a. auf Web-Sites verfolgen zu können, sind sog. *Sessions* zu definieren. Unter einer *Session* versteht man die Sitzung eines Nutzers in einem Web-Auftritt, seine ganze Navigation durch die Web-Seiten und der veröffentlichten Informationen. Während der Session wird der Nutzer vom System identifiziert, es werden Informationen über seine Vorgeschichte, sein Nutzungsverhalten und seine Zugangsrechte im System gespeichert. Dies macht unmittelbar zentrale Nachteile des *HTTP* deutlich.
HTTP wurde als ein verbindungsloses Protokoll definiert: der Browser wendet sich mit einem Abruf (*HTTP GET*) an den Server, der mit der gewünschten Information antwortet. Wenn eine Web-Seite aus mehreren Elementen - Bilder, Sounds, etc. - besteht, wird für sie jeweils eine neue, separate Verbindung aufgebaut. Möchte der Nutzer mit einem Link auf eine andere Seite „weitersurfen", wird wieder ein neuer *Request* gestartet, der von dem vorherigen völlig unabhängig ist. Dies bedeutet, daß *HTTP* „keine Sessions verwaltet", diese müssen mit anderen Methoden verwirklicht werden.
Hierzu gibt es heute folgende Möglichkeiten:

- Bei den sog. **Client-Side Lösungen** werden die zur Verfolgung von Sessions erforderlichen Informationen als *Sessionvariablen* auf der Client-Seite gespeichert. Diese Verfahren bieten den Vorteil, daß die Belastung des Servers geringer ist, aus der Sicht

der Zuverlässigkeit sind sie aber weniger zufriedenstellend. Folgende Methoden sind bekannt:

- Eine ganz einfache Methode ist die *Speicherung von Sessionvariablen in versteckten Feldern* in *HTML*-Formularen. Versteckte Felder sind für den Nutzer unsichtbar, sie sind nur in dem *HTML*-Quellcode zu sehen. Ein Nachteil dieses Mechanismus ist, daß er die Verwendung von *HTML*-Formularen voraussetzt. Folgender Code zeigt ein Beispiel für versteckte Felder.

  ```
  <FORM>
  ...
     <INPUT TYPE="HIDDEN"
            NAME="who_am_i"
            VALUE="Zoltan_Fiala">
  ...
  </FORM>
  ```

- Eine weitere Methode für die Weiterreichung von Sessionvariablen ist, daß sie in die URI eingebaut werden. Dieses Verfahren ist bei der Anwendung der *HTTP-GET* Methode einsetzbar. Hier sieht die Übertragung einer Sessionvariablen folgendermaßen aus:

  ```
  <A HREF="nextpage.html?who_am_i=Zoltan_Fiala">
  ```

  Der große Nachteil von dieser Methode ist, daß die zu übertragenden Sessiondaten nicht verschlüsselt sind. Diese Gefahr kann durch entsprechende Kodierung verringert werden, in dem die Daten auf der Seite des Senders kodiert und beim Empfänger dekodiert werden. (Verschiedene fundierte Kodierungsmechanismen bieten z.B. die *mcrypt*-Funktionen von *PHP4*.)

- Eine verbreitete Mehode ist die Verwendung von *Cookies*. Es handelt sich um Informationen in textuellen Schlüssel/Werte-Paaren, die die Web-Anwendungen auf den Klienten ablegen, um für spätere Besuche Informationen zu speichern. Viele Web-Programmierumgebungen - z.B. *PHP* - bieten Funktionen zur Behandlung von *Cookies*.

- Bei den **Server-Side Lösungen** werden die Sessionvariablen auf der Server-Seite abgelegt. Diese Methoden sind sicherer und auch eleganter, habel jedoch eine entsprechende Belastung des Servers zur Folge. Möglichkeiten sind:

  - Die Session-Variablen können auf der Serverseite im Dateisystem abgelegt werden. Dies kann bei hohen Nutzerzahlen zu einer substantieller Belastung des Servers führen. Gleichzeitige Schreib-Zugriffe auf dieselbe Datei sollten vermieden werden.
  - Die eleganteste und auch komplizierteste Methode ist die Speicherung von Sessionvariablen in einer Datenbank. Dieses Verfahren fordert hohe Programmierungskenntnisse und den Einsatz leistungsfähiger Server.

– Da der Bedarf an personalisierten Web-Auftritten immer größer wird, bieten verschiedene Firmen und Organisationen Server-Programmierschnittstellen, die die Einrichtung und die Verwaltung von Sessionvariablen sehr vereinfachen. Solche Möglichkeiten bietet z.B. die Firma *Microsoft* mit der Schnittstelle *ISAPI* (Internet Server Application Programming Interface). Eine kostenlose Alternative dazu ist die Skriptsprache *PHP* („PHP Hypertext Preprocessor"), die ab der Version 4 die Verwaltung von Session-Variablen und -funktionen unterstützt. (Siehe [Kra00].)

2.2.5 Workflow und Rollenverteilung

Die Erstellung und Verwaltung von Web-Seiten setzt die Zusammenarbeit von Personen aus verschiedenen Bereichen voraus: an einem Web-Auftritt sind Programmierer, Techniker , Grafiker, Designer, Redakteure und Projektleiter tätig. Redakteure sind für die textuellen Inhalte, Grafiker und Designer für das Layout und die eingebetteten multimedialen Elemente, Techniker und Programmierer für die Export-, Import- und Programmierschnittstellen, Projektleiter für die Überwachung und die Planung der Tätigkeiten zuständig. Die Veröffentlichung neuer Inhalte oder Online-Dienste macht eine Vielzahl koordinierter Interaktionen unter diesen Personen und eine klare Definition von einem Rollenkonzept notwendig, das die Arbeitsabläufe des Teams wiederspiegelt.
Das Workflowmanagement in einem *Content Management System* ist für die *Koordination und die Synchronisation der Tätigkeiten der Arbeitsgruppen* verantwortlich. Es muß im wesentlichen die folgenden Aufgaben erfüllen:

- Definition und Verwaltung von Arbeitsrollen
- Synchronisation der Arbeitsabläufe
- Implementation von Benachrichtigungs- und Freigabemechanismen

Workflows und *Rollen* behandelt Abschnitt 2.3.1.

2.2.6 Multiple Access - Verwaltung mehrfacher Zugriffe

Ein *CMS* vereint ein Team in der gemeinsamen Arbeit an einer Web-Site. Die Möglichkeit gleichzeitiger Zugriffe auf ein und denselben Content kann so nicht ausgeschlossen werden. Gleichzeitige Lese-Zugriffe auf Inhalte führen zu keinen Problemen, Schreib-Zugriffe dürfen aber nicht überlappend erfolgen, um die Konsistenz der Daten zu behalten. Eine wichtige Funktion eines *CMS* ist also die Unterstützung von *geschütztem Editieren*.
Geschütztes Editieren bedeutet die Sperrung der zu bearbeitenden Assets. Wenn ein Mitarbeiter ein Asset auswählt, wird es ihm sozusagen ausgeliehen und für andere gesperrt. Diesen Schritt nennt man *Check-Out*. Solange der Mitarbeiter am Asset arbeitet, kann es von anderen nicht editiert werden. Nach Beendigung der Arbeit gibt er das Asset zurück, dies nennt man *Check-In*.

2.2.7 Staging

Die Entwicklung von großen Web-Auftritten ist eine koordinierte Zusammenarbeit von Entwicklern und Designern. Die Web-Seiten müssen ständig geändert bzw. weiterentwickelt werden, um den *Content* aktuell halten zu können. Die Verwaltung der bereits veröffentlichten Inhalte und die Entwicklung der neuen Dienste werden parallel ausgeführt.
Der Grundgedanke von *Staging* ist *die Trennung der veröffentlichten und der in Entwicklung befindlichen Inhalte und Dienste*. Die neuentwickelten Seiten, Programme, etc. sind zunächst nur für die Entwickler sichtbar, erst nach einer entsprechenden Genehmigung dürfen sie auch „live“ im Internet erscheinen. So kann es vermieden werden, daß halbfertige, noch nicht getestete Programme Störungen im „live-Bereich” verursachen.

Staging wird in der Praxis durch die Replikation von Inhalten und die vollständige Trennung der Entwicklungs- bzw. Live-Bereiche realisiert. In einem *WCMS* bedeutet dies den Einsatz mehrerer - meistens zweier - Web-Server, die alle auf ihre eigenen, separierten Datenbestände zugreifen.
Der *Entwicklungs-Server* , der sich meistens im Intranet der Entwickler befindet, verwaltet die in Bearbeitung befindlichen Inhalte. Jegliche Veränderungen oder Neuentwicklungen, die die Entwickler der Web-Site vornehmen, sind zunächst nur auf dem Entwicklungs-Server zu sehen.
Der *Live Web-Server* ist dagegen extern, d.h. für das ganze Internet erreichbar. Er verwaltet nur getestete, genehmigte Web-Seiten und Inhalte, die theoretisch keine Fehler enthalten und durch entsprechende Sicherheitsmechanismen geschützt sind.
Die Veröffentlichung neuer Inhalte und Dienste kann erst dann erfolgen, wenn sie genehmigt wurden. Die fertigen Inhalte werden vom *Entwicklungs-Server* auf den *Live Web-Server* kopiert und für alle Internet-Nutzer freigegeben.

2.2.8 Caching

Die Trennung von Inhalt und Layout - siehe 2.2.1 - erfordert die Verwendung dynamisch erstellter Web-Seiten. Das bedeutet, daß die zum Browser gesendeten *WWW*-Seiten bei jedem Abruf (*Request*) neu generiert werden müssen. Der Web-Server ruft den *TemplateProcessor* jedes Mal auf, der die gerade gewünschte Seite anhand der Layout-Vorgaben und der aktuellen Inhalte zur Laufzeit zusammenstellt.
Ein bedeutender Nachteil dieses Mechanismus ist, daß die Generierung der Seiten je nach ihrer Komplexität unterschiedlich zeitaufwendig sein kann. Die für die dynamische Erstellung einer Seite benötigte Zeit erhöht natürlich die Antwortzeit des Web-Servers. Bei Web-Sites mit einer Vielzahl von Besuchern kann es zu einer Überbelastung des Servers und dadurch zu sehr hohen durchschnittlichen Antwortzeiten kommen.
Caching in einem *CMS* bedeutet die vorübergehende statische Speicherung bestimmter Web-Seiten auf dem Web-Server. Diese Seiten werden nach ihrer Erstellung als statische Seiten im Cache abgelegt. Bei dem Abruf einer „gecachten” Seite sendet der Server die schon fertige Seite an den Browser zurück, ohne den *TemplateProcessor* wieder aufrufen zu müssen.
Da durch *Caching* die Dynamik der Web-Seiten wesentlich eingeschränkt wird, ist die Veröffentlichung von „gecachten” Seiten natürlich nur begrenzt möglich. Sobald die Inhalte,

die hinter einer „gecachten" Seite stehen, verändert werden, muß die Seite aktualisiert und wieder zusammengestellt werden. Es gibt jedoch einige typische Situationen, wo die Verwendung von „gecachten" Seiten durchaus sinnvoll ist:

- Seiten, die *ausschließlich* statische Informationen enthalten und dadurch nicht zeitvariant sind, sollen auf jeden Fall „gecacht" werden.
- Viele Seiten veröffentlichen Informationen, die *nur in bestimmten Zeitabständen* - einmal pro Tag, einmal pro Stunde, etc. - verändert werden. Es ist empfehlenswert, diese für diese Zeitperioden zu „cachen" und nur in den gegebenen Zeitabständen zu aktualisieren. So kann die Belastung des Web-Servers reduziert werden, ohne daß die Aktualität der veröffentlichten Inhalte fraglich wird.
- Es ist auch möglich, nur bestimmte Assets auf einer Web-Seite zu „cachen", andere dagegen immer neu zu generieren. In diesem Fall wird die gesamte Seite in sog. *Fragmente* aufgeteilt, die alle unterschiedlich behandelt werden. Bei dem Abruf der Seite werden ihre „gecachten" Fragmente aus dem statischen Speicher geholt, die anderen werden dagegen wieder dynamisch zusammengestellt. Dieses Verfahren setzt natürlich die Definition von Fragmenten bzw. ihre entsprechende logische Beschreibung mittels *Meta-Daten* voraus.

2.2.9 Zusammenfassung

In diesem Abschnitt wurden die wichtigsten Merkmale eines *WCMS* zusammengefaßt. Es wurde auf wesentliche *CM*-Funktionen kurz eingegangen, um einen Einblick in die allgemeine Problematik von *Content Management* zu gewähren. In den folgenden werden aktuelle Forschungsaktivitäten, Standards und Trends auf dem Gebiet *Content Management* dargelegt.

2.3 Forschungen, Standards, Trends im *Content Management*

Content Management ist eine der neuesten Disziplinen in der Informatik. Das bedeutet einerseits, daß es sich sehr rasant entwickelt, andererseits fehlt es noch an Standards und allgemeinen theoretischen Ansätzen, die in der Welt der *CM*-Systeme als Wegweiser dienen könnten.
Content Management ist außerdem sehr eng verbunden mit anderen Teilgebieten der Informatik und der Mathematik, seien es die Kryptographie, die Netzwerk- bzw. Informationstheorie oder die Multimediatechnik. *CM* stützt sich auf die Forschungsergebnisse dieser Wissenschaften, des weiteren stellt es auch neue Fragen an die Forscher dieser Fachgebiete. In diesem Abschnitt wird versucht, anhand ausgewählter Beispiele in die aktuellen Forschungsrichtungen Einblick zu gewähren. Zunächst wird ausführlicher auf das *Workflowmanagement* und die Unterstützung von *Teamarbeit* eingegangen. Anschließend werden einige andere interessante Ergebnisse kurz erwähnt.

2.3.1 Workflow-Management in CM-Systemen

Die Workflow-Komponente eines *CMS* ist für die Koordination und die Synchronisation der Tätigkeiten der einzelnen Arbeitsgruppen zuständig. In diesem Abschnitt werden die *Workflowmechanismen* in *CM*-Systemen diskutiert.

Workflowmanagement - Definitionen

Definition 2.3.1 *Ein* Workflow *stellt einen technisch umfassend unterstützten Arbeitsablauf dar, der ausgehend von einem auslösenden Ereignis entlang einer definierten Kette von Teilschritten bis zu einem definierten Arbeistergebnis führt, wobei der Grad der Vervollständigung des Arbeitsergebnisses mit jedem einzelnen Arbeitsschritt zunimmt. [Gie98]*

Definition 2.3.2 Workflow-Management *ist für die Definition, Verwaltung und Ausführung von Workflows zuständig.[Hol95]*

Definition 2.3.3 *Unter* Content-Workflow *versteht man die Automatisierung der Vorgänge der Aufbereitung, Erstellung, Publikation und Verwaltung der Inhalte in einem* CMS.

Aufgaben von Workflowmanagement

1. **Definition und Management von Rollen:**

   Das *CMS* ordnet den Mitarbeitern verschiedene *Rollen* zu, d.h. sie haben unterschiedliche Aufgaben und Verantwortungen. Die Mitarbeiter haben nur zu den für sie relevanten Assets und Werkzeugen Zugang, sie sehen verschiedene *Views* der Inhalte - siehe Abbildung 2.5. Das System überprüft die Zugangsrechte der angemeldeten Nutzer und läßt diese nur auf die für sie erreichbaren Inhaltsbestandteile zugreifen. So hat z.B. ein Nachrichten-Redakteur nicht die Möglichkeit, das Layout einer Seite zu verändern oder die im Hintergrund der Web-Site laufenden dynamischen Programme zu editieren.

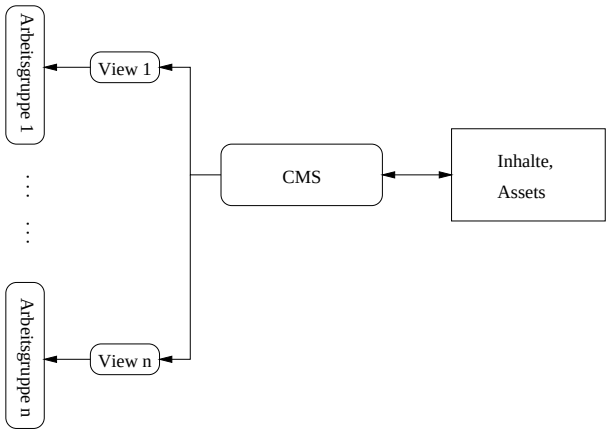

Abbildung 2.5: Verschiedene Views der Assets

2. **Synchronisation der Arbeitsabläufe:**

   Da die veröffentlichten Inhalte praktisch von allen Mitgliedern de Arbeitsgruppen bearbeitet werden müssen, ist eine festgelegte Synchronisation ihrer Arbeit notwendig. Die einzelnen Arbeitsbereiche sind voneinander stark abhängig, erst wenn der eine seine Arbeit ausgeführt hat, kann der andere mit seinen Tätigkeiten anfangen.

   Ein *CMS* ermöglicht die Definition von Arbeitsabläufen, von sogenannten *Workflows*. Es definiert eine Anzahl von Rollen mit den entsprechenden Zugangsrechten und Verantwortungen. Die Mitarbeiter der einzelnen Arbeitsbereiche erhalten die zu bearbeitenden Unterlagen und die Instruktionen vom System. Erst wenn sie ihre Arbeit ausgeführt haben, werden die Mitglieder der weiteren Gruppen vom *CMS* automatisch benachrichtigt und mit den entsprechenden, zu bearbeitenden Inhalten versehen.

3. **Benachrichtigungsmechanismen:** Die Synchronisation der Arbeitsschritte benötigt die Implementation eines Benachrichtigungssystems, die die Kommuniaktion zwischen den Mitarbeitern der einzelnen Rollen ermöglicht.

   Das *CMS* stellt eine Anzahl vordefinierter Nachrichten zur Verfügung, die die Mitarbeiter versenden können, um einander über den jeweiligen Stand ihrer Arbeit zu benachrichtigen, um Rückkoppelung zu bitten, etc. Außerdem besteht meistens auch die Möglichkeit, selbstdefinierte Nachrichten untereinander zu versenden.

   Diese Nachrichten können mit verschiedenen Methoden verwirklicht werden. Einige Systeme verwenden systeminterne Meldungen, andere dagegen basieren auf externen Lösungen, z.B. auf dem Mail-System. Eine andere Klassifizierung ist die Unterscheidung zwischen *synchronen* (z.B. Pop-Up-Fenster, Tonsignale) und *asynchronen* Meldungen (z.B. E-Mail).

4. **Freigabemechanismen:** Eine der wichtigsten Aufgaben vom Workflowmanagement ist die Implementation von verschiedenen Freigabemechanismen.

   Ziel jedes *CM*-Systems ist die Veröffentlichung von formatierten Inhalten. Diese werden nach ihrer Erstellung überprüft, und wenn sie den Anforderungen der Projektleiter entsprechen, können sie freigeschaltet und publiziert werden. Die Freischaltung ist meistens die Verantwortung von Chefredakteuren und kann nach verschiedenen Prinzipien erfolgen. Mögliche Freigabemechanismen sind:

Freigabemechanismen in CM-Systemen

- Das **Vier-Augen-Prinzip** ist eine einfach implementierbare und deshalb oft eingesetzte Methode für die Genehmigung und Freischaltung von Inhalten. Es geht um die Kommunikation von zwei Personen - *vier Augen* -, z.B. dem Redakteur und dem Chefredakteur.

  Folgende Liste und Abbildung 2.6 auf Seite 21 veranschaulichen die Theorie des *Vier-Augen-Prinzips*.

  1. Der Redaktuer erstellt die zu publizierende Präsentation und bittet um Genehmigung des Chefredakteurs per „Knopfdruck”.

2. Der Chefredakteur überprüft die zu publizierenden Inhalte
   (a) Ist die Präsentation zufriedenstellend, so kann sie freigeschaltet und im Internet veröffentlicht werden. Dies erfolgt wieder per „Knopfdruck".
   (b) Ist sie nicht zufriedenstellend, so wird sie dem Redakteur zurückgeschickt und von diesem revidiert.
3. Der Redakteur bekommt eine Rückmeldung über die Freischaltung oder wird zur Wiederbearbeitung der Präsentation aufgefordert.

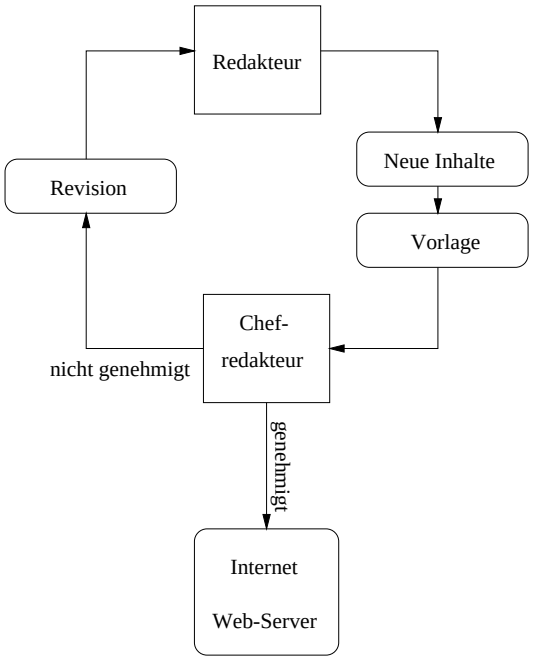

Abbildung 2.6: Das Vier-Augen-Prinzip

- **Erweiterungen vom Vier-Augen-Prinzip:**

  Das *Vier-Augen-Prinzip* ist in der Praxis häufig zu einfach. Bei der Zusammenarbeit vieler Gruppen und Entwickler sind kompliziertere Mechanismen erforderlich. Aus diesem Grunde existieren einige Erweiterungen dieses Prinzips.

  Das *X-Augen-Prinzip* Hier wird die Präsentation nicht von einem Chefredakteur, sondern von einer Gruppe von Personen genehmigt. Im strengsten Falle kann ein Inhalt erst dann veröffentlicht werden, wenn alle ihr Einverständnis gegeben haben. Eine andere Möglichkeit ist *Voting*, wobei die Freischaltung nach einem bestimmten Prozentsatz an positiven Bestätigungen erfolgen kann.

  Verschachteltes *4-Augen-Prinzip* Diese Erweiterung bedeutet die mehrmalige, rekursive Anwendung des *Vier-Augen-Prinzips*. Wie schon erwähnt, werden die zu veröffentlichenden Seiten meistens in verschiedenen Arbeitsphasen - von verschiedenen Arbeitsgruppen - erstellt. Es kann am Ende jeder Erstellungsphase

eine Genehmigung nach dem *Vier-Augen-Prinzip* erfolgen. So kann ein „Arbeitsergebnis" eine gegebene Phase erst dann verlassen, falls es in dem Stadium für funktionstüchtig erklärt wird.

Hybrides Verfahren Die beiden Erweiterungen können natürlich auch kombiniert angewandt werden – *verschachteltes X-Augen-Prinzip*.

Verschiedene Workflow-Modelle in *CM*-Systemen

Die veschiedenen *CM*-Systeme bieten je nach ihrer Komplexität unterschiedliche Workflowmodelle. Diese können grundsätzlich in drei Gruppen aufgeteilt werden.

- Die einfachsten sind die sog. **manuelle**n **Workflow-Modelle**. Hier findet kein Automatismus statt, die Definition von Rollen, sowie die Synchronisation der Arbeitsprozesse liegt vollständig in der Verantwortung der Mitarbeiter.
- Viele *CMS* verfügen über vordefinierte, **statische Workflow Mechanismen**. Hier wird für die Erstellung und die Genehmigung von Inhalten entweder das *Vier-Augen-Prinzip* oder eine seiner Weiterentwicklungen implementiert. Diese Methode ist zwar ein wesentlicher Fortschritt gegenüber dem manuellen Verfahren, die Nutzer des *CMS* haben jedoch nicht die Möglichkeit eigene Prozesse zu definieren, die an die individuellen Aufgaben angepaßt sind. Der Vorteil der statischen Methoden liegt in ihrer Übersichtlichkeit, und daß sie in vielen Fällen wirklich ausreichend sind.
- Moderne *CM*-Systeme bieten die Definition **selbstdefinierte**r **Workflows**. Sie erlauben die Modellierung, die Implementierung und die Überwachung komplexer Arbeitsabläufe. In diesen Fällen enthält das *CMS* ein eingebautes *Workflow Management System (WMS)*.

Phasen des Workflowmanagements

Die wichtigsten Phasen der Implementierung von Workflows sind:

1. **Analyse, Modellierung und Definition**: In dieser Phase werden die Arbeitsabläufe und -strukturen identifiziert, die Teilnehmer der einzelnen Arbeitsschritte festgelegt und die Vorgänge modelliert. Die einzelnen Prozesse werden mit einem sog. *Workflow-Definiton-Tool* definiert.

   Workflow-Definition-Tools werden von den verschiedenen Software-Herstellern unterschiedlich implementiert. Die *Workflow Management Coalition* [Hol95] hat deswegen ein *Meta-Modell* zur Definition von Prozessen in *Workflows* vorgestellt, das die Interoperabilität der verschiedenen Systeme erleichtern kann. Das sog. *Basic Process Definition Meta-Model* wird in der Abbildung 2.7 auf Seite 23 dargestellt.

   Die *Definition von einem Workflow* (Workflow Type Definition) besteht aus einer bestimmten Anzahl von *Aktivitäten* (Activity), die von den Teilnehmern mit unterschiedlichen *Rollen* (Role) ausgeführt werden. Bei der Ausführung von *Aktivitäten* werden

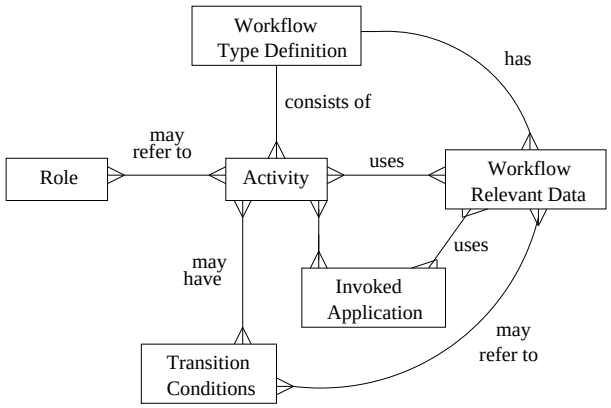

Abbildung 2.7: Basic Process Definition Meta-Model[Hol95]

Anwendungen (Invoked Application) aufgerufen. Die Übergänge zwischen Aktivitäten sind mit *Übergangskriterien* (Transition Condition) verbunden, des weiteren wird der ganze *Workflow* von *relevanten Workflow-Informationen* (Workflow Related Data) gesteuert(, siehe [Hol95]).

2. **Implementierung**: Hier werden die definierten Vorgänge mit den eingebauten Workflow-Werkzeugen in die Praxis übersetzt. Für die Implementierung gibt es zwei wichtige Modelle, das *Push-* bzw. das *Pull-Modell*.

   Definition 2.3.4 *Bei dem* Push-Modell *werden die zu bearbeitenden Inhaltskomponenten immer von dem jeweiligen Sender an den Empfänger weitergereicht. Der momentane Bearbeiter leitet die Assets an den nächsten Bearbeiter in der definierten Reihenfolge weiter.*

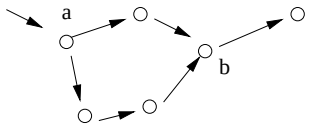

Abbildung 2.8: Das Push-Modell

Bei dieser Lösung kann es zu Versionsproblemen kommen, wenn bei paralleller Bearbeitung die Arbeitsschritte und Informationen unterschiedliche Wege durchlaufen. Das Beispiel in der Abbildung 2.8 zeigt, daß die Inhalte vom Punkt a auf zwei verschiedenen Routen zum Punkt b weitergeleitet werden, d.h. die Assets sollen verdoppelt werden. Zwischen den Punkten a und b ist es also nicht möglich, den aktuellen Stand der Inhalte eindeutig zu bestimmen.

Definition 2.3.5 *Beim* Pull-Modell *müssen die jeweiligen Empfänger die Inhalte selbst aus einer gemeinsamen Datenbank abrufen. Die Arbeit wird von einem zentralen Verwaltungssystem* synchron *gesteuert, das die betroffenen Arbeitsgruppen immer rechtzeitig informiert und die zu bearbeitenden Assets für diese zur Verfügung stellt (,siehe Abbildung 2.9 auf Seite 24).*

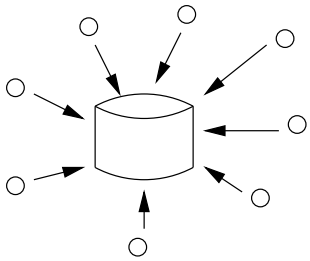

Abbildung 2.9: Das Pull-Modell

Ein wesentlicher Vorteil dieser Methode ist, daß die Assets zentral verwaltet werden. Die Handhabung eventueller Versions- bzw. Konsistenzprobleme ist damit einfacher.

3. **Monitoring**: In dieser Phase werden die Workflows kontrolliert und auf geeignete Weise visuell dargestellt. Die gewonnenen Informationn werden in der *Redesign-Phase* benutzt.

4. Die letzte Phase ist das **Redesign**. Bestehende Workflow-Modelle werden optimalisiert bzw. an sich ändernde Abläufe angepaßt.

Werkzeuge zur Realisierung von *WM*-Funktionen in *Content Management Systemen*

Ein *WMS* muß verschiedene Werkzeuge bzw. Funktionen für die Automatisierung der Arbeitsabläufe bereitstellen. Die heute in *Content Management Systemen* verwendeten sind [BZTZ00]:

- **Benachrichtigungen:** Das *CMS* kann Nachrichten generieren, wenn eine Änderung eintritt oder eine Aktion ausgelöst wurde. Diese Nachrichten sind meistens *E-Mails* oder systeminterne Anzeigen, z.B. Texte in *Pop-Up-Fenster*n. Da *E-Mail* von allen Systemen unterstützt wird, ist die Implementierung dieses Tools einfach.

- **Statusanzeigen und Filter** werden genutzt, um Zustände zu überprüfen. Diese können auch dann verwendet werden, wenn keine Aktionen ausgelöst wurden und beruhen auf den mit den einzelnen Objekten verknüpften Meta-Informationen. Mit *Statusanzeigen* kann man z.B. feststellen, wann ein Inhalt das letzte Mal bearbeitet wurde. *Filter* erweitern die Statusanzeigen um intelligente Verknüpfungen. Sie ermöglichen, daß die Objekte anhand der Statusinformationen gefiltert werden können.

- **Aufgabenlisten:** Mit dem Einsatz sogenannter *To-Do-Listen* ist es möglich, die benötigten Arbeitsschritte im vorhinein zu erfassen. Aufgabenlisten sind personalisiert und zeigen für jeden Mitarbeiter die nächsten zu erfüllenden Aufgaben an.

- Die **Protokollierung** der Arbeitsschritte dient dazu, daß die Aktivitäten der einzelnen Rollen nachvollgezogen werden können. Das *CMS* „schreibt" die Nutzeraktivitäten in sog. *Log-Dateien*, die später verarbeitet werden, um z.B. Statistiken erstellen zu können.

- **Previews** dienen dazu, daß Veränderungen an Objekten sofort betrachtet werden können. Sie ermöglichen, daß Entwickler, die neue Web-Seiten programmieren, ihre Programme testen und auf Korrektheit überprüfen können, ohne diese freigeschaltet zu haben.

- **Vertreterregeln:** Es kann vorkommen, daß Mitarbeiter wegen Abwesenheit ihre Aufgaben nicht erfüllen können. Einige modene *CMS* ermöglichen es, daß für alle beteiligten Nutzer Stellvertreter definiert werden, die im Notfall benachrichtigt und eingesetzt werden können.

2.3.2 Groupware und Group Awareness

Die Aufgabe des *Workflowmanagements* ist die Definition und die zeitliche Synchronisation der Arbeitsvorgänge bei der Erstellung und Pflege von Inhalten. Es beschreibt den Ablauf der verschiedenen den Inhalt nutzenden bzw. verändernden Arbeitsschritte.
Es ist auch sehr oft der Fall, daß gewisse Inhalte von verschiedenen Teilnehmern mit unterschiedlichen Rollen gleichzeitig benutzt und bearbeitet werden. Das *CMS* muß in diesem Fall dafür sorgen, daß die gemeinsame Bearbeitung der Inhalte nicht zu Inkonsistenzen führt bzw. daß die Mitarbeiter über die Aktionen der anderen benachrichtigt werden. D.h. das *CMS* muß auch sog. *Groupware*-Funktionen erfüllen.

Definition 2.3.6 Groupware *ist Mehrbenutzer-Software, die zur Unterstützung von kooperativer Arbeit entworfen und genutzt wird und die es erlaubt, Informationen und (sonstiges) Material auf elektronischem Wege zwischen den Mitgliedern einer Gruppe koordiniert auszutauschen oder gemeinsame Materialien in gemeinsamen Speichern koordiniert zu bearbeiten. [Obe91]*

Definition 2.3.7 *Unter* Group Awareness *(Gruppenbewußtsein) versteht man das Wissen darüber, was in der Gruppe gerade los ist bzw. was in der Vergangenheit in der Gruppe gemacht wurde.*

(In [GGR96] werden vier Aspekte von *Group Awareness* behandelt: *Social-*, *Informal-*, *Group-Structural-* and *Workspace*-Awareness.)
Eine der Aufgaben von *CM*-Systemen ist die Untersüztung vom *Gruppenbewußtsein* der Teilnehmer. Diese müssen erfahren, welche Tätigketen von ihren Kollegen ausgeführt werden. Zu diesem Zweck werden ebenfalls *Statusanzeigen*, *Pop-Up-Fenster*, *Tonsignale*, *E-Mail*-Nachrichten etc. verwendet. Weitere Informationen zu *Workspaces* und *Group Awareness* sind zu finden in [GGR96] und [Lor97].

2.3.3 Personalisierung mit dem *Universal Content Service*

Im Abschnitt 2.2.4 wurde ein wichtiger Aspekt der Personalisierung vorgestellt: Der Inhaltsanbieter soll die veröffentlichten Inhalte in einem, an den jeweiligen Nutzer angepaßten, Format darstellen. Die Besucher einer Web-Site benutzen verschiedene Paradigmen und Technologien um auf die Inhalte zuzugreifen, abhängig davon ob sie zu Hause, im Büro oder

unterwegs sind. Nutzer mit einem mobilen Telefon brauchen andere Präsentationen als andere, die vor einem Farbbildschirm sitzen. Diese Vielfalt an Schnittstellen ist ein Problem für die *Content-Provider*, sie müssen veschiedene Versionen der Inhalte in verschiedenen Formaten herstellen.
Katherine Curtis und Oliver Draper definieren in ihrer Arbeit [CD99] den sog. *Universal Content Service* (UCS) zur Personalisierung von *Content*. Das *UCS* versieht die Inhalte mit *Meta*-Daten und implementiert die *Universal Content Engine*, die die Inhalte anhand der Meta-Daten in einem für den Nutzer geeigneten Format präsentiert. In ihrem Artikel stellen sie auch eine mögliche Implementation des *UCS* vor. Diese wird in der Abbildung 2.10 gezeigt.

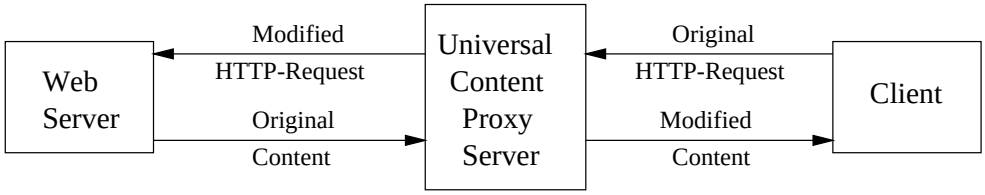

Abbildung 2.10: Universal Content Service [CD99]

Die *Universal Content Engine* wird in Form eines *Proxy-Server*s zwischen dem Browser und dem Web-Server realisiert. Diese schickt die Anfragen vom Browser modifiziert an den Web-Server weiter. Der Web-Server sendet die gefragten Inhalte mit den Meta-Daten zunächst zum Proxy-Server zurück. Dieser konvertiert die Inhalte in eine Form, die für den Nutzer geeignet ist und leitet sie an ihn weiter.
Die Autoren speichern die Inhalte auf dem Web-Server in *XML*-Format. Neben den herkömmlichen *HTML*-Tags können sie drei weitere Tags verwenden: <content>, <alternative> and <original>. <original> enthält das ursprüngliche (default) Format der Inhalte, <alternative> beschreibt seine weiteren Versionen. Das folgende Beispiel beschreibt eine Bedienungs-Anleitung, die abhängig von der dem Nutzer zur Verfügung stehenden Bandbreite entweder als Video-Sequenz oder als Text mit Bild präsentiert wird.

```
<content priority=1>
  <original bandwidthRating=4>
       For assembly, follow <a href="demo.mpg">video</a>
  </original>
  <alternative bandwidthRating=2>
       1. Assemble frame from 4 flat pieces
       2. Screw shelves together
       Final product should resemble <img src="chair.gif">
  </alternative>
</content>
```

2.3.4 iWebDB

iWebDB - The Integrated Web Database Framework- ist ein integriertes Web-Datenbank- bzw. *Content Management System*, das auf einem *ORDBMS* (object-oriented relational database system) basiert und an der Universität Kaiserslautern entwickelt wurde. Eine Beschreibung ist zu finden in [Loe] und [LN99], hier werden der Aufbau und die wichtigsten Komponenten diskutiert.
iWebDB ist ein *CMS*, das ein objektorientiertes Datenbanksystem verwendet. Objektorientierte Datenbanksysteme haben den großen Vorteil, daß sie alle Funktionalitäten von Relationalen Datenbanksystemen (RDBMS) besitzen, daneben aber auch erweiterbar sind. Unter dieser Erweiterungsfähigkeit versteht man die *benutzerdefinizierten Datentypen* (UDT) , die *benutzerdefinierten Funktionen* (UDF) und die *benutzerdefinierten Indexstrukturen* . In *CM*-Systemen, die die verschiedensten Inhalte verwalten und indizieren sollen, ist die Definition zusammengesetzter, eigener Datentypen sehr vorteilhaft.

Komponenten von *iWebDB*

Die Komponenten von iWebDB werden in der Abbildung 2.11 dargestellt.

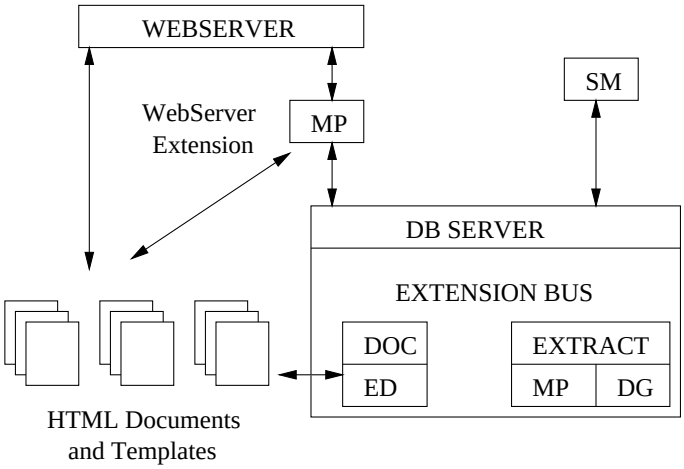

Abbildung 2.11: iWebDB-Komponenten[LN99]

- Das Modul *iWebDB/Doc* stellt die Grundfunktionalität zur DB-internen Dokumentenverwaltung zur Verfügung. Hier werden *UDT*s und Analysefunktionen in Form von *UDF*s bereitgestellt.
- *iWebDB/ED* (external data) ermöglicht den Zugriff auf externe Daten- bzw. Dateiquellen über die *SQL*-Schnittstelle.
- *iWebDB/Extractor* ergänzt *iWebDB/Doc* und *iWebDB/ED* um Analyse- und Suchfunktionen. Des weiteren ermöglicht es die Indizierung der Inhalte für Suchmaschinen.

- *iWebDB/MP* (MacroProcessor) ist eine Schnittstelle, die den Zugriff auf Datenbanken aus Web-Seiten ermöglicht. Dieses Modul bietet einen MacroProcessor, der spezielle *Tags* in Web-Dokumente einfügen kann.
- *iWebDB/DG* (DocumentGenerator) dient zum automatisierten Erstellen und Updaten von Web-Dokumenten. Es verwendet sog. *Trigger*, die weiter unten beschrieben werden.
- *iWebDB/SM* (Site Manager) ist das grafische Werkzeug zur Administration des Systems.

Document Generation mit *iWebDB/DG*

Content Management Systeme verwenden in erster Linie dynamisch generierte Seiten, die nach jedem Abruf „on-the-fly" erzeugt werden - siehe 2.2. Da diese Methode lange Antwortzeiten verursachen kann, arbeiten viele Web-Sites nach einem anderen Prinzip: sie aktualisieren ihre Inhalte in gegebenen Zeitabständen. Dieses zweite Verfahren ist weniger ressourcenintensiv, die Aktualität der Inhalte ist aber nicht immer garantiert.
iWebDB/DG verwendet eine dritte Methode, es arbeitet mit *DB-Triggers*. Diese bewirken, daß bei der Modifizierung von Datenbanktabellen - (*INSERT-, DELETE- bzw. UPDATE-Kommandos*) - alle betroffenen Web-Seiten ohne Nutzerinteraktion automatisch neu generiert werden. Trigger werden für die einzelnen Datenbanktabellen definiert und können benutzerdefinierte Funktionen aufrufen, die die Generierung der Seiten durchführen. Folgender Quell-Code zeigt einen *Trigger*, der beim Einfügen neuer Daten in die Tabelle *news* aktiviert wird[Loe].

```
CREATE TRIGGER docGenerate
AFTER INSERT ON news REFERENCING NEW AS n
(EXECUTE PROCEDURE DocumentGenerator
('INSERT','AFTER','news',n.id));
```

Nachdem der *Document Generator* aufgerufen wird, such er die zu generierenden Seiten, erstellt sie wieder und speichert das Ergebnis.
Das *DG*-Modul wurde als eine benutzerdefinierte Funktion in das *DBMS* integriert. Dies erfordert, daß das verwendete Datenbanksystem erweiterbar ist. *iWebDB* wurde mit Hilfe des *Informix Dynamics Servers* implementiert.

2.3.5 Content Management in verteilten Umgebungen

Das schnelle Wachstum des Internet und der in ihm veröffentlichten Inhalte führt zu einer immer größeren Belastung von Web-Servern. Content-Anbieter, die ihre Systeme skalierbar, sicher und effektiv machen möchten, denken verstärkt an verteilte Architekturen.
Anbieter wie *Alta Vista*, *Yahoo*, *Netscape*, etc. verwenden sog. *Cluster*-Architekturen. Ihre Dienste laufen nicht mehr auf einem einzigen Rechner, sondern auf einer Gruppe verteilter Server, die eine *heterogene* Umgebung bilden. Diese Systeme sind deshalb skalierbar, weil

man in einem Cluster je nach Bedarf weitere Server hinzufügen kann; des weiteren sind sie auch weniger störanfällig, da beim Ausfall eines Servers die anderen noch immer funktionstüchtig bleiben und die Aufgaben des ausgefallenen Servers übernehmen können.
Die Verwendung verteilter Systeme hat aber auch wesentliche Nachteile. Um die in einem *Cluster* veröffentlichten Inhalte effektiv zu managen, braucht man kompliziertere Verfahren. Einige von ihnen werden in diesem Abschnitt behandelt.

NFS

Eine einfache Methode der Verwaltung von Inhalten ist ihre Speicherung in einem zentralisierten Netzwerk-Datei-System *(NFS - Network File System)*. Die einzelnen Server erreichen die Assets über das Kommunikationsnetzwerk des Clusters. Der große Vorteil dieses Verfahrens ist, daß die Inhalte einheitlich verwaltet werden können. Das System bleibt aber sehr verletzlich, Störungen im NFS blockieren den ganzen *Cluster*.

Replikation

Viele Web-Sites verwenden eine andere Methode und replizieren ihre Inhalte, so daß diese auf allen Servern gespeichert werden. Diese Lösung hat wesentliche Nachteile:

- Die einzelnen Server brauchen große Speicherkapazitäten
- Die Server im Cluster sollen möglichst *homogen* sein, da alle Dienste auf allen Servern laufen sollen.
- Die Administration der Seiten wird wesentlich erschwert, weil bei der Aktualisierung von Inhalten alle Exemplare auf allen Servern aktualisiert werden müssen. Es kann rasch zu Konsistenzproblemen kommen.

Semantisch partitionierte Systeme

Chu-Sin Yang und Mon-Yen Lou präsentieren in ihrer Arbeit [CSY00] eine Lösung, die auf der semantischen Partitionierung von Inhalten und Diensten beruht.
Der Administrator des verteilten *CMS* kann die verschiedenen Inhalte auf veschiedenen Servern ablegen. Die Partitionierung der Inhalte erfolgt entweder nach ihrem Typ (statische *HTML*-Seiten, *CGI*-Skripte, multimediale Dokumente, etc.) oder nach anderen Aspekten wie z.B. ihrer Priorität. So kann die heterogene Struktur der zur Verfügung stehenden Technik ausgenutzt werden: dynamisch generierte Seiten werden auf leistungsfähigen Rechnern plaziert, multimediale Assets sollen dagegen auf Systemen abgelegt werden, die für „real-time" Anwendungen optimiert sind. [1] Auch die verschiedenen Technologien (*Microsoft ASP*, *PHP*, *Java Servlets*, etc.) setzen verschiedene Architekturen und Plattformen voraus.
Die Autoren stellen zudem eine effiziente Management-Umgebung vor, die in folgender Abbildung skizziert wird.

[1] Diese semantische Partitionierung hat statistische Gründe. [CSY00] erwähnt, daß große Dateien (bis zu 64KB) nur 0.3% vom Content ausmachen, jedoch 53.9% der Speicherkapazität benötigen. Der Anteil der Zugriffe auf diese Dateien beträgt nur 0.1% aller Abrufe.

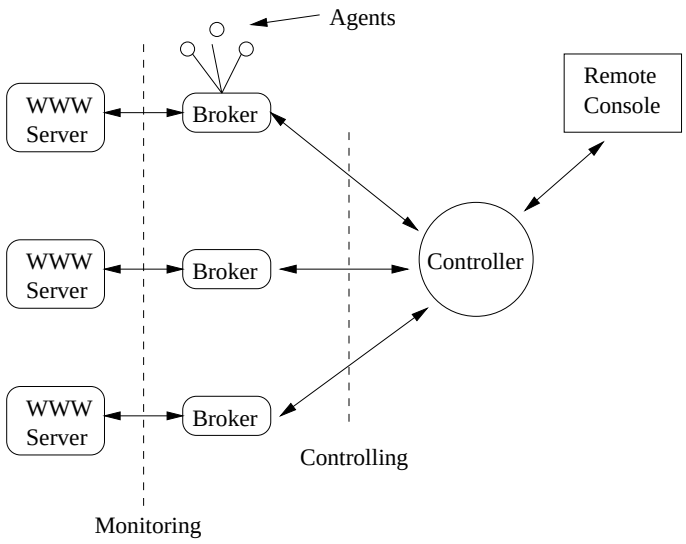

Abbildung 2.12: Partitioniertes System[CSY00]

Die Komponenten der Umgebung sind:

- Der **Broker** ist eine *Java-Applikation*, die die Schnittstelle zu den einzelnen Servern implementiert. Jeder *Broker* administriert einen Server und liefert Informationen darüber an den *Controller*.
- Die **Agent**s verwirklichen administrative Funktionen in Form von *Java*-Klassen. Die *Broker* können die *Agents* von dem *Controller* herunterladen.
- Der **Controller** verarbeitet die Abfragen des Administrators und ruft die *Broker* auf, die die entsprechenden Aufgaben mit der Hilfe der *Agents* ausführen.
- Die **Remote Console** ist die graphische Benutzerschnittstelle für den Administrator, die als *Java*-Applet verwirklicht wurde.

Diese Struktur hat mehrere Vorteile:

- Die Implementation der *Broker* und des *Controllers* in *Java* unterstützt die Anwendbarkeit heterogener Systeme.
- Da *Java* die Anwendung von herunterladbaren, ausführbaren Programmen (downloaded executable content) unterstützt, ist die Erweiterung von Broker-Programmen sehr einfach.

2.3.6 Protokolle im *CM*

In diesem Abschnitt werden zwei in *Content Management Systemen* oft eingesetzte Protokolle vorgestellt, die standardisierte Im- bzw. Exportschnittstellen für digitale Informationen bieten.

Das $Z39.50$ Protokoll ist älter und hat seinen Ursprung in der Verbindung zu bibliothekarischen Datenbankanwendungen. Mit der Entwicklung von *Content Management* findet es auch auf diesem Gebiet immer mehr Anwendung.
Das *ICE*-Protokoll ist in den letzten Jahren entstanden und zielt in erster Linie auf *Web Content Management Systeme* ab.

Z39.50 Protokoll

Z39.50 ist ein Standard zur Verbindung von Datenbankanwendungen. Er entstand 1984 und wird vom *ZIG* (Z39.50 Implementors Group) betreut. 1992 wurde er so überarbeitet, daß er mit dem *ISO*-Standard *ISO 10162/1163 SR* kompatibel ist. Obwohl der Standard als ein *OSI*-Protokoll definiert wurde, werden all seine Anwendungen fast ausschließlich als Anwendungsprotokoll über *TCP/IP* verwendet.

Z39.50 erlaubt die weltweite Suche in heterogenen Datenbanken, unabhängig von ihrem Typ und ihrer Struktur.
Die mit dem Protokoll kommunizierenden Datenbanken implementieren einen sogenannten *Z*-Server. Der *Z*-Server übersetzt die Inhalte der Datenbanken in ein standardisiertes Format, das von den *Z*-Clients verstanden wird. Zu den meisten *Z*-Servern wird auch ein *HTTP/Z39.50*-Gateway geschrieben, damit sie aus dem *WWW* erreichbar werden.
Abbildung 2.13 zeigt den Ablauf der Kommunikation.

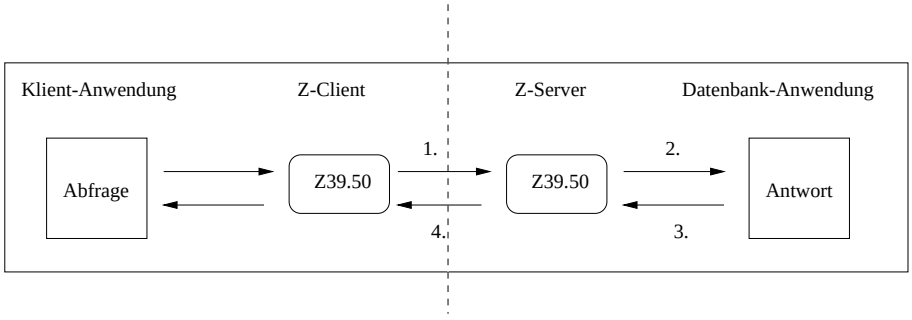

Abbildung 2.13: Kommunikation via Z39.50

1. Der *Z*-Client übersetzt die Anfrage des Nutzers in *Z39.50* und leitet sie an die Zieladresse weiter.
2. Der *Z*-Server verarbeitet die Abfrage so, daß sie auch für die Datenbankanwendung „verständlich” wird
3. Die Datenbankanwendung verarbeitet die Abfrage und liefert die Ergebnisse an den *Z*-Server zurück
4. Der *Z*-Server liefert die Ergebnisse in *Z39.50*-Format an den *Z*-Client, der Nutzer bekommt die Informationen vom *Z*-Client.

Das Protokoll bietet u.a. die folgenden Dienste:

- **Initial Service:** Die Initialisierung der Verbindung, Voraussetzung für nachfolgende Dienste.
- **Search Service:** Suchanfrage vom Klient an den Server, deren Ergebnis eine Ergebnismenge (*ResultSet*) ist
- **Present Service:** Setzt eine erfolgreiche Suchanfrage voraus und erhält aus den Suchergebnissen gezielte Informationen
- **DeleteResultSet Service:** Ermöglicht das Löschen von Elementen aus der Ergebnismenge
- **AccessControl Service:** Dienst zur Authentifizierung des Klients
- **Sort Service:** Ermöglicht das Sortieren der Ergebnismenge nach bestimmten Kriterien
- **Concurrent Operations Service:** Dieser Dienst erlaubt das Abschicken von mehreren Suchanfragen, bevor die zugehörige Suchantwort (*Search Response*) eingegangen ist.
- **Termination Service:** Wird vom Client initiiert und beendet sofort alle aktiven Operationen

Die weiteren Dienste werden ausführlicher in [Age00] beschrieben. *Z39.50* findet heute im Bereich *Content Management* immer mehr Verwendung. Ein gutes Beispiel dafür ist der *SIM Content Manegement Server* von *RMIT Multimedia Database Systems* - siehe [AMKF+00].

ICE - Information and Content Exchange

Ein wichtiges Ziel von Content-Anbietern ist es, ihre Inhalte in einer standardisierten Form ihren Kunden im Internet zur Verfügung zu stellen. Bis vor einigen Jahren waren für den Export von Daten zu Geschäftspartnern kundenspezifische Austauschschnittstellen erforderlich. Da der Bedarf an automatisiertem Austausch von Inhalten zwischen Unternehmen stark gewachsen ist, war es nötig, einen allgemeinen Standard für diesen Zweck zu entwickeln. Mit diesem Ziel haben einige namhafte Unternehmen, zusammengeschlossen zur *ICE Authoring Group* [ICE00], im Jahr 1998 das *ICE*-Protokoll entwickelt. Die Standardisierung erfolgte im Oktober 1998 durch das *W3C* .

ICE ist ein *Protokoll zum kontrollierten, automatisierten Austausch von Online-Inhalten zwischen Geschäftspartnern*. Es ist ein bidirektionales Protokoll, welches *XML*-basierende Nachrichten zwischen Systemen versendet, um die Übertragung zu steuern. Das Format der Daten spielt dabei keine Rolle, von normalem Text, *HTML* bis *XML* ist alles denkbar.

Überblick: Das *ICE*-Protokoll ermöglicht den Austausch von Inhalten zwischen Servern, wobei *XML* zur Repräsentation der Nachrichten verwendet wird. Zur Übertragung der Daten werden etablierte Protokolle wie *HTTP/SSL* und *TCP/IP* benutzt. Die beiden kommunizierenden Seiten konvertieren ihre Dokumente - unabhängig von ihrem Format - in *ICE*-Nachrichten, die auf eine standardisierte Weise transportiert und auf der Empfängerseite dekodiert werden können.

ICE folgt dem *Request - Response*-Modell. Das Protokoll definiert zwei Rollen: es gibt einen *Syndicator* (Inhalt-Anbieter) - und einen *Subscriber* (Inhalt-Abonnent) - siehe Abbildung 2.14. Der *Syndicator* greift auf ein *Content Management System* zu, extrahiert Inhalte für die Verbreitung und integriert ankommenden Content. Ein *Syndicator* kann seine Inhalte mehreren *Subscriber*n anbieten, und ein *Subscriber* kann Inhalte von mehreren *Syndicator*s erhalten.

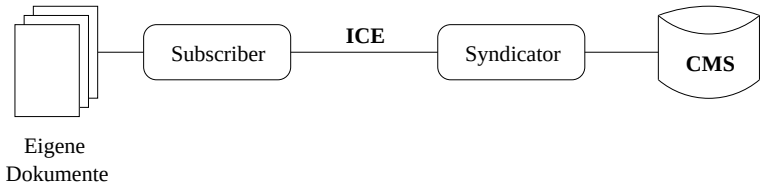

Abbildung 2.14: Übertragung von Dokumenten mit ICE

Der Datenaustausch zwischen den beteiligten Systemen erfolgt auf zwei Ebenen:

1. Zunächst wird eine **Subscription** eingerichtet. Allgemeine Informationen, wie Beginn- und Enddatum, Zeitpunkte und Frequenzen der Übertragung sowie alternative Lieferadressen werden abgestimmt.
2. Erst dann kommt es zur **Datenlieferung**, wobei Inhalte in Form von *ICE-Paketen* ausgetauscht werden.

Einrichtung von Subscriptions Der erste Schritt ist die Anfrage des *Subscribers* nach einem Katalog, in dem der *Syndicator* seine Angebote spezifiziert (*ice-get-catalog*).

Der *Subscriber* wählt Informationen aus dem vom *Syndicator* gelieferten Katalog aus und nennt seine Auswahl mit *ice-offer*. Ist der *Syndicator* weiterhin damit einverstanden, antwortet er mit einer *Subscription*. Er kann den Antrag natürlich auch ablehnen. In diesem Fall müssen genaue Parameter-Verhandlungen erfolgen, um die *Subscription* zu ermöglichen.

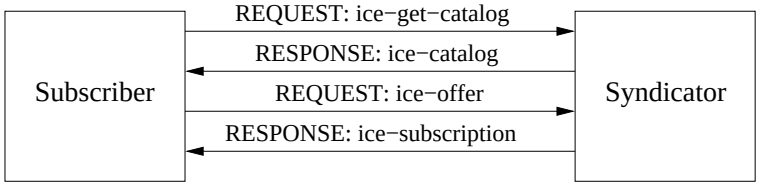

Abbildung 2.15: Einrichtung einer Subscription

Datenlieferung Nach der Einrichtung der *Subscription* kommt es zur tatsächlichn Übertragung der Daten. Abhängig davon, ob die Transaktion vom *Syndicator* oder vom *Subscriber* initiiert wird, spricht man von einem *Push*- bzw. einem *Pull*-Mechanismus. Bei jeder Übertragung werden die Daten in *Pakete* gefaßt, wobei in einem Paket mehrere

Lieferungen eingebettet sein können. Es sind auch *Update*-Vorgänge möglich, wobei ein Paket nur die seit der letzten Paketlieferung geänderten Inhalte enthält. Der *Syndicator* kann sich den Erhalt eines Pakets vom *Subscriber* bestätigen lassen.

Bezüglich der Größe der in einem Paket übertragenen Datenmengen gibt es keine Beschränkungen. Die Inhalte werden durch *ice-item*s räpresentiert, die sowohl textuelle als auch binäre Daten enthalten können. Der `CONTENT-TRANSFER-ENCODING`-Tag eines Items gibt Auskunft über seine Kodierung, der `ITEM-ID`-Tag identifiziert den Inhalt eindeutig. Binäre Inhalte werden meist mit dem *base64*-Verfahren kodiert.

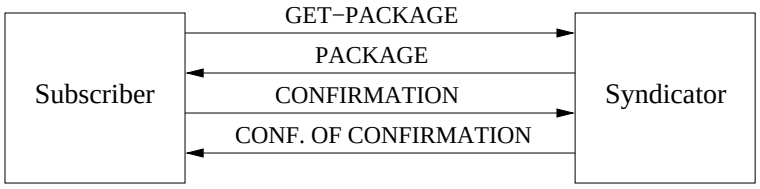

Abbildung 2.16: ICE-Pull-Mechanismus

Weitere Merkmale von ICE Neben den beschriebenen Funktionen bietet *ICE* weitere Funktionalitäten.

- Wenn ein *Subscriber* keinen ständigen Netzzugang hat oder die Übertragung unterbrochen wird, können die Pakete auf der *Syndicator*-Seite in Warteschlangen eingereiht werden.
- Der *Subscriber* kann beim *Syndicator* von einzelnen Items eine Kopie anfordern, falls diese zerstört wurden.
- Zur Unterstützung der Fehlersuche kann der *Syndicator* alle *ICE*-Aktionen protokollieren.

2.3.7 Zusammenfassung

Im Abschnitt 2.3 wurden einige Forschungsarbeiten und Trends auf dem Gebiet *Content Management* vorgestellt. Ziel war es zu zeigen, daß *CM* mit einer Vielzahl von interessanten Disziplinen der Informatik verknüpft ist.

Nach der theoretischen Analyse von *Content Management Systeme* werden nun einige Kriterien genannt, anhand deren vorhandene *CM*-Werkzeuge klassifiziert werden können.

2.4 Kriterien zur Bewertung von *Content Management Systemen*

In diesem Abschnitt werden die Kriterien zur Evaluation von *Content Management Systemen* genannt. Die im Kapitel 3 vorgestellten *CM*-Werkzeuge werden dann im Abschnitt 3.4 anhand dieser Gesichtspunkte miteinander verglichen.

In den folgenden Unterpunkten wird je ein Kriterium genannt, bzw. es werden Richtlinien zur Klassifizierung der Systeme angegeben.

Systemanforderungen an den CM-Server

- **Plattformen:** Auf welchen Betriebssystemen läuft der Server? Besteht die Möglichkeit der Portierung zu anderen Plattformen?
- **Hardwareanforderungen:** Wie hohe Anforderungen an die Hardware stellt das System? Fordert es bestimmte herstellerspezifische Hardware-Bauteile (z.B. Prozessor)? Empfohlene Mindestkonfiguration?
- **Datenhaltung:** Welche Art der Datenverwaltung wird verwirklicht? Welche Verzeichnisdienste und/oder Datenbankschnittstellen werden unterstützt?
- **Web-Server:** Ist das System an einen bestimmten Web-Server gebunden? Welche Möglichkeiten der Portierung zu anderen *HTTP*-Servern bestehen?

Systemanforderungen an den CM-Client

- **Web-basierte Schnittstelle:** Ist der Client web-basiert? Wenn nicht, welche Werkzeuge sind nötig?
- **Plug-ins:** Welche Browser-Plug-ins setzt der Client voraus?
- **Plattform:** Welche Plattformen unterstützt der Client?

Installationsaufwand

- **Aufwände:** Wie aufwendig ist die Konfiguration des *CMS*? Gibt es automatisierte Installationsroutinen? Wie ausführlich werden die Installationsschritte dokumentiert?
- **Erforderliche Kenntnisse, Schulungsaufwand:** Welche Kenntnisse sind zur Verwendung des *CMS* erforderlich? Wie schnell kann man das System erlernen? Sind spezialisierte Fachleute nötig?

User-Interface

- **Eingabeschnittstelle:** Was für eine Eingabeschnittstelle verwendet das *CMS*?
- **Views, Voransichten:** Sind verschiedene Views und/oder Voransichten unterstützt?

Werkzeuge zur Seitenerstellung

- **Dynamische Seiten:** Unterstützt das *CMS* die Erstellung dynamischer Seiten?
- **Caching:** Kann man Inhalte „cachen”? Wenn ja, welche Caching-Strategien werden ermöglicht?
- **APIs, Sprachen:** Welche Programmierschnittstellen und Werkzeuge können genutzt werden?

Importschnittstellen

- **Formate:** Welche Dokumentenformate kann man ins *CMS* importieren? Wie wird die Konversion der Inhalte verwirklicht? Welche Standards und Standardanwendungen werden unterstützt?

Versionierung

- **Versionierung, Archivierung:** Wird Archivierung unterstützt? Welche Inhalte kann man versionieren? Kann man archivierte Inhalte ex- bzw. importieren?

Workflow

- **Rollen:** Kann man Rollen definieren? Welches Rollen- bzw. Rechtesystem wird implementiert?
- **Multiple Access, Locking:** Werden mehrfache Zugriffe geregelt? Wird Locking unterstützt? Auf welche Weise? Wie werden Deadlocks vvermieden, erkannt?
- **Benachrichtigungsmechanismen:** Welche systeminterne Benachrichtigungsmechanismen gibt es? Sind diese synchron oder asynchron? Wie werden sie verwirklicht?
- **Genehmigungsinstanzen, Freigabe:** Wie werden Inhalte genehmigt? Welche Freigabemechanismen werden unterstützt? Kann man eigene Workflows definieren?

Technische Unterstützung

- **Dokumentation:** Wie gut ist das System dokumentiert? In welchem Format liegt die Dokumentation vor? *Online-Help*?
- **Support** Wie ist die technische Unterstützung? Welche zusätzlichen Kosten entstehen?

Kostenmodell

- **Kosten:** Wieviel kostet das *CMS*? Wie sehen die Kosten- bzw. Lizenzmodelle aus?
- **Laufende Kosten:** Welche laufenden Kosten entstehen bei der Verwaltung des *CMS*?

Referenzen

- **Referenzen:** Wo wird das *CMS* eingesetzt? Beispiele aus dem akademischen bzw. kommerziellen Bereich?

Kapitel 3

Content Management Systeme im Vergleich

3.1 Kommerzielle Lösungen

In diesem Abschnitt werden kommerzielle Systeme vorgestellt, deren Zielgruppe die großen Web-Sites sind und die auf dem amerikanischen bzw. europäischem Markt immer öfter eingesetzt werden.

3.1.1 Vignette Content Manager Server

Der *Content Manager Server* der Firma *Vignette* ist eines der am längsten auf dem Markt erhältlichen *CMS*. Es verwirklicht die meisten der zuvor genannten *CMS*-Funktionen, so wie *Trennung von Inhalt und Layout*, *Caching*, *Staging*, *Workflowmanagement*, etc. Die Entwickler arbeiten auf der *Client-Seite* mit einer *Java*-basierten Benutzeroberfläche, die auch mit dem Browser intensiv kommuniziert und neben den Suchefunktionen auch *Seiten-Previews* gewährt.
Als besonderes Merkmal enthält der *Content Manager Server* weitgehende Personalisierungsmöglichkeiten, auch für anonyme Benutzer. Es arbeitet mit Templates, die neben *HTML*-Code auch *Tcl*-Skripte - siehe [Wel00] -, *ASP*- oder *JSP*-Elemente enthalten und die Seiten dynamisch generieren, außerdem steht ein *WYSIWYG* Seiteneditor zur Verfügung. Der Server läuft auf *Windows NT* bzw. *Solaris* Plattformen, unterstützt die Web-Server *IIS*, *Netscape* und *Apache* und die Datenbanken *Oracle*, *Sybase* und *MS SQL Server* .
Vignette Content Management Server unterstützt weitgehend die Definition von *Workflows* und *Rollen*, sowie von *Tasks*, d.h. Arbeitsabläufen, die zeitlich synchronisiert werden können.

Komponenten vom *V-CMS*

Abbildung 3.1 auf Seite 39 zeigt die Komponentenstruktur vom *V-CMS*.

Content Management Server Der *CMS* bildet den Mittelpunkt des Systems und verwaltet jegliche Objekte, wie Templates, Projekte, Nutzer, Inhalte, Workflows etc. Er ver-

wendet die *Systemdatenbank* zur Speicherung aller relevanten Informationen. Jedes System enthält genau einen *Content Manager Server*.

Content Delivery Server (CDS) Der CDS ist für die Kommunikation mit dem *Web-Server* sowie für die Veröffentlichung der Inhalte zuständig. Er generiert die Seiten anhand der Templates dynamisch. In den meisten Szenarien werden zwei *CDS* eingerichtet, der eine für die Entwickler, der andere für den „Live-Bereich". Der *CDS* enthält zwei weiter Module: den *DocRoot Manager* für das Caching und die *ApplicationEngine*, die für die dynamische Seitengenerierung verantwortlich ist.

WebServer Module Das *WebServer Module* kommuniziert mit dem *Page Generator* und sendet die fertigen Seiten an die eigentlichen Web-Server (*Netscape* , *IIS* oder *Apache*).

Observation Management Server (OMS) Der *OMS* sammelt Informationen über die Besucher der Web-Site. Er erfüllt die meisten Aufgaben der *Personalisierung* - siehe 2.2.4 - und dient auch der Anfertigung von Statistiken. Eine Web-Site kann natürlich nur einen *OMS* enthalten

Multi-Channel Server (MCS) Der *MCS* ist ein optionaler Teil vom *V-CMS* und verwirklicht *Content-Delivery* durch *Push-Mechanismen*. Er implementiert sog. *Channels* für den Export der Inhalte an Pager, mobile Telefone, E-Mail Clients und realisiert auch einen *ICE-Syndicator* .

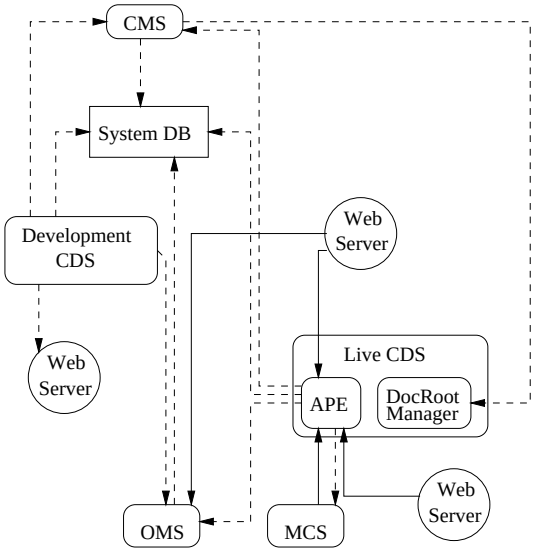

Abbildung 3.1: Komponenten vom V-CMS[Sto00]

Ein wesentlicher Nachteil des *Vignette CMS* ist die komplizierte, manchmal unübersichtliche Rechtesystem, bzw. daß keine kompletten Projekte versioniert werden können. Weitere Probleme bereitet auch der systeminterne *Tcl*-Interpreter wegen des Mangels an *Debug*-Funktionen. Die einfach handhabbare Benutzeroberfläche und die Vielzahl der Funktionen machen das System dennoch sehr attraktiv, obwohl seine Kosten ziemlich hoch sind. (Die

Version 5.6 vom *Vignette CMS* enthält drei *Application Programming Environments für* ASP, Tcl, Java Beans/JSP.)

3.1.2 Gauss VIP Content Manager

Der *VIP Content Manager* [Gau00] ist ein professionelles Managementsystem für den Aufbau und die Pflege von Web-Sites. Es ist eine der Kernkomponenten von *Gauss VIP*, der *Versatile Internet Platform*.
Da der *Gauss Content Manager* auf *Java* basiert, ist er plattformübergreifend einsetzbar und kann mit allen gängigen Web-Servern kommunizieren. Er unterstützt den Qualitätsmanagement-Standard *ISO 9000* bei den wichtigsten *CM*-Aktivitäten, wie *Assetmanagement*, *Zugriffsverwaltung*, *Protokollierung*, *Versionierung*, etc.
Das System basiert auf dem sog. *Drei-Server-Konzept*, d.h. es besteht aus drei getrennten Serverprozessen für *Pflege*, *Qualitätssicherung* und *Produktionsbetrieb* - siehe Abbildung 3.2.

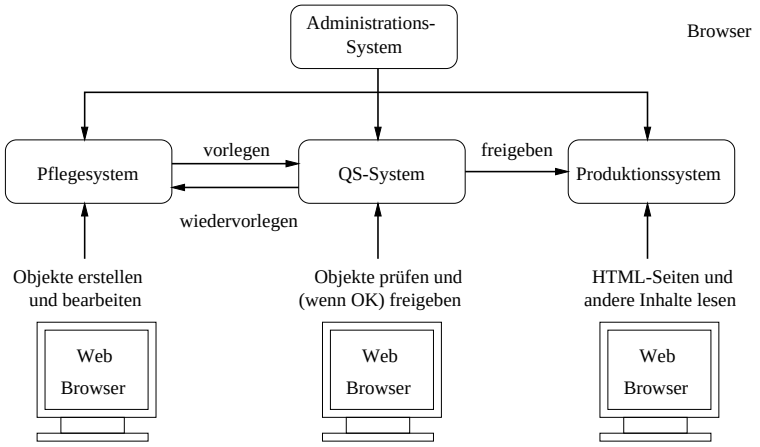

Abbildung 3.2: Das Drei-Server-Konzept[Gau00]

- Das **Pflegesystem** stellt die redaktionelle Schnittstelle dar, hier werden die Inhalte erstellt und bearbeitet. Des weiteren ermöglicht es den Import der gängigsten Dokumentenformate.
- Im **QS-System** werden die Inhalte inhaltlich und formal geprüft. Hier wird entschieden, ob das Objekt an das Pflegesystem zurückgegeben oder im Produktonssystem freigegeben wird.
- Das **Produktionssystem** stellt die freigegebenen Seiten der Web-Site für die Nutzung durch Endbenutzer bereit.

Neben diesen drei Systemen gibt es auch ein sog. **Administrationssystem**, das die administrativen Aufgaben erfüllt.

Die Inhalte werden mit *Objekten* modelliert. Die wichtigsten Objekte sind die sog. *Thema-Objekte*, die als Kombinationen von *HTML*-Seiten und Verzeichnissen vorgestellt werden können, und in erster Linie zur Stukturierung der Assets dienen. Die Zugriffsverwaltung erfolgt mit der Definitionen von Nutzern und Gruppen, über die der Zugang zu den Objekten geregelt wird.
Die dynamischen Templates werden mit der Hilfe von *JSP (Java Server Pages)* erstellt. Die Java-Technologie ermöglicht auch die Integration externer Anwendungen ins *VIP*-System.
Die Erstellung und Bearbeitung von Objekten mit dem *VIP'Content Manager* sowie deren Veröffentlichung im Produktionssystem unterliegen einem fest definierten Workflow. Die Definition von eigenen Workflows und Arbeitsabläufen ist im *Gauss VIP' Content Manager* leider nicht gestattet.
Um dynamische, personalisierte Seiten auf der *Versatile Internet Platform* herzustellen, kann der *VIP Portal Manager eingesetzt werden.*

3.1.3 NPS - Network Productivity System

Das *Network Productivity System* der deutschen Firma *Infopark* ist ein leistungsfähiges, preiswertes Redaktionssystem mit vielen *Content Management* Funktionen. Es bietet eine automatisierte, leicht handhabbare Software-Oberfläche für die Verwaltung und den Austausch der Inhalte in einem Web-Auftritt.
Eines der wichtigsten Merkmale von *NPS* ist die flexible Organisation von *Content* - siehe [Inf01]. Der *NPS-CMS-Kernel* - siehe Abbildung 3.3 auf Seite 42 - bietet eine objektorientierte Oberfläche für die Adressierung der Inhalte, die physikalisch in einer Datenbank oder im Dateisystem abgelegt werden. Er implementiert eine eigene *XML-DTD* zur Strukturierung von *Content* und ermöglicht somit die plattformunabhängige Kommunikation mit den *WWW*-Servern und den Redaktionsklienten. Die Nutzer haben des weiteren die Möglichkeit, die Inhalte mit beliebigen Meta-Attributen zu versehen, bzw. sowohl die gängigsten Dokumentenformate - *MS Office*, *Adobe PDF* -, als auch bereits existierende Web-Seiten ohne jegliche *HTML*-Kenntnisse ins System zu importieren. Einer der weiteren Vorteile von *NPS* ist die Verwaltung und Verfolgung von Hyperlinks, wodurch falsche oder nicht mehr existierende Verknüpfungen vermieden werden können.
Als Exportmechanismen werden *XML*, *XML/EDI*- und *ICE*-Schnittstellen implementiert.
Die Benutzer führen ihre Tätigkeiten standardmäßig über einen *Web-Browser* aus, die *XML*-basierte Schnittstelle ermöglicht daneben die Erweiterung des Systems mit anderen Klienten.
Ein sehr nützliches Werkzeug für das Management der Inhalte ist die *Tcl*-basierte Kommandozeilenoberfläche, die mittels eingebauter *Tcl*-Funktionen die Automatisierung vieler Aufgaben gestattet.
Die Benutzerverwaltung erfolgt entweder intern oder mit dem Einsatz des *LDAP*-Dienstes. *NPS* implementiert ein gruppenorientiertes Rechtekonzept mit frei definierbaren Benutzerattributen, diese können auch bei der Definition von *Workflows* und *Rollen* genutzt werden. *NPS* erlaubt die Erstellung selbstdefinierter Arbeitsabläufe bzw. sogenannter *To-Do-Listen* und implementiert u.a. auch das *Mehr-Augen-Prinzip*.

NPS ermöglicht in erster Linie die Publikation und den Austausch der verschiedensten Do-

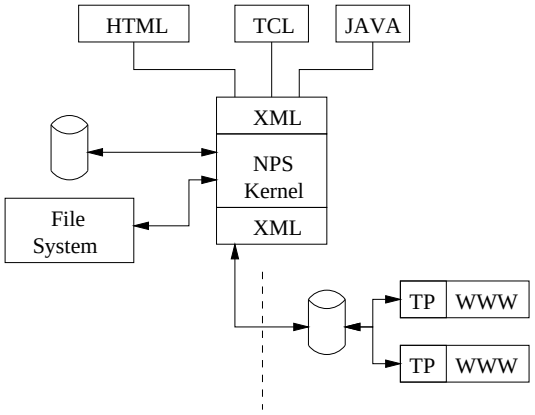

Abbildung 3.3: Network Productivity System

kumententypen. Die Programmierung von dynamischen, personalisierten Web-Seiten wird dagegen nicht so gut unterstützt, wie z.B. bei *Vignette CMS* oder *OpenCMS*. Der Autor ist der Meinung, daß *NPS* eher ein Redaktionssystem als eine Plattform für die Erstellung dynamischer Web-Auftritte mit viel Programmierlogik ist. (Für die Erstellung personalisierter Web-Auftritte bietet *Infopark* das Produkt *NPS-Portal-Manager* an.)

3.2 Open-Source Lösungen

Mit der Entwicklung des Internet und der Kommunikationsgesellschaft gewinnen *Open-Source* Software-Lösungen immer mehr an Bedeutung. Viele Forschungs- und Entwicklergruppen bieten Plattformen und Entwicklungswerkzeuge, die auch als Quell-Code frei erhältlich und nicht lizenzabhängig sind. Mit dem Wachstum des Internet werden solche Werkzeuge vor allem in der Welt des *WWW* auch im kommerziellen Bereich immer öfter verwendet.
Mit der rasanten Entwicklung von *Web Content Management* sind auch die ersten *CM*-Systeme erschienen, die gegenüber den genannten kostspieligen Lösungen auf *OpenSource*-Software basieren und zum Teil auch selbst als Quell-Code frei verfügbar sind. Diese bieten folgende Vorteile:

Niedrige Kosten: Da die meisten Komponenten dieser Systeme frei zur Verfügung stehen, bedeutet ihre Installation keine Kosten für den Anwender. Es ist jedoch nicht zu vergessen, daß neben den fixen Installationskosten auch mit laufenden Wartungskosten zu rechnen ist, die natürlich auch bei Open-Source Lösungen sehr hoch werden können.

Anpassungs- bzw. Erweiterungsmöglichkeit Ein riesiger Vorteil von *Open-Source* Software ist die Möglichkeit der Erweiterung. Da die Programme in Quell-Code vorliegen, ist ihre Weiterentwicklung bzw. Anpassung an vorhandene Systemkomponenten für den interessierten Entwickler möglich.

Die Arbeit mit *Open-Source* Lösungen hat daneben auch einige Nachteile:

Mangel an technischer Unterstützung Der Vertrieb von *Open-Source-Software* ist von der Vermarktung kommerzieller Produkte sehr unterschiedlich. *Open-Source*-Werkzeuge können kostenlos angeschafft werden, gleichzeitig fehlt es auch an Garantien und an technischer Unterstützung, die bei den kommerziellen Produkten auch zusätzlich bezahlt werden sollen.

Komplizierte Konfiguration Der Mangel an technischer Unterstützung bzw. die Tatsache, daß *Open-Source*-Lösungen heute oft auf noch weniger verbreiteten, ebenfalls freien Plattformen - z.B. *Linux* - laufen, bedeuten auch mehr Installations- und Konfigurationsaufwand und setzen den Einsatz besonders qualifizierter Fachkräfte voraus.

In diesem Abschnitt werden einige *CM*- bzw. Redaktionssysteme vorgestellt, die ausschließlich auf *OpenSource* Softwarekomponenten basieren.

3.2.1 OpenCMS

OpenCMS ist ein frei auch als Quell-Code verfügbares *CMS*, das aus dem Internet heruntergeladen werden kann. Ursprünglich wurde es von der „*mindfakt interaktive medien ag*" und der Firma „*Glo BE technology*" entwickelt, heute wird es von einer Gruppe von freiwilligen Entwicklern namens *OpenCMS Group* gepflegt.
OpenCMS basiert 100%-ig auf *Java*, d.h. es ist plattformunabhängig. Um das System zu verwenden, muß man zuvor eine *Java Virtual Machine*, einen Web-Server mit einer *Java-Servlet-Engine*, den *XML*-Parser *Xerces* und die Datenbank *MySQL* installieren. All diese Komponenten sind frei verfügbar.
Abbildung 3.4 zeigt den logischen Aufbau von *OpenCMS* [Pro01].

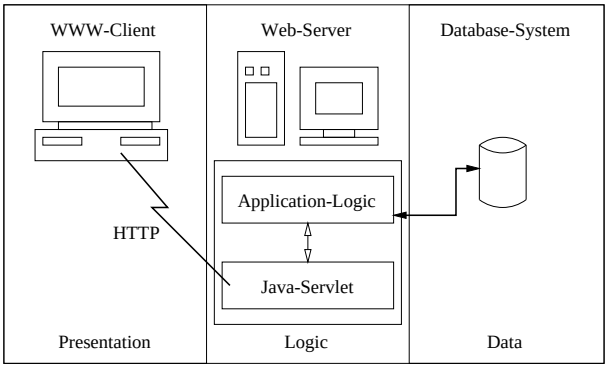

Abbildung 3.4: Aufbau von OpenCMS[KGRH00]

Die **Präsentationsschicht** von *OpenCMS* ist die Oberfläche eines Web-Browsers. Diese Benutzerschnittstelle nützt umfassend *JavaScript* und verwendet ein *Active-X-Control* als *WYSIWYG HTML*-Editor. Die **logische Schicht** wird vom Web-Server dargestellt, der *Java-Servlets* verwendet. Diese Servlets implementieren die Schnittstellen sowohl zur *Präsentations*- als auch zur **Datenschicht**.

OpenCMS verwaltet die Arbeiten in Projekten. Die Projektmitglieder können verschiedenen Rollen zugeordnet werden, gesteuert von *Projektleiter*n. Die Projektleiter sind auch für die Freigabe der Inhalte zuständig. *OpenCMS* bietet eine leicht übersichtliche und gut verwendbare Oberfläche für die Definition von einfachen Workflows und implementiert ein auf *E-Mail* basierendes Benachrichtigungssystem.
Die Erstellung von Templates, die die Web-Seiten dynamisch generieren, erfolgt in *OpenCMS* mit dem Einsatz von *XML* und *Java-Servlets*. Die Entwickler können die Seiten neben dem *WYSIWYG*-Editor auch textuell bearbeiten und die Ausführung von *Servlets* mit *OpenCMS*-spezifischen *XML*-Tags kontrollieren. Das System bietet eine Vielzahl von gut dokumentierten Funktionen und *XML*-Elementen, die die Pogrammierung der Seiten ermöglichen. Zur Erstellung von Templates siehe [Pro01].

Ein wesentlicher Nachteil von *OpenCMS* ist der Mangel an Funktionen für die Versionierung und Erstellung „gecachter" Inhalte. Da der *WYSIWYG*-Editor *Active-X*-Controlls verwendet, kann dieser nur mit *MS Internet Explorer* benutzt werden, was auch die Plattformunabhängigkeit des Systems einschränkt.

3.2.2 Midgard

Midgard ist ein wenig bekanntes, dennoch sehr leistungsfähiges *Content Management* Werkzeug, das auf der Skriptsprache *PHP*, der Datenbank *MySQL* und dem Web-Server *Apache* basiert. Es steht unter der *LGPL*-Lizenz, ist also auch als Quell-Code frei verfügbar[1]. Die Weiterentwicklung von *Midgard* wird von der Organisation *Midgard Project Ry* betreut.

Midgard unterstützt die koordinierte Zusammenarbeit an einer Web-Seite, in dem es die drei Bereiche *Logik*, *Layout* und *Content* trennt. Es implementiert Baumstrukturen für die sog. *Page*-, *Style*- und *Topic*-Elemente, die von den *Programmierern*, den *Designern* und den *Inhalts-Editoren* bearbeitet werden. Die einzelnen Elemente können über die *Midgard API* programmiert werden. Das Zugriffs- bzw. Rechtesystem verwaltet einzelne Nutzer und Nutzergruppen und ist der Benutzervewaltung von *UNIX*-Systemen sehr ähnlich.

Die meistverwendete *CM*-Funktion von *Midgard* ist die Verwaltung von *Themen (Topics)* und *Artikeln (Article)*. Diese formen eine hierarchische Baumstruktur mit einem Hauptthema und verschiedenen, verschachtelten Unterthemen, wobei die „Blätter" der Bäume die Artikel sind, die textuelle Inhalte enthalten.

Midgard besteht aus 4 wichtigen Komponenten:

- **midgard-lib** implementiert die grundsätzlichen Funktionen, die die Interpretierung der *Midgard-API*, die Verwaltung von Datenbankanbindungen, die Manipulation von Zeichenketten und das Speichermanagement ermöglichen.
- **mod_midgard** stellt ein Modul des *Apache*-Web-Servers dar. Es bearbeitet die Aufrufe und filtert bzw. verarbeitet jene, die auf eine *Midgard*-Seite zugreifen wollen.
- **midgard-php** ist ein *PHP*-Patch und realisiert zusätzliche Funktionalität in *PHP*.

[1] Eine kompilierte, binäre Version von *Midgard* befindet sich in den neuen Versionen von *Mandrake Linux*.

- **midgard-data** enthält die Administrationsschnittstelle *Asgard* und die *Midgard*-Datenbank.

Die Generierung einer Seite mit *Midgard* erfolgt wie in der Abbildung 3.5 dargestellt:

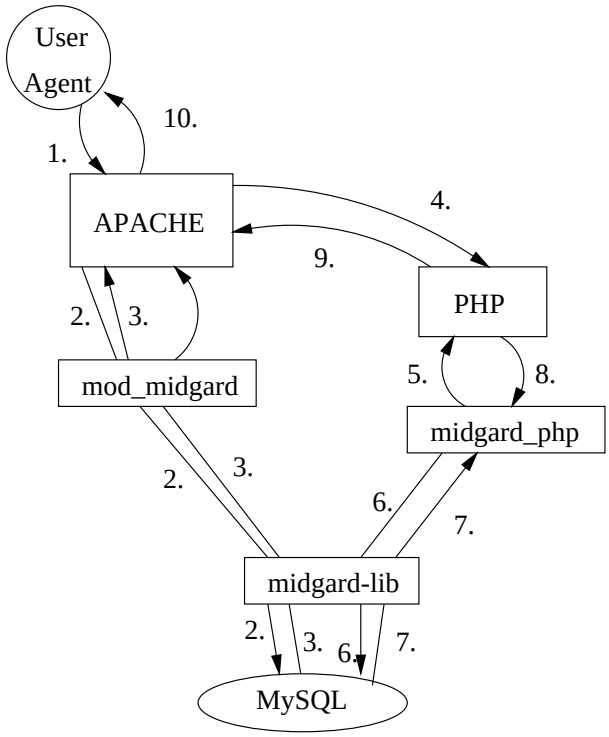

Abbildung 3.5: Midgard Data Flow[MID01]

- **1.** Der Aufruf vom Browser wird vom Modul *mod_midgard* entgegengenommen
- **2.** *mod_midgard* findet heraus, ob der Abruf auf ein *Midgard-Objekt* gezielt ist
- **3.** es werden die entsprechenden *Page*- und *Style*-Elemente über **midgard-lib** aus der Datenbank gelesen
- **4.** *PHP* fängt mit der Generierung der Seite an
- **5,8.** *PHP* verarbeitet die *Midgard*-spezifischen Funktionen
- **6-7.** Diese werden vom Modul *midgard-lib* interpretiert
- **9-10.** Das fertige Dokument wird an den Web-Server zurückgegeben

3.2.3 Zope

Zope (Z Object Publishing Environment) ist ein *OpenSource*-Werkzeug für die Erstellung und die Verwaltung von Web-Seiten. Es ist eigentlich kein richtiges *Content Management*

System, es implementiert jedoch einige wesentlichen *CM*-Funktionen. Die Betreuung erfolgt durch die Firma *Digital Creations* und durch viele freiwilligen Programmierern.
Zope wurde in der Skriptsprache *Python* implementiert, d.h. es ist auf vielen Plattformen nutzbar. Von der Web-Seite der *Zope-Community* sind binäre Distributionen für *Windows, Solaris und Linux*, so wie alle *Zope*-Quellcodes erhältlich.

Zope besteht aus folgenden Teilen:

Zope WebServer: *Zope* enthält einen eigenen Web-Server, kann aber auch mit anderen *HTTP*-Servern zusammenarbeiten, die *CGI* unterstützen.

Web-basierte Schnittstelle: Die Nutzer von *Zope* arbeiten mit dem *Zope Management Interface* über einen Web-Browser. So wird die Plattformunabhängigkeit des Systems weitgehend unterstützt.

Objektendatenbank: *Zope* implementiert eine eigene relationale Datenbank namens *Gadfly*, die der Verwaltung der Inhalte dient. Sie wurde von *Aaron Winters* in der Skriptsprache *Python* geschrieben. Das System bietet des weiteren auch Schnittstellen zu anderen relationalen Datenbanken an, z.B. wie *Oracle, SyBase, MySQL, PostgreSQL*.

Scripting Language Support: *Zope* bietet die Integration verschiedener Server-Skriptsprachen zur Erstellung dynamischer Seiten an.

Zope implementiert eine eigene Server-Side Skriptsprache für die Erstellung von dynamischen Seiten. Diese Sprache heißt *DTML (Document Template Markup Language)* und dient in erster Linie zum Erstellen kleiner Skripte für die Gestaltung des Layout der Seiten. Um komplexere Server-Skripte zu schreiben, kann man in *Zope* andere Sprachen wie z.B. *Python* oder *Perl* einbinden.
Die Inhalte in *Zope* werden in sog. *Folder*n strukturiert, welche auch verschachtelt werden können. Die einzelnen Assets sind als *Zope*-Objekte in den Foldern abgelegt. Neben den vordefinierten Objekten kann man auch eigene Objekte definieren. Die Nutzer des Systems können zu Rollen zugeordnet werden. Unter einer Rolle in *Zope* wird eine Gruppe von Benutzern mit ähnlichen Aufgaben und Berechtigungen verstanden. Ein Nutzer kann auch Mitglied mehrerer Gruppen sein.
Wie schon erwähnt, ist *Zope* kein richtiges *Content Management System* sondern eher ein Redaktionssystem, es verwirklicht jedoch wichtige Funktionen für die Verwaltung von Web-Inhalten. Da *Zope* auch als Quell-Code verfügbar ist, kann man es nach eigenen Bedürfnissen erweitern bzw. an seiner Weiterentwicklung teilnehmen. *Zope* wird wegen seiner einfachen Handhabbarkeit in erster Linie bei der Pflege der Web-Seiten kleinerer Unternehmen eingesetzt.

3.3 Andere Aspekte

3.3.1 Knowledge Management mit Mindaccess

Die bisher betrachteten Werkzeuge waren in erster Linie sog. *Web Content Management Systeme*, die die Bereitstellung und die Pflege von Web-Inhalten ermöglichen. Da aber *Web*

Content Management nur ein Teilgebiet von *Content Management* ist, wird hier ein Werkzeug betrachtet, das einen anderen wichtigen Aspekt von *CM* vertritt.
Mindaccess ist ein Produkt der deutschen Firma *Insiders Information Management*, das in Zusammenarbeit mit Forschungseinrichtungen, insbesondere mit dem *Deutschen Forschungszentrum für Künstliche Intelligenz* entwickelt wurde. Es ist im Grunde genommen ein *Knowledge Management* Produkt, das die *semantische Bearbeitung* von Inhalten ermöglicht.
Mindaccess bietet Dienste zur Erschließung, Organisation, Bereitstellung und Klassifikation von Informationen aus textbasierten Dokumenten. Es ermöglicht die inhaltsbezogene Verarbeitung von Assets und erfüllt solche Aufgaben, wie:

- Implementierung von Such-Funktionen
- Index- und Term-Generierung
- Dokumenten-Recherche und -Klassifikation
- Feature-Extraktion
- Termsuche
- Clustering

Mindaccess kann textuelle Inhalte semantisch verknüpfen, mit seiner Hilfe werden Tendenzen und versteckte Korrelationen zwischen Dokumenten offenbar. Es wird möglich, komplexe Recherchen, Ähnlichkeiten zwischen Texten bzw. Textsammlungen zu erfassen, die oft die Voraussetzung für Trendanalysen in einem Unternehmen darstellen.
Mindaccess ist ein gutes Beispiel dafür, daß Content Management mit einer Vielzahl von informatischen Teilgebieten wie *Knowledge Management*, *Künstliche Intelligenz* oder *Data Mining* eng verknüpft ist. Für weitere Informationen über *Mindaccess* siehe [Ins00].

3.4 Zusammenfassung - Bewertung der Systeme

Es wurde in den vorherigen Abschnitten versucht, einige heute verwendeten *Content Management Systeme* kurz vorzustellen. Neben den kommerziellen, kostspieligen Systemen wurden auch frei zur Verfügung stehende *Open-Source* Werkzeuge diskutiert.
Es ist festzustellen, daß die vorgestellten Produkte verschiedene Funktionen in den Mittelpunkt stellen. *NPS* und *Gauss* zielen z.B. in erster Linie auf die Verwaltung und die Einbindung von Dokumenten, *Midgard* oder *V-CMS* bieten dagegen eine Vielzahl von Programmierfunktionen zur Erstellung von dynamischen, personalisierten Web-Sites. Einen dritten Aspekt vertritt *Mindaccess*, das die semantische Verarbeitung der zur Verfügung stehenden Daten unterstützt.
Es ist auch nicht zu bezweifeln, daß die meisten *Open-Source*-Lösungen noch nicht so leistungsfähig sind, wie ihre „kommerziellen Rivalen". Sie vertreten zwar auch viele fundierte Ideen, es ist jedoch keines dieser Systeme ein richtiges *CMS*, das alle wichtigen *CM*-Funktionen verwirklicht. Aus diesem Grunde wird im Kapitel 4 eine weitere preisgünsige

Lösung vorgestellt, die in dieser Arbeit entstanden ist und viele weitere *CM*-Funktionen realisiert.
Eine wichtige Anforderung an ein heutiges *CM*-System ist die Plattformunabhängigkeit. Diese wird heute auf der Client-Seite fast immer verwirklicht, die Servers stützen sich dagegen noch oft auf plattformspezifischen Software-Bauelementen. Eine klar erkennbare Tendenz ist auch, daß neben den auf *Java* basierenden Technologien auch gewisse Skriptsprachen, wie *PHP*, *JSP*, *Perl*, *Python* oder *Tcl* immer mehr an Bedeutung in der plattformunabhängigen Programmierung gewinnen, die *Java* vor allem aus der Sicht der Geschwindigkeit übertreffen.
Ein wichtiger Gesichtspunkt beim Einsatz eines *CMS* sind die entstehenden Kosten. Die meisten kommerziellen Lösungen sind heute nur für große Unternehmen mit entsprechend großem Kapital und Personal erreichbar, auf der anderen Seite bereiten auch *Open-Source*-Lösungen noch immer viele Schwierigkeiten bezüglich der Konfiguration und der Pflege der Werkzeuge. Die Marktuntersuchungen zeigen eindeutig, daß der Bedarf, Inhalte wirtschaftlich managen zu können, auch bei mittelgroßen und kleinen Unternehmen immer größer ist. Gerade in diesem Bereich ist also eine wesentliche Entwicklung in der Zukunft zu erwarten.

In den folgenden Tabellen werden die wichtigsten Eigenschaften der besprochenen Werkzeuge systematisch aufgelistet.
(Für Informationen über weitere *CM*-Werkzeuge siehe die *Deutsche Content Manager Site* [CON00] oder die Studie [BSW00].)

Content Management Systeme im Vergleich I.

	Vignette CMS	**NPS 4.0**	**OpenCMS**
Anbieter	Vignette Corp., Austin	Infopark AG	OpenCMS Group
Systemanforderungen-Server			
Plattform	Solaris, Windows NT	Solaris, Linux, WinNT/2000	Alle mit Java-VM/Servlets
Hardware	hoch	moderat	hoch
Datenhaltung	Sybase, Oracle, Informix MS SQL	Sybase, Oracle, Informix MS SQL	MySQL
Web-Server	Netscape, Apache, IIS	alle mit CGI	Apache, IIS
Systemanforderungen-Client			
Schnittstelle	Java-Applikation	Web-Browser, Kommandozeile	Web-Browser
Plug-Ins	Java	Java	Java
Plattform	alle mit JVM	alle	alle
Benutzerschnittstelle			
Eingabeschnittstelle	Java-Applikation, Web-Browser	Web-Browser	Web-Browser
Voransicht	ja	ja	Browser
Werkzeuge zur Seitenerstellung			
Dynamische Seiten	ja	nein, nur mit Portal Manager	ja
Caching	ja	nein	nein
APIs,Sprachen	Tcl, ASP, JSP	Tcl	Java
Importschnittstellen			
Formate	kann implementiert werden	Text, XML, Formate mit Konverter, Batch-Upload	exportierte Inhalte
Workflow			
Versionierung	ja	ja	nein
Rollen	ja	ja	ja
Freigabe	mehrstufig, Mehr-Augen-Prinzip	selbstdefinierte Workflows	Freigaberecht an Gruppen
Locking	ja	ja	ja
Kosten			
Preismodell	nach CPU-Leistungsklasse ab ca. 320 000 DM	nach Server CPUs und Benutzer ab 50 000 DM	frei
Referenzen			
	Deutsche Telekom, ZDNet	Siemens, Dresdner Bank	

Content Management Systeme im Vergleich II.

	Gauss VIP	**Zope**	**Midgard**
Anbieter	Gauss Interprise AG	Zope.org	Midgard Project Ry
Systemanforderungen-Server			
Plattform	alle mit Java-VM/Servlets	Windows, Linux, Solaris	Unix, Linux
Hardware	hoch	moderat	moderat
Datenhaltung	Dateisystem bzw. O/JDBC	eigene Datenbank externe relationale	MySQL
Web-Server	IIS, Netscape Apache, Lotus Domino	alle mit CGI	Apache
Systemanforderungen-Client			
Schnittstelle	Java-Applikation	Web-Browser	Web-Browser
Plug-Ins	Java		
Plattform	alle mit Java-VM	alle	alle
Benutzerschnittstelle			
Eingabeschnittstelle	Java-Applikation	Web-Browser	Web-Browser
Voransicht	ja	ja	nein
Werkzeuge zur Seitenerstellung			
Dynamische Seiten	mit Portal Manager	ja	ja
Caching	nein	nein	nein
APIs,Sprachen	SSJS, PHP, JSP	DTML, Perl, Python	PHP-API mit vielen eigenen Funktionen
Importschnittstellen			
Formate	diverse Formate, Import ganzer Web-Sites		
Workflow			
Versionierung	ja	ja	ja, mit Repligard
Rollen	ja	ja	ja
Freigabe	frei definierbar	einfach	
Locking	ja		
Kosten			
Preismodell	per Server, Clients und Datenbanklizenz extra	frei	frei
Referenzen			
	Bayerische Landesbank Luftfahrtbundesamt	ZopeZen, Digital Garage	European Telecom Business Portal, Linux Germany

Kapitel 4

Konzeption eines preiswerten *CMS*

In den vorangegangenen Kapiteln wurden die wichtigsten Eigenschaften und Merkmale von *Content Management Systemen* zusammengefaßt. Dabei wurde auch die Frage der Anwendbarkeit von sogenannten „low-cost“ Lösungen diskutiert.
Als praktischer Teil dieser Diplomaufgabe soll ein *Content Management System* (*EasyContent*) entworfen und zum Teil implementiert werden, das die wichtigsten Funktionalitäten eines *CMS* in einer preiswerten Umgebung darstellt. In diesem Abschnitt wird die Architektur und funktionelle Struktur konzipiert. Kapitel 5 befaßt sich dann mit den Fragen der Implementation, d.h. der Umsetzung der Konzepte in die Praxis.

4.1 *EasyContent* – Der Name

Der Name *EasyContent* weist darauf hin, daß es um ein einfach zu handhabendes, dennoch sehr nützliches Werkzeug geht. Während der Konzeption wurde versucht, die wichtigen *CM*-Funktionen auf eine übersichtliche, leicht verständliche Weise in das System zu integrieren.

4.2 *EasyContent* – Zielsetzungen

EasyContent ist ein *Web Content Management System*, das ausschließlich auf *Open-Source* Software-Lösungen basiert. Es stellt eine Oberfläche für die Verwaltung von *Web-Auftritten* und *Web-Inhalten* dar.

- *EasyContent* ist eine **Plattform zur Verwaltung von Inhalten** einer Web-Site. Es implementiert Funktionen zur Speicherung, Strukturierung, Einbindung, Veröffentlichung und Archivierung digitaler Assets.
- *EasyContent* bietet des weiteren eine **Programmieroberfläche für die Erstellung statischer und dynamischer Web-Seiten**. Mit der Hilfe von *EasyContent*-Templates wird es möglich, datenbankgestützte, personalisierte „up-to-date” Web-Dokumente zu erstellen und auf eine effektive Weise im Intra- bzw. Internet zu publizieren.

- *EasyContent* ist außerdem ein Entwicklungswerkzeug, das die **koordinierte Zusammenarbeite mehrerer Entwickler** an einem Web-Auftritt erlaubt. Es bietet **grundsätzliche Workflow-Mechanismen** für die Automatisierung der Arbeitsabläufe im Bereich *Content Management*.

Die wesentlichsten in *EasyContent* implementierten *CM*-Funktionen sind:

- *die projektorientierte Entwicklung von Web-Auftritten*
- *die Trennung von Inhalt und Layout*
- *das Management digitaler Assets*
- *die Automatisierung von Freigabemechanismen durch die Implementierung von Content-Workflows und Staging*
- *die zentrale Benutzer- bzw. Zugriffsverwaltung und die Personalisierung der Inhalte*
- *die effektive Publikation der Inhalte mittels Caching*

Bei der Konzeption dieser Funktionen wurden folgende Gesichtspunkte berücksichtigt:

- **Anwendung freier Softwarekomponenten:** Bei der Konzeption von *EasyContent* ist die ausschließliche Anwendung freier Software-Komponenten eines der wichtigsten Gesichtspunkte.

  Im Abschnitt 3 hat es sich beim Vergleich verschiedener *Content Management Systeme* herausgestellt, daß die heutigen „low-cost" *CMS* meistens noch nicht so leistungsfähig sind, wie ihre kommerziellen Rivalen. Ein wichtiges Ziel dieser Arbeit ist es zu zeigen, daß wichtige *CM*-Funktionen auch in einer preiswerten Umgebung verwirklicht werden können. Aus diesem Grunde werden bereits bei der Konzeption auch technologische Fragen behandelt - siehe 4.3.

- **Skalierbarkeit und allgemeine Einsetzbarkeit:** *EasyContent* organisiert und verwaltet die Arbeiten in sog. Projekten. Diese und die internen Nutzer- bzw. Gruppenverwaltungsfunktionen ermöglichen die Definition von separaten Arbeitsschritten und die synchronisierte, parallelle Zusammenarbeit mehrerer Entwickler. So ist *EasyContent* auch bei der Pflege großer Inhaltsmengen und komplexer Web-Anwendungen einsetzbar.

- **Weitgehende Plattformunabhängigkeit:** *EasyContent* ist ein Werkzeug, das sowohl auf der Client- als auch auf der Server-Seite plattformunabhängig ist, es nutzt auf allen Plattformen verfügbare Software-Komponenten. Die Bedienung des Systems erfolgt mittels einer web-basierten Oberfläche, die die Anwendung auf einer Vielzahl von Plattformen ermöglicht. Diese Oberfläche benutzt Funktionen, die von den meisten heute verwendeten Web-Browsern unterstützt werden. Für die genaueren Anforderungen an die Client-Anwendungen siehe Abschnitt 5.2.2.

- **Einfache, übersichtliche Bedienung:** *EasyContent* bietet eine übersichtliche Oberfläche und eine einfach verständliche sowie handhabbare Arbeitsumgebung für alle Nutzer sowohl im Entwicklungs-, als auch im Redaktions- und Verwaltungsbereich.
- **Unterstützung von gängigen Programmierstandards:** Das System basiert auf bekannten, verbreiteten Programmierstandards, die bei der Entwicklung der *EasyContent*-Web-Seiten eingesetzt werden können. So ist es für Einsteiger nicht nötig, neue Markup- bzw. Programmiersprachen zu erlernen. Des weiteren bietet das System auch eigene *EasyContent*-spezifische Funktionen, die bei der Arbeit sehr nützlich sind.

4.3 Technologische Fragen

Da *EasyContent* in einer preiswerten Software-Umgebung implementiert werden soll, ist es bereits bei der Konzeption des Systems wichtig, das Für und Wider der zur Verfügung stehenden Technologien abzuwägen, um eine leistungsfähige Arbeitsumgebung zu schaffen. In diesem Abschnitt werden also technologische Entscheidungen getroffen.

4.3.1 Betriebssystem

Im Abschnitt 4.2 wurde schon erwähnt, daß *EasyContent* auf der Server-Seite möglichst plattformunabhängig werden soll. Außerdem ist es wichtig, daß es auch in einer *OpenSource*-Umgebung läuft.
Aus diesem Grunde wird der *EasyContent*-Prototyp unter *Linux* verwirklicht. (Für die genauere Beschreibung der Distribution siehe Abschnitt 5.) Da *Linux* eigentlich „das Unix für den PC" ist, kann man den Prototypen auch unter anderen „*Unix*-like" Systemen wie z.B. *Sun Solaris* verwenden.
In der heutigen Zeit kann man in der PC-Welt auch auf die 32bit Betriebssysteme der Firma *Microsoft* nicht verzichten. Aus diesem Grunde werden später die Schritte einer eventuellen Portierung zu *Windows* angegeben.

4.3.2 Web-Server

Da der Web-Server eine der wichtigsten Komponenten eines *CMS* ist, ist seine Wahl von entscheidender Wichtigkeit. Der Web-Server in *EasyContent* muß die folgenden Anforderungen erfüllen:

- Er soll niedrige Kosten haben
- Er soll schnell und leistungsfähig sein
- Er soll einfach erweiterbar und konfigurierbar sein

In *EasyContent* wird deshalb der weltweit führende *Apache*-Webserver eingesetzt. Dieser ist frei auch als Quell-Code auf allen Plattformen verfügbar und bietet sehr fundierte Konfigurationsmöglichkeiten. (Am 1. April 2001 war *Apache* nach dem *Web Server Survey* der Firma *E-Soft Inc.* auf 74.17% der deutschen Domänen eingesetzt[ESO01].)

4.3.3 Programmierschnittstelle

Die Wahl der Programmierschnittstelle ist vielleicht die wichtigste Frage bei der Konzeption eines *CMS*. Ein wichtiges Kennzeichen eines *Web Content Management Systems* ist nämlich die Generierung dynamischer, datenbankgestützter Seiten, hinter denen eine Programmierlogik steht.
Es gibt heute eine Vielzahl an Server-Programmierschnittstellen, die bei der Konzeption von *EasyContent* verglichen wurden.

- Eine oft verwendete Umgebung für die Erstellung von dynamischen Web-Sites ist *ASP (Active Server Pages)* der Firma *Microsoft*. Dieses Werkzeug hat den Nachteil, daß es eine *Windows*-Plattform voraussetzt, von Portabilität und niedrigen Kosten kann also nicht die Rede sein.

- Die plattformunabhängige Programmierung bedeutet heute in erster Linie den Einsatz von Werkzeugen, die auf der Sprache *Java* basieren. Für serverseitige Programme bietet *Java* zwei Möglichkeiten: *Java Servlets* bzw. *JSP (Java Server Pages)*. *Servlets* sind kompilierte *Java*-Programme, die auf dem Web-Server laufen, *Java Server Pages* sind dagegen dynamische Template-Skripte, die neben *HTML*-Code auch *Java*-Befehle beinhalten und bei jedem Abruf interpretiert werden.

  In *EasyContent* wurde auf den Einsatz von *Java*-basierten Technologien vor allem aus Geschwindigkeits- und Leistungsgründen verzichtet.

- Eine nächste Möglichkeit ist der serverseitige Einsatz von bekannten Skriptsprachen, wie *Perl*, *bash* oder *Tcl*. Diese Sprachen entstanden in einer Zeit, als der Begriff *Web* noch unbekannt war. Da sie sehr schnelle, leistungsfähige Werkzeuge für die Verarbeitung von textuellen Zeichenketten sind, werden sie auf *Web-Servern* oft eingesetzt. Leider ist die Lesbarkeit dieser Skripte oft nicht einfach und die Programmierung vor allem für Anfänger kompliziert.

- In *EasyContent* wird als Programmierwerkzeug die Skriptsprache *PHP (PHP Hyper-Text Preprocessor)* eingestzt. Sie wurde 1994 von *Rasmus Lerdorf* entwickelt und zielt direkt auf die Programmierung von Server-seitigen Programmen ab. *PHP* bietet ein leistungsstarke *C*-ähnliche Syntax und einen breiten Funktionsumfang für die veschiedensten Programmierfunktionen - siehe [Kra00].

4.3.4 Datenhaltung

Die letzte Frage bei der Konzeption von *EasyContent* ist die Datenhaltung. Es ist eine Datenbank zu wählen, die ebenfalls frei zur Verfügung steht, Schnittstellen für Server-Programme bietet und besonders leistungsfähig ist.
In *EasyContent* wird *MySQL* als Systemdatenbank eingesetzt. *MySQL* ist ein *Open Source RDBMS (Relational Database Management System)*, das unter der *GPL-Lizenz (GNU Generic Public License)* steht. Um in kommerziellen Anwendungen benutzt zu werden, muß es lizenziert werden, die Kosten sind aber mit den Preisen der kommerziellen Produkte nicht

zu vergleichen. *MySQL* wurde für schnelle *SQL*-Abfragen optimiert, auf der anderen Seite wurden bisher noch keine *Transaktions*- bzw. *Rollback*-Mechanismen unterstützt. Erst in letzter Zeit kann man in *MySQL* sogenannte *BDB-Tabellen* (*Berkeley DB*) erstellen, die auch Transaktionen unterstützen.

Die Kombination von *PHP*, *Apache* und *MySQL* ist heute eine der leistungstärksten Web-Programmierumgebungen. Diese Konfiguration ist in allen Internet-Plattformen einsetzbar.

4.4 *EasyContent* – Architektur

EasyContent basiert auf dem *Client-Server*-Ansatz - siehe Abbildung 4.1.
Im Mittelpunkt des Systems steht der *EasyContent*-Server, der für die Verwaltung der Inhalte bzw. der Nutzer und die Koordination der Arbeitsabläufe zuständig ist. Die Benutzer des Systems kommunizieren mit dem Server über die im Intra- bzw. Internet verteilten Klientanwendungen.
Die Kommunikation zwischen dem Server und den Klientprogrammen erfolgt über *HTTP*, d.h. die Benutzer arbeiten mit ihren *Web-Browsern*. Auf diese Weise wird die Plattformunabhängigkeit der Klienten weitestgehend garantiert.

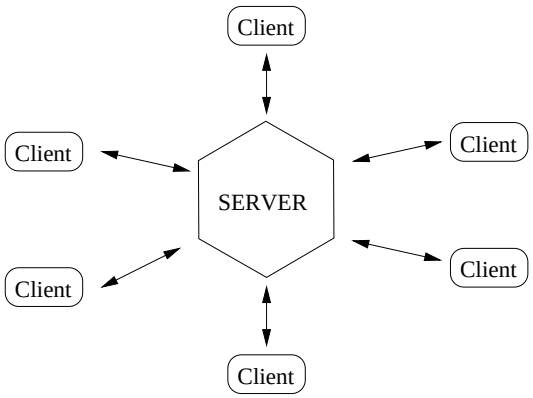

Abbildung 4.1: Client-Server-Ansatz

4.5 *EasyContent* – Systemkomponenten

Die wichtigsten Komponenten von *EasyContent* werden in der Abbildung 4.2 auf Seite 56 dargestellt. Der Abbildung folgt eine kurze Zusammenfassung der Funktionen, die in den nächsten Abschnitten ausführlicher besprochen werden.

Die System-Datenbank verwaltet alle Informationen bezüglich der Projekte, der Templates, der Versionen, der Nutzer, etc. Die weiteren Komponenten des Systems ermitteln alle Informationen aus dieser Datenbank.

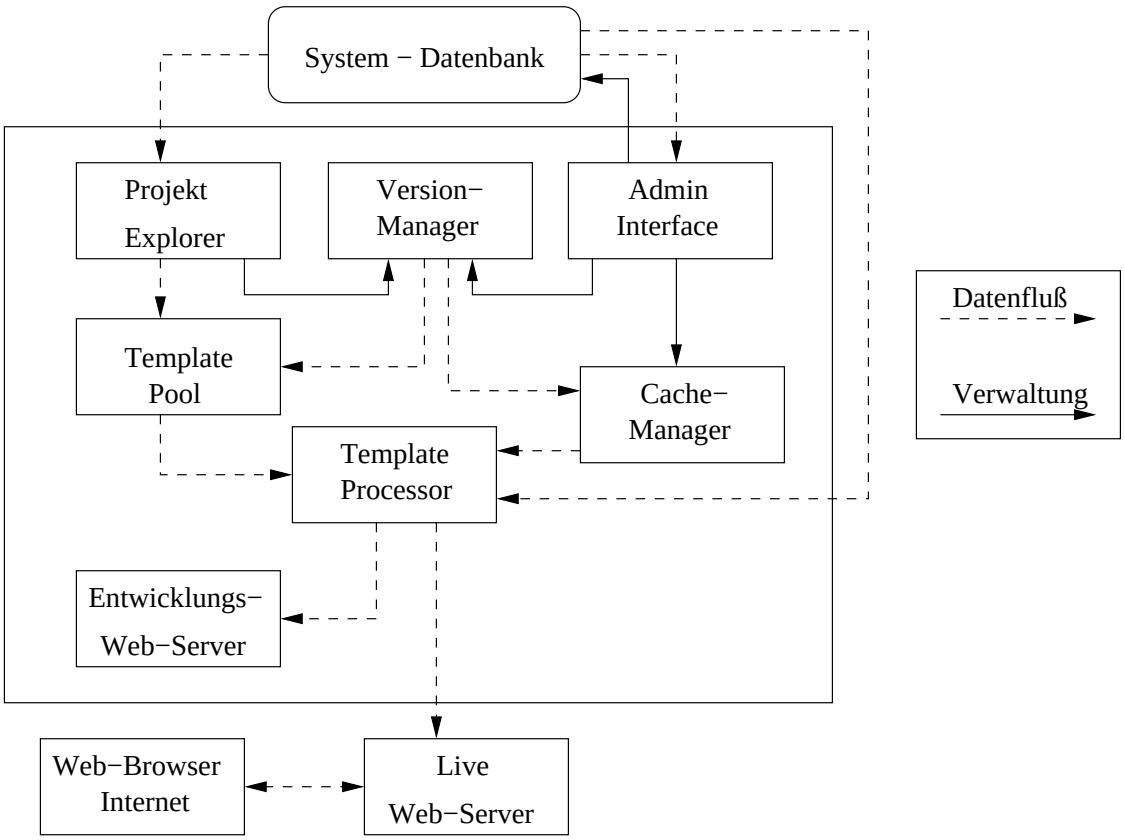

Abbildung 4.2: Die Systemkomponenten

Der Projekt-Explorer ermöglicht die Verwaltung von Projekten, Templates und Assets - siehe 4.6. Die Entwickler bzw. die Redakteure haben die Möglichkeit, Projekte, Assets und die Templates zu bearbeiten, Inhalte zu editieren, neue Inhalte bzw. Versionen von alten Inhalten zu erstellen.

Im Template-Pool befinden sich die bereits aktuellen Versionen der Web-Seiten. Die beiden Web-Server greifen auf den *Template-Pool* zu, um die Web-Seiten an die Browser zu liefern.

Das Admin-Interface ermöglicht dem Administrator neue Projekte zu erstellen, neue User anzulegen, die Zugriffsrechte der Benutzer auf die einzelnen Projekte zu regeln, Versionen von Projekten und Unterprojekten zu speichern, die inneren Datenbanken zu manipulieren bzw. Statistiken zu evaluieren. Hier findet auch die Genehmigung, die Freigabe und die eventuelle Sperrung der Templates statt.

Der Version-Manager ist für die Verwaltung der Versionen zuständig. Es können sowohl Versionen von einzelnen Templates und Assets als auch welche von ganzen Projekten und Unterprojekten angelegt werden.

Der TemplateProcessor bedient die beiden Web-Server. Er generiert die zu veröffentlichenden Web-Seiten anhand der Templates bzw. der Inhalte in der Datenbank und steuert die Arbeit des *Cache-Managers*.

Der Cache-Manager ist für die Erstellung und die Regenerierung von statischen, „gecachten" Seiten zuständig.

Der Entwicklungs-Server veröffentlicht die bereits von den Nutzern erstellten Templates, auf ihn können nur die Entwickler zugreifen. Alle neu erstellten oder aktualisierten Templates erscheinen sofort auf dem Entwicklungs-Server.

Der Live Web-Server ist dagegen der öffentliche Web-Server. Hier dürfen nur noch die fertigen, genehmigten Inhalte erscheinen.

4.6 Projekte, Templates und Assets

Die Arbeit in *EasyContent* erfolgt in *Projekten*. Unter einem *Projekt* versteht man eine zusammengesetzte Aufgabe, die von einer Gruppe von *EasyContent-Nutzern* ausgeführt wird. So eine Aufgabe kann sowohl die Entwicklung einer komplexen Web-Site, als auch die Erstellung einer einfachen *HTML*-Seite sein.
Um Projekte übersichtlich und strukturiert verwalten zu können, kann man diese ineinander verschachteln. Jedes Projekt kann beliebig viele *Unterprojekte* enthalten, die wieder in weitere *Unterprojekte* unterteilt werden können, d.h. die Projekte bilden eine *hierarchische Baumstruktur*. Diese Darstellungsweise unterstützt die hierarchische Projektarbeit, die bei der Erstellung von Web-Seiten erforderlich ist. Die Entwicklung einer komplexen Web-Site kann nämlich oft in getrennte Arbeitsschritte aufgeteilt werden, die von verschiedenen Entwicklern ausgeführt werden.
Die *Projekte* und *Unterprojekte* enthalten *Templates* und *digitale Assets*.

Definition 4.6.1 Templates *sind Layout-Vorgaben - siehe 2.2.1 - die die Web-Seiten dynamisch generieren. Sie ermöglichen, daß die veröffentlichten* HTML-*Seiten zur Laufzeit erstellt werden und den aktuellen Stand der Inhalte präsentieren.*

Definition 4.6.2 *Unter* Assets *versteht man in* EasyContent *beliebige digitale Inhalte. Diese können einfache* Texte, HTML-*Codes,* strukturierte Texte, *sowie beliebige mutimediale Elemente (Grafiken, Images, Sound, Video, Animation) sein, und werden mit* Meta-Daten, *wie z.B. ihrem* MIME-*Typ klassifiziert.*

Die *Projekte*, *Templates* und *Assets* können in *EasyContent* mit logischen Pfadnamen angesprochen werden. Ihre Struktur ist dem Aufbau eines hierarchischen Dateisystems ähnlich, wo die Projekte den Verzeichnissen und die Templates bzw. die Assets den Dateien entsprechen. Die Abbildung 4.3 auf Seite 58 zeigt die Hierarchie von Projekten, Templates und Assets.

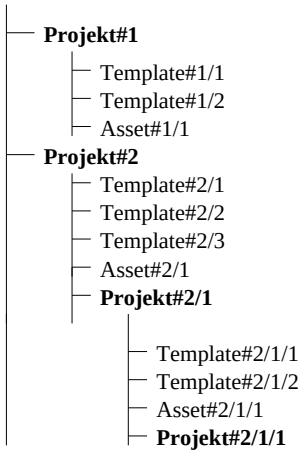

Abbildung 4.3: Projekte, Templates und Assets

4.7 Benutzer- und Zugriffsverwaltung

4.7.1 *EasyContent* – Benutzer

Die Benutzer des Systems können grundsätzlich in drei Gruppen eingeordnet werden: *EasyContent-Entwickler*, *EasyContent-Redakteure* und *Administratoren*.

- Die *EasyContent-Entwickler* des *CMS* arbeiten *ausschließlich* auf der Web-basierten Oberfläche. Diese ermöglicht ihnen, neue Templates anzulegen, bestehende Templates zu editieren, die erstellten Seiten zu testen und versionieren, etc. Die Nutzer arbeiten in sogenannten Projektgruppen - siehe 4.6.

- Die *EasyContent*-Redakteure sind für die Bereitstellung und die Bearbeitung der digitalen Assets verantwortlich. Sie verfügen über keine Programmierkenntnisse und kümmern sich nur um die Erstellung der digitalen Inhalte, nicht aber um ihre Einbindung in die Web-Site. Sie arbeiten auch an der Web-basierten Oberfläche.

- Die *Administratoren* verfügen über alle Funktionen, die Entwickler und Redakteure besitzen, des weiteren haben sie zusätzliche Verantwortungen, die sich vor allem auf die Verwaltung von Nutzern und Gruppen - 4.7.2 -, auf das Management von Projekten - siehe 4.6 -, bzw. auf die Pflege der Inhalte, der Datenbanken und der Versionen beziehen.

Abbildung 4.4 auf Seite 59 zeigt die Struktur der Nutzergruppen. Die *Administratoren* verwalten die Projekte, in denen *Entwickler* und *Redakteure* tätig sind.

4.7.2 Zugriffsverwaltung über Projektgruppen

Wie schon erwähnt, wird in *EasyContent* in sog. Projektgruppen gearbeitet. Die Benutzer des Systems können zu Projektgruppen zusammengefaßt werden, die je an einem Projekt

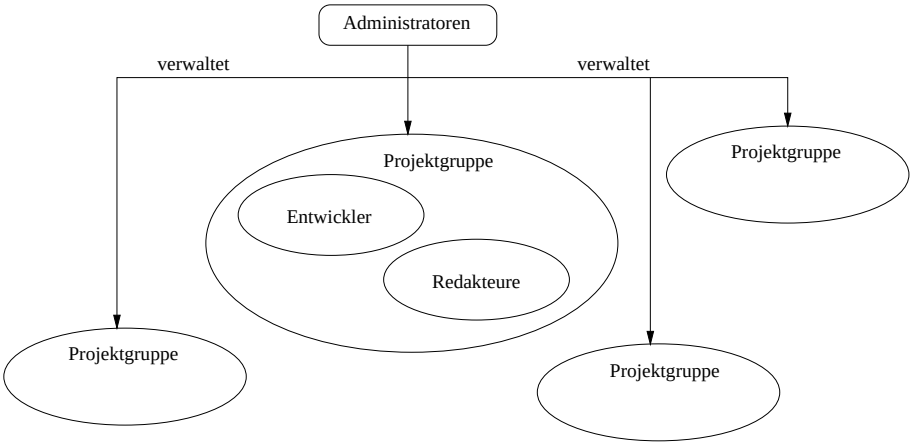

Abbildung 4.4: Entwickler, Redakteure und Administratoren

tätig sind. Das bedeutet, daß die Teilnehmer der Projektgruppe auf die Unterprojekte, auf die Templates bzw. auf die Assets des betroffenen Projektes zugreifen, diese bearbeiten und modifizieren können.
Die Zugriffsverwaltung der *EasyContent-Entwickler* erfolgt über diese Projektgruppen. Das heißt, daß die Rechte nicht per Nutzer, sondern per Projektgruppe vergeben werden. Um diesen Mechanismus zu verstehen, muß zunächst der Begriff *Top-Level-Projekt* definiert werden.

Definition 4.7.1 *Unter einem* Top-Level-Projekt *versteht man ein Projekt, daß sich in der hierarchischen Projektstruktur auf dem obersten Level befindet. D.h. die Projekt-Hierarchie besteht aus disjunkten Bäumen, wobei die Wurzel eines jeden Baumes ein* Top-Level-Projekt *ist. Der Inhalt eines* Top-Level-Projektes *ist die Gesamtheit aller Unterprojekte, Templates und Assets, die sich in der Baumstruktur unter dem* Top-Level-Projekt *befinden - siehe Abbildung 4.5 auf Seite 60.*

Die *EasyContent*-Projektgruppen werden zu den *Top-Level-Projekten* zugeordnet. D.h. ein Nutzer einer bestimmten Projektgruppe hat Zugang zu allen Unterprojekten, Templates und Assets, die sich in der hierarchischen Projektstruktur unter dem gegebenen *Top-Level-Projekt* befinden. Es ist die Aufgabe des Administrators, die Nutzer zu den einzelnen *Top-Level-Projekten* zuzuordnen bzw. sie aus diesen zu entfernen.
Sobald jemand Mitglied einer Projektgruppe ist, darf er unter dem betroffenen *Top-Level-Projekt* neue Unterprojekte,Templates und Assets erstellen, diese editieren bzw. Versionen der einzelnen Inhalte verwalten. Das Löschen von Projekten, Unterprojekten und Templates ist eine Verantwortung des Administrators.

Das Rechtesystem von *EasyContent* mag auf den ersten Blick etwas einfach scheinen. Viele Systeme, die ebenfalls mit hierarchischen Projektstrukturen arbeiten - z.B. *Vignette Content Management Server* - implementieren nämlich kompliziertere Rechtesysteme, wobei die Rechte der Nutzer auch für die einzelnen Unterprojekte vergeben werden können. Da aber diese Lösungen bei einer hohen Anzahl von Nutzern und Unterprojekten kompliziert

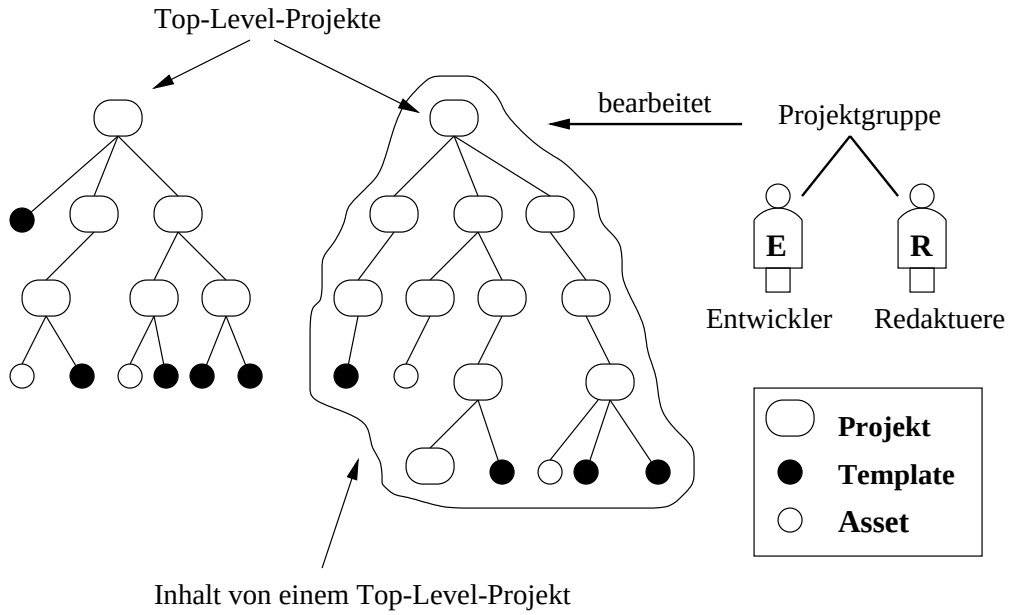

Abbildung 4.5: Die Zugriffsrechte beziehen sich auf die Top-Level-Projekte

und in der Praxis schwer handhabbar sind, verwendet *EasyContent* diese einfachere, jedoch übersichtlichere Methode.

4.8 Assetmanagement in *EasyContent*

Die wichtigsten Aufgaben der *Assetmanagementkomponente* von *EasyContent* sind:

1. die strukturierte Darstellung der Inhalte
2. die Verwaltung der Zugriffe auf die Inhalte
3. die Versionierung der Inhalte
4. Datenbankverwaltung

4.8.1 Strukturierte Darstellung der Inhalte

Wie schon im Abschnitt 4.6 erwähnt, werden die Inhalte in *EasyContent* in einer Baumhierarchie verwaltet. Es können *Projekte* und *Unterprojekte* definiert werden, die Seiten-Vorlagen (*Templates*) und digitale *Assets* enthalten können. Unter digitalen Assets kann man die verschiedensten Medientypen verstehen: von einfachen *ASCII*- bzw. *HTML*-Texten bis zu komplexen multimedialen Objekten ist hier alles denkbar.
Die Aufgabe von *EasyContent* ist die Adressierung dieser Inhalte. Die Projekte, Templates und Assets werden mit logischen Namen versehen, um eindeutig identifiziert und angesprochen werden zu können. Als logische Namen für die Templates und Assets werden ihre

Pfadnamen in der Baumhierarchie verwendet. Das ausgewählte Projekt in der Abbildung 4.6 hat so z.B. den logischen Namen `a-b-c`. Diese Bezeichnung kann überall verwendet werden, um dieses Projekt ansprechen zu können.

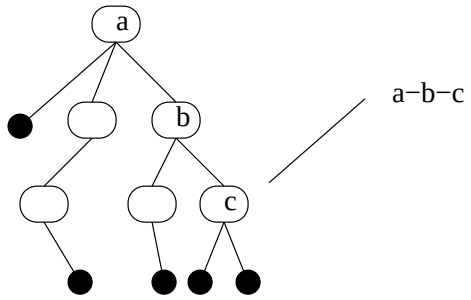

Abbildung 4.6: Adressierung der Inhalte

Die Benutzer von *EasyContent* können in der Baumhierarche mit Hilfe des *Projekt-Explorers* navigieren - siehe Abbildung 4.7 auf Seite 62. Diese grafische Benutzeroberfläche zeigt die Projekte, Templates und Assets mit ihren Eigenschaften und erlaubt dem Nutzer, sich in der Baumhierarchie zu bewegen.
Die Implementierung der Trennung der logischen und der physikalischen Adressierung der Inhalte war eine der schwierigsten Aufgaben bei der Programmierung von *EasyContent*. Diese wird im Abschnitt 5.3.2 ausführlich erklärt.

4.8.2 Die Verwaltung der Zugriffe auf die Inhalte

EasyContent verwaltet die Zugriffsrechte der Nutzer auf die einzelnen Templates und Assets. Beim Zugriff auf ein Asset wird bestimmt, ob der Nutzer berechtigt ist, im zugehörigen *Top-Level-Projekt* zu arbeiten. Im *Projekt-Explorer* sieht der Nutzer nur die für ihn erreichbaren Inhalte. Die Rechte der Nutzer und der Administratoren werden in der folgenden Tabelle dargestellt:

	Bearbeitung von Templates	Bearbeitung von Assets	Unterprojekte erstellen	Top-Level-Projekte erstellen	Ganze Projekte Versionieren
Entwickler	√		√		
Redakteur		√	√		
Administrator				√	√

- Die Zugriffsrechte der *Entwickler* in einer Projektgruppe beziehen sich auf die Programmierung der Templates. Entwickler können Unterprojekte erstellen, statische und dynamische Web-Seiten programmieren, in die sie die von den *Redakteuren* erstellten und verwalteten Assets einbinden können. Das Anlegen von Versionen einzelner Templates ist ebenfalls die Verantwortung der *Entwickler*.

- Die *Redakteure* kümmern sich um die digitalen Assets, sie können diese hinzufügen und modifizieren. Sie können sich die von den *Entwicklern* erstellten Web-Seiten anschauen, die dynamischen Skripte dürfen sie jedoch nicht verändern.

- Die *Administratoren* sind an der Programmierung der Templates und der Erstellung der Assets nicht beteiligt. Sie können neue Projektgruppen erstellen, in dem sie *Top-Level-Projekte* definieren und zu diesen *Entwickler* und *Redakteure* zuordnen. Des weiteren kümmern sie sich auch um die Versionierung von ganzen Projekten und die Freigabe der Inhalte.

Die Abbildung 4.7 illustriert, wie ein Bild-Asset mit seinen Attributen für einen *EasyContent*-Redakteur im *Projekt-Explorer* angezeigt wird.

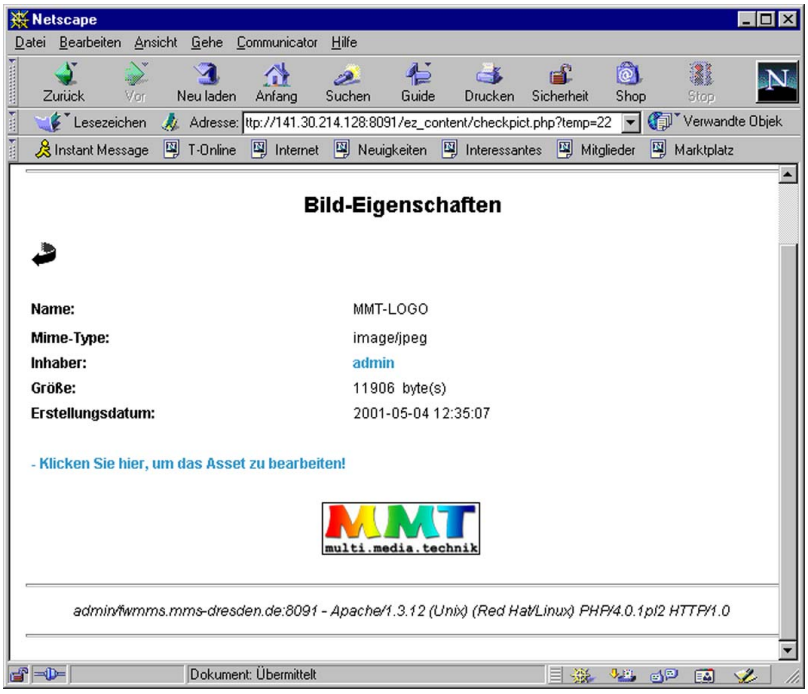

Abbildung 4.7: Darstellung eines Bild-Assets im *Projekt-Explorer*

4.8.3 Versionierung der Inhalte

Abschnitt 2.2.2 stellte bereits die Wichtigkeit der Versionierung heraus. Es ist eine wesentliche Anforderung an ein *CMS*, die Gesamtheit aller Inhalte zu einem bestimmten Zeitpunkt archivieren zu können. Um das Konzept des *EasyContent-Versionsmanager*s zu verstehen, werden hier die wichtigsten Aspekte der Versionierung in einem *CMS* behandelt.

Die *Trennung von Inhalt und Layout* - siehe 2.2.1 - bedeutet, daß eine Web-Seite aus verschiedenen Komponenten besteht. Die Templates und die verschiedenen Assets befinden sich meistens in einer Datenbank bzw. im Dateisystem und werden erst zur Laufzeit zusammengebaut. Um zu wissen, wie eine Präsentation zu einem gegebenen Zeitpunkt aussah, müssen also alle ihrer Komponenten in ihrem damaligen Zustand aufbewahrt werden.
Eine andere Herausforderung an die Versionierung ist die Archivierung ganzer Projekte. Die Inhalte müssen in ihrer ehemaligen Hierarchie gespeichert und wiederhergestellt werden.

Dabei sollen sowohl *absolute* als auch *relative* Referenzen und Verknüpfungen zwischen den einzelnen Elementen erhalten bleiben. Die Versionierung in einem *CMS* ist also ein komplizierter Vorgang.

Die heute gängigen *CMS* - siehe Kapitel 3 - verwirklichen verschiedene Versionierungsstrategien, einige unterstützen die Versionierung gar nicht. Andere ermöglichen es, einzelne Templates oder Assets als Version abzulegen bzw. *Backups* von ganzen Datenbanken zu erstellen. Es ist jedoch fast nirgendwo möglich, daß ganze Projekte und Unterprojekte zusammen archiviert werden. Bei der Konzeption der Versionierung in *EasyContent* wird deshalb dieser Aspekt in den Mittelpunkt gestellt.

EasyContent bietet die Möglichkeit, alte Versionen von Templates, Assets bzw. Projekten speichern und wiederherstellen zu können. Diese Aufgaben wurden in der Systemkomponente *Version-Manager* implementiert. Die Versionierung in *EasyContent* erfolgt auf zwei Ebenen.

1. Auf der einen Seite können die *Entwickler* und *Redakteure* in einer Projektgruppe *Versionen von einzelnen Templates und Assets* speichern.

2. Auf der anderen Seite haben die Administratoren des Systems die Möglichkeit, ganze Projekte bzw. Unterprojekte als *Projekt-Versionen* zu speichern bzw. diese wiederherzustellen.

Versionierung einzelner Inhalte

Wenn ein Nutzer ein beliebiges Template oder Asset bearbeitet, hat er die Möglichkeit, dieses als Version zu speichern. Dazu braucht er einfach auf die Schaltfläche „*Als Version speichern*" zu klicken. So entsteht eine Version des Inhaltes, die nur vom Administrator gelöscht werden kann. Der Nutzer hat des weiteren die Möglichkeit, beliebige alte Versionen des bearbeiteten Inhaltes wiederherzustellen.
Die Versionen von den einzelnen Templates bzw. Assets existieren nur so lange, wie die Inhalte selbst! Wenn der Administrator das gegebene Asset - oder das Unterprojekt, in dem es sich befindet - löscht, so verschwinden auch seine Versionen. Um Versionen „für die Ewigkeit" aufbewahren zu können, soll der Administrator ganze Projekte bzw. Unterprojekte archivieren.
Die Versionen von Templates und Assets werden in Datenbanktabellen mit entsprechenden *Meta*-Informartionen gespeichert.

Versionierung von (Unter)projekten

Eine komplexere Methode für die Archivierung von mehreren Templates und Assets ist das Erstellen von Versionen ganzer Projekte bzw. Unterprojekte durch den Administrator.

Wie schon erwähnt, haben die Projekte und Unterprojekte in *EasyContent* eine hierarchische Baumstruktur. Um ein (Unter)projekt zu archivieren, muß der Administrator dieses auswählen und die Option „*absichern*" aktivieren. So werden alle (Unter)Projekte, Templates und

Assets, die sich unterhalb des ausgewählten Unterprojektes befinden archiviert - siehe Abbildung 4.8. Eine gesicherte Projekt-Version ist also praktisch auch ein kleiner Baumgraf, eventuell mit vielen Unterprojekten und Inhalten.
Die Wiederherstellung einer Projekt-Version bedeutet, daß diese wieder in die globale Projektstruktur von *EasyContent* eingefügt wird. Der Administrator hat hier die Möglichkeit, ein (Unter)Projekt zu bestimmen, unter dem die Projekt-Version wiederhergestellt wird. So kann eine Version - d.h. ein Teilbaum - an einer beliebigen Stelle der Baumstruktur wieder eingefügt und verwendet werden.

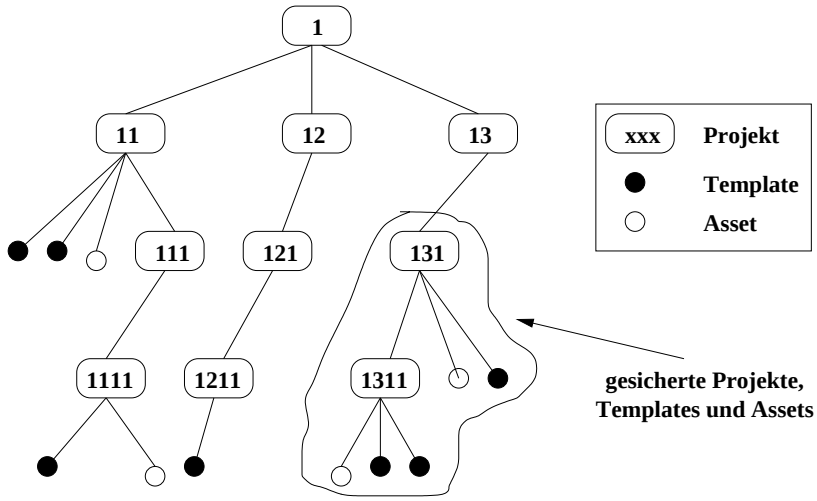

Abbildung 4.8: Sicherung vom Projekt „131”

4.8.4 Datenbankverwaltung

Eine der Zielsetzungen der Konzeption von *EasyContent* war es, eine allgemeine Web-Programmieroberfläche zu schaffen. Das System soll die Programmierung von komplexen, datenbankgestützten Web-Auftritten unterstützen.
Aus diesem Grunde können die Entwickler in ihre dynamischen Seiten nicht nur die genannten internen *EasyContent*-Assets einbinden, sie haben auch die Möglichkeit, auf externe Datenbanken zuzugreifen.

Die *Administratoren* von *EasyContent* haben die Möglichkeit, über eine web-basierte Oberfläche beliebige relationale Datenbanktabellen zu erstellen, bzw. diese zu verwalten. Diese Tabellen haben mit denen in der *EasyContent*-Systemdatenbank nichts zu tun, sie können beliebig bearbeitet, modifiziert oder sogar gelöscht werden. Die *Entwickler* können diese Tabellen aus ihren Templates erreichen, ihre Inhalte in einem „web-gerechten” Format darstellen.
Die Möglichkeit, eigene Datenbanken im *CMS* zu definieren und zu verwalten, macht *EasyContent* zu einem allgemeinen Programmierwerkzeug.

4.9 Personalisierung in *EasyContent*

In *EasyContent* wurden verschiedene Aspekte der Personalisierung konzipiert.

Interne Personalisierung

Die interne Personalisierung bedeutet, daß die Entwickler und Redakteure auf einer personalisierten Oberfläche arbeiten. Sie können nur jene Inhalte sehen und bearbeiten, zu denen sie wirklich Zugang haben.
Abbildung 4.9 zeigt die web-basierte Oberfläche des *Projekt-Explorers*. Die Verzeichnisse kennzeichen die Projekte, die weiteren Zeichnungen die Templates und die Assets. Ein Nutzer hat im *Projekt-Explorer* nur zu den Projekten und Unterprojekten Zugang, deren Mitglied er ist. Es wird auch angezeigt, wenn ein Asset oder Template von einem anderen Benutzer bearbeitet wird. In diesem Fall kann er den gegebenen Inhalt natürlich nicht editieren. Der *Projekt-Explorer* versucht also auch *Groupware Awareness* zu unterstützen, indem er die Attribute der Inhalte auf eine leicht verständliche Weise darstellt.

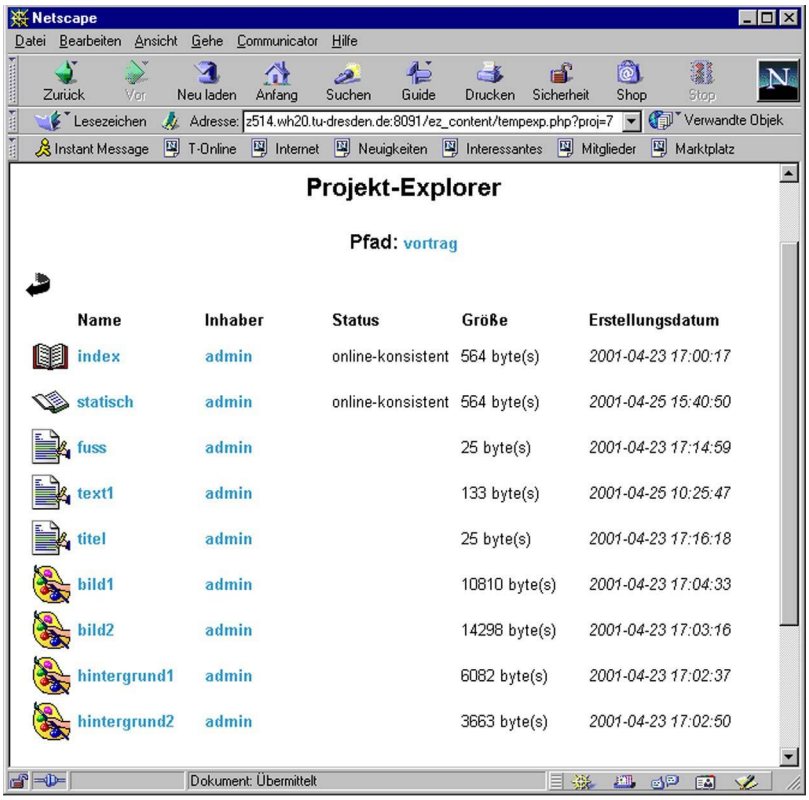

Abbildung 4.9: Der Projekt-Explorer

Externe Personalisierung

Die externe Personalisierung in *EasyContent* bedeutet die Möglichkeit der Erstellung von Web-Seiten, die an die Eigenschaften der Besucher aus dem Internet angepaßt sind.

Die Erstellung personalisierter Seiten ist mit dem Einsatz von *Templates* möglich. Die Entwickler können Skripte für den Server erstellen, die gegebenfalls Datenbankabfragen initieren, um personalisierte Inhalte zu publizieren. Da *EasyContent* u.a. auch ein allgemeines Programmierwerkzeug ist, können die *EasyContent*-Seiten beliebige dynamische Programme auf der Server-Seite laufen lassen.
Um mehr über den Einsatz der Templates zu erfahren, siehe Abschnitt 5.3.1.

4.10 Workflows in *EasyContent*

Wie schon erwähnt, bietet *EasyContent* grundsätzliche Mechanismen für die Koordination der Zusammenarbeit der Mitarbeiter. Diese werden in diesem Abschnitt beschrieben.

4.10.1 Die Verwaltung mehrfacher Zugriffe

Die Projekte in *EasyContent* enthalten Templates und Assets, die gleichzeitig von mehreren Benutzern bearbeitet werden. Es erfolgen parallele Zugriffe auf die gleichen Inhalte, sowohl Lese- als auch Schreib-Zugriffe. Um die Konsistenz der Inhalte sicherzustellen, sollen gleichzeitige Schreibzugriffe auf die gleichen Daten vermieden werden.

Dies wird in *EasyContent* durch die Realisierung von sog. *Check-In*- bzw. *Check-Out*-Mechanismen verwirklicht. Wenn ein Nutzer auf ein Template oder Asset zugreifen möchte, um dieses zu bearbeiten, überprüft das System, ob die zu bearbeitende Inhaltskomponente nicht an jemanden ausgeliehen ist. Falls die gewünschte Komponente von einem anderen Nutzer bearbeitet wird, sind weitere Zugriffe nicht gestattet. Das wird in der grafischen Benutzeroberfläche durch eine Warnung angezeigt.
Wenn der gewünschte Inhalt gerade frei ist, kann er bearbeitet werden. Das System leiht ihn an den Nutzer aus, für andere Mitarbeiter wird er gesperrt - es erfolgt ein *Check-Out*. Nach der Bearbeitung soll der Nutzer das bearbeitete „Arbeitsstück" an die Gemeinschaft zurückgeben (*Check-In*).

4.10.2 Freigabe nach dem *4-Augen-Prinzip*

Eines der Ziele bei der Konzeption von *EasyContent* war, die Verwaltung der Nutzer und deren Zugriffsrechte auf die einzelnen Projekte und Inhalte möglichst einfach zu halten. Entsprechend sind auch die Freigabemachanismen gestaltet.
Das System realisiert das *4-Augen-Prinzip* - siehe 2.3.1. Dabei übernimmt die Rolle des Chefredakteurs der *EasyContent*-Administrator. Die Genehmigung eines neuen Templates bzw. Inhaltes erfolgt folgendermaßen:

1. Die *EasyContent*-Entwickler bzw. Redakteure fertigen Templates bzw. Assets an, die sie per Knopfdruck zur Kontrolle vorlegen können.

2. Darüber erhält der *EasyContent*-Administrator eine System-Meldung - siehe 4.10.3. Er kann sich eine Vorschau der erstellten Web-Seite anschauen und dann eine Entscheidung über eine eventuelle Freischaltung treffen.

3. Der Administrator kann die Web-Seite ebenfalls per Knopfdruck freischalten oder zur Bearbeitung zurückgeben. Die entsprechenden Nachrichten werden an den betroffenen Entwickler bzw. Redakteur gesendet.

Im System wird der Zustand des Templates mit seinem Status angegeben. Das Modell in der Abbildung 4.10 zeigt die möglichen Zustände, bzw. die definierten Übergänge. Mögliche Zustände sind:

1. *Offline und unter Bearbeitung*
2. *Offline und zur Kontrolle vorgelegt*
3. *Online und konsistent* (Der Inhalt ist freigeschaltet und wurde seit der Freischaltung nicht bearbeitet.)
4. *Online und verändert* (Die Web-Seite ist online, im Hintergrund wurde sie aber verändert.)
5. *Online und zur Kontrolle vorgelegt* (Die Web-Seite ist online, wurde aber verändert und wieder zur Kontrolle vorgelegt.)

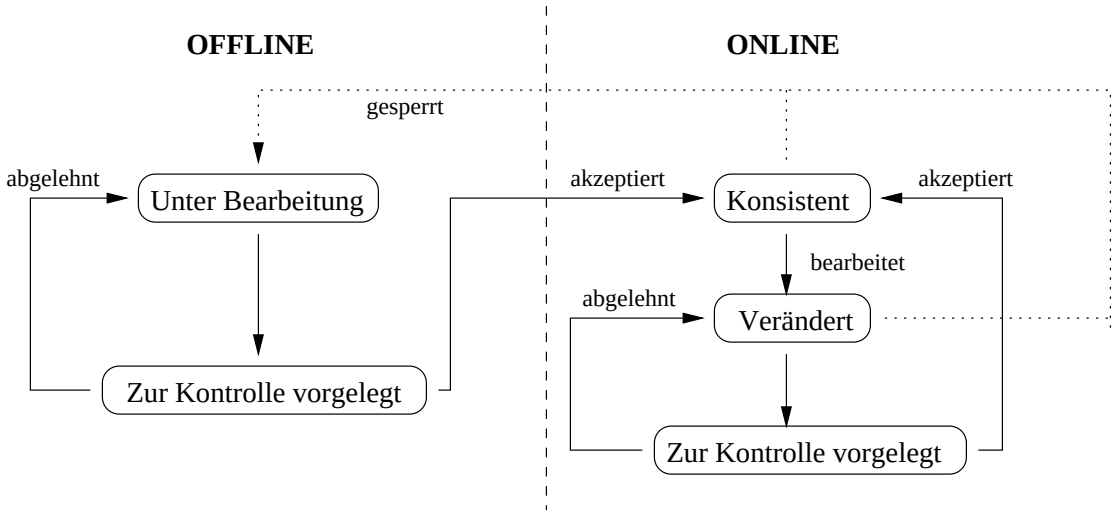

Abbildung 4.10: Zustandsänderungen von Templates

4.10.3 System-Meldungen in *EasyContent*

Die Kommunikation der Mitarbeiter wird in *EasyContent* mit systeminternen Meldungen realisiert. Diese sind asynchrone, textuelle Nachrichten, die zwischen den Mitarbeitern verschickt werden können.

Die *EasyContent*-Meldungen funktionieren ähnlich wie *E-Mail*. Sie haben einen Absender, einen bzw. mehrere Empfänger, einen Betreff, einen textuellen Inhalt und ein Datum. Auf

der grafischen Oberfläche gibt es ein einfaches, integriertes Werkzeug für die Erstellung von Nachrichten, mit dessen Hilfe die Nutzer beliebige Inhalte einander zusenden können. Die Erfassung von Meldungen an ganze Projektgruppen ist auch per Knopfdruck möglich.

Neben den freidefinierten Meldungen zwischen Mitarbeitern werden auch Meldungen vom System generiert. Z.B. wenn ein Administrator die Freischaltung eines Inhaltes genehmigt, wird eine Rückmeldung an den betroffenen Nutzer automatisch generiert. *EasyContent* nützt die Möglichkeiten des implementierten Meldungsystems also weitestgehend aus. Über die praktische Implementierung der Meldungen wird im Abschnitt 5.3.6 geschrieben.

4.11 Staging in *EasyContent*

In diesem Abschnitt wird die Konzeption von *Staging* in *EasyContent* besprochen.

Wie schon erwähnt, erfolgt die Entwicklung in *EasyContent* in sog. Projektgruppen. Diese bestehen aus *Entwicklern* und *Redakteuren*, die für die Erstellung der Skripte bzw. der digitalen Assets zuständig sind.
Die Entwickler und Redakteure arbeiten im geschützten Bereich, d.h. sie haben erst nach einer entsprechenden Authentifizierung Zugriff auf die Inhalte. Ihre Entwicklungen sind zunächst auch nur in diesem geschützten Bereich sichtbar, erst nach einer Freigabe werden sie auch im Live-Bereich erscheinen.
Um die Entwicklungs- und die Live-Bereiche in einem *CMS* zu trennen, ist es nötig, die Gesamtheit aller Daten zu replizieren. Der Transport der Inhalte zwischen den beiden Bereichen ist die Aufgabe der Administration.

In den meisten kommerziellen Lösungen werden zwei Web-Server verwendet, die auf ihre eigenen Datenbestände zugreifen. Der *Entwicklungs*-Server befindet sich im Intranet der Nutzer, der *Live*-Server dagegen im Internet - siehe Abbildung 4.11 auf Seite 69. Der Austausch der Daten erfolgt meistens über *HTTP* oder *FTP* und setzt die Implementierung sehr komplexer Transaktionsvorgänge voraus. Dieses Verfahren haben wir z.B. bei *V-CMS* genützt.
Im Kapitel 3 wurden auch Systeme behandelt - z.B. *OpenCMS* -, die *Staging* viel einfacher implementieren. Bei diesen wird das *CMS* von einem Web-Server bedient, sowohl die internen als auch die externen Daten befinden sich in einer einzigen Systemdatenbank bzw. in einem einzigen Dateisystem. Die Zugriffe aus dem Entwicklungs- bzw. Live-Bereich werden von einer „Programmierlogik" gesteuert - siehe Abbildung 4.12 auf Seite 69. Diese Methode ist aus der Sicht der Implementierung wesentlich einfacher, es hat aber auch Nachteile:

- Die Datenbestände des Entwicklungs- bzw. des Live-Servers sind physikalisch nicht separiert. Die zentrale Verwaltung der Daten macht das System störungsanfälliger
- Da auf den Web-Server auch alle aus dem Internet zugreifen können, sind für den Schutz der inneren Daten strenge Maßnahmen nötig.

In *EasyContent* wird eine dritte Alternative verwirklicht, die die Vorteile der genannten Lösungen kombiniert und in der Abbildung 4.13 auf Seite 70 dargestellt wird.

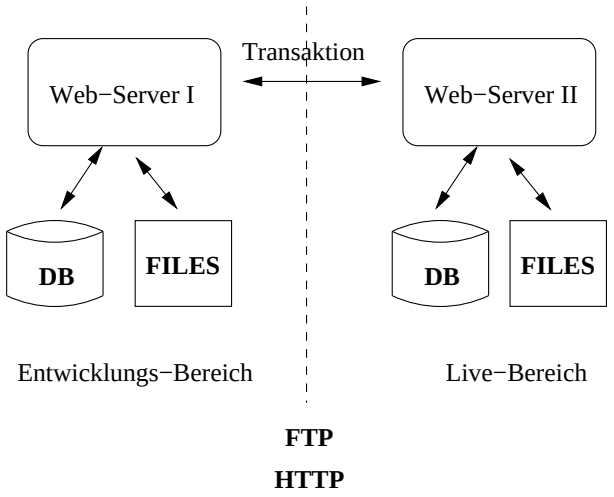

Abbildung 4.11: Staging mit Replikation

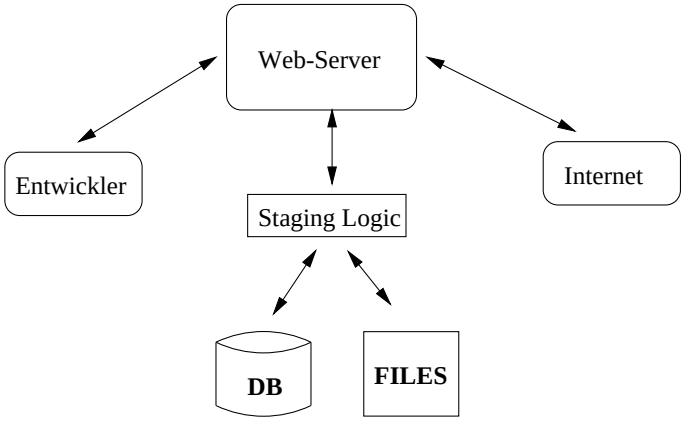

Abbildung 4.12: Staging mit einem Web-Server

EasyContent verwendet zwei separate Web-Server. Diese laufen auf einem einzigen Rechner, wobei der Entwicklungs-Server auf einem für die Außenwelt nicht zugänglichen Port arbeitet. Die Server greifen auf dynamische Seiten zu, die in ihren eigenen Datenverzeichnissen abgelegt sind, d.h. es gibt von allen Templates zwei existierende Exemplare.
Die gemeinsame Systemdatenbank verwaltet alle Inhalte und die mit ihnen verbundenen Meta-Daten. Die ursprünglichen Exemplare der auf den beiden Servern befindlichen Templates und Assets sind hier gespeichert.
Diese Implementierung von Staging hat gewisse Vorteile:

- Die beiden Arbeitsumgebungen können durch den Einsatz zweier Server auch physikalisch vollständig getrennt werden. Die Konfiguration der Zugriffsrechte auf die Server kann aus dem *CMS* gesteuert werden.
- Die gemeinsame Systemdatenbank ermöglicht, daß alle Daten zentral verwaltet werden können. Der eventuelle Ausfall der Datenbank blockiert zwar das ganze System, die Verwaltung und der Transport der Daten ist aber einfach handzuhaben.

- Die Möglichkeit, die beiden Server auf einem Rechner zu installieren, bietet sich für kleinere Sites an.

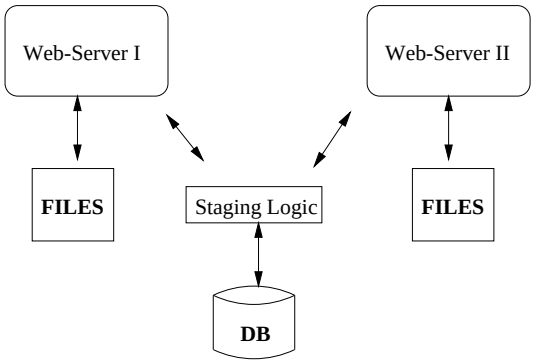

Abbildung 4.13: Staging in *EasyContent*

4.12 Caching in *EasyContent*

Eines der wichtigesten Kennzeichen eines *WCMS* ist *Caching*, d.h. die vorübergehende statische Speicherung gewisser Inhalte - siehe 2.2.8. Dies ist nötig, um die Belastung der Datenbank zu reduzieren.
Bei der Konzeption eines *Caching*-Systems sind folgende Fragen zu klären:

- Welche Inhalte sollen „gecacht" werden?
- Wie werden „gecachte" Inhalte gespeichert?
- Wie lange sind „gecachte" Inhalte gültig?

Die wichtigste Frage von Caching ist, welche Inhalte „gecacht" werden sollen. Es dürfen nämlich theoretisch nur jene Seiten „gecacht" werden, deren Inhalte noch aktuell sind. Ein „ideales" *WCMS* sollte die Seiten also nur so lange statisch speichern, bis sie verändert und wieder generiert werden sollen.

Ein interessanter Ansatz für die Automatisierung von *Caching* wurde im Abschnitt 2.3.4 mit *iWebDB* dargestellt. Hier werden die Web-Seiten erst dann aktualisiert, wenn die hinter ihnen stehenden Datenbanktabellen modifiziert werden. Dieser Mechanismus beruht auf der Verwendung von *Datenbank-Triggers* und benötigt deshalb spezielle Datenbanken. Außerdem ist sein Einsatz in vielen praktischen Fällen kompliziert:

- Bei einer hohen Anzahl von dynamischen, datenbankgestützten Seiten ist es schwer zu wissen, welche Seiten von welchen Datenbanktabellen abhängig sind. Mit der Zunahme der Zahl der Seiten und der Datenbanktabellen wird die Administration sehr aufwendig.

- Dynamische Seiten beruhen in praktischen Fällen nicht nur auf Datenbanken, sondern auf Inhalten im Dateisystem bzw. auf anderen Web-Servern im Intra- bzw. Intranet. Es ist fast unmöglich, die Veränderung dieser Quellen automatisch nachvollziehen zu können.

In den meisten heute verwendeten *CM*-Systemen müssen die Entwickler entscheiden, ob eine Seite „gecacht" werden soll. Dies gilt auch für *EasyContent*.

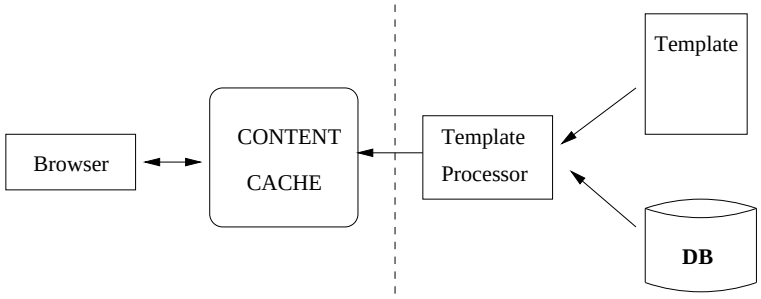

Abbildung 4.14: Caching in *EasyContent*

Die *EasyContent*-Templates können mit dem Attribut „gecacht" versehen werden. Die auf dynamischen Templates basierenden Web-Seiten werden dann folgendermaßen erstellt:

- Bei dem ersten Zugriff auf die Seite wird diese wie üblich vom *TemplateProcessor* generiert. Gleichzeitig wird die fertige Seite sofort statisch im *WCMS* gespeichert.
- Bei den folgenden Zugriffen auf die Seite wird der *TemplateProcessor* nicht mehr aufgerufen. Die zuvor zusammengebaute, statische Seite wird sofort an den Browser zurückgeschickt.
- Nachdem die statische Seite ihre Aktualität verliert, kann diese vom Administrator neu generiert - *regeneriert* - werden. Dabei wird der *TemplateProcessor* erneut aufgerufen und der statische Speicher - siehe Abbildung 4.14 - mit neuem Inhalt geladen.

Die Regenerierung einer „gecachten" Seite ist also die Aufgabe des Admnistrators. Diese Aufgabe kann in *EasyContent* sowohl manuell als auch automatisch erfolgen. Der Administrator hat die Möglichkeit, die Termine der Aktualisierungen des *Cache* festzulegen, wobei er sowohl periodische als auch aperiodische Folgen von Zeitpunkten definieren kann. (Er kann z.B. festlegen, daß die Seite mit den aktuellen Lotto-Zahlen jede Woche am Samstag Abend neu generiert werden soll.)

4.13 Zusammenfassung

In diesem Abschnitt wurden die wichtigsten in *EasyContent* realisierten Konzepte dargestellt. Nun wird es darum gehen, wie diese in einer preiswerten Umgebung in die Praxis umgesetzt werden können.

Kapitel 5

Implementation

Im Kapitel 4 wurden die grundsätzlichn Konzepte von *EasyContent* dargelegt. Hier wird beschrieben, wie die einzelnen *CM*-Funktionen in der gewählten preiswerten Software-Umgebung realisiert wurden.

5.1 Beschränkungen gegenüber der Konzeption

Es wurde ein **Prototyp** von *EasyContent* implementiert. Seine Realisierung erfolgte in jeder Hinsicht anhand der entworfenen Konzepte, es gibt dennoch einige Ideen, die im jetztigen Prototypen noch nicht umgesetzt wurden. Es ist jedoch zu betonen, daß das entstandene Werkzeug alle wichtigen *CM*-Funktionen verwirklicht, auf welche bei der Konzeption großer Wert gelegt wurde.
Die Vereinfachungen gegenüber den Konzepten sind folgende:

- Im Abschnitt 4.7.1 wurde beschrieben, daß in *EasyContent Entwickler*, *Redakteure* und *Administratoren* tätig sind. Im Prototypen wurden die beiden Rollen *Entwickler* und *Redakteur* als eine zusammengesetzte Rolle realisiert, die sowohl auf die Templates als auch auf die Assets Zugriff haben.

- Im Abschnitt 4.6 wurde erwähnt, daß die *EasyContent*-Projekte beliebige digitale Assets enthalten können. Dieses Konzept wurde auch nur beschränkt realisiert. Im *EasyContent*-Prototypen wurde nur die Verwaltung von textuellen bzw. Bild-Assets verwirklicht.

5.2 Die Softwareumgebung des Prototypen

Im Abschnitt 4.3 wurden technologische Entscheidungen der Konzeption von *EasyContent* beschrieben. Nun wird die Software-Umgebung des verwirklichten Prototypen spezifiziert. Der Prototyp basiert *ausschließlich* auf kostenlosen Software-Komponenten.

Die Basis des Systems ist das Betriebssystem *Linux*. Bei der Implementation wurde die Version 7.0 von *RedHat Linux* - siehe [Red00], [KD00] - gewählt, doch das System funktio-

niert auch mit anderen *Linux*-Distributionen. Auf eine eventuelle Portierung nach *Microsoft Windows* wird später im Kapitel 5.5 eingegangen.
Der verwendete Web-Server ist *Apache* 1.3.12. Dieser ist der aktuell am meisten verwendete *HTTP*-Server. Die Umstellung auf andere Web-Server ist in dieser Version nicht gestattet, weil das realisierte System viele innere Dienste und Eigenschaften von *Apache* nützt. Da *Apache* auf den meisten Plattformen – *Linux, Unix, Sun Solaris, Windows, etc.* – frei zur Verfügung steht, ist seine Anwendung eine sinnvolle Wahl.
Die interne Datenbank von *EasyContent* ist *MySQL* - siehe [YRK99]. Auf dem Prototypensystem läuft die Version 3.23.
Der Kern des Systems ist die Skriptsprache *PHP4* ([Kra00]). Die Programme verwenden viele *PHP4*-spezifische Funktionen und können deshalb mit einem alten *PHP3*-Interpreter nicht genutzt werden.
Als Text-Editor für die Erstellung des Quellcodes und der Dokumentation wurde *Emacs* [Cam99] verwendet. Die Abbildungen wurden *xfig* und *gimp* [Neu00] gezeichnet. Die Diplomarbeit wurde in LaTeX [FCG98] geschrieben.

5.2.1 Systemanforderungen - Server

Hier werden die Software-Werkzeuge genannt, die auf dem *EasyContent*-Server installiert und konfiguriert werden sollen.

- *Apache Web Server*
- *Skriptsprache* PHP4
- Das Programm *wget* zum Download kompletter Datenbäume
- MySQL - *Datenbankserver*

Die Installation und die Konfiguration dieser Werkzeuge ist den entsprechenden Dokumentationen zu entnehmen.

5.2.2 Systemanforderungen - Client

Als Klientanwendung für *EasyContent* sind alle Browser einsetzbar, die

- eine graphische Oberfläche bieten
- *Datei-Upload*s unterstützen
- *JavaScript1.1* verstehen

Die meisten heute verwendeten Web-Browser erfüllen diese Kriterien. Bei der Implementierung wurden folgende Browser getestet:

- *Netscape Navigator* ab Version 4.0
- *Microsoft Internet Explorer* ab Version 4.0
- *Konqueror Web Browser* 1.9.8

5.3 Die Realisation der *CMS* - Funktionen

In diesem Abschnitt wird ausführlich erklärt, wie die einzelnen *CMS*-Funktionen in *Easy-Content* realisiert wurden. Es wird auf die Verwendung und die Konfiguration der angewandten „low-cost" Software-Elemente eingegangen.

5.3.1 Trennung von Inhalt und Layout mit *PHP4*

Die Trennung von Inhalt und Layout - siehe Abschnitt 2.2 - erfolgt in *EasyContent* mit Templates. Diese sind Layout-Vorgaben, die die Generierung der Seiten anhand der aktuellen Web-Inhalte steuern. Die Templates werden im *EasyContent*-Prototypen mit der Skriptsprache *PHP4* erstellt, d.h. der *TemplateProcessor* entspricht im Grunde genommen dem im System installierten *PHP*-Interpreter.
Unter *Linux* kann *PHP4* entweder als *CGI*-Programm oder als *Apache*-Modul installiert werden. Im Prototypen geht es um die Konfiguration als *Apache*-Modul.

Die Generierung einer dynamischen Web-Seite mit dem Einsatz von *PHP4* erfolgt - siehe Abbildung 5.1 - auf folgende Weise:

1. Der Browser ruft eine *PHP*-Seite ab
2. Der Web-Server ruft den installierten *PHP*-Interpreter auf, der den *PHP*-Code verarbeitet. Der *PHP*-Interpreter stellt die Seite anhand eventueller Datenbankeinträge und anderer aktuellen Inhalte dynamisch zusammen.
3. Der Web-Server schickt die Ausgabe des *PHP*-Interpreters zum Browser zurück

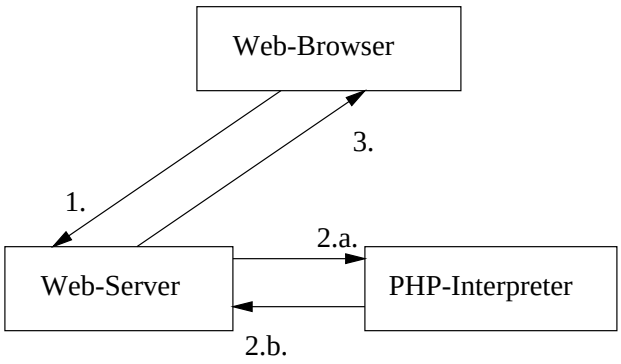

Abbildung 5.1: Ausführung eines *PHP*-Programms

Das folgende Programm-Beispiel zeigt ein *PHP4*-Template, das die Inhalte der Datenbanktabelle *TestTabelle* aus der *MySQL*-Datenbank *TestDB* auf eine Web-Seite schreibt. (Die Zeilen, die mit dem Zeichen # beginnen, sind Kommentare.)

```
<HTML>
  <BODY>
```

```
    <?
      #es wird eine Verbindung zum MySQL-Server aufgebaut
      $conn=mysql_pconnect("my_server","my_login","my_password");

      #es wird eine Datenbank ausgewaehlt
      $db=mysql_select_db("TestDB",$conn);

      #Datensaetze werden gelesen
      $sql="select * from TestTabelle";
      $result=mysql_query($sql,$conn);

      #Datensaetze werden verarbeitet
      while($row=mysql_fetch_array())
      {
        echo $row["feld1"] . " " . $row["feld2"] . "<BR>";
      }
    ?>
  </BODY>
</HTML>
```

Neben der Funktionalität von *PHP4* stehen den Entwicklern auch einige *EasyContent*-spezifische Funktionen zur Verfügung. Diese sind :

- Die Funktion $curl()$ dient zur Erstellung von Hyperlinks zwischen den Templates. Diese hat drei Parameter: den logische Namen des Zieltemplates, den Text des Hyperlinks und die übergebenen Parameter. Sie kann folgendermaßen verwendet werden: $curl(''zieltemplate.ez'','' name'','' param1 = value1'')$.
- Die beiden Funktionen $inserttext()$ bzw. $insertpic()$ dienen dazu, textuelle Assets bzw. Bilder in eine Web-Seite einzufügen. Der Code $inserttext(''projekt1 - text1'')$ in einem Template bewirkt, daß der Inhalt des textuellen Assets $projekt1 - text1$ auf die zu erstellende Web-Seite geschrieben wird.

5.3.2 Logische Adressierung der Templates

Wie schon im Abschnitt 4.6 erwähnt, werden die Projekte, die Assets und die Templates in *EasyContent* in einer Baumstruktur verwaltet. Theoretisch können beliebig viele Projekte und Unterprojekte erstellt werden, es gibt keine Beschränkungen bezüglich der Tiefe und der Größe des Projektenbaumes. Die einzelnen Inhalte werden mit ihren Namen adressiert, diese Namen repräsentieren den jeweiligen Pfad in der Baumstruktur. So ist z.B. der logische Name des Templates *MyTemplate* in dem Top-Level-Projekt *MyProject* `/MyProject-MyTemplate.ez`. (Die Endung `.ez` bedeutet, daß es um ein *EasyContent*-Template geht, die Namen der Assets haben keine solchen Endungen.)

Die physikalische Speicherung der Inhalte auf dem Web-Server erfolgt „flach”. Sowohl die aktuellen Templates und Assets als auch ihre Versionen befinden sich in der *MySQL*-Datenbank, wo sie mit ihren numerischen Template-*ID*s - Template-Identifikatoren - eindeutig adressiert werden können.
Des weiteren werden die jeweils aktuellen Versionen der Templates auch in einem Verzeichnis - `/temppool` - des Dateisystems abgelegt, weil der *Apache*-Webserver grundsätzlich nur auf Dateien im Verzeichnissystem zugreifen kann. Dieses Verzeichnis entspricht dem sog. *Temppool*, das im Abschnitt 4.5 definiert wurde.
Die Speicherung der Templates in dem Verzeichnis `/temppool` hat eine flache Struktur. Das Template mit dem numerischen Identifikator `xxx` wird hier unter dem Namen `/temppool/xxx.php` abgelegt. Unabhängig davon, wie kompliziert die logische Baumstruktur der Templates wird, bleibt ihre physikalische Speicherung dennoch flach und recht einfach.

Aufgabe des *EasyContent*-Prototypen ist die Abbildung der logischen Namen der Inhalte auf die entsprechenden physikalischen Adressen. Dies wird hier am Beispiel der Templates erklärt.
Das *CMS* soll die vom Browser erhaltene logische Adresse verarbeiten und die dazugehörige Datei aus dem Verzeichnis `/temppool` heraussuchen. Diese Datei wird auf der Serverseite vom *PHP-Parser* verarbeitet und an den Browser zurückgeschickt. Abbildung 5.2 veranschaulicht diesen Vorgang.

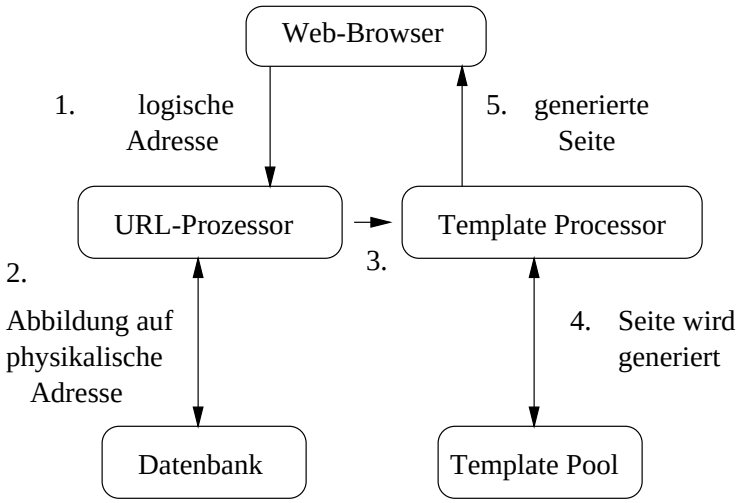

Abbildung 5.2: Abbildung der logischen Namen auf die physikalischen Adressen

Die einzelnen Schritte in der Abbildung haben folgende Bedeutung:

1. Der Browser wendet sich mit einer logischen Adresse, also der *URL* an das CMS
2. Der *URL-Processor* kommuniziert mit der System-Datenbank und bestimmt die physikalische Adresse des Templates
3. Die physikalische Adresse des Templates wird dem *TemplateProcessor* übergeben.

4. Der *PHP-Parser* nimmt das Template aus dem *Template Pool* und interpretiert es

5. Die generierte *HTML*-Seite wird schließlich an den Browser zurückgesandt

Um diese Abbildung zwischen logischen und physikalischen Adressen zu ermöglichen, bietet der *Apache*-Webserver mehrere Möglichkeiten.

Logische Namenabbildung mit dem Modul *mod_rewrite*

Das Modul *mod_rewrite* des *Apache*-Webservers ist ein komplexes Werkzeug für die Bearbeitung der *URL*s „on-the-fly". Es implementiert eine auf regulären Ausdrücken basierende *Rewrite Engine*. Mit der Hilfe des Moduls lassen sich die *URL*s umschreiben (*rewrite*) und auf die physikalische Sicht des Dateisystems abbilden.
Die genaue Funktionsweise des Moduls ist in der *Apache*-Referenz [Fou00] oder in [Bow00] ausführlich besprochen, hier wird nur ein praktisches Beispiel genannt.
Nehmen wir an, daß wir alle *URL*s mit der Endung `.ez` auf die Templates in *EasyContent* abbilden wollen. Die folgenden Einträge in der Konfigurationsdatei von *Apache* ermöglichen, daß alle solchen *URL*s als Eingabe-Parameter an das *PHP4*-Skriptprogramm `TemplateProcessor.php` weitergegeben werden. Das Programm kann anhand der Eingabe den entsprechenden Eintrag aus der Datenbank holen, den *PHP4*-Interpreter aufrufen, die Seite generieren und das Ergebnis an den Browser zurücksenden.

```
RewriteEngine On
RewriteBase /ez_content/
RewriteRule ^(.+\.ez)/$ /TemplateProcessor.php
```

Logische Namenabbildung mit dem Modul *mod_mime*

Eine weitere, ebenfalls fundierte Methode für die Abbildung logischer Namen ist der Einsatz des Moduls *mod_mime*. Im Prototypen von *EasyContent* wird diese Methode implementiert.
MIME steht für **Multipurpose Internet Mail Extensions**. Es ist ein Mechanismus zum Transport von Dateien mit verschiedenen Dateitypen im Internet. Die zu übertragenden Dokumente haben einen *MIME*-Typ, der sowohl beim Sender als auch beim Empfänger bekannt ist. Die beiden kommunizierenden Seiten können mit der Hilfe vom *MIME*-Typ eines Dokumentes herausfinden, wie dieses verarbeitet werden soll.
Mit dem Modul *mod_mime* des *Apache*-Servers ist es möglich, neue *MIME*-Typen einzurichten bzw. Aktionen, die beim Abruf eines Dokumentes mit einem bestimmten *MIME*-Typ ausgeführt werden sollen, zu definieren.
In unserem Beispiel wollen wir die *URL*s mit der Endung `.ez` verarbeiten. Aus diesem Grunde fügen wir der *Apache*-Konfigurationsdatei `httpd.conf` folgende Zeilen hinzu:

```
AddType application/x-ez_content .ez
Action application/x-ez_content /ez_content/TemplateProcessor.php
```

Die Direktive `AddType` bewirkt, daß der *Apache*-Webserver die Dateien mit der Endung `.ez` zum *MIME*-Typ *application/x-ez_content* zuordnet. Wenn also der Browser ein Dokument mit der Endung `.ez` abruft, so weiß der Server, daß es sich um ein Dokument mit dem Typ *application/x-ez_content* handelt.
Der Befehl `Action` in der zweiten Zeile bestimmt nun, was der Server mit einem Dokument mit dem *MIME*-Typ *application/x-ez_content* machen soll. Alle Zugriffe auf Dokumente von diesem Typ werden an das Skriptprogramm `TemplateProcessor.php` weitergeleitet.
Im Endeffekt wird beim Abruf einer Seite mit der Endung `.ez` das Skriptprogramm `TemplateProcessor` gestartet. Es bekommt vom *Apache*-Webserver u.a. die folgenden wichtigen Parameter:

$PATH_INFO Diese Variable enthält den ursprünglichen logischen Namen, der vom Browser abgerufen wurde.

$PATH_TRANSLATED Diese Variable zeigt auf die physikalische Adresse, die vom Server tatsächlich aufgerufen wird.

Wenn also bei der oben beschriebenen Konfiguration der Browser die Seite `/MyProjekt-Foo.ez` abruft, so hat die Variable $PATH_INFO den Wert `/MyProjekt-Foo.ez`, und die Variable $PATH_TRANSLATED den Wert `TemplateProcessor.php`. Das aufgerufene Skriptprogramm kann anhand dieser Infomationen das entsprechende Template aus der *MySQL*-Datenbank heraussuchen und die Seite generieren.

5.3.3 Versionierung mit *MySQL*

Im Abschnitt 4.8.3 wurde das Konzept der Versionierung der Inhalte in *EasyContent* vorgestellt. Nun wird die Verwirklichung dieser Ideen diskutiert.

Die Speicherung der Versionen erfolgt im *EasyContent*-Prototypen in der *MySQL*-Datenbank. Hier werden alle relevanten Informationen über die archivierten (Unter-)Projekte und Templates abgelegt.
Die Versionen von ganzen Projekten sind auch einzelne Bäume, d.h. es sollen ganze Baumstrukturen in den Datenbanktabellen dargestellt werden. Aus diesem Grunde werden in der Systemdatenbank u.a. die folgenden Tabellen verwendet - siehe Abbildung 5.3 auf Seite 79.

- Die Datenbanktabelle `EZ_PROJ_VERSION` verwaltet Informationen über das ganze versionierte Unterprojekt, wie z.B. Erstellungsdatum, kurze Bemerkung des Erstellers etc.
- Ein versioniertes (Unter)Projekt besteht aus einer hierarchischen Struktur von Projekten, die in der Tabelle `EZ_PPROJ_VERSION` beschrieben werden. Da Projekte beliebig tief ineinander geschachtelt werden können, sind die einzelnen Einträge in der Tabelle miteinander rekursiv verknüpft.
- Jedes Projekt enthält eine Vielzahl von Assets und Templates, die in den Tabellen `EZ_TEMP_VERSION`, `EZ_TEXT_VERSION`, `EZ_BIN_VERSION` beschrieben werden.

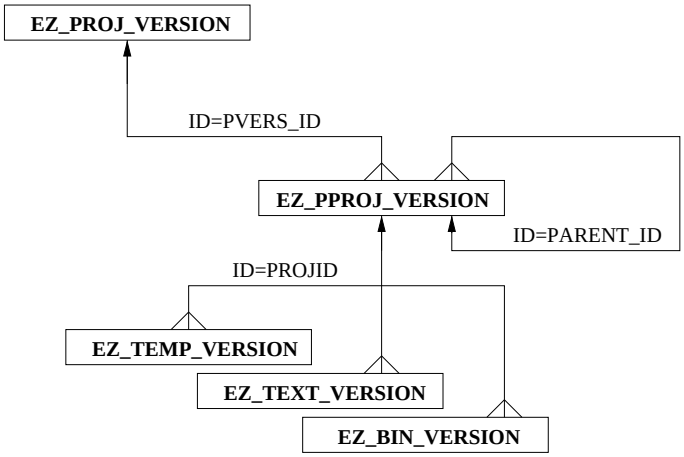

Abbildung 5.3: Speicherung der Versionen

Bei der Speicherung eines (Unter)Projektes muß *EasyContent* alle unter dem gegebenen Projekt befindlichen Unterprojekte, Templates und Assets rekursiv suchen und archivieren. Da die einzelnen Einträge auch untereinander verknüpft sind, ist die Reihenfolge ihrer Speicherung sehr wichtig. Ein bestimmtes Unterprojekt darf natürlich nicht archiviert werden, solange sein „Vater" *(Ancestor)* noch nicht versioniert wurde, d.h. der Algorithmus muß systematisch vorgehen.
Der *Versionen-Manager* besucht die zu archivierenden Assets und Templates nach der sog. *Breadth First Search* Strategie, oder *Breitensuche*. Bei der *Breitensuche* wird von vornherein die Grundstrategie eingesetzt, zunächst alle nächsten Knoten eines Knoten zu besuchen und dann erst in die Tiefe zu gehen - siehe Abbildung 5.4. Eine genauere Erklärung des Algorithmus befindet sich z.B. in [RIS98].

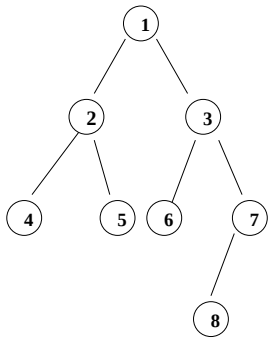

Abbildung 5.4: Breitensuche - Breadth First Search

5.3.4 Datenbankverwaltung mit *phpMyAdmin*

Im Abschnitt 4.8.4 wurde vorgestellt, daß die *Entwickler* in *EasyContent* die Möglichkeit haben, aus ihren Templates auf eigene relationale Datenbanken zuzugreifen. Dadurch wird die Programmierung datenbankgestützter, dynamischer Web-Seiten weitestgehend unterstützt.

Die Erstellung und die Manipulierung dieser Datenbanken ist die Verantwortung des *Administrators*, der diese Aufgabe auf einer web-basierten Oberfläche durchführt.

Im Prototypen von *EasyContent* wird die Verwaltung der Datenbanken mit der Integration von *phpMyAdmin* verwirklicht. *phpMyAdmin* ist eine in *PHP* implementierte web-basierte Oberfläche für die Bearbeitung von *MySQL*-Datenbanken, die von *Tobias Ratschiller* entwickelt wurde und frei zur Verfügung steht.
Durch eine entsprechende Konfiguration ist *phpMyAdmin* in *EasyContent* integriert. Die Administratoren können *phpMyAdmin* aus der web-basierten Adminoberfläche aufrufen und so die Datenbanken manipulieren.

Die Anwendung von *phpMyAdmin* ist wieder ein gutes Beispiel dafür, wie einfach und elegant man aus *Open-Sorce*-Bauelementen komplizierte Systeme zusammenbauen kann. Die ausführliche Erklärung der Funktionen von *phpMyAdmin* ist hier nicht beabsichtigt, für die Dokumentation siehe [Rat00], [RG00].

5.3.5 Personalisierung mit *htaccess*

htaccess ist ein eingebauter Dienst des *Apache*-Webservers für die Personalisierung von Nutzern. Mit der Hilfe von *htaccess* ist es möglich, bestimmte Bereiche der Web-Site durch Authentifizierung zu schützen. Das bedeutet, daß die Nutzer sich mit ihren Login-Namen und Kennworten anmelden sollen, um Zugang zu den Seiten im geschützten Verzeichnis zu erhalten. Nach einer erfolgreichen Anmeldung werden sie während ihrer ganzen Sitzung - *Session* - mit der Umgebungsvariablen `$REMOTE_USER` identifiziert.

In diesem Abschnitt wird zunächst auf die Arbeitsweise von *htaccess* im allgemeinen eingegangen.
Nehmen wir an, daß wir das virtuelle Verzeichnis `mycontent` durch Authentifizierung schützen wollen. So müssen wir zunächst in die Konfigurationsdatei `httpd.conf` folgende Zeilen eintragen.

```
Alias /mycontent/ "/var/www/html/mycontent/"
<Directory "/var/www/html/mycontent">
      AllowOverride AuthConfig FileInfo
</Directory>
```

Dadurch wird das virtuelle Verzeichnis erstellt und die Verwendung von *htaccess* im Verzeichnis genehmigt. Als folgender Schritt muß im Dateiverzeichnis `/var/www/html/mycontent` eine Datei namens `.htaccess` mit folgendem Inhalt angelegt werden:

```
AuthName "Authorization for MyContent"
AuthType Basic
AuthUserFile /usr/local/passwd/.htpasswd
require valid-user
```

Der Eintrag `AuthName` bestimmt nur die Aufschrift des Login-Fensters, das bei der Authentifizierung erscheint. `AuthType` bedeutet, daß eine einfache Authentifizierung im sog.

Basic-Mode erfolgt. Der Eintrag `AuthUserFile` zeigt auf die Datei, in der die Login-Namen mit den entsprechenden Kennworten abgelegt sind. `require valid-user` bedeutet, daß zum geschützten Bereich nur jene Nutzer Zugang haben, die sich erfolgreich anmelden können.
Nun müssen nur noch die entsprechenden Login-Kennwort-Paare in die Datei `.htpasswd` im Verzeichnis `/usr/local/passwd/` eingetragen werden. Dazu verwendet man das Kommandozeilenprogramm *htpasswd*, das ein Teil aller *Apache*-Distributionen ist.

```
>htpasswd /usr/local/passwd/.htpasswd username
>New password:
>Re-type new password:
```

Dieses Kommando bewirkt, das zur Datei `.htpasswd` der Login-Name *username* und das entsprechende kodierte Kennwort hinzugefügt werden.

In *EasyContent* werden die Nutzer mit *htaccess* authentifiziert. Außerdem werden in der Datenbanktabelle `EZ_USER` - siehe 5.4 - die wichtigsten Informationen bezüglich der einzelnen Nutzer gespeichert.
Bei der Anmeldung eines Nutzers wird auf der Server-Seite die Variable `$REMOTE_USER` gesetzt. In der Tabelle `EZ_USER` entspricht der Wert dieser Variablen dem Feld *login*. Wenn der Administrator einen neuen Nutzer hinzufügt, wird automatisch sowohl ein neuer Eintrag in der Tabelle *ez_user* als auch einer in der Datei `.htpasswd` erzeugt.
So wird ein zweistufiger Mechanismus für die Zugangskontrolle der Nutzer implementiert - siehe Abbildung 5.5 auf Seite 82.

1. Die Nutzer, die in *EasyContent* arbeiten wollen, müssen sich zunächst über *htaccess* authentifizieren. Nach einer erfolgreichen Anmeldung setzt der *Apache*-Server die Variable `$REMOTE_USER`.

2. Ist die Variable `$REMOTE_USER` gesetzt, so kann *EasyContent* die genauen Zugangsrechte des Nutzers zu den System-Ressourcen anhand der Datenbanktabelle `EZ_USER` ermitteln. Das System bestimmt den numerischen Identifikator des Nutzer und speichert ihn in der Sessionvariablen *who_am_i*.

5.3.6 Implementierung der Workflowmechanismen

Check-In und *Check-Out*

Um mehrfache Zugriffe zu unterstützen, werden die Inhalte in *EasyContent* mit *Meta-Attributen* versehen, die ebenfalls in der Systemdatenbank verwaltet werden. Diese sind:

- Jedes Template, Text- bzw. Bild-Asset wird mit dem *boolschen* Attribut `LOCKED` beschrieben. Dieses gibt an, ob der Inhalt von jemandem bearbeitet wird oder nicht. (Die Templates, text- bzw. Bild-Assets werden in den *MySQL*-Tabellen `EZ_TEMPLATE`, `EZ_TEXTASSET` und `EZ_BINASSET` gespeichert, siehe 5.4)

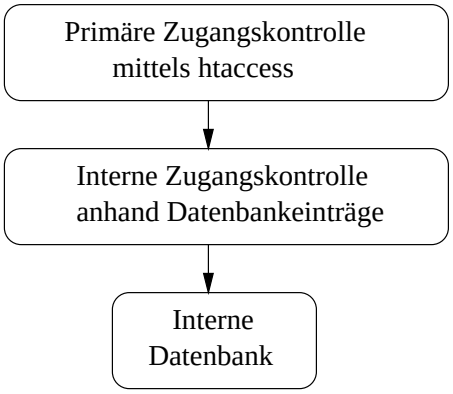

Abbildung 5.5: Zweistufige Zugangskontrolle in *EasyContent*

- Wenn der Inhalt an jemanden ausgeliehen ist, so wird das Attribut `LOCKING_USER` gesetzt und zeigt den numerischen Identifikator des Bearbeiters an.

Die grafische Oberfläche des Projektexplorers wird für jeden Nutzer dynamisch generiert. Anhand der Attribute werden die einzelnen Templates und Assets unterschiedlich angezeigt. Es werden z.B. verschiedene Schriftarten, Ikone und Farben verwendet, um das Attribut eines Inhaltes grafisch darzustellen.

Die Realisierung des *4-Augen-Prinzips*

Die Implementierung des *4-Augen-Prinzips* erfolgt ebenfalls mit *Meta-Daten*. Jedes Template wird mit einem numerischen Status versehen, der in der Datenbanktabelle `EZ_TEMPLATE` unter dem Namen `STATUS` gespeichert wird. Dieses Feld kann die folgenden Werte haben - vgl. 4.10.2:

- 0: offline und unter Bearbeitung
- 1: offline und zur Kontrolle vorgelegt
- 2: online und konsistent
- 3: online und verändert
- 4: online, verändert und zur Kontrolle vorgelegt

Benachrichtigungen

In dem *EasyContent*-Prototypen wurde ein interner, asynchroner Benachrichtigungsmechanismus für den Informationsaustausch zwischen den Mitarbeitern verwirklicht.

Die Nachrichten werden in der MySQL-Datenbanktabelle `EZ_MESSAGE` abgelegt. Diese Tabelle hat die folgenden Felder:

- `ID:` Eindeutiger numerischer Identifikator der Meldung

- `VON:` ID des Absenders
- `AN:` ID des Empfängers
- `BETREFF:` Betreff der Meldung
- `GELESEN:` Dieses Attribut zeigt an, ob die Meldung schon gelesen ist
- `WANN:` Datum und Uhrzeit der Versendung der Meldung
- `TEXT:` Der Text der Meldung

Das Senden und Empfangen von Meldungen wird vollständig von der internen Programmierlogik gesteuert:

1. Die Erstellung einer Meldung bedeutet, daß ein neuer Eintrag in der Datenbanktabelle `EZ_MESSAGE` erzeugt wird
2. Die web-basierte Oberfläche zeigt die neuen Meldungen für die einzelnen Nutzer an
3. Nach dem Lesen einer Meldung wird das Attribut `GELESEN` des entsprechenden Eintrages auf 0 gesetzt.
4. Beim Löschen einer alten Meldung wird der entsprechende Eintrag aus der Datenbank entfernt.

Die datenbankgestützte Implementierung der Meldungen hat gegenüber der Anwendung externer *E-Mail*-Lösungen einige Vorteile:

- Die Installation externer *Mail*-Systeme ist nicht erforderlich.
- Da alle Meldungen in einer einzelnen Datenbank abgelegt werden, kann das ganze Benachrichtigungssystem zentral verwalten werden.
- Die Erstellung von *Mailing-Lists* ist einfach.
- Die Versendung von systeminternen Nachrichten ist durch entsprechende Programmierung möglich.

5.3.7 Staging mit *Apache*

Im Abschnitt 4.11 wurde das Konzept von *Staging* in *EasyContent* schon vorgestellt. In dem Prototypen wurde *Staging* folgendermaßen implementiert:
Der Entwicklungs- und der Live Web-Server befinden sich auf dem gleichen Rechner. Im Grunde genommen laufen im System zwei Exemplare von *Apache*, das eine auf dem Port 80, das andere auf dem Port 8091.
Der Live Web-Server verwendet die Standard-Portnummer 80. Der Zugriff auf die Dokumente auf diesem Server ist so für das ganze Internet gestattet.

Die Portnummer des Entwicklungs-Servers ist 8091. Der Zugriff auf die Seiten auf diesem ist erst nach einer entsprechenden Authentifizierung mit *htaccess* erlaubt. Außerdem ist es mit einer entsprechenden Konfiguration des Netzwerkes möglich, Dienste mit dieser Portnummer nur für das lokale *LAN* erreichbar zu machen.

Die Konfiguration der Portnummer erfolgt in den *Apache*-Konfigurationsdateien `httpd.conf` und `httpd2.conf`:

```
Port 80      #Eintrag in der Datei httpd.conf
Port 8091    #Eintrag in der Datei httpd2.conf
```

Beim Starten von *EasyContent* werden zwei Examplare von *Apache* mit verschiedenen Konfigurationsdateien gestartet. Im Prototypen erfolgt das mit der Ausführung der folgenden Befehle:

```
$/usr/sbin/httpd -f /etc/httpd/conf/httpd.conf
$/usr/sbin/httpd -f /etc/httpd/conf/httpd2.conf
```

5.3.8 Caching mit *wget*

Wie schon erklärt, kann man in *EasyContent* sowohl „gecachte" als auch „nicht-gecachte" Templates erstellen. Der *TemplateProcessor* überprüft beim Abruf jeder Seite, ob sie „gecacht" ist oder nicht und handelt dementsprechend. Nicht „gecachte" Seiten werden immer wieder neu generiert, „gecachte" Seiten dagegen nur einmal und dann als statischer *HTML*-Code gespeichert.

Im Abschnitt 5.3.2 wurde beschrieben, daß die fertigen Templates standardmäßig als reine *PHP4*-Dateien im Verzeichnis `/temppool` abgelegt werden. Mit dem Einsatz von „gecachten" Templates wird dieser Mechanismus erweitert. Neben den *PHP4*-Skripten werden im Verzeichnis `/temppool` auch schon fertige, statische *HTML*-Seiten gespeichert, die als Output der „gecachten" Templates entstanden sind. Das Template mit dem Identifikator `yyy` wird hier also zweifach abgelegt: sowohl dynamisch als `/temppool/yyy.php`, als auch statisch als `/temppool/yyy.html`.

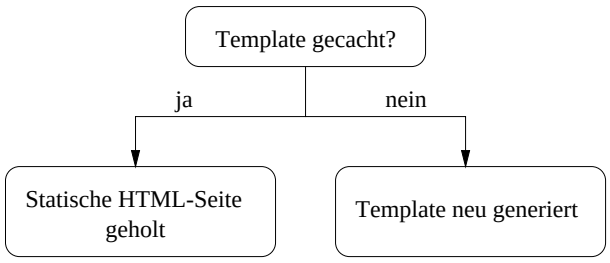

Abbildung 5.6: „Gecachte" bzw. „nicht-gecachte" Seiten

Die Generierung von Templates erfolgt mit *wget*. Es handelt sich um ein freies Software-Werkzeug, das das nicht-interaktive Herunterladen von Web-Sites ermöglicht. Mit *wget* kann

man große Web-Seiten mit komplexen Verzeichnisstrukturen ohne jegliche Interaktion rekursiv besuchen und ihre Inhalte in reinem *HTML*-Format speichern.
Bei der Erstellung der statischen *HTML*-Seiten ruft also das CMS die zu generierende *PHP*-Code mit *wget* auf. Die Ausgabe von *wget* wird als *HTML*-Datei im Verzeichnis `/temp-pool` gespeichert. Beim nächsten Abruf einer „gecachten" Seite wird diese Datei ohne Bearbeitung an die Browser geschickt.
Das Flußdiagramm in der Abbildung 5.6 auf Seite 84 zeigt das Verhalten des *TemplateProcessor*s beim Abruf von „gecachten" und „nicht-gecachten" Seiten.

5.4 Das Datenbankmodell des Systems

In diesem Abschnitt wird das Datenbankmodell des Systems vorgestellt. Eine ausführliche Besprechung der einzelnen Tabellen und Felder ist hier nicht beabsichtigt, es werden nur die wichtigsten Strukturen der *EasyContent*-Datenhaltung vorgestellt.

Abbildung 5.7 zeigt die Tabellen, die für die hierarchische Speicherung von Projekten, Templates bzw. von textuellen und binären Assets zuständig sind. Die Projekte können rekursiv ineinandergeschachtelt werden, die Templates und Assets referenzieren die Projekte mit ihrem `PROJEKT_ID`.

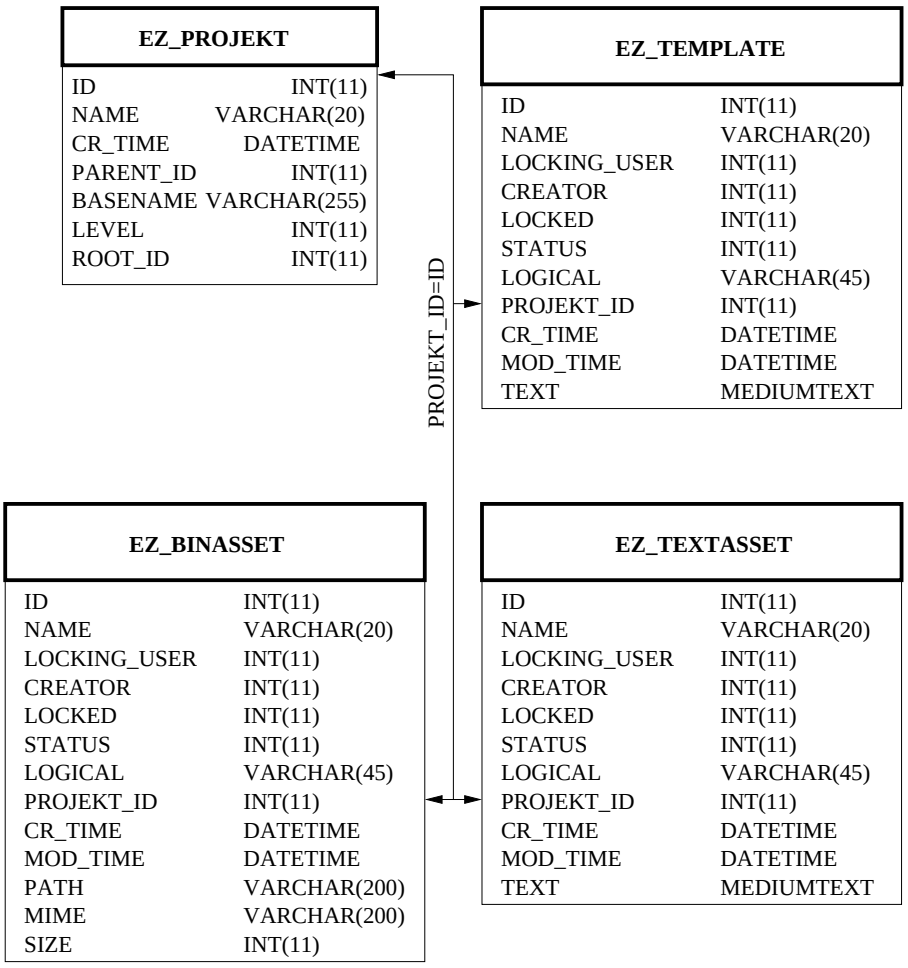

Abbildung 5.7: Projekte, Templates, textuelle und binäre Assets

Die nächste Abbildung (Abb. 5.8, Seite 87) zeigt die Zuordnung der Nutzer zu den Projekten. In der Tabelle `EZ_USER` werden die Nutzer verwaltet. Zu einem Projekt können mehrere Benutzer gehören, außerdem kann ein Nutzer Mitglied mehrerer Projekte sein. Es besteht also eine $n : n$-Beziehung zwischen den beiden Datenbankrabellen `EZ_PROJEKT` und `EZ_USER`, die durch eine dritte Tabelle (`EZ_MITGLIEDER`) logisch miteinander verknüpft sind.

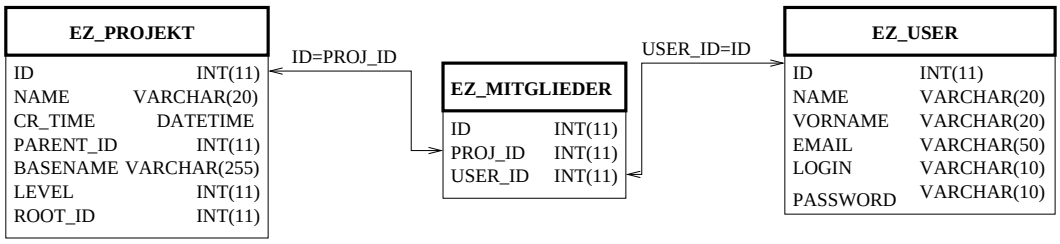

Abbildung 5.8: Zuordnung der Nutzer zu den Projekten

Die letzte Abbildung (5.9) stellt die Verwaltung von Versionen dar. Über diesen Mechanismus wurde schon ausführlich im Abschnitt 5.3.3 geschrieben, hier werden die betroffenen Tabellen noch einmal detailliert dargestellt.

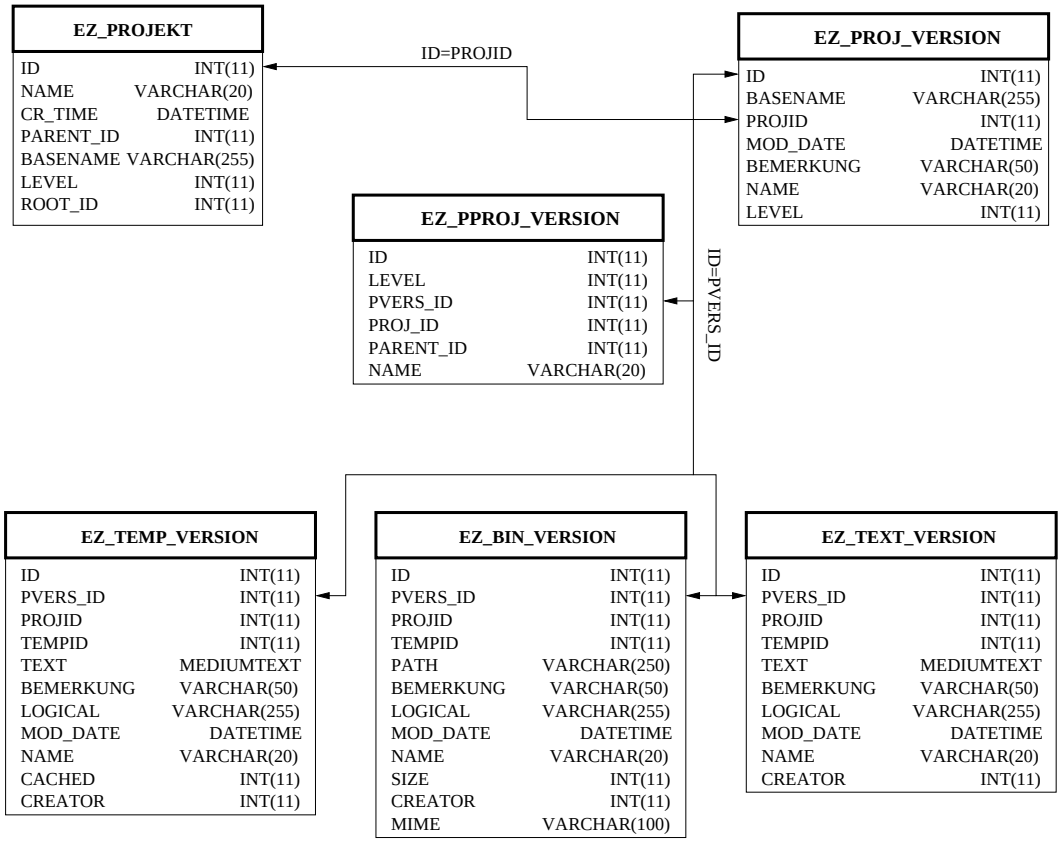

Abbildung 5.9: Die Datenbanktabellen der Versionierung

5.5 Die Möglichkeit der Portierung zu *Windows*

Der Prototyp von *EasyContent* wurde unter dem Betriebssystem *Linux* entwickelt. Es wurde jedoch bereits bei der Konzeption des Systems die Plattformunabhängigkeit als eine der wesentlichsten Zielsetzungen genannt.
Da die 32-bit *Windows*Umgebungen heute eine sehr wichtige Rolle spielen, kann man auf ihre Anwendung nicht verzichten. Aus diesem Grunde werden hier die Möglichkeiten einer eventuellen Portierung zu Windows diskutiert.

5.5.1 Portierung des Web-Servers

Da der *Apache*-Webserver auch unter *Windows* existiert, ist seine Portierung mit ziemlich geringem Aufwand möglich. Es sollen lediglich nur die beiden Konfigurationsdateien `httpd.conf` bzw. `httpd2.conf` der beiden *EasyContent*-Webserver verändert und an die *Windows*-Umgebung angepaßt werden. (Die Veränderung bedeutet meistens einfach nur die Modifizierung der entsprechenden Pfadnamen der betroffenen Verzeichnisse und Dateien im Dateisystem.)

5.5.2 Portierung der Programmierlogik

Die Skriptsprache *PHP4* ist auch weitestgehend plattformunabhängig. *PHP*-Interpreter stehen für *Windows*-Umgebungen sowohl als Quell-Code als auch vorkompiliert zur Verfügung. Die Portierung der Programmierlogik bedeutet auch in diesem Fall nur die Modifizierung jener Skripte, die auf das Dateisystem zugreifen.

5.5.3 Datenhaltung unter *Windows*

Die letzte Frage, die bei einer eventuellen Portierung diskutiert werden soll, ist die Implementierung der Datenhaltung in *Windows*-Umgebungen. Da *MySQL* auf 32-bit *Windows*-Plattformen einsetzbar ist, ist es auch kein Problem. Ein eventueller Datentransfer zwischen *MySQL*-Datenbanken auf verscheidenen Plattformen ist mit der Hilfe von *mysqldump* - siehe [YRK99] - auch ziemlich einfach.

5.6 Zusammenfassung

In diesem Kapitel wurden die interessantesten Aspekte und Ideen der Implementierung von *EasyContent* zusammengefaßt.
Ziel war es nicht, ein ausführliches Benutzer- bzw. Programmierhandbuch anzufertigen. Es wurde versucht darzustellen, wie preiswerte Software-Komponenten konfiguriert und miteinander integriert werden können, um grundsätzliche *CM*-Funktionen zu realisieren. Eine ausführlichere Anleitung für Benutzer ist u.a. auch der in *EasyContent* implementierten *Online-Hilfe*-Funktion zu entnehmen.

Kapitel 6

Diskussion und Ausblick

In dieser Arbeit wurde versucht, einen Überblick in das Thema „*Content Management*" zu geben.

Zunächst wurden die wichtigsten Eigenschaften von *Web Content Management* zusammenfassend diskutiert, daneben wurde auf einige Forschungsergebnisse in diesem Umfeld eingegangen.
Im zweiten Teil der Arbeit wurden Kriterien zur Evaluierung von Content Management Systemen aufgestellt, anhand deren heute vorhandene Werkzeuge untersucht und klassifiziert wurden. Großer Wert wurde dabei auf die Analyse preiswerter *Open-Source*-Lösungen gelegt.
Als praktischer Teil der Arbeit wurde das *WCMS EasyContent* konzipiert und in einer preiswerten Umgebung realisiert. Anschließend, im Kapitel 5 wurden die wichtisgten Probleme und Lösungen der technischen Implementierung dargestellt.

Zum Schluß werden einige Vorschläge zur Erweiterung des entwickelten Systems genannt. Des weiteren wird ein Ausblick auf mögliche künftige Forschungsarbeiten gegeben.

6.1 Erweiterungsmöglichkeiten für *EasyContent*

Es wurde versucht, ein System zu implementieren, das eine Vielzahl von *CM*-Funktionen realisiert. *EasyContent* bietet eine einfache, übersichtliche, benutzerfreundliche Arbeitsumgebung für die Verwaltung digitaler Web-Inhalte. Die in der Aufgabenstellung genannten Zielsetzungen wurden bei der Implementierung des Werkzeuges erfüllt.

Nun sollen einige Aspekte bzw. eventuelle Besserungen genannt werden, die für Weiterentwicklung von *EasyContent* relevant sind:

Verwaltung von Streaming-Media Formaten: Multimediale Anwendungen spielen in der Welt des Web eine zunehmende Rolle. Immer mehr Anbieter stellen Dokumente in Streaming-Formaten bereit, die von einer Web-Site herunterladbar sind. Um diese zu verwalten, ist der Einsatz von Streaming-Servern erforderlich. Es wäre eine interessante Aufgabe, die Verwaltung solcher zeitvarianter Inhalte - z.B. durch die Einbindung entsprechender Anwendungen - zu konzipieren.

Im- und Exportschnittstellen für gängige Dokumentenformate: Die automatische Einbindung von gängigen Dokumentenformaten - wie *Microsoft Word*, *PowerPoint* oder *PDF* - ist heute eine wichtige Anforderung an ein *CMS*. Im Kapitel 3 wurden einige Systeme genannt, die diese Funktion unterstützen. Eine wichtige Aufgabe wäre zu klären, welche Werkzeuge zur Verfügung stehen, die in *EasyContent* integriert werden könnten.

Gleichzeitig wäre es auch nützlich, die im System verwalteten Daten in herkömmlichen Formaten exportieren zu können. Dies könnte mit den Software-Komponenten, die *EasyContent* verwendet, durchaus möglich sein. Die Skriptsprache *PHP4* bietet eine Vielzahl von Funktionen für die Erstellung von *PDF*-Dokumenten, des weiteren auch zur Programmierung der *COM*- bzw. *DCOM*-Schnittstellen, die die Integration entsprechender Exportobjekte ermöglichen.

XML-basierte Schnittstelle für Datenexport: Im Abschnitt 2.3.6 wurde kurz das Protokoll *ICE* vorgestellt, das eine standardisierte, *XML*-basierte Schnittstelle für den Datenaustausch zwischen Inhalts-Anbietern und Kunden ermöglicht. Die Konzeption bzw. Implementierung von einem *ICE-Syndicator* in *EasyContent* wäre eine interessante Aufgabe.

Umstellung auf transaktionssichere Datenbanken mit *MySQL* Bei der Konzeption von *EasyContent* in 4.3.4 wurde erwähnt, daß *MySQL* die Erstellung von transaktionssicheren Datenbanken standardgemäß nicht unterstützt. Seit der letzten Version von *MySQL* können aber auch sog. *BDB*-Tabellen angelegt werden, die transaktionssicher sind. Eine Umstellung auf solche Tabellen würde die Sicherheit des Systems wesentlich steigern.

Verbesserung der Editorenoberfläche Bei der Implementierung von *EasyContent* wurde der Schwerpunkt auf die Realisierung von den grundsätzlichen *CM*-Funktionen gelegt. Die browser-basierte Schnittstelle für die Bearbeitung der Templates bzw. der Text-Inhalte ist ziemlich einfach, neben der Möglichkeit von Datei-Uploads steht zur Zeit nur ein Textfeld für die Erstellung textueller Inhalte zur Verfügung. Es ist wichtig, diese Schnittstelle zu verbessern, und zwar durch die Integration eines Text-Editors. Die Einbindung eines entsprechenden *Java*-Applets wäre nicht kompliziert. Die *Swing*-Klassen von *Java* stellen auch fertige, konfigurierbare *WYSIWYG-HTML*-Editoren bereit, deren Integration auch sehr nützlich sein könnte.

6.2 Forschungsmöglichkeiten in *Content Management*

Content Management ist eine neue Disziplin mit vielen Forschungsfragen. In diesem letzten Abschnitt werden einige genannt.

Content Management außerhalb des Webs: Da *Content Management* heute vor allem in der Welt des *World Wide Web* Anwendung findet, wurden in dieser Arbeit in erster

Linie *Web Content Management Systemen* betrachtet. Es ist aber zu betonen, daß *Web Content Management* nur ein spezielles Teilgebiet von *Content Management* ist.

Die effiziente Verwaltung von Informationen spielt auch in anderen Bereichen eine große Rolle. Vor allem firmeninterne *Document Management Systeme*, die Informations- und Kommunikationsprozesse innerhalb von Unternehmen verwalten, gewinnen immer mehr an Bedeutung. In diesem Bereich entstehen auch viele „CM-spezifische" Probleme, deren Lösung wichtige Themen für künftige Forschungsarbeiten bereitet.

Content Management und Wissensmanagement: Eine ebenfalls wichtige Problematik stellt die kontextbezogene Verwaltung von Informationen dar. Die zur Verfügung stehenden Inhalte sollen auch semantisch bearbeitet werden, um Zusammenhänge zwischen Informationen zu erkennen, intelligente Such- und Recherchefunktionen in Datenbeständen zu implementieren oder z.B. Tendenzen aus Dokumenten erfassen zu können. *Content Management* soll also immer mehr Anknüpfungpunkte an solche Disziplinen wie *Wissensmanagement* oder *Data Mining* haben.

Adaptive Benutzeroberflächen Die im Abschnitt 2.2.4 vorgestellte Personalisierung, d.h. die Anpassung der Präsentation von Inhalten an die Anforderungen der Nutzer ist eines der wichtigsten Merkmale von *CM*-Systemen. Die Inhalte sollen nicht nur gespeichert und verwaltet, sondern auch veröffentlicht werden, wobei die verschiedensten Präsentationsformen unterstützt werden sollen.

Das Erstellen von adaptiven Benutzeroberflächen stellt wichtige Fragen: Wie können Eigenschaften und Präferenzen von Nutzern erfaßt werden? Wie kann die Bereitstellung der Informationen adaptiv angepaßt werden? Unter welchen Umständen ist der Einsatz „gecachter" Seiten auf personalisierten Seiten möglich? All diese Fragen bereiten interessante Forschungsmöglichkeiten.

Verwaltung komponentbasierter multimedialer Dokumente Ein weiteres, relevantes Forschungsthema ist die Verwaltung von komplexen, komponentbasierten Dokumenten. Multimediale Dokumente sind heute aus mehreren, zusammengehörenden Teilobjekten zusammengesetzt, die alle einzeln konfiguriert werden können. Es ist eine wichtige Frage, wie diese Inhalte als Gesamtheit erfaßt, klassifiziert und in einem *Content Management System* echtzeitgeschützt verwaltet werden können.

Glossar

Admin-Interface Komponente von →*EasyContent*, die die Ausführung von administrativen Aufgaben über eine webbasierte Oberfläche ermöglicht *Seite 56*

Apache Der weltweit führende Open-Source *HTTP*-Server, der u.a. auch in →*EasyContent* eingesetzt wird *Seite 73*

Assets Beliebige digitale Inhaltskomponenten in einem →*CMS* *Seite 8*

Assetmanagement Systematische Speicherung, Strukturierung, Verfolgung und Kontrolle der in einem →*CMS* verwalteten →*Assets* *Seite 11*

Cache-Manager Komponente von →*EasyContent*, die „gecachte" Seiten verwaltet *Seite 57*

Caching Vorübergehende, statische Speicherung bestimmter Inhalte auf der Server-Seite in einem →*CMS* *Seite 17*

Check-In Rückgabe eines bearbeiteten →*Assets* an die Gemeinschaft *Seite 16*

Check-Out Das Ausleihen eines →*Assets* an einen Mitarbeiter *Seite 16*

CM siehe →*Content Management* *Seite 7*

CMS siehe →*Content Management System* *Seite 8*

Content Information gespeichert auf einem Datenträger *Seite 7*

Content-Life-Cycle Lebenszyklus von Inhalten auf einer Webseite, der nach dem theoretischen Ansatz in die Phasen *Analyse und Planung*, *Erstellung*, *Kontrolle und Freigabe*, *Publikation*, *Aktualisierung* und *Archivierung* aufgeteilt wird *Seite 8*

Content Management Gesamtheit der geschäftlichen und technischen Prozesse der Aufbereitung, Abfrage, Verwaltung und Veröffentlichung von →*Content* *Seite 7*

Content Management System Komplexe Softwareplattform für die Unterstützung von →*CM*-Funktionen *Seite 8*

Cookie Logische Information in Schlüssel/Wert-Paaren, die die Web-Anwendungen auf den Klienten ablegen *Seite 15*

EasyContent Ein von *Zoltán Fiala* entwickeltes →*WCM*-Werkzeug, das ausschließlich auf Open-Source Softwarekomponenten basiert *Seite 51*

Entwicklungsserver Komponente von →*EasyContent*, die die Entwicklung von neuen Web-Inhalten unterstützt *Seite 57*

Geschütztes Editieren Mechanismus für die Handhabung mehrfacher Schreibzugriffe in einem →*CMS* *Seite 16*

Groupware Mehrbenutzer-Software, die zur Unterstützung von kooperativer Arbeit entworfen und genutzt wird *Seite 25*

Groupware Awareness Das Wissen darüber, was in der Arbeitsgruppe gerade los ist bzw. was in der Vergangenheit in der Gruppe gemacht wurde *Seite 25*

htaccess Dienst des →*Apache*-Servers zur Erstellung geschützter Bereiche auf dem Web-Server, die erst nach einer entsprechender Authentifizierung zugänglich sind *Seite 80*

ICE - Information and Content Exchange Auf *XML* basierendes Protokoll zum standardisierten Datenaustausch zwischen Geschäftspartnern *Seite 32*

Live Web-Server Komponente von →*EasyContent*, die für die Veröffentlichung von freigegebenen, genehmigten Inhalten zuständig ist *Seite 57*

Meta-Data Logische Information für die Beschreibung von →*Content* *Seite 12*

Midgard Open-Source →*WCM*-Werkzeug *Seite 44*

mod_action Modul des →*Apache*-Servers, das die Zuordnung von verschiedenen Inhaltstypen zu serverseitigen Aktivitäten erlaubt *Seite 77*

mod_rewrite Modul des →*Apache*-Servers, das die Manipulierung von *URL*s ermöglicht *Seite 77*

MySQL Open-Source Datenbank, die für schnelle *SQL*-Abfragen optimiert wurde und u.a. auch in →*EasyContent* eingesetzt wird *Seite 73*

Network Productivity System Kommerzielles →*WCMS* der deutschen Firma *Infopark Seite 41*

OpenCMS Open-Source →*WCM*-Werkzeug *Seite 43*

PHP Open-Source serverseitige Programmierungebung für die Erstellung dynamischer, datenbankgestützter Webseiten *Seite 73*

Projekt Grundbegriff von →*EasyContent* für die Bündelung zusammengehörender →*Assets* *Seite 57*

Projekt-Explorer Komponente von →*EasyContent*, die die Erstellung von →*Template*s und →*Assets* ermöglicht *Seite 56*

Push-Modell Modell zur Implementierung von →*Workflows*, bei dem die zu bearbeitenden →*Asset*s vom jeweiligen momentanen Bearbeiter an den nächsten Mitarbeiter weitergereicht werden *Seite 23*

Session Gesamtheit der Tätigkeiten eines Nutzers auf einer Website (Sitzung) *Seite 14*

Staging Trennung der veröffentlichten und der unter Entwicklung stehenden Inhalte in einem →*CMS* *Seite 17*

System-Datenbank Komponente von →*EasyContent*, die alle relevanten Daten über Nutzer und Inhalte speichert *Seite 55*

Template-Pool Komponente von →*EasyContent*, die die aktuellen Versionen von Web-Seiten enthält *Seite 56*

TemplateProcessor Komponente von →*EasyContent*, die die Web-Seiten anhand der im →*Template-Pool* befindlichen →*Templates* und entsprechender Datenbankeinträge erstellt *Seite 57*

Top-Level Projekt Ein →*Projekt* in →*EasyContent*, das sich in der hierarchischen Baumstruktur auf dem obersten Level befindet und in der Zugriffsverwaltung eine wichtige Rolle spielt *Seite 59*

Trennung von Inhalt und Layout Wichtigs Kennzeichen von →*CM*-Systemen für die Erstellung aktueller, datenbankgestützter Webseiten *Seite 9*

Vignette Content Management Server Kommerzielles →*WCMS* der Firma *Vignette* *Seite 38*

Versionierung Systematische Archivierung von Inhalten in einem →*CMS* *Seite 13*

Version-Manager Komponente von →*EasyContent*, die die Erstellung, Verwaltung und das Wiederherstellen von Versionen realisiert *Seite 56*

Vier-Augen-Prinzip Einfach implementierbare Methode für die Genehmigung und Freischaltung von Inhalten *Seite 20*

VIP Content Manager Kommerzielles →*WCMS* der Firma *Gauss* *Seite 40*

WCM siehe →*Web Content Management* *Seite 8*

Web Content Gesamtheit der Inhalte einer Web-Seite *Seite 7*

Web Content Management Systematische Verwaltung und Administration von →*Web Content* *Seite 8*

Workflow Ein Workflow stellt einen technisch umfassend unterstützten Arbeitsablauf dar, der ausgehend von einem auslösenden Ereignis entlang einer definierten Kette von Teilschritten bis zu einem definierten Arbeistergebnis führt, wobei der Grad der Vervollständigung des Arbeitsergebnisses mit jedem einzelnen Arbeitsschritt zunimmt. *Seite 19*

Z39.50 Protokoll zur Verbindung von bibliothekarischen Datenbankanwendungen, die in →*CM* immer mehr an Bedeutung gewinnt *Seite 31*

Zope Open-Source Redaktionswerkzeug mit vielen →*WCM*-Funktionen *Seite 45*

Abbildungsverzeichnis

Literaturverzeichnis

[Age00] International Standard Maintenance Agency. *Z39.50*, 2000. http://lcweb.loc.gov/z3950/agency/.

[AMKF+00] Tim Arnold-Moore, Alan Kent, Michael Fuller, Ron Sacks-Davis, and Neil Sharman. Architecture of a content management server for xml document applications. *Web Information Systems Engineering (WISE 2000), Hong Kong, China. 97-108,* 2000.

[Bow00] Rich Bowen. *Apache Server Unleashed*. Sams Publishing, 2000. ISBN 0672318083.

[BSW00] Hans-Jörg Bullinger, Erwin Schuster, and Stephan Wilhelm. *Content Management Systeme - Auswahlstrategien, Architekturen und Produkte*. Verlagsgruppe Handelsblatt GmbH, Wirtschaftswoche, Düsseldorf, 2000.

[BZTZ00] Heino Büchner, Oliver Zschau, Denis Traub, and Rik Zahradka. *Web Content Management - Websites professionell betreiben*. Galileo Business Bücher, 2000. ISBN 3934358861.

[Cam99] Debra Cameron. *Emacs kurz und gut*. O'Reilly, 1999. ISBN 3897212110.

[CD99] Katherine Curtis and Oliver Draper. Multimedia content management - provision of validation and personalisation services. *ICMCS - International Conference on Multimedia Computing and Systems, Florence, Italy*, 1999.

[CON00] CONTENTMANAGER. *Die Deutsche Content Management Site*, 2000. http://www.contentmanager.de/.

[CSY00] Mon-Yen Luo Chu-Sing Yang. A content placement and management system for distributed web-server systems. *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on, 2000*, 2000.

[ESO01] *Web Server Survey*. Security Space, http://www.securityspace.com/s_survey, 2001.

[FCG98] Wettl Ferenc, Sudár Csaba, and Mayer Gyula. *LaTeX kezdőknek*. Panem Kft., Budapest, 1998.

[Fou00] The Apache Software Foundation. *Apache Web Server Documentation*, 2000. http://www.apache.org/docs/.

[Gau00] *Gauss VIP Content Manager User's Manual*, 2000. Gauss Interprise, http://www.gauss-interprise.com.

[GGR96] C. Gutwin, S. Greenberg, and M. Roseman. *Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets and Evaluation*, 1996. People and Computers XI HCI'96, Conference held at Imperial College.

[Gie98] Olaf Gierhake. *Integriertes Geschäftsprozessmanagement: effektive Organisationsgestaltung mit Workflow-, Workgroup- und Dokumentenmanagement-Systemen*. Vieweg Verl, Braunschweig, Wiesbaden, 1998.

[Hol95] David Hollingsworth. *The Workflow Reference Model*, 1995. Workflow Management Coalition, http://www.aiim.org/wfmc/, Document Number TC00-1003.

[ICE00] ICE. *Information and Context Exchange*, 2000. http://www.icestandard.org/.

[Inf01] Infopark AG. *NPS Data Sheet*, 2001. http://www.nps.de.

[Ins00] Insiders Information Management. *Mindaccess - neue Horizonte erkunden*, 2000. http://www.im-insiders.de.

[KD00] Olaf Kirch and Terry Dawson. *Linux Network Administrator's Guide*. O'Reilly UK, 2000. ISBN 1565924002.

[KGRH00] Alexander Kandzior, Dietmar Galle, Ralf Ruf, and Dierk Himstedt. *Benutzerhandbuch OpenCMS Version 4.0*, 2000. http://www.opencms.com.

[Kra00] Jörg Krause. *PHP4*. Hanser Verlag, München, 2000. ISBN 3-446-21546-8.

[LN99] Henrik Loeser and Rittner N. iwebdb - integrated web content management based on object-relational database technology. *Int. Database Engineering and Applications Symposium (IDEAS'99), Montreal, Canada, pp. 92-97*, 1999.

[Loe] Henrik Loeser. Keeping web pages up-to-date with sql:1999. *Int. Databse Engineering and Application Symposium (IDEAS 2000), Yokohama, Japan, September 2000, pp. 219-223*.

[Lor97] Alexander Lorz. *Entwicklung eines Zusammenarbeitsmodells für ein multimediales Mehrbenutzerautorensystem*, 1997. Belegarbeit an der TU-Dresden.

[MID01] *Midgard Documentation*. Midgard Content Management, http://www.midgard-project.org, 2001.

[Neu00] Sven Neumann. *Gimp - kurz & gut, 2.Auflage*. O'Reilly, 2000. ISBN 3897212234.

[Obe91] H. Oberquelle. *Kooperative Arbeit und Computerunterstützung: Stand und Perspektiven.* Verlag für angewandte Psychologie Göttingen, Stuttgart, 1991.

[Pro01] OpenCMS Project. *OpenCMS*, 2001. OpenCMS Documentation, http://www.opencms.com.

[Rat00] Tobias Ratschiller. *phpMyAdmin Documentation*, 2000. http://www.phpmyadmin.de.

[Red00] *Das Offizielle Red Hat Linux Installationshandbuch*, 2000. http://www.redhat.com.

[RG00] Tobias Ratschiller and Till Gerken. *Web Application Development with PHP4.0*. New Riders Publishing, Indianapolis, 2000. ISBN 0735709971.

[RIS98] L. Rónyai, G. Ivanyos, and R. Szabó. *Algoritmusok*. Typotex, Budapest, 1998. ISBN 963-9132-16-0.

[Sto00] *V/5 E-business Platform - Platform Installation and Configuration Guide*. Vignette Inc., http://www.vignette.com, 2000.

[Wel00] Brent B. Welch. *Practical Programming in Tcl and Tk.* Prentice Hall PTR, 2000. ISBN 0130220280.

[YRK99] Randy Jay Yarger, George Reese, and Tim King. *MySQL and mSQL*. O'Reilly and Associates Inc., 1999. ISBN 1565924347.