Otto-von-Guericke-Universität Magdeburg

Thema:

Konzeption und Implementierung eines Content Repository nach JSR 170 auf Basis von Peer-to-Peer-Netzwerken und Webservices

Diplomarbeit

Fakultät für Informatik
Arbeitsgruppe Wirtschaftsinformatik

Betreuer:	Prof. Dr. rer. pol. habil. Hans-Knud Arndt (FIN/ITI)
Themensteller:	Dipl.-Wirt.-Inform. Sebastian Herden, Wirtschaftsinformatik (FIN/ITI)
vorgelegt von:	Torsten Brandt

Inhaltsverzeichnis

Abbildungsverzeichnis **vi**

Abkürzungsverzeichnis **vii**

1 Einleitung **1**

2 Content Management als Teilbereich des Informationsmanagements **5**

2.1 Informationsmanagement 6

2.2 Enterprise Content Management 12

2.3 Repositories als Speicherort für ECM-Systeme 16

2.4 Implikationen für die Content-Repository-Konzeption 20

3 Technische Grundlagen **23**

3.1 Verwendete Techniken/Hilfsmittel für die Konzeption 24

3.1.1 Unified Modeling Language 24

3.1.2 Extensible Markup Language 29

3.1.3 Patterns 31

3.2 Content Repositories nach JSR 170 34

3.2.1 Repository Information Model 36

3.2.2 API 39

3.2.3 Repository Engine 41

3.2.4 Persistente Datenspeicher für den JSR 170 42

3.2.5 Implikationen für die Konzeption eines Content Repository 42

3.3 Data Source 44

3.3.1 Peer-to-Peer-Netzwerke 44

3.3.2 Merkmale von Peer-to-Peer-Netzwerken 47

3.3.3 Routing- und Suchverfahren in Peer-to-Peer-Netzwerken 51

3.3.4 Implikationen für die Konzeption der Data Source Layer 59

3.4 Service Layer 61

3.4.1 Webservices 63

3.4.2 Webservicetechniken 64

3.4.3 Verwendung von Webservices 66

3.4.4 Implikationen für die Konzeption der Service Layer 68

4 Konzept eines Content Repository auf Basis eines P2P-Netzwerks und Webservices 70

4.1 Die Komponenten des Konzepts 72

4.1.1 Komponente ContentRepository 73

4.1.2 Komponente P2PPersistence 75

4.1.3 Komponente Webservice 81

4.2 Interaktion der Komponenten 85

4.2.1 Speichern von Content im Content Repository 85

4.2.2 Aufrufen von Content aus dem Content Repository . . 86

4.2.3 Suchen nach Content im Content Repository 87

4.3 Verteilung der Artefakte des Konzepts 88

5 Evaluierung des Konzepts 91

5.1 Artefakt ContentRepository 91

5.2 Artefakt P2PNode 93

5.3 Artefakt P2PPersistenceManager 94

5.4 Artefakt Webservice 96

5.5 Artefakt Anwendung 97

6 Zusammenfassung und Ausblick 99

Literaturverzeichnis 101

Abbildungsverzeichnis

2.1 Elemente der Semiotik 6

2.2 Modell des Information Managements 9

2.3 Steuerungszentrierte Anwendungen 17

2.4 Integrierte steuerungszentrierte Anwendungslandschaft 17

2.5 Inhaltszentrierte Anwendungslandschaft 18

2.6 Inhaltszentrierte Anwendungslandschaft mit Content Repository 19

2.7 Service Layer Pattern 21

3.1 UML Darstellung des Service Layer Pattern 33

3.2 Service Layer Pattern - ApplicationLayer 34

3.3 JSR 170: Das Datenmodell 37

3.4 Exemplarische Instanz eines RIM 39

3.5 Sequenzdiagramm der Benutzung des Content Repository . . . 40

3.6 Service Layer Pattern - Data Source Layer 44

3.7 Merkmale eines Peer-to-Peer-Netzwerks 47

3.8 Übersicht der Netzwerktopologien in Peer-to-Peer-Netzwerken 49

3.9 Merkmalsausprägungen im Morphologischen Kasten für Napster 53

3.10 Aufbau des Napster-Netzwerks 53

3.11 Merkmalsausprägungen im Morphologischen Kasten für Gnutella 54

3.12 Exemplarische Aufbaustruktur eines Flooding-Netzwerks . . . 55

3.13 Suche noch Inhalten im Gnutella-Netzwerk 56

3.14 Merkmalsausprägungen im Morphologischen Kasten für CAN . 57

3.15 Content Adressable Network 58

3.16 Service Layer Pattern - Service Layer 61

3.17 Aufruf eines Webservice 67

4.1 Die Verteilungsknoten des Konzepts 70

4.2 Die Komponenten des Konzepts 72

4.3 Komponentendiagramm: Content Repository 73

4.4 Komponentendiagramm: Peer-to-Peer-Persistenz 75

4.5 Probleme der konsistenten Datenhaltung in Peer-to-Peer-Netzwerken 77

4.6 Komponentendiagramm: Webservice 81

4.7 Sequenzdiagramm: Ticket-System 82

4.8 Komponentendiagramm: Überblick über alle Komponenten des Konzepts 84

4.9 Sequenzdiagramm: Speichern von Content 85

4.10 Sequenzdiagramm: Abrufen von Content 86

4.11 Sequenzdiagramm: Suche nach Content 87

4.12 Komponentendiagramm: Zuordnung der Komponenten zu Artefakten 88

4.13 Verteilungsdiagramm der Knoten und Artefakte des Konzepts 90

5.1 Klassendiagramm der Content-Typen des Prototypen 92

5.2 Abbildung der Operationen der Persistenzschnittstelle auf die Operationen des Peer-to-Peer-Netzwerks 95

5.3 Bildschirmfoto der Client-Anwendung 98

Abkürzungsverzeichnis

AIIM	Association for Information and Image Management
API	Application Programming Interface
CAN	Content Addressable Network
DHT	Distributed Hash Tables
ECM	Enterprise Content Management
ECMS	Enterprise Content Management System
JCR	Java Content Repository
JSR	Java Specification Request
P2P	Peer-to-Peer
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WCM	Web Content Management
XML	Extensible Markup Language

Kapitel 1

Einleitung

Durch die Bedeutung von Information als Produktionsfaktor im betrieblichen Leistungserstellungsprozess ist das Informationsmanagement eine elementare Aufgabe der Unternehmensführung (vgl. Pietsch et al. (2004, S. 39ff) und Krcmar (2005, S. 17)). Ein Teilbereich des Informationsmanagements ist das Content Management, in dem auf operativer Ebene Content-Management-Systeme einsetzt werden. Diese nutzen Repositories (deutsch: *Ablage*, *Speicher*), beispielsweise Datenbanken oder das Dateisystem, um den verwalteten Content zu speichern (vgl. Ehlers, 2003, S. 114).

Nach Mertens et al. (2005, S. 6) bedingt die steigende technische und logische Vernetzung von Content in den Anwendungen eines Unternehmens einen Trend zur Integration. Diese kann, wie Kapitel 2 zeigen wird, für Content-Management-Systeme auf Daten- und Funktionsebene durch die Nutzung eines Content Repository erfolgen. Dabei bedarf die Anbindung einer Vielzahl von Anwendungen an ein Content Repository einer offenen und programmiersprachenunabhängigen Schnittstelle zu den Funktionen des Repository. Diese Arbeit soll klären, ob eine solche Anbindung durch Webservices realisierbar ist.

Auf Grund des Umfangs des Content-Bestands oder aus Sicherheitsüberlegungen kann die dezentrale Speicherung des Content sinnvoll sein (vgl. Krcmar, 2005, S. 259). Nach Milojicic et al. (2003, S. 8) können Peer-to-Peer-Netzwerke durch die Ausnutzung von in immer größerer Kapazität

verfügbarem Speicher in immer kleineren Rechnern eine dezentrale Datenhaltung bieten. In dieser Arbeit soll daher überprüft werden, ob sich Peer-to-Peer-Netzwerke für die Verteilung des Content eines Content Repository durch die Ausnutzung von in immer größerer Kapazität verfügbarem Speicher in immer kleineren Rechnern eignen, um die Datenhaltung eines Content Repository zu verteilen. Dabei ist insbesondere zu prüfen, ob ein Peer-to-Peer-Netzwerk die von einem Content Repository benötigte Verfügbarkeit und Konsistenz des Content erbringen kann.

Das Ziel dieser Arbeit ist die Dokumentation eines Konzepts für ein Informationssystem, welches auf Basis offener Standards ein Content Repository mit verteilter Datenhaltung sowie einer offenen und programmiersprachenunabhängigen Schnittstelle zu anderen Anwendungen beinhaltet. Dazu wird ein konzeptionell-deduktives Vorgehen angewandt (vgl. Wilde und Hess, 2007, S. 282ff). Aus der deduktiven Analyse der Anforderungen eines Content-Management-Systems an seine Datenhaltung entsteht ein Konzept für ein solches Informationssystem. Untersetzt wird dieses Konzept durch eine prototypische Implementierung, welche die Umsetzbarkeit des Konzepts verifiziert (vgl. Wieczorrek und Mertens, 2008, S. 76f).

Nach Mertens et al. (2005, S. 1) umfasst die Wirtschaftsinformatik die Konzeption, Entwicklung, Einführung, Wartung und Nutzung von Informationssystemen. Damit ist der Gegenstand dieser Arbeit, die Dokumentation eines Konzepts für ein Informationssystem zur Unterstützung des Informationsmanagements, ein originäres Aufgabengebiet der Wirtschaftsinformatik.

Diese Arbeit ist wie folgt gegliedert:

Kapitel 2 – Content Management als Teilbereich des Informationsmanagement

Im zweiten Kapitel werden das Informationsmanagement sowie das Content Management als Teilbereich des Informationsmanagements eingeführt und aus einer Betrachtung zweier Ansätze zur Konzeption von Content-Management-Anwendungen Implikationen für die Konzeption eines Content Repository abgeleitet.

Kapitel 3 – Technische Grundlagen

Im dritten Kapitel werden die technischen Grundlagen für die Konzeption erläutert. Dazu werden die verwendete Modellierungssprache Unified Modeling Language (UML), Patterns zur Unterstützung der Konzeption und die Extensible Markup Language (XML) als Basis von im Weiteren genutzten Protokollen und Ablageformaten vorgestellt.

Darüber hinaus werden in diesem Kapitel Content Repositories, Peer-to-Peer-Netzwerke und Webservices als Bestandteile des Konzepts eingeordnet und detailliert erörtert.

Kapitel 4 – Konzeption eines Content Repository auf Basis eines Peer-to-Peer-Netzwerks

Im vierten Kapitel wird das Konzept des zu konzipierenden Informationssystems vorgestellt. Dazu werden die enthaltenen Komponenten, ihre Interaktionsbeziehungen und ihre Verteilung auf unterschiedliche Ressourcen erklärt.

Kapitel 5 – Evaluierung des Konzepts

Im fünften Kapitel wird die prototypische Implementierung des Konzepts erklärt indem die gewählte Realisierung der in Kapitel 4 eingeführten Komponenten vorgestellt wird.

Kapitel 6 – Zusammenfassung und Ausblick

Im sechsten Kapitel werden die Aufgabenstellung und Ergebnisse der

Arbeit zusammengefasst. Dabei wird geprüft, ob die in dieser Arbeit getroffenen Annahmen, Peer-to-Peer-Netzwerke und Webservices seien geeignet, die Datenhaltung eines Content Repository zu verteilen und dessen Anbindung an andere Anwendungen bereitzustellen, zutreffend sind. Im Anschluss wird ein Ausblick gegeben, wie eine weitere Entwicklung des Konzepts aussehen kann.

Kapitel 2

Content Management als Teilbereich des Informationsmanagements

In diesem Kapitel werden die fachlichen Grundlagen zur Einordnung dieser Arbeit in das Informationsmanagement gegeben. In Abschnitt 2.1 wird das Informationsmanagement als elementarer Bestandteil der Unternehmensführung vorgestellt und das Content Management als Teilbereich des Informationsmanagements eingeordnet.

Im folgenden Abschnitt wird der Begriff Content Management erläutert und auf das Enterprise Content Management (ECM) und die funktionalen Anforderungen an die Datenhaltung eines Enterprise-Content-Management-Systems (ECMS) fokussiert.

In Abschnitt 2.3 werden zwei Ansätze zur Konzeption einer Anwendung für das Enterprise Content Management beschrieben und dabei der Nutzen der Verwendung eines Content Repository erläutert.

Die sich aus den Abschnitten 2.2 und 2.3 für die Konzeption eines Content Repository ergebenden Implikationen werden in Abschnitt 2.4 zusammengefasst.

2.1 Informationsmanagement

Der Begriff *Informationsmanagement* ist zusammengesetzt aus den Begriffen *Information* und *Management*. Daher wird im Folgenden vor der Bestimmung des Begriffs Informationsmanagement eine Definition von Information und Management gegeben.

Information wird in der Wissenschaft unterschiedlich definiert (vgl. Becker, 1999, S. 545f). Insbesondere das dazu häufig genutzte Begriffstripel *Daten*, *Information* und *Wissen* ist dabei zu untersuchen. So definiert Heinrich (2005, S. 7) Information als handlungsbestimmendes Wissen über historische, gegenwärtige und zukünftige Zustände der Wirklichkeit und Vorgänge in der Wirklichkeit. Im Gegensatz dazu bezeichnen Völker et al. (2007, S. 59f) Information als mit Bedeutung versehene Daten und Wissen als durch Verarbeitung, Filterung und Bewertung vernetzte Information.

Die Semiotik, (Zeichenlehre. Von griech. *Sema* = Zeichen), bietet mit ihren Elementen *Syntaktik*, *Sigmatik*, *Semantik* und *Pragmatik* eine Möglichkeit den Begriff Information zu beschreiben (vgl. Krcmar (2005, S. 16) und Morris (1988, S. 85)). Die Beziehungen zwischen diesen Elementen sind in Abbildung 2.1 dargestellt.

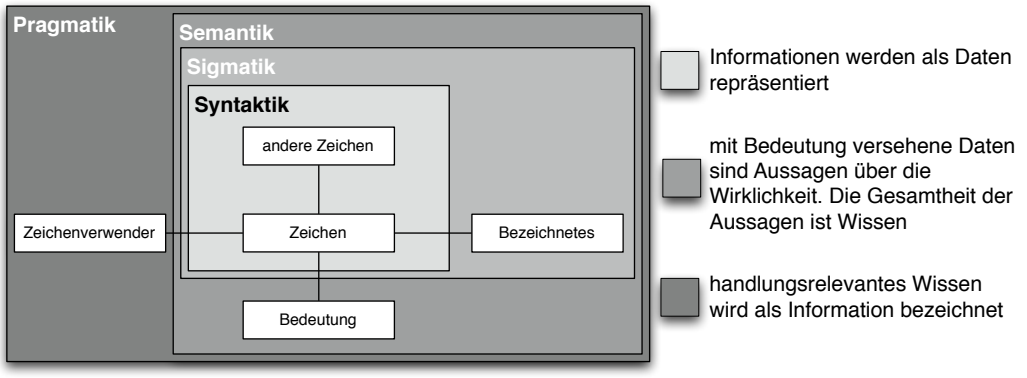


Abbildung 2.1: Elemente der Semiotik
Nach Berthel (1975)

Syntaktik Mit der Syntaktik werden die Beziehungen zwischen den Zeichen eines Sprachsystems beschrieben.

In der Syntaktik werden Informationen als Daten repräsentiert, deren Symbole in Datenstrukturen kodiert werden (vgl. Krcmar (2005, S. 16) und Hansen (1996, S. 6)).

Semantik Die Semantik ist die Wissenschaft von der Bedeutung der Zeichen sowie der Beziehung von Zeichen zum bezeichneten Objekt (Sigmatik) (vgl. Krcmar, 2005, S. 16).

Um die Bedeutung von Daten erfassen zu können, wird Kenntnis über das verwendete Kodierungsschema benötigt. Diese Kenntnisse sind „gesicherte Aussagen über Objekte und Sachverhalte der Welt“ (vgl. Kuhlen, 1999, S. 115). Die Gesamtheit dieser Aussagen wird Wissen genannt (vgl. Herden et al., 2006, S. 9).

Pragmatik Die Relation von Zeichen und dem Zeichenverwender wird mit Hilfe der Pragmatik untersucht.

Im pragmatischen Sinn wird eine Aussage dann als Information aufgefasst, wenn sie eine Wirkung auf die Wissensstruktur einer Person hat. Eine Wirkung hat eine Aussage dann, wenn sie für die Person neu ist und im aktuellen situativen Kontext von Nutzen ist (vgl. Herden et al., 2006, S. 9).

In Einbeziehung dieser Elemente der Pragmatik definiert Szyperski (1980, S. 904) Information als „Aussagen, die den Erkenntnis- bzw. Wissensstand eines Subjektes (Informationssubjekt/-benutzer) über ein Objekt (Informationsgegenstand) in einer gegebenen Situation und Umwelt (Informationsumwelt) zur Erfüllung einer Aufgabe (Informationszweck) verbessern.“

Kampffmeyer (2003, S. 6) und Kretzschmar (2007, S. 198) definieren den Begriff *Content* als Information, die in elektronischen Systemen genutzt wird. Aus der semiotischen Betrachtung des Begriffs Information folgt, dass Information eines Informationssubjekts und einer Informationsumwelt bedarf. In

einem elektronischen System können also nur Daten als Basis von Information verwaltet werden. In dieser Arbeit soll daher Content als durch Daten repräsentierte Aussagen über Objekte und Sachverhalte in einer Organisation verstanden werden.

Für das Informationsmanagement ist der Management-Begriff in seiner funktionalen Ausprägung maßgeblich (vgl. Krcmar, 2005, S. 23). Daher wird in dieser Arbeit das Management nach Heinrich (2005, S. 7) als Leitungshandeln in einer Betriebswirtschaft angesehen.

Aus dem erarbeiteten Verständnis der Begriffe Information und Management kann das Informationsmanagement somit nach nach Heinrich (2005, S. 8) als Leitungshandeln in einer Betriebswirtschaft in Bezug auf Information und Kommunikation angesehen werden.

Krcmar präzisiert das Leitungshandeln auf das Management der Informationswirtschaft, der Informationssysteme, der Informations- und Kommunikationstechniken sowie der übergreifenden Führungsaufgaben mit dem Ziel des bestmöglichen Einsatzes von Information. Krcmar benennt das Informationsmanagement somit sowohl als Management- als auch als Technikdisziplin (vgl. Krcmar, 2005, S. 49).

Es existieren differenzierte Ansätze und Konzepte, die unterschiedliche Sichten auf das Informationsmanagement abbilden. Darunter fallen problemorientierte, aufgabenorientierte oder prozessorientierte Ansätze (vgl. Krcmar, 2005, S. 28ff). Das von Krcmar (2005, S. 47ff) entwickelte Modell des Informationsmanagement liefert einen ganzheitlichen Ansatz, der Aspekte aus den zuvor genannten Ansätzen aufgreift und soll daher nachfolgend erläutert werden.

Das Modell des Informationsmanagements nach Krcmar gliedert sich in die drei Ebenen *Management der Informationswirtschaft*, *Management der Informationssysteme* und *Management der Informations- und Kommunikationstechnik* sowie den *Führungsaufgaben des Informationsmanagements* als

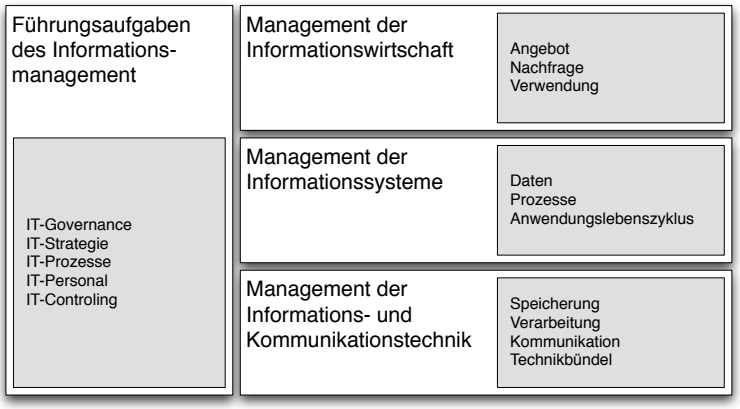


Abbildung 2.2: Modell des Information Managements (nach Krcmar, 2005, S. 47)

alle Ebenen betreffende Querschnittsfunktion (vgl. Abbildung 2.2). Im Folgenden werden die Inhalte dieser vier Elemente des Modells beschrieben (vgl. Krcmar, 2005, S. 48):

Informationswirtschaft In der Ebene der Informationswirtschaft werden Informationsnachfrage und -angebot in den verschiedenen Verwendungsbereichen einer Unternehmung festgelegt. Die daraus resultierenden Anforderungen sind die Basis für die Entscheidungen in der Ebene der Informationssysteme.

Somit erfolgt auf dieser Ebene in Anlehnung an Wollnik (1988, S. 38) das Management des *Informationseinsatzes*.

Informationssysteme In der Ebene der Informationssysteme erfolgt das Management von Daten, Prozessen und Anwendungslebenszyklen, um den Informationsbedarf zu decken. Dabei werden Anforderungen an die Informations- und Kommunikationstechnik spezifiziert.

Somit erfolgt auf dieser Ebene in Anlehnung an Wollnik (1988, S. 38) das Management des *Informationsbedarfs*.

Informations- und Kommunikationstechnik Auf der untersten Ebene erfolgt das Management der physischen Basis der Anwendungsland-

schaft. Dies umfasst die Bereitstellung und Verwaltung der Informationsinfrastruktur.

Auf dieser Ebene erfolgt in Anlehnung an Wollnik (1988, S. 38) das Management der *Informationsinfrastruktur*.

Führungsaufgaben des Informationsmanagements Umfasst die Gestaltung der IT-Governance sowie der IT-Strategie und das Management des IT-Personals und der IT-Prozesse. Darüberhinaus ist das IT-Controling eine Querschnittsaufgabe der Informationsmanagements.

Ziel des Informationsmanagements ist nach Krcmar (2005, S. 49) der bestmögliche Einsatz von Information in Hinblick auf die Unternehmensziele. Um dieses zu erreichen, ist es nötig, dass das Wissen der Unternehmung als Basis der Information zur richtigen Zeit am richtigen Ort bereitgestellt wird. Das gesammelte Wissen setzt sich aus einzelnen Aussagen über Sachverhalte im Unternehmen (Content) zusammen. Content Management ist demnach als das Management von Aussagen zu verstehen. Damit ist das Content Management ein Teilbereich des Informationsmanagement.

Im Management der Informationssysteme umfasst das Content Management somit die Organisation der Daten (als Repräsentation der Aussagen) und der Prozesse, mit Hilfe derer die Aussagen zur richtigen Zeit am richtigen Ort bereitgestellt werden. Die Daten können dabei unterschiedlich stark strukturiert sein. Es wird dabei *strukturierter Content* von *unstrukturiertem Content* unterschieden. *Strukturierter Content* folgt in seinem Aufbau einem zuvor festgelegten Muster. Beispiele hierfür sind in Datenbanken abgelegte Inhalte. Ihr Aufbau ist dabei durch ein Datenbankschema vorgegeben. *Unstrukturierter Content* ist Content, der keinen Aufbauregeln folgt. Ihr Inhalt kann nicht direkt erschlossen werden, da keine Metadaten oder keine Trennung von Inhalt und Metadaten existiert (vgl. Kampffmeyer, 2003, S.6).

Im folgenden Kapitel werden die Anforderungen näher erläutert, die sich aus dem Content Management für die Informationssysteme und daraus abgeleitet für die Informations- und Kommunikationstechnik ergeben.

2.2 Enterprise Content Management

Der Begriff *Content Management* wird von Herstellern und Benutzern von Content-Management-Lösungen unterschiedlich verwendet. Kampffmeyer (2003, S. 7) identifiziert drei Hauptströmungen für Content Management:

Content Management als Content Syndication Unter dem Begriff der Content Syndication lässt sich die Vermarktung und Verbreitung von Content zusammenfassen. Dies umfasst den Schutz, die Verwaltung, Verteilung sowie, im Falle von kommerziellem Content, die Abrechnung für dessen Nutzung (vgl. Kampffmeyer (2003, S. 7) und Kretzschmar (2007, S. 216)).

Beispiele für Content Syndication sind RSS-Feeds von Weblogs (Blogs) und Nachrichtenseiten aber auch das Anbieten von Podcasts (vgl. Richards, 2006, S. 521).

Web Content Management (WCM) In diesem Fall wird Content Management häufig als Begriff für das Management von Web-Inhalten verwendet (vgl. Maass und Stahl, 2008, S. 152). Das Web Content Management bezeichnet dabei die Verwaltung von Content auf internetbasierten Webseiten und Portalen (vgl. Kampffmeyer, 2003, S. 8).

Enterprise Content Management (ECM) Nach Kampffmeyer (2003, S. 11) ist das Ziel des ECM, dass der Content eines Unternehmens über eine einheitliche Plattform bereitgestellt und verwaltet wird und so eine einheitliche Regelung für den Zugriff auf Content ermöglicht wird.

Durch den unternehmensweiten Ansatz des Enterprise Content Management umfasst es die beiden anderen Begriffsausprägungen des Content Managements. Deshalb wird im Folgenden der Begriff Enterprise Content Management detaillierter erläutert und die im Rahmen eines Enterprise-Content-Management-Systems benötigten Funktionen erarbeitet.

De Carvalho beschreibt die Aufgabe des Enterprise Content Management in der Integration der Aufgaben des Managements von strukturiertem und unstrukturiertem Content sowie zugehöriger Software und Metadaten in Softwarelösungen für die Produktion, Publikation, Nutzung und Speicherung von Content in Organisationen (vgl de Carvalho, 2008, S. 174).

Damit umfasst das Enterprise Content Management u. a. die Funktionalität des Archiv- und Dokumentenmanagement (vgl. Kampffmeyer, 2003, S. 10).

ECM-Systeme und -Teilsysteme lassen sich in die folgenden, auf der ECM-Definition der AIIM[1] basierenden, Kategorien einteilen (vgl. Kampffmeyer, 2003, S. 16ff):

Capture (Erfassen) Die Komponenten der Capture-Kategorie beinhalten Funktionen zur Erstellung, Erfassung und Aufbereitung von analogen und elektronischen Informationen. Ziel dieser Komponenten ist, die erfassten Informationen zur Weiterbearbeitung oder Archivierung den Manage-Komponenten bereitzustellen. Dies umfasst Techniken wie Optical Charakter Recognition (OCR), Barcodes oder den Import von XML-Dokumenten anderer Anwendungen zur Erfassung und Verschlagwortung mit dem Ziel der Klassifikationen und Kategorisierung zur inhaltlichen Erschließung.

Manage (Verwalten) Manage-Komponenten werden zur Verwaltung, Bearbeitung und Nutzung von Content verwendet. Dazu wird ein einheitliches Berechtigungssystem zum Schutz vor unbefugtem Zugriff benötigt. Die Manage-Kategorie umfasst Aufgaben wie das Dokumentenmanagement, das Web Content Management, die Ablage- und Archivverwaltung sowie die Vorgangsbearbeitung. Nur Manage-Komponenten greifen auf die Komponenten der Store- und Preserve-Kategorie zu.

[1] Association for Information and Image Management
http://www.aiim.org Stand: 16.03.2009

Store and Preserve (Speichern und Sichern) Die Komponenten dieser Kategorien dienen zur Speicherung von Content in Repositories und werden daher hier zusammengefasst. Nicht archivierungswürdiger oder -pflichtiger Content wird dabei von Store-Komponenten gespeichert, die langfristige, unveränderbare und damit revisionssichere Speicherung erfolgt durch Preserve-Komponenten (vgl. Kampffmeyer, 2003, S. 24f und S. 27).

Da von diesen Komponenten Repositories genutzt werden, sind sie Hauptbetrachtungsgegenstand bei der folgenden Analyse der Anforderungen, die an die Datenhaltung eines ECM-Systems gestellt werden.

Deliver (Ausgabe) Komponenten in dieser Kategorie dienen der Bereitstellung von Content für andere Systeme oder Personen. Sie umfassen daher Transformations- und Ausgabetechniken wie XSLT und PDF sowie Techniken zur Content Syndication.

Der Inhalt dieser Arbeit umfasst die Konzeption eines Content Repository. Aus der vorangegangenen Betrachtungen der Kategorien für Komponenten in Enterprise-Content-Management-Systemen lässt sich schließen, dass nur mit Komponenten der Store- und Preserve-Kategorie auf Repositories zugegriffen wird. Diese Komponenten werden nur von Komponenten der Manage-Kategorie genutzt. Eine Untersuchung dieser beiden Kategorien liefert die folgenden funktionalen Anforderungen an ein Repository:

Speichern, Suchen, Zugreifen Die Speicherung von Content ist eine elementare Funktion eines Repository. Dabei muss das Merkmal der unterschiedlich starken Strukturierung von Content beachtet werden, um eine gleichermaßen performante Verarbeitung zu ermöglichen. Weiter ist das Suchen nach Content sowie der Zugriff auf Content essentiell für ein Repository (vgl. Ehlers, 2003, S. 113).

Medienneutrale Datenhaltung bezeichnet die Ablage von Content in einem Format, dass dessen Wieder- und Weiterverwendung durch eine

Vielzahl von Anwendungen ermöglicht. Dieses erfordert eine Trennung von Struktur, Inhalt und Layout (vgl. Kretzschmar (2007, S. 18) und Milleg und Wagner (2001, S. 37)).

Zugriffsberechtigungen für den Content. Für jeden Content muss festlegbar sein, ob er von einem bestimmten Benutzer oder einer Benutzergruppe gelesen, verändert oder gelöscht werden darf (vgl. Smolnik (2007, S. 28) und Ehlers (2003, S. 116)).

Sperrverwaltung (Locking) Zur Ermöglichung eines Mehrbenutzerbetriebs und der konsistenten Nutzung des Content ist es notwendig, einzelne Content-Elemente für die Zeit der Bearbeitung zu sperren. Dieses Sperren und Freigeben nach der Bearbeitung wird in Anlehnung an vergleichbare Mechanismen in Quellcodeverwaltungssystemen Check-out bzw. Check-in genannt (vgl. Ehlers, 2003, S. 114).

Versionsmanagement wird benötigt, um frühere Stände eines Content zu speichern und wieder abzurufen. Dies ist wichtig für eine revisionssichere Speicherung des Content (vgl. Kampffmeyer, 2003, S. 21).

Nach Ehlers (2003, S. 113) ist dabei das Speichern, Suchen und Zugreifen auf Content grundlegende Funktionalität eines jeden Repository. Die anderen genannten Anforderungen sind optional. Ehlers bezeichnet sie als Content-Management-spezifische Funktionen.

Im folgenden Abschnitt wird erklärt, wie Repositories in einer ECM-Anwendung genutzt werden und wie durch die Zusammenführung von grundlegenden und Content-Management-spezifischen Funktionalitäten eine vereinfachte Anwendungslandschaft geschaffen werden kann.

2.3 Repositories als Speicherort für Enterprise-Content-Management-Systeme

Im Enterprise Content Management wird Content in Repositories abgelegt. Repositories besitzen als elementare Funktionalität das Speichern, Suchen und Zugreifen auf abgelegte Inhalte (vgl. Abschnitt 2.2).

Bei der Konzeption einer ECM-Anwendung mit Repository kann auf steuerungszentrierte (engl. *control centric*) und inhaltszentrierte (engl. *content centric*) Ansätze zurückgegriffen werden (vgl. Fielding, 2005, S. 3). Dabei wird bei dem steuerungszentrierten Ansatz die von einer Anwendung zu erbringende Funktionalität in den Mittelpunkt gestellt. Diese Funktionalität wird spezifiziert und durch anwendungsspezifische Implementierungen bereitgestellt. Die zugrunde liegenden Daten werden dazu strukturiert und in einem proprietären Datenformat in einem Repository gespeichert. Eine medienneutrale Datenhaltung, wie sie als funktionale Anforderung an ein Repository am Ende von Abschnitt 2.2 erarbeitet wurde, ist nicht vorgesehen. In Abbildung 2.3 wird eine aus drei Anwendungen bestehende Anwendungslandschaft beispielhaft dargestellt, deren Anwendungen nach dem steuerungszentrierten Ansatz entworfen sind. Jede der Anwendungen umfasst dabei sowohl die anwendungsspezifische Funktionalität als auch die zuvor als Content-Management-Funktionalität bezeichneten Funktionen. Das Repository bietet die elementaren Funktionen Speichern, Suchen und Zugreifen. Die Nutzung proprietärer Datenformate und die Bindung an ein Repository führt dazu, dass der Wechsel auf eine andere Anwendung nur mit hohen und schwer kalkulierbaren Kosten für die Migration der Daten und ohne Gewissheit eines Erfolges möglich ist (Lock-In-Effekt vgl. Dixit und Nalebuff (1997, S. 246)).

Ein erster Schritt das Ziel des ECM, eine einheitliche Bereitstellung und Verwaltung des Content (vgl. Abschnitt 2.2), zu erreichen, besteht in der Integration der Daten einer Anwendungslandschaft (vgl. Mertens et al., 2005, S. 7). Die Vermeidung einer redundanten Datenhaltung führt zu einer konsistenten

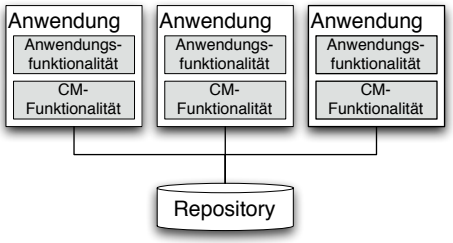


Abbildung 2.3: Steuerungszentrierte Anwendungen

Datenhaltung, ohne, dass eine Synchronisierung der Datenbestände erfolgen muss. Verwenden die Anwendungen einer Anwendungslandschaft jeweils eigene Datenformate, so bedarf es Übersetzern, die Datenformate anderer Anwendungen für die jeweilige Anwendung lesbar machen (vgl. Messaging, Message Translator und Canonical Data Model Pattern in Hohpe und Woolf (2003)). Dies ist in Abbildung 2.4 dargestellt. Die anwendungsspezifische und Content-Management-Funktionalität bleibt dabei in jeder Anwendung erhalten.

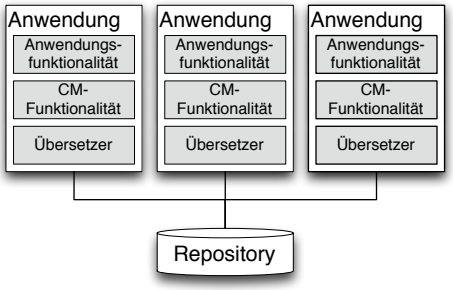


Abbildung 2.4: Integrierte steuerungszentrierte Anwendungslandschaft

Mit dem inhaltszentrierten Ansatz wird der Content, dessen Struktur und die Identifikation von allgemein benötigten Funktionalitäten, wie der Sperrverwaltung, dem Versionsmanagement sowie einem Zugriffsberechtigungssystem (vgl. Abschnitt 2.2), in das Zentrum gestellt. Ziel des Entwerfens nach dem inhaltszentrierten Ansatz ist ein allgemeines, erweiterbares Datenmodell sowie eine Programmierschnittstelle (engl. Application Programming Interface (API)) (vgl Fielding, 2005, S. 3f). In Abbildung 2.5 ist ein inhaltszentrierter Entwurf einer Anwendungslandschaft dargestellt.

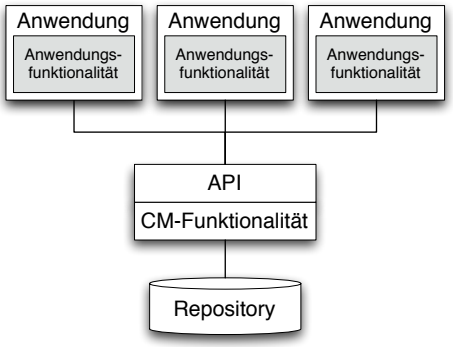


Abbildung 2.5: Inhaltszentrierte Anwendungslandschaft

Die dort veranschaulichte Auslagerung der Content-Management-Funktionalität ist die Zusammenfassung fachlich zusammenhängender Funktionen und kann daher als Funktionsintegration angesehen werden (vgl. Mertens et al., 2005, S. 7). Da nach dem Entwurf ein allgemeines Datenmodell existiert, werden keine Übersetzer für den Zugriff auf die Daten benötigt. Daher wird durch den inhaltszentrierten Ansatz die Datenintegration vereinfacht. Weitere Erläuterungen zu den Integrationsarten finden sich bei Mertens et al. (2005, S. 7ff) sowie Herden et al. (2006, S. 11ff).

In einer inhaltszentrierten Anwendungslandschaft nutzen Anwendungen für das Management des genutzten Content die über die API bereitgestellte Content-Management-Funktionalität. Anwendungsspezifische Funktionen, die dadurch nicht abgedeckt sind, werden von der Anwendung bereitgestellt. Dies erlaubt ein modulares Enterprise-Content-Management-System, bei dem Teilkomponenten (bei gleichem anwendungsspezifischen Funktionsumfang) ausgetauscht werden können und somit den Lock-In-Effekt minimieren (vgl. Röwekamp, 2001, S. 15). Modulare Systeme erfordern die vorherige Definition von Standards für gemeinsam genutzte Komponenten, in diesem Fall für die API der Content-Management-Funktionalität. Nur eine standardisierte API kann als verlässliche Basis für die Nutzung einer wiederverwendbaren Komponente dienen und einen Beitrag zum Investitionsschutz bei der Anwendungsentwicklung und -einführung leisten (vgl. Turowski (2003, S. 3) und Großmann und Koschek (2005, S. 183)).

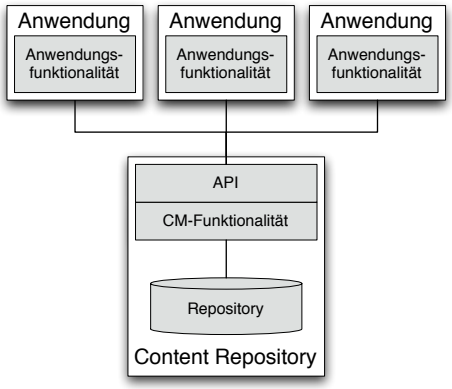


Abbildung 2.6: Inhaltszentrierte Anwendungslandschaft mit Content Repository

Eine Komponente, die sowohl die elementaren Funktionen eines Repository als auch die Content-Management-Funktionalität zusammenfasst und über eine API zugängig macht, wird *Content Repository* genannt (vgl. Ehlers (2003, S. 113f) und Abbildung 2.6).

Für das Enterprise Content Management ist ein inhaltszentrierter Ansatz dem steuerungszentrierten Ansatz vorzuziehen, da er durch die Bereitstellung aller Daten durch genau ein Repository und die Bündelung der Content-Management-Funktionalität in einer Komponente die vom Enterprise Content Management angestrebte einheitliche Plattform für die Bereitstellung und das Management von Content ermöglicht.

Im nachfolgenden Kapitel werden die Implikationen, die sich aus den bisherigen Betrachtungen für die Konzeption eines Content Repository ergeben, zusammengefasst.

2.4 Implikationen für die Content-Repository-Konzeption

Aus einer Definition und Betrachtung der Aufgaben des Informationsmanagements in Abschnitt 2.1 lässt sich das Content Management als Teilbereich des Informationsmanagements ableiten. Als umfassendste Form des Content Management wurde das Enterprise Content Management identifiziert. Die informationstechnische Unterstützung des ECM erfolgt durch ECM-Systeme, deren Content-Speicherung in Repositories geschieht. Für ECMS wurden Content-Management-spezifische und grundlegende Repositoryfunktionen identifiziert. Diese Funktionen lassen sich in einem Content Repository zusammenfassen.

Aus Abschnitt 2.3 und insbesondere aus der in Abbildung 2.6 dargestellten inhaltszentrierten Anwendungslandschaft ergeben sich drei Aufgabenbereiche für die Konzeption eines Content Repository :

Konzeption eines Content Repository Ein Content Repository muss die grundlegende Funktionalität eines Repository, das Speichern, Suchen und Zugreifen auf Content, abdecken. Darüber hinaus muss ein Content Repository weitere Content-Management-spezifische Funktionen bieten (vgl. Ende Abschnitt 2.2). Dies sind eine medienneutrale Datenhaltung, die Verwaltung von Zugriffsberechtigungen, Sperrverwaltung und ein Versionsmanagement.

Wahl eines passenden Repository zur Speicherung des Content Der Datenspeicher eines Content Repository muss ein sicheres Ablegen und Wiederfinden des Content garantieren, d. h. ein einmal abgelegter Content darf nicht verloren gehen.

Anbindung des Content Repository an ECM-Anwendungen Um ein Content Repository von einer Vielzahl an Anwendungen in unterschiedlichen Programmiersprachen und auf unterschiedlichen Plattfor-

men nutzen zu können, muss diese Anbindung flexibel und plattformunabhängig sein, um eine einfache und programmiersprachenunabhängige Nutzung des Content zu ermöglichen.

Bei der Konzeption kann auf Muster (engl. *patterns*) zurückgegriffen werden (vgl. Herden und Zwanziger, 2004, S. 1). Patterns stellen Lösungen für häufig wiederkehrende Probleme dar. Sie beschreiben „Best Practices“ oder besonders gelungene Lösungen und dokumentieren so Erfahrungen für deren Wiederverwendung durch andere Personen (vgl. Buschmann et al., 1998, S. 1). Detailliertere Erläuterungen zu Patterns werden in Kapitel 3.1.3 gegeben.

Ein Pattern zur Konzeption eines Informationssystems ist das Layer Pattern (vgl. Buschmann et al., 1998, S. 32). Das Layer Pattern sieht vor, dass ein System, dessen Größe eine Zerlegung erfordert, in Gruppen von Teilaufgaben zerlegt wird. Dazu werden diese Teilaufgaben konzeptionell verschiedenen Schichten zugeordnet. Jede dieser Schichten nutzt dabei die Funktionen der nächsttieferen Schicht über eindeutige Schnittstellen oder Protokolle.

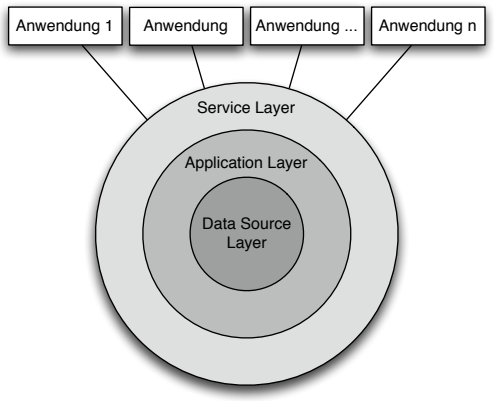


Abbildung 2.7: Service Layer Pattern

Für die Konzeption eines Content Repository mit seinen drei Aufgabenbereichen eignet sich ein spezialisiertes Layer Pattern, das Service Layer Pattern. Es kann genutzt werden, wenn mehrere Anwendungen die von einem Informationssystem bereitgestellten Funktionen nutzen (vgl. Abbildung 2.7). In

der Service Layer wird dabei die Systemgrenze des Informationssystems sowie die verfügbaren Operationen aus der Perspektive aufsetzender Anwendungen definiert (vgl. Fowler, 2002, S. 133). Die von der Service Layer angebotenen Operationen werden in der Application Layer realisiert. In dieser Arbeit wird die Application Layer durch ein Content Repository umgesetzt. Die Application Layer verwendet die Data Source Layer zur Speicherung seiner Daten.

Im folgenden Kapitel werden die technischen Grundlagen der Konzeption der einzelnen Schichten erläutert und in Kapitel 4 das Konzept eines Content Repository auf Basis von Peer-to-Peer-Netzwerken und Webservices vorgestellt.

Kapitel 3

Technische Grundlagen

In diesem Kapitel werden die technischen Grundlagen dieser Arbeit vorgestellt. In Abschnitt 3.1 werden die eingesetzte Modellierungssprache Unified Modeling Language (UML) beschrieben, die Extensible Markup Language (XML) als Metaformat der im Folgenden verwendeten Nachrichten und als Grundlage eingesetzter Technologien eingeführt sowie der Nutzen von Patterns in der Softwareentwicklung vorgestellt.

Die für die Konzeption eines Content Repository benötigten technischen Grundlagen und Standards werden in Abschnitt 3.2 erläutert. Der anschließende Abschnitt beschreibt Peer-to-Peer-Netzwerke allgemein und erörtert für den Einsatz als Repository wichtige Merkmale.

In Abschnitt 3.4 werden als Abschluss dieses Kapitels Techniken zur Frontend-Integration eines Content Repository erläutert und Webservices im Detail betrachtet.

3.1 Verwendete Techniken und Hilfsmittel für die Konzeption

3.1.1 Unified Modeling Language

Wie in Kapitel 2.4 beschrieben, kann ein Content Repository als Teil eines Enterprise-Content-Management-Systems eingesetzt werden. Damit ist es Teil eines Informationssystem im betrieblichen Umfeld. Nach Herden et al. (2006, S. 57) ist ein solches System komplex und kompliziert und daher nur beherrschbar, wenn es ausreichend abstrahiert wird. Die Abstraktion erfolgt dabei durch die Erstellung eines Modells (vgl. Steinmüller, 1993, S. 30).

Ein Modell ist eine Abbildung einer Struktur oder eines zeitlichen Vorgangs der Realwelt für Zwecke eines Subjekts (vgl. Rautenstrauch und Schulze, 2003, S. 225). Es besitzt nach Stachowiak (1973, S. 131ff) ein *Abbildungs-*, ein *Reduktionsmerkmal* und ein *pragmatisches Merkmal*:

Abbildungsmerkmal Ein Modell bildet ein Original ab. Ein Original kann dabei ein materielles Objekt, aber auch nicht-materielle Gegenstände wie Pläne, Vorgänge, Zustände, Ideen, Einschätzungen oder Theorien sein (vgl. Hesse und Mayr, 2008, S. 380).

Reduktionsmerkmal Ein Modell ist abstrahiert. Es enthält nur einen Teil der Attribute des Originals. Die Auswahl der Attribute, die in das Modell aufgenommen werden, ist Ergebnis der Ausprägung des Pragmatischen Merkmals.

Pragmatisches Merkmal Mit dem pragmatischen Merkmal wird abgebildet, dass die Modellierung abhängig vom Modelladressaten ist. Je nach Zielgruppe des Modells werden andere Attribute des Originals in das Modell übernommen.

Die in dieser Arbeit zur Beschreibung der Modelle verwendete Modellierungssprache ist die Unified Modeling Language (UML) der Object Management Group (OMG)[1]. Sie bietet grafische Notationselemente zur Modellierung, Dokumentation, Spezifizierung und Visualisierung von Informationssystemen (vgl. Rupp et al., 2007, S. 12). Die UML wird verwendet, da sie eine weit verbreitete Modellierungssprache ist (vgl. Hesse und Mayr, 2008, S. 379) und daher einem weiten Personenkreis verständlich ist. Nach Rupp et al. (2007, S. 1f) stellt sie somit eine gemeinsame Sprache zur Modellierung unter Softwareentwicklern dar.

Die UML-Spezifikation besteht aus vier Bestandteilen, der *Infrastructure*, *Superstructure*, *Object Constraint Language* und *Diagram Interchange* (vgl. OMG, 2007a, S. 9). Die Teile Object Constraint Language und Diagram Interchange werden in dieser Arbeit nicht genutzt und daher nicht näher erläutert.

Die Infrastructure beschreibt Elemente zur Modellierung auf sehr abstraktem Niveau. Diese Elemente werden z. B. in der MetaObject Facility der OMG[2] und in der Superstructure verwendet (vgl. Duc, 2007, S. 50f).

In der Superstructure werden die offiziellen Diagrammtypen und damit die grafischen Sprachelemente und deren Syntax und Semantik auf der Benutzerebene definiert. Wenn von UML gesprochen wird, ist daher meist die Superstructure gemeint (vgl. Herden et al., 2006, S. 58). Die Diagrammtypen sind Metamodelle der von Benutzern erstellen Modelle.

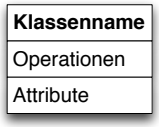
Die Diagrammtypen der UML Superstructure unterteilen sich in Struktur- und Verhaltensdiagramme (vgl. OMG, 2007b, S. 15). Mit Strukturdiagrammen wird der statische Aufbau eines Informationssystems abgebildet. Dies umfasst die Abbildung von Teilsystemen, deren Funktion und Beziehungen zu anderen Teilen des Systems. Mit Verhaltensdiagrammen werden zeitli-

[1] http://www.omg.org Stand: 16.03.2009
[2] http://www.omg.org/mof Stand: 16.03.2009

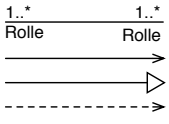
che Abfolgen in der Kommunikation von Systemteilen oder Reaktionen auf eintretende Ereignisse beschrieben.

Im Folgenden werden die in dieser Arbeit verwendeten Strukturdiagrammtypen und die darin verwendeten Elemente näher erläutert. Die dabei genutzten deutschen Diagramm- und Elementnamen sind Rupp et al. (2007) entnommen. Für eine vollsändige Beschreibung sei auf die UML-Spezifikation verwiesen (vgl. OMG (2007a) und OMG (2007b)).

Klassendiagramm (engl. *Class Diagram*) In einem Klassendiagramm werden Klassen eines Informationssystems und deren Beziehungen modelliert. In den Abbildungen 3.3 und 5.1 sind Klassendiagramme zu sehen. In ihnen werden die folgenden Elemente verwendet:

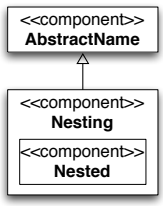


Klasse Mit Klassen werden Objekte eines Informationssystems mit gleichen Attributen und Methoden beschrieben. Die UML repräsentiert sie durch Rechtecke die den Klassennamen beinhalten. Operationen und Attribute werden in darunterliegenden Rechtecken dargestellt. Die Angabe von Operationen und Attributen ist optional.

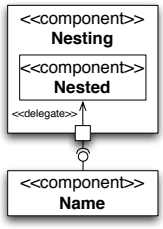


Beziehungen *Allgemeine Beziehungen* werden durch Linien, die zwei Klassen verbinden, dargestellt. Beziehungen können durch Kardinalitäten und Rollen an den Beziehungsenden genauer spezifiziert werden. Ein Pfeil am Ende einer Beziehung legt eine Navigationsrichtung fest.
Linien mit einer nicht ausgefüllten dreieckigen Pfeilspitze repräsentieren *Generalisierungsbeziehungen*. Dabei ist die Klasse auf die der Pfeil zeigt die Oberklasse, die von der anderen Klasse spezialisiert wird.
Abhängigkeiten zwischen zwei Klassen werden durch eine *Abhängigkeitsbeziehung* als gestrichelte Linien mit Pfeilspitze dargestellt. Dabei ist die Klasse auf die der Pfeil zeigt die unabhängige Klasse.

Komponentendiagramm (engl. *Component Diagram*)In einem Komponentendiagramm werden die Komponenten eines Informationssystem sowie die Schnittstellen zwischen ihnen modelliert. In den Abbildungen 3.1 und 4.12 sind Komponentendiagramme zu sehen. In ihnen werden die folgenden Elemente verwendet:

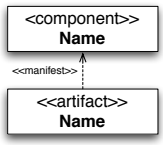


Komponente Eine Komponente ist ein modularer Systemteil, der seinen Inhalt kapselt. Sie wird durch ein Rechteck mit dem Schlüsselwort ≪*component* ≫ dargestellt. Komponenten können andere Klassen oder Komponenten enthalten, welche die Funktionalität der Komponente anbieten. Gemeinsame Funktionen mehrerer Komponenten können in *abstrakten Komponenten* zusammengefasst werden, die durch eine kursive Schreibweise des Namens gekennzeichnet werden. Abstrakte Komponenten und deren Spezialisierungen werden über Generalisierungsbeziehungen (vgl. Klassendiagramm) verbunden.



Schnittstellen und Ports Komponenten kommunizieren über *Schnittstellen*. Angebotene Schnittstellen werden durch eine Linie mit einem Kreis am Ende, benötigte Schnittstellen durch einen Halbkreis dargestellt.

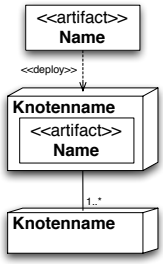
Nutzt eine eingebettete Komponente eine externe Schnittstelle, so wird die eingebettete Komponente durch einen Delegationskonnektor mit einem außen sichtbaren *Port*, dargestellt durch ein Viereck, verbunden.



Artefakt Ein Artefakt stellt eine physische Informationseinheit, beispielsweise Quellcode oder ausführbare Binärdateien, dar. Es wird durch ein Rechteck mit dem Schlüsselwort ≪*artifact* ≫ repräsentiert. In Komponentendiagrammen können Artefakte verwendet werden, um die Realisierung einer Komponente zu modellieren. So wird dargestellt durch welche physikalische Informationseinheit eine Komponente im Informationssystem repräsentiert wird. Dies ist durch eine *Manifestations-*

beziehung, eine gestrichelte Linie mit Pfeilspitze und dem Schlüsselwort «*manifest* » dargestellt.

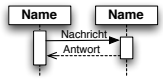
Verteilungsdiagramm (engl. *Deployment Diagram*)In einem Verteilungsdiagramm werden die Hardwareeinheiten eines Informationssystem sowie die Verteilung der Artefakte auf die Hardwareeinheiten modelliert. In den Abbildungen 4.1 und 4.13 sind Verteilungsdiagramme zu sehen. In ihnen werden die folgenden Elemente verwendet:



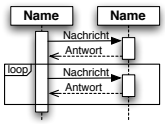
Knoten Ein Knoten repräsentiert eine Ressource, die zur Ausführung von Artefakten genutzt werden kann. Dies können Hardwareeinheiten oder auch Ausführungsumgebungen sein. Knoten werden durch Quader dargestellt, die den Namen des Knotens enthalten. Knoten können durch *Kommunikationspfade*, dargestellt durch Linien, verbunden werden um Nachrichten auszutauschen. Kommunikationspfade können durch Kardinalitäten spezifiziert werden. Die Verteilung der Artefakte auf Knoten kann durch zwei Notationen beschrieben werden. Zum Einen durch die Schachtelung eines Artefakts in einen Knoten, zum Anderen durch eine *Verteilungsbeziehung*. Sie wird als gestrichelte Linie mit Pfeilspitze und dem Schlüsselword «*deploy* » dargestellt.

Die nachfolgend beschriebenen Sequenzdiagramme gehören zu den Verhaltensdiagrammen (vgl. OMG, 2007b, S. 15):

Sequenzdiagramm (engl. *Sequence Diagram*) In Sequenzdiagrammen wird der Informationsaustausch zwischen Kommunikationspartnern in einem Informationssystem dargestellt. Dabei können unter anderem zeitliche Abfolgen und Schleifen modelliert werden. In den Abbildungen in Kapitel 4.2 sind Sequenzdiagramme zu sehen. In Ihnen werden die folgenden Elemente verwendet:



Lebenslinie, Ausführungssequenz und Nachricht Eine *Lebenslinie* repräsentiert einen Teilnehmer der Kommunikation. Sie wird durch ein Rechteck mit dem Namen des Teilnehmers und einer gestrichelten Linie, die die Lebenszeit des Teilnehmers abbildet, repräsentiert.
Führt ein Kommunikationsteilnehmer während seiner Lebenszeit Methoden aus, so wird dies durch *Ausführungssequenzen* dargestellt, die als senkrechte Balken abgebildet werden.
Informationsflüsse und Methodenaufrufe werden als Nachrichten modelliert. Dabei werden synchrone Nachrichten, bei denen der Sender auf die Antwort des Empfängers wartet und asynchrone Nachrichten, bei denen nicht gewartet wird, unterschieden. Synchrone Nachrichten werden durch eine Linie mit ausgefüllter dreieckiger Spitze dargestellt, die entsprechende Antwort durch eine gestrichelte Linie mit Pfeilspitze.



Schleife Werden Nachrichten in einer Schleife versendet, lässt sich dies als *kombiniertes Fragment* modellieren. Diese Fragmente kennzeichnen Bereiche, für die bestimmte Regeln gelten. Eine Ausprägung eines kombinierten Fragments ist der *loop*. Er ist durch ein Rechteck dargestellt, das in der linken oberen Ecke in einem Fünfeck durch den Bezeichner *loop* benannt ist. Die innerhalb des Rechtecks dargestellten Nachrichten werden in einer Schleife ausgeführt.

3.1.2 Extensible Markup Language

Die Extensible Markup Language (XML) ist ein plattformunabhängiger Standard, um Daten in einem strukturierten Format zu beschreiben (vgl. Richards, 2006, S. 1). XML ist eine Metasprache, welche die Basis für die Entwicklung weiterer Auszeichnungssprachen bildet. XML-basierte Sprachen sind etwa XHTML als XML-kompatible Neuformulierung von HTML oder SVG als Standard zur Beschreibung von Vektorgrafiken (vgl. Pemberton et al.

(2002) und Ferraiolo et al. (2003)). Zur Formulierung eigener XML-Sprachen werden Dokumenttypdefinitionen genutzt. Jedem XML-Dokument kann ein Dokumenttyp zugewiesen werden, der den Aufbau eines Dokuments definiert.

Bei der Entwicklung von XML wurde als Ziel formuliert, dass XML-Dokumente einfach über das Internet versendet werden können. Zudem sollen XML-Dokumente einfach zu schreiben, zu verarbeiten und die Daten in einer menschenlesbaren Form kodiert sein (vgl. Bray et al., 2008, Abschnitt 1.1).

Die Nutzung von XML ist nicht auf einzelne Anwendungsbereiche beschränkt. XML-basierte Sprachen können in der IT-Versorgung aller Anwendungsgebiete eingesetzt werden (vgl. Mertens, 2003, S. 1). XML ist somit als Basistechnik zu sehen, auf der XML-basierte Anwendungen aufsetzen.

Dabei wird XML sowohl als Austauschformat als auch als Format zur Speicherung von Daten verwendet (vgl. Goik et al., 2007, S. 99f):

Austauschformat Sollen Nachrichten zwischen Kommunikationspartnern, beispielsweise verschiedenen Anwendungen, ausgetauscht werden, so ist es notwendig, dass Sender und Empfänger das Format, in dem die Nachrichten verschickt werden, verstehen und gleich interpretieren (vgl. Melzer et al., 2007, S. 71).

Ist den Kommunikationspartnern die Dokumenttypdefinition eines XML-Dokuments bekannt, kennen sie den Aufbau und die Bedeutung des Inhalts des Dokuments. Damit ist sichergestellt, dass eine einheitliche Interpretation der Daten oder Nachrichten erfolgt.

Durch die Möglichkeit des konvertierungsfreien Versandes von XML-Dokumenten über das Internet wurde XML dort zum Standard für Austauschformate (vgl. Huemer, 2001, S. 13).

Datenspeicherung XML ist ein Standard zur Beschreibung strukturierter Daten. Daher eignen sich XML-Dokumente zur Speicherung von strukturiertem Content (vgl. Kapitel 2.1) in einer Anwendung.

XML-Dokumente können dabei mit oder ohne expliziter Vorgabe des Aufbaus durch eine Dokumenttypdefinition erstellt werden. Wird keine Dokumenttypdefinition festgelegt, so ist das XML-Dokument nur durch die ertellende Anwendung zu verarbeiten, da der Aufbau des Dokuments nur dieser Anwendung bekannt ist.

Die Verwendung einer Dokumenttypdefinition erlaubt es jeder Anwendung, die diese Definition kennt, das Dokument zu verarbeiten. Somit erlaubt eine Dokumenttypdefinition die Nutzung von Dokumenten durch eine Vielzahl von Anwendungen. Dies macht das Dokument Anwendungsunabhängig und daher langlebig (vgl. Goik et al., 2007, S. 100).

3.1.3 Patterns

Bei der Entwicklung von Informationssystemen werden Softwareentwickler immer wieder mit der Umsetzung gleicher oder ähnlicher Anforderungen konfrontiert (vgl. Herden und Zwanziger, 2004, S. 1). Nach Buschmann et al. (1998, S. 1) ist die menschliche Reaktion darauf die Rückbesinnung auf bereits gefundene Lösungen eines ähnlichen Problems. Es erfolgt also ein Denken in Problem-Lösung-Paaren.

Die Abstraktion konkreter Problem-Lösungs-Paare führt zu Mustern (engl. *patterns*). Christopher Alexander (1979, S. 247f) definiert ein Muster als dreiteilige Regel, die eine Beziehung zwischen einem bestimmten Kontext, einem Problem und einer Lösung beschreibt. Damit stellen sie wiederverwendbare „Best Practices“ für die Lösung eines Problems in einer bestimmten Situation dar (vgl. Herden et al. (2006, S. 66) und Buschmann et al. (1998, S. 8)).

Durch das Rückgreifen auf bewährte Lösungsstrategien wird die Entwicklung komplexer Informationssysteme vereinfacht. Da nicht für jedes Problem eine eigene Lösung erdacht wird, sondern ein erprobtes Pattern genutzt wird, verkürzt sich die Dauer der Entwicklung und die Qualität des Informationssystems wird erhöht (vgl. Buschmann et al., 1998, S. 7).

Thematisch ähnliche Patterns werden zu Patternsprachen zusammengefasst. Diese stellen eine Fachsprache dar, deren Verwendung die Diskussion über komplexe Sachverhalte vereinfacht (vgl. Adams et al., 2002, S. 11). Voraussetzung dafür ist, dass alle Beteiligten das jeweilige Pattern sowie dessen Bedeutung und Inhalt kennen. Dazu müssen Patterns in einer angemessenen Form dargestellt werden. Ein Konzept zur Beschreibung eines Pattern bieten Meszaros und Doble (1996, S. 6f). Danach besteht eine Patternbeschreibung aus einem *Namen*, einer Beschreibung des *Problems* und des *Kontexts*, einer Auflistung einwirkender *Kräfte* sowie der Schilderung der *Lösung*. Dabei identifiziert der Name das Pattern. Das Problem und die Situation, in der das Problem auftritt, werden in der Problemstellung und dem Kontext umrissen. Die Kräfte beschreiben Restriktionen, die bei der Lösung beachtet werden müssen.

Darüber hinaus identifizieren Meszaros und Doble (1996, S. 8f) weitere optionale Bestandteile einer Patternbeschreibung. Dazu gehören Beispiele, Verweise auf andere Patterns sowie Quellcodebeispiele einer Umsetzung der Patternlösung.

Das in dieser Arbeit zur Strukturierung des Informationssystems genutzte Service Layer Pattern (vgl. Kapitel 2.4) lässt sich nach Fowler (2002, S. 133) und Herden et al. (2006, S. 162) wie folgt darstellen:

Service Layer

Problem

Die aufsetzenden Komponenten benutzen oftmals die gleichen Zugriffsmechanismen wie Zugriffskontrollen, Transaktionsmechanismen, Koordination der Application Layer etc. Die Implementation dieser Mechanismen in separaten Komponenten verursachen Redundanzen.

Kontext

Eine Software soll durch Layer (vgl. Buschmann et al., 1998, S. 32) strukturiert werden. Dabei setzen mehrere Komponenten auf der Application Layer auf.

Lösung

Eine Service Layer definiert eine Systemgrenze und die verfügbaren Operationen aus der Perspektive der aufsetzenden Komponenten.

Das Service Layer Pattern lässt sich in der UML wie in Abbildung 3.1 dargestellt abbilden. Jede Schicht ist durch einen Komponente repräsentiert, die über klar definierte Schnittstellen miteinander verbunden sind.

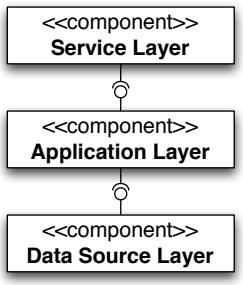


Abbildung 3.1: UML Darstellung des Service Layer Pattern

Anhand dieser Strukturierung werden in den folgenden Kapiteln Softwarelösungen und Techniken vorgestellt, die zur Realisierung dieser Schichten verwendet werden können.

3.2 Content Repositories nach JSR 170

Die Application Layer des konzipierten Informationssystems wird durch ein Content Repository umgesetzt (vgl. Kapitel 2.4 und Abbildung 3.2). Der Begriff Content Repository wurde von David Nüscheler, Chief Technology Officer der Firma Day[3], geprägt. Er bezeichnet ein Content Repository als ein high-level Information-Management-System, welches eine Erweiterung traditioneller Datenrepositories ist. Zusätzlich zu den Funktionalitäten eines Datenrepository werden damit so genannte *„content services“* wie Versionierung, Volltextsuche, feingranulare Zugriffskontrolle, Kategorisierung von Content und Event Monitoring implementiert (vgl. Nüscheler, 2003). Die *„content services“* umfassen die in Kapitel 2.2 als für ein ECMS zwingend benötigt identifizierte Funktionalität.

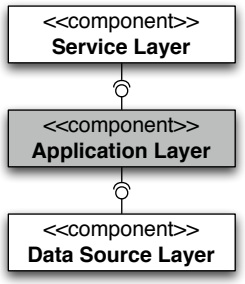


Abbildung 3.2: Service Layer Pattern - ApplicationLayer

Nüscheler (2003) nennt als erklärtes Ziel der Entwicklung von Content Repositories die Vereinfachung des Datenaustauschs zwischen Anwendungen, die Inhalte produzieren und Anwendungen, die Inhalte nutzen. Als Beispiel sei hier das Erstellen von Inhalten in einem Content-Management-System und deren Publizierung in einem Portal genannt. Mit der Spezifizierung einer Content Repository API sollte der Grundstein für eine industrieweite Content-Infrastruktur, bei welcher der produkt- und herstellerunabhängige Zugriff auf Content möglich ist, geschaffen werden (vgl. Nüscheler und Piegaze, 2005, S. 11).

[3] http://www.day.com Stand: 16.03.2009

Damit erfüllt ein Content Repository die am Ende von Kapitel 2.2 genannten grundlegenden und Content-Management-spezifischen Anforderungen an ein Repository. Seine Funktion als Grundstein einer Content-Infrastruktur unterstützt das Ziel des ECM, eine einheitliche Plattform für den Zugriff und das Management von Content zu schaffen (vgl. Kapitel 2.2).

Ein Content Repository bildet damit die Applikationslogik ab, die der Application Layer des Service Layer Pattern zugeordnet wird (vgl. Abbildung 3.2).

Ein Content Repository besteht nach Bernstein (1997, S. 1ff) aus:

Repository Information Model Ein Repository Information Model ist das Datenmodell eines Repository. Mit ihm werden die Inhalte des Repository beschrieben.

API Eine API erlaubt den Zugriff auf die Funktionalität der Repository-Implementierung.

Repository Engine Die Repository Engine ist die Implementierung der API. Sie verwaltet die Inhalte des Repository.

Persistenter Datenspeicher Ein persistenter Datenspeicher oder eine Schnittstelle zu einem persistenten Datenspeicher wird benötigt, um die Inhalte des Repository dauerhaft zu speichern.

Content Repositories wurden mit dem Java Specification Request 170 (JSR 170) durch den Java Community Process (JCP) spezifiziert. Der JCP ist ein auf Initiative von Sun Microsystems[4] definierter formaler Standardisierungsprozess, den Weiterentwicklungen an der Programmiersprache Java durchlaufen, bevor sie offizieller Bestandteil der Sprache werden.

Ausgehend von der Definition eines Content Repository nach Nüscheler (2003) und der Bestimmung der Bestandteile eines Repository von Bernstein

[4] http://www.sun.com Stand: 16.03.2009

(1997) beinhaltet der im Juni 2005 verabschiedete Standard JSR 170 eine technische Spezifikation, in der das Repository Information Model und die API sowie eine Referenzimplementierung beschrieben sind und ein Technology Compatibility Kit zum Testen der Konformität von Implementierungen zum Standard (vgl. Curran, 2004).

Der JSR 170 ist ein Standard für Java Content Repositories. Die im Standard enthaltene API wird auch für andere Programmiersprachen umgesetzt. Derzeit erfolgt im Rahmen der Entwicklung des Content-Management-Systems Typo3[5] die Entwicklung eines Content Repository in PHP, welches die JSR 170 API nutzt (vgl. Dambekalns, 2007, S. 6). Für .NET und PEARL existieren Wrapper um auf ein Java Content Repository zuzugreifen[6,7]. Ein Wrapper ummantelt eine Softwarekomponente, beispielsweise um unerwünschte Eigenschaften zu verbergen (vgl. Fischer und Hofer, 2008, S. 934). In diesem Fall wird verborgen, dass das Content Repository in Java implementiert ist und so von .NET- oder PEARL-Komponenten genutzt werden kann.

Den angeführten Umsetzungen ist gemein, dass sie sich auf den JSR 170 beziehen und dessen API nutzen. Nach derzeitigem Kenntnisstand ist der JSR 170 der einzige Standard für Content Repositories. Daher soll in den folgenden Abschnitten ein Content Repository nach JSR 170 anhand der vier Bestandteile eines Repository nach Bernstein beschrieben werden.

3.2.1 Repository Information Model

Das Repository Information Model ist das Datenmodell des Java Content Repository und spezifiziert Struktur und Semantik des im Content Repository verwalteten Content (vgl. Bernstein, 1997, S. 1). Der JSR 170 spezifiziert ein Metamodell, nach dem das Repository Information Model eines Con-

[5] http://typo3.org Stand: 16.03.2009
[6] http://issues.apache.org/jira/browse/JCR-675 Stand : 16.03.2009
[7] http://cpan.uwinnipeg.ca/dist/Java-JCR Stand: 16.03.2009

tent Repository aufgebaut ist (vgl. Abbildung 3.3). Ein Repository besteht aus mindestens einem Workspace. Ein Workspaces enthält eine von anderen Workspaces separierte baumartige Item-Struktur. Jeder Workspace besitzt einen Wurzelknoten (engl. *rootnode*). Dieser Knoten kann beliebig viele andere Knoten als Kindknoten besitzen.

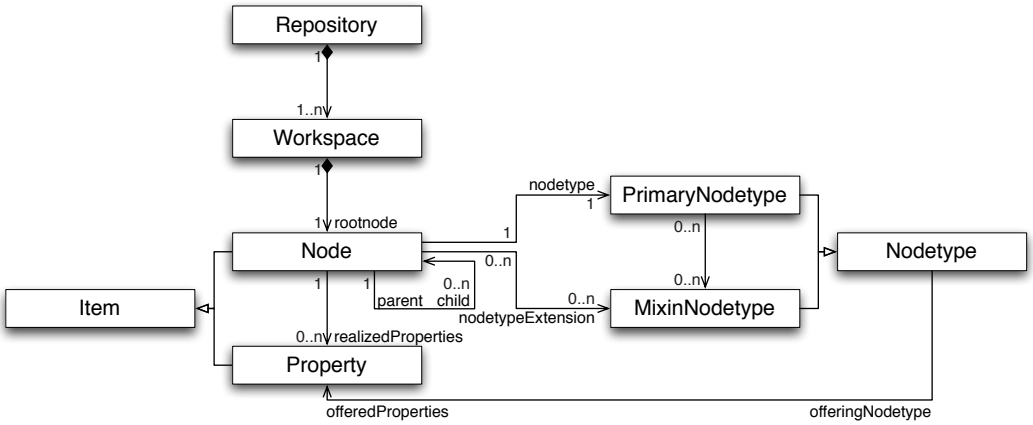


Abbildung 3.3: Das Metamodell des JSR 170 für Datenmodelle eines Repository

Der Aufbau eines Knotens eines Workspaces wird durch seinen Knotentyp (engl. *nodetype*) festgelegt. Dieser Aufbau wird durch einen so genannten *PrimaryNodetype* bestimmt, der definiert, welche Knotentypen Kindknoten dieses Knotens sein dürfen und welche Eigenschaften (engl. *properties*) ein Knoten besitzt. Darüber hinaus kann ein Knoten einen *MixinNodetype* besitzen, der zusätzliche Eigenschaften für diesen Knoten definiert. In einem Repository Information Model dienen Knoten zur Strukturierung des Content, Eigenschaften zur Speicherung der Daten.

Nach Rückel et al. (2007, S. 85) erfolgt die Erstellung des Repository Information Model in vier Schritten:

Ermitteln der Content-Typen In diesem ersten Schritt werden Content-Typen, also inhaltlich ähnlicher Content mit gleicher Struktur, ermittelt und diese Struktur im Repository Information Model abgebildet

(Klassifizierung). Für ein JSR 170 konformes Content Repository ist das Ergebnis dieser Klassifizierung die Definition von Nodetypes.

Erstellen einer Ablagestruktur Der zweite Schritt ist die Erstellung einer Ablagestruktur. Dabei wird festgelegt, welche Content-Typen in der Baumstruktur des Content Repository unter welchen Content-Typen abgelegt werden dürfen. So lässt sich zum Beispiel festlegen, dass ein Content des Content-Typs „Artikel“ nur unter einem Content des Content-Typs „Artikelverzeichnis“ abgelegt werden darf.

Spezifizieren von Metadatenattributen Metadaten werden zur Beschreibung des Content genutzt. Mögliche Metadaten sind Daten über den Autor, das Erstellungsdatum, Sprache oder Format des Content. Eine Auflistung von Metadatenattributen zur Beschreibung eines Content bietet das Dublin Core Metadata Element Set (vgl. Dublin Core Metadata Initiative, 2008).

Ein Content Repository nutzt diese Metadaten zur Verwaltung des Content, beispielsweise um den gesamten von einem Autor in einem bestimmten Zeitraum erstellten Content zu finden (vgl. Dornfest und Brickley, 2001, S. 192).

Festlegen der Zugriffsrechte Als letzten Schritt der Erstellung eines Repository Information Model werden die Zugriffsrechte auf die Inhalte des Content festgelegt. Dabei wird definiert, welche Person oder Rolle Rechte für das Erstellen, Ändern, Lesen oder Löschen eines Content hat (vgl. Seufert (2002, S. 3)).

Eine Instanz des so definierten Repository Information Model als Datenmodell des Content Repository bildet den Content als Baum ab. Abbildung 3.4 zeigt eine exemplarische Instanz eines Repository Information Model. Dabei werden Knoten durch Kreise und Eigenschaften durch Rechtecke repräsentiert. Es ist erkennbar, dass die Daten in den Eigenschaften gespeichert sind.

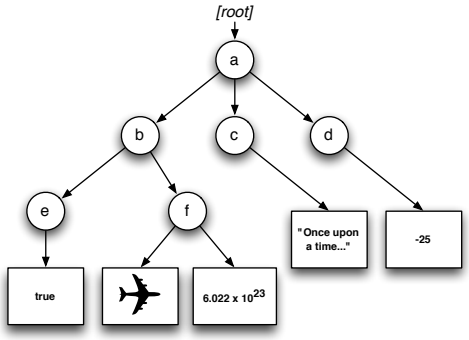


Abbildung 3.4: Exemplarische Instanz eines Repository Information Model (nach Nüscheler, 2003, S. 16)

3.2.2 API

Neben einer Beschreibung des Repository Information Model beinhaltet die technische Spezifikation des JSR 170 eine API, welche die Methoden für den Zugriff auf das Content Repository beschreibt.

Um das Content Repository benutzen zu können, muss zuvor eine Anmeldung (engl. *login*) erfolgen. Dazu stellt die API eine *login*-Methode des Repository-Objekt bereit. Zum Login wird ein Berechtigungsnachweis (engl. *credential*), beispielsweise eine Kombination von Benutzername und Passwort, sowie der Name des Workspaces, bei dem die Anmeldung erfolgen soll, erwartet. Bei einer erfolgreichen Anmeldung wird ein Session-Objekt zurückgeliefert. Mit diesem Session-Objekt kann der Benutzer auf jeden Content des Content Repository zugreifen, für dessen Zugriff er autorisiert ist.

Am Ende der Benutzung des Repository meldet sich der Benutzer mit der *logout*-Methode wieder ab. In Abbildung 3.5 wird diese Abfolge in einem Sequenzdiagramm dargestellt.

Die Methoden der API sind in drei Level eingeteilt (vgl. Nüscheler und Piegaze, 2005, S. 12):

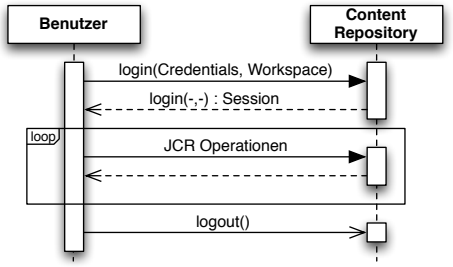


Abbildung 3.5: Sequenzdiagramm der Benutzung des Content Repository

Level 1 - lesender Zugriff Beinhaltet das Bewegen im Content-Baum, das Durchsuchen des Repository, das Lesen von Inhalten sowie den Export des Content in ein XML-Dokument. Um das Durchsuchen des Content zu beschleunigen, erstellt das Content Repository einen Index über den verwalteten Content. Die Methoden in Level 1 reichen aus, um Anwendungen zu schreiben, die nur lesend auf den Content zugreifen und keinen neuen Content erstellen. Beispiele hierfür sind Portlets, die Inhalte eines Content-Management-Systems in einem Portal anzeigen.

Level 2 - schreibender Zugriff Beinhaltet das Einfügen, das Ändern, das Löschen und das Importieren von Inhalten. Mit den Methoden des Level 2 ist es möglich Anwendungen zu erstellen, die eigene Inhalte erstellen sowie bestehende Inhalte ändern.

Optionale Funktionalität Der dritte Level sind die optionalen Funktionalitäten eines Content Repository. Sie beinhaltet das Sperren, Versionieren und Überwachen von Inhalten. Des Weiteren werden Transaktionen unterstützt und die Abfrage des Content über SQL-Statements beschrieben.

In den Level 1 und 2 werden die grundlegenden Funktionen eines Repository umgesetzt (vgl. Kapitel 2.2). Die optionalen Funktionalitäten umfassen die Content-Management-spezifischen Funktionen. Damit sind sie nach Nüscheler (2003) das Merkmal, durch das sich Content Repositories von

anderen Repositories unterscheiden. Die optionalen Funktionalitäten bilden die Content-Management-Funktionalität ab, die im inhaltszentrierten Ansatz Gegenstand der Funktionsintegration sind (vgl. Kapitel 2.3).

3.2.3 Repository Engine

Wie in Abschnitt 3.2 gezeigt, implementiert die Repository Engine die Methoden für den Zugriff auf den Content. Es stellt also die Implementierung der API dar. Der JSR 170 umfasst neben der technischen Spezifikation eine Referenzimplementierung des Standards. Diese Referenzimplementierung wurde von der Firma Day entwickelt und dann von der Apache Software Foundation[8] unter dem Namen Jackrabbit[9] weiter entwickelt. Jackrabbit implementiert die volle Funktionalität der JSR 170 Level 1 und 2 sowie die optionalen Funktionalitäten und ist frei erhältlich. Weitere freie Implementierungen bieten JBoss im JBoss Portal[10] und Alfresco Software in der Alfresco Content Plattform[11].

Aus der anfänglichen Referenzimplementierung entwickelte die Firma Day ihr kommerzielles Content Repository Day CRX (Content Repository Extreme), welches gegenüber Jackrabbit weitere Zugriffsprotokolle und erweiterte Zugriffskontrollen erlaubt. Weitere kommerzielle Implementierungen des JSR 170 sind von IBM mit dem IBM WebSphere Portal[12] sowie eXo Plattform mit dem eXo JCR[13] erhältlich.

[8] http://www.apache.org Stand: 16.03.2009
[9] http://jackrabbit.apache.org Stand: 16.03.2009
[10] http://www.jboss.org/jbossportal Stand: 16.03.2009
[11] http://www.alfresco.com/products/platform Stand: 16.03.2009
[12] http://www.ibm.com/websphere/portal Stand: 16.03.2009
[13] http://www.exoplatform.com Stand: 16.03.2009

3.2.4 Persistente Datenspeicher für den JSR 170

Zur dauerhaften Speicherung des Content benötigt ein Content Repository einen persistenten Datenspeicher oder eine Schnittstelle zu einem solchen Datenspeicher (vgl. Abschnitt 3.2). Den in Abschnitt 3.2.3 genannten Implementationen des JSR 170 ist gemein, dass sie keinen persistenten Datenspeicher vorschreiben, sondern eine Schnittstelle anbieten, um so beliebige Datenspeicher zu erlauben. Der Zugriff auf den Datenspeicher erfolgt über eine Implementierung dieser Schnittstelle. Diese Schnittstelle ist nicht standardisiert und daher herstellerabhängig.

Derzeit existieren für die genannten Implementierungen Lösungen, welche die Speicherung des Content eines Content Repository als XML-File im Dateisystem, in einer Datenbank oder auch in anderen Anwendungen wie OpenText Livelink, Microsoft Sharepoint oder IBM Lotus Notes ermöglichen[14].

Ein Datenspeicher auf Basis eines Peer-to-Peer-Netzwerks, wie er Bestandteil dieser Arbeit ist, existiert nach derzeitigem Kenntnisstand nicht. Im folgenden Kapitel wird daher auf die Grundlagen von Peer-to-Peer-Netzwerken eingegangen.

3.2.5 Implikationen für die Konzeption eines Content Repository

Ein Content Repository umfasst die grundlegenden Funktionalitäten eines Repository und erweitert diese um Content-Management-spezifische Funktionen (vgl. Kapitel 2.4). Dazu besteht ein Content Repository aus einem Repository Information Model, einer API, einer Repository Engine und ei-

[14] http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ Stand: 16.03.2009
http://www.day.com/content/day/en/products/crx/jcr_connectors.html
Stand: 16.03.2009

ner Schnittstelle zu einem persistenten Datenspeicher. Ein Konzept, dass ein Content Repository umfasst, muss diese Elemente enthalten.

3.3 Data Source

Ein Content Repository benötigt einen Datenspeicher, in dem der Content gespeichert wird (vgl. Kapitel 3.2). Im Service Layer Pattern entspricht dieses der Data Source Layer (vgl. Abbildung 3.6).

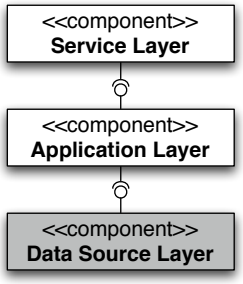


Abbildung 3.6: Service Layer Pattern - Data Source Layer

Wie in Abschnitt 3.2.3 genannt, werden dazu Datenbanken, XML-Dateien im Dateisystem oder andere Anwendungen genutzt. Ein Content Repository mit Peer-to-Peer-Netzwerk als Datenspeicher existiert derzeit nicht. Daher werden Peer-to-Peer-Netzwerke in den folgenden Abschnitten näher erläutert und in Kapitel 4 als Datenspeicher des konzipierten Content Repository genutzt.

3.3.1 Peer-to-Peer-Netzwerke

Internetbasierte Anwendungen müssen sich den Herausforderungen eines steigenden Datenaufkommens und immer weiter steigender Teilnehmerzahlen stellen. So nahm der Datendurchsatz am DE-CIX Internetknoten zwischen 2007 und 2008 um 300% zu[15], die Zahl der Internetnutzer in Deutschland stieg zwischen 2005 und 2008 um 20% (vgl. Arbeitsgemeinschaft Online Forschung (2005, S. 6) und Arbeitsgemeinschaft Online Forschung (2008, S.5)).

[15] http://www.de-cix.net/content/network/Traffic-Statistics.html Stand: 16.03.2009

Steinmetz und Wehrle (2004, S. 1) identifizieren drei grundlegende Anforderungen an Anwendungen, die das Internet nutzen. Sie müssen *skalierbar* sein um ein Wachstum der Teilnehmerzahlen bewältigen zu können. Daher ist darauf zu achten, dass Engpässe bei Bandbreite, Speicherplatz und Rechenkapazität vermieden werden. Anwendungen müssen *sicher* und *verlässlich* sein. Dazu muss die Anwendung selber gegen Angriffe geschützt sein und ein zuverlässiger Schutz der Daten der Benutzer gewährleistet werden. Als dritte Anforderung identifizieren Steinmetz und Wehrle eine verbesserte *Flexibilität* und *Dienstgüte*. Dabei wird die Umsetzbarkeit und Integration neuer Dienste, wie Gruppenkommunikation und Mobilität (vgl. Stoica et al., 2002; Zhuang et al., 2003), in vorhandene Anwendungen als entscheidende Kriterien für den Erfolg neuer Internet-Technologien gewertet.

Das Internet funktioniert prinzipiell nach dem Client-Server-Prinzip (vgl. Rautenstrauch und Schulze, 2003, S. 183). Nach dem Client-Server-Prinzip lassen sich zwei Teilnehmer einer Kommunikation in Dienstanbieter (Server) und Dienstnachfrager (Client) einteilen (vgl. Lassmann, 2006, S. 147f). Für internetbasierte Anwendungen bedeutet dies in der Regel, dass der Internetnutzer mit Hilfe seines Browsers (Client) eine Webseite von einem Webserver (Server) abruft oder mit Hilfe seines E-Mail-Programms (Client) seine Mails von einem Mailserver (Server) abruft.

In einer Client-Server-Architektur sind Server zentrale Instanzen, deren Ressourcen sich als potentielle Engpässe erweisen können. Die Zentralität eines Servers macht ihn zu einem Single-Point-of-Failure im Falle eines Angriffs, beispielsweise durch eine Denial-of-Service-Attacke (DOS-Attacke), bei der versucht wird, den Server in seiner Fähigkeit den Dienst anzubieten einzuschränken oder die Ausbringung des Dienstes ganz zu verhindern (vgl. Huang und Gouda, 2006, S. 25).

Eine Client-Server-Architektur kann die von Steinmetz und Wehrle formulierten Anforderungen an eine internetbasierre Anwendung auf Grund der

Verwendung eines Servers als zentralisierten Diensterbringer nicht mehr in allen Fällen umfassend erfüllen.

Peer-to-Peer-Netzwerke (P2P-Netzwerke) sind Netzwerke, deren Teilnehmer gleichberechtigt sind. Sie werden Peers (deutsch: *Gleichgestellter*) genannt, da sie im Gegensatz zur Trennung in Client und Server sowohl Anfragen von anderen Peers bearbeiten als auch eigene Anfragen stellen können (vgl. Mahlmann und Schindelhauer, 2007, S. 6f).

Nach Milojicic et al. (2003, S. 1) realisiert ein Peer-to-Peer-Netzwerk eine Funktion durch die Nutzung dezentralisierter Ressourcen. Dabei werden *Distributed Computing*, *Content Sharing* und *Communication and Collaboration* als mögliche Funktionen gennant.

Distributed Computing Die von Distributed-Computing-Anwendungen, wie SETI@Home genutzten Ressourcen sind von Netzwerkteilnehmern bereitgestellte CPU-Zyklen (vgl. Andersen, 2001, S. 67ff).

Content Sharing Content-Sharing-Anwendungen nutzen Festplattenspeicher oder Inhalte der Teilnehmer (vgl. Dabek et al. (2001) und Oram (2000)).

Communication-and-Collaboration-Anwendungen Anwendungen mit dieser Funktionalität basieren darauf, dass Personen als Ressource angesehen werden, die ortsunabhängig und dezentral an etwas teilhaben (vgl. Miller, 2001, S. 77ff).

Shirky (2001, S. 22f) ergänzt die Definition von Milojicic um die Feststellung, dass genutzte Ressourcen von Peers mit unsicherer Konnektivität und unvorhersehbarer Netzwerkadresse bereitgestellt werden. Das bedeutet, dass die Ressourcen von Teilnehmern bereitgestellt werden, die nur unregelmäßig Teil des Netzwerks sind und dabei unterschiedliche Netzwerkadressen besitzen, weil diese entweder von einem Internetprovider zugewiesen werden oder unterschiedliche Internetprovider verwendet werden.

Ein Peer-to-Peer-Netzwerk realisiert also eine Funktion durch die Nutzung dezentraler Ressourcen, die von Peers mit unsicherer Konnektivität und unvorhersehbarer Netzwerkadresse bereitgestellt werden.

Ein Peer-to-Peer-Netzwerk, dass eine Content-Sharing-Funktion mit als Ressource bereitgestelltem Festplattenspeicher realisiert, kann als Datenspeicher eines Content Repository genutzt werden.

Nach Ngo et al. (2001, S. 3) existieren verschiedene Architekturen für Peer-to-Peer-Netzwerke, die sich in Ihren Merkmalen unterscheiden. Im nachfolgenden Abschnitt werden diese Merkmale von Peer-to-Peer-Netzwerken vorgestellt.

3.3.2 Merkmale von Peer-to-Peer-Netzwerken

Herden et al. (2006) stellen fest, dass jedes Anwendungsgebiet von Peer-to-Peer-Netzwerken unterschiedliche Charakteristika aufweist. In Abbildung 3.7 sind Merkmale von Peer-to-Peer-Netzwerken und deren mögliche Ausprägungen in einem morphologischen Kasten dargestellt. Im Folgenden werden die Merkmale und Ausprägungen näher erläutert:

Merkmal	Merkmalsausprägung					
Netzwerk-topologie	zentralisiert		dezentralisiert			
			hybrid		vollständig dezentralisiert	
Datenhaltung	strukturiert		teilstrukturiert		unstrukturiert	
Suchmethode	zentrales Verzeichnis		Flooding		Document Routing	
Identität	Local			Remote		
Anonymität	organisatorisch			technisch		
	Autor	Anbieter	Nachfrager	Server	Dokument	Anfrage
Performanz	Replikation		Caching		Intelligent Routing	
Sicherheit	Multi-Key-Encryption	Sandboxing	DRM	Reputation	Firewall	

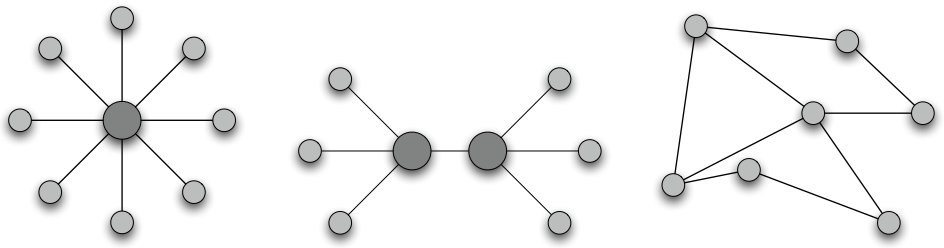
Abbildung 3.7: Merkmale eines Peer-to-Peer-Netzwerks
(Herden et al., 2006, S. 73)

Netzwerktopologie Mit der Netzwerktopologie wird die Topologie, also die Struktur und der Aufbau eines Netzwerks beschrieben. Dabei werden zentralisierte und dezentralisierte Topologien unterschieden. Diese, sowie eine hybride Netzwerktopolgie, werden in Abbildung 3.8 dargestellt. Dabei entsprechen Punkte den Teilnehmern in einem Netzwerk und Linien Verbindungen zwischen zwei Teilnehmern.

Zentralisierte Topologien besitzen eine zentrale Instanz, etwa als Verzeichnisdienst oder Authentifizierungsserver (vgl. Yang und Garcia-Molina, 2002; Miller, 2001, S. 9; S. 85). *Dezentrale Topologien* werden weiter in *vollständig dezentrale* und *hybride* Strukturen unterschieden. In vollständig dezentralen Netzwerken sind alle Teilnehmer gleichberechtigt und kommunizieren direkt miteinander. Sie sind also Peers im eigentlichen Sinne. Ressourcen und Berechtigungen werden vom Netzwerk selbst verwaltet. Hybride Topologien bestehen aus Peers sowie sogenannten Superpeers, auch Supernodes genannt. Diese stellen mehr Ressourcen als Peers bereit und können daher erweiterte Services anbieten. Sie dienen beispielsweise als Gateways zum Peer-to-Peer-Netzwerk, an denen sich Peers anmelden (vgl. Abbildung 3.8(b)) und übernehmen so erweiterte Routingaufgaben (vgl. Yang und Garcia-Molina (2003, 4f) und Baset und Schulzrinne (2006, S. 1)). Weitere Zusatzdienste von Supernodes können ein erweitertes Caching, d. h. die redundante Speicherung von Ressourcen eines Teilnetzes oder die Verwaltung der Suchergebnisse eines Teilnetzes sein (vgl. Xu und Hu (2003, S. 3) und Mahlmann und Schindelhauer (2007, S. 244)).

Datenhaltung Die Art der Datenhaltung beschreibt den Aufbau der Verwaltung der Daten im Netzwerk.

Besitzt ein Netzwerk eine strukturierte Datenhaltung, so bestehen Regeln, die festlegen wo im Netzwerk sich ein Inhalt befindet. Bei teilstrukturierter Datenhaltung existieren Hinweise im Netzwerk, die auf den Ablageort eines Inhalts schließen lassen. In einer unstrukturierte Datenhaltung sind keine Informationen zum Ablageort von Inhalten bekannt (vgl. Herden et al., 2006, S. 74). Je nach Art der Datenhal-



(a) zentral (b) hybrid (c) vollständig dezentral

Abbildung 3.8: Übersicht der Netzwerktopologien in Peer-to-Peer-Netzwerken

tung sind unterschiedliche Suchverfahren nötig, um die Inhalte wieder zu finden.

Suchmethode Mit der verwendeten Suchmethode wird die von einem Peer-to-Peer-Netzwerk verwendete Methode zur Suche nach Inhalten beschrieben. Nach Milojicic et al. (2003, S. 10) werden die Suche unter Verwendung eines zentralen Verzeichnisses, das Flooding und Document Routing unterschieden.

Die Suche mit einem *zentralen Verzeichnis* setzt eine zentralisierte Netzwerktopologie voraus, da eine zentrale Instanz das Verzeichnis verwalten muss. Eine Suchanfrage wird immer an diese Instanz gesendet und von dieser beantwortet. Im *Flooding* wird die Suchanfrage vom suchenden Peer aus solange an benachbarte Peers weitergegeben, bis ein Suchergebnis feststeht. Das *Document Routing* basiert darauf, dass bei einer (teil-) strukturierten Datenhaltung Rückschlüsse auf die Position des gesuchten Inhalts gezogen werden können.

Abschnitt 3.3.3 geht bei der Betrachtung von Routing- und Suchmethoden näher auf diese drei Merkmale eines Peer-to-Peer-Netzwerks ein.

Performanz Mit der Performanz eines Peer-to-Peer-Netzwerks wird beschrieben, wie schnell ein Inhalt im Peer-to-Peer-Netzwerk gefunden wird und wieviel Bandbreite dazu verwendet wird. Die Performanz

kann durch *Replikation*, *Caching* und *Intelligent Routing* verbessert werden (vgl. Milojicic et al., 2003, S. 16).

Durch *Replikation* werden mehrere Kopien eines Inhalts im Netzwerk erstellt. Da der dem Nachfrager am nächsten gelegene Anbieter des Inhalts gewählt werden kann, wird so der Weg zwischen Nachfrager und Anbieter eines Inhalts verringert, was die Performanz steigert.

Wichtig ist, dass bei der redundanten Haltung von Daten Änderungen an einer der Kopien auch auf die restlichen Kopien übertragen werden, um einen konsistenten Zustand der Inhalte des Netzwerks sicherzustellen.

Durch *Caching* werden bereits bearbeitete Anfragen an einen Peer gespeichert, so dass bei einem erneuten Auftreten dieser Anfrage die Antwort direkt gegeben werden kann, ohne sie an andere Peers weiterleiten zu müssen. Dies senkt das Nachrichtenaufkommen und damit die benutzte Bandbreite in einem Peer-to-Peer-Netzwerk.

Ziel des *Intelligent Routing* ist, dass angefragte Inhalte gezielt aufgefunden werden. Dazu kann bei der Anfrage direkt bestimmt werden, von welchem Peer ein Inhalt verwaltet wird (vgl. dazu die Suchmethode *Document Routing*).

Identität Das Merkmal Identität beschreibt, ob die Identifizierung eines Benutzers oder Peers lokal oder entfernt erfolgt. Lokal bedeutet, dass die Identifizierung auf dem Rechner, auf dem der Peer läuft, durchgeführt wird. Entfernt meint, dass die Identifizierung auf einer anderen, eventuell zentralen Instanz geschieht. Eine detailliertere Beschreibung des Merkmals Identität bieten Ngo et al. (2001, S. 5).

Anonymität In Peer-to-Peer-Netzwerken werden Inhalte verteilt auf allen Peers des Netzwerks gespeichert. Dabei kann es wichtig sein, dass Anbieter und Nachfrager eines Inhalts oder der Inhalt selbst nicht identifiziert werden können, z.B. um eine Zensur zu verhindern (vgl. Langley, 2001, S. 123). Eine eingehende Auflistung und Beschreibung von Methoden zur Erreichung von Anonymität in Peer-to-Peer-Netzwerken

bieten Dingledine et al. (2001b, S. 157ff) und Mahlmann und Schindelhauer (2007, S. 209ff).

Sicherheit Das Merkmal Sicherheit umfasst Mechanismen, um ein Peer-to-Peer-Netzwerk gegen Missbrauch zu schützen. Missbrauch umfasst dabei die Einbringung von Fehlinformation oder Inhalt niederer Qualität (Dingledine et al., 2001a), den Angriff auf die Verfügbarkeit des Netzwerks oder den unberechtigten Zugriff auf Inhalte des Netzwerks (vgl. Mahlmann und Schindelhauer, 2007, S. 199).

Eine Beschreibung der in Abbildung 3.7 aufgelisteten Mechanismen liefern Waldman und Rubin (2001, S. 242ff) sowie Milojicic et al. (2003, S. 17).

3.3.3 Routing- und Suchverfahren in Peer-to-Peer-Netzwerken

Durch ihre variablen Konnektivität und das Merkmal der Anonymität sind Teilnehmer in einem Peer-to-Peer-Netzwerk nicht mehr durch ihre Lokation (Netzwerkadresse) zu erreichen (vgl. Abschnitt 3.3.1 und 3.3.2). In Peer-to-Peer-Netzwerken muss das Ziel einer Nachricht also nicht eine spezielle Netzwerkadresse, sondern die gesuchte Ressource selbst sein (vgl. Steinmetz und Wehrle, 2004, S. 52). Es erfolgt eine Adressierung der Ressource, nicht eine Adressierung des Netzwerkteilnehmers, der die Ressource bereitstellt (engl. *resource centric addressing*).

Daher ist das Routing, also das Finden eines Wegs zum Empfänger einer Nachricht im Netzwerk, nicht Abhängig von der Netzwerkadresse des Teilnehmers (engl. *location based*) sondern Abhängig vom gesuchten Inhalt (engl. *content based*).

Die benötigten Protokolle zur Kommunikation in einem Netzwerk sind im Standard ISO/IEC 7498-1 der International Organisation for Standardizati-

on (ISO) im so genannten ISO/OSI-Modell festgehalten (vgl. ISO/IEC (1994, S. 32ff). Der Quasi-Standard für die im ISO/OSI-Modell definierte Netzwerkschicht zum Routing von Nachrichten ist das Internet Protocol (IP) (vgl. Rautenstrauch und Schulze, 2003, S. 171). Das Internet Protocol findet den weg zum Ziel abhängig von dessen Netzwerkadresse (vgl. Comer, 2008, S. 93). In einem Peer-to-Peer-Netzwerk können also die Routing-Methoden der Netzwerkschicht und des Internet Protocol nicht genutzt werden.

Peer-to-Peer-Netzwerke setzen als *Overlay-Netzwerke* auf IP-Netzwerken auf (vgl. Darlagiannis et al., 2004, S. 371) und benötigen daher eigene, inhaltsbasierte Routing-Mechanismen, die für eine gesuchte Ressource den Peer, der diese Ressource bereitstellt, bestimmt und dessen Netzwerkadresse an die unterliegende Netzwerkschicht des IP-Netzwerks übergibt.

Das Routing in Peer-to-Peer-Netzwerken entspricht also immer der Suche nach der gewünschten Ressource und dem Peer, der diese bereitstellt. Daher werden im Folgenden die drei in Abschnitt 3.3.2 genannten Suchmethoden in Peer-to-Peer-Netzwerken erläutert.

Suche über ein zentrales Verzeichnis

Napster, eines der ersten Peer-to-Peer-Netzwerke auf Basis eines Verzeichnisses, ist eine im Jahr 1999 von Shawn Fanning entwickelte Anwendung, um Dateien auf anderen Rechnern zu finden und herunter zu laden. Es ist als Anwendung mit zentralisierter Netzwerktopologie, unstrukturierter Datenhaltung und einer Suche über ein zentrales Verzeichnis aufgebaut (vgl. Abbildung 3.9). Wie Abbildung 3.10 verdeutlicht, besitzt ein solches Netzwerk einen zentralen Server, der Informationen von jedem Client im Netzwerk nutzt, um einen Index über alle im Netzwerk verfügbaren Dateien zu erstellen, der unter anderem für jeden Dateinamen die IP-Adressen der Peers beinhaltet, die diese Datei bereitstellen.

Merkmal	Merkmalsausprägung		
Netzwerk-topologie	zentralisiert	dezentralisiert	
		hybrid	vollständig dezentralisiert
Datenhaltung	strukturiert	teilstrukturiert	unstrukturiert
Suchmethode	zentrales Verzeichnis	Flooding	Document Routing

Abbildung 3.9: Merkmalsausprägungen im Morphologischen Kasten für Napster – dunkle Bereiche markieren Eigenschaften des Napster-Netzwerks

Eine Suche im Netzwerk erfolgt immer über den zentralen Server. Eine Anfrage (engl. *query*) wird an den Server geschickt und dort mit Hilfe des Index verarbeitet. Der Server sendet eine Liste von Netzwerkteilnehmern, welche die gewünschte Datei anbieten, an den anfragenden Client als Antwort (engl. *reply*) zurück. Danach stellt dieser Client eine Verbindung zu einem der Clients aus der Liste her und läd die Datei direkt, also von Peer zu Peer, herunter (vgl. Shirky, 2001, S. 29).

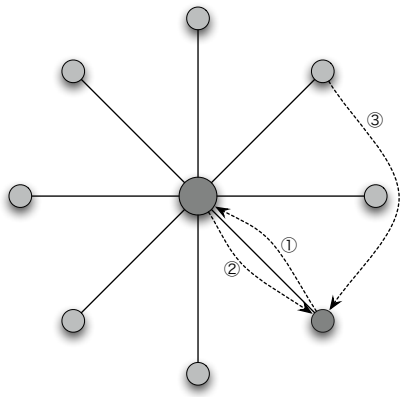


Abbildung 3.10: Aufbau des Napster-Netzwerks
(1) Query (2) Reply (3) direkter Download
(nach Mahlmann und Schindelhauer, 2007, S. 56)

Die Suche über ein zentrales Verzeichnis ermöglicht eine schnelle und effiziente Suche in den verfügbaren Inhalten (vgl. Mahlmann und Schindelhauer, 2007, S. 57). Die partielle Nutzung einer Client-Server-Struktur schafft aber einen Angriffspunkt für Denial-of-Service-Angriffe und einen Single-Point-of Failure. Steigende Teilnehmerzahlen werfen das Problem der Skalierbarkeit

auf, da der Server mit steigenden Teilnehmerzahlen eine steigende Anzahl an Anfragen verarbeiten muss (vgl. Abschnitt 3.3.1). Die Nutzung eines zentralen Server widerspricht dem Merkmal der Anonymität (vgl. Abschnitt 3.3.2), da ein Verantwortlicher für den Server festzustellen ist. Dadurch wurde das Napster-Netzwerk angreifbar und im Februar 2001 als freies Netzwerk abgestellt (vgl. Röttgers, 2003, S. 23).

Suche über Flooding

Als Beispiel eines Peer-to-Peer-Netzwerks, dass Flooding zur Suche nach Inhalten verwendet, soll hier das Gnutella-Netzwerk vorgestellt werden. Wie Napster ist Gnutella ein File-Sharing-Netzwerk mit unstrukturierter Datenhaltung. Es besitzt aber statt eines zentralen Servers eine komplett dezentrale Netzwerktopologie (vgl. Mahlmann und Schindelhauer (2007, S. 57) und Abbildung 3.11).

Merkmal	Merkmalsausprägung		
Netzwerk-topologie	zentralisiert	dezentralisiert	
		hybrid	vollständig dezentralisiert
Datenhaltung	strukturiert	teilstrukturiert	unstrukturiert
Suchmethode	zentrales Verzeichnis	Flooding	Document Routing

Abbildung 3.11: Merkmalsausprägungen im Morphologischen Kasten für Gnutella – dunkle Bereiche markieren Eigenschaften des Gnutella-Netzwerks

Die Aufbaustruktur eines Netzwerkes dieser Art entsteht zufällig, abhängig davon, welche Teilnehmer sich im Netz befinden. Eine ausführliche Beschreibung des Aufbauprozesses eines Flooding-Netzwerks liefern Mahlmann und Schindelhauer (2007) auf den Seiten 58ff. Abbildung 3.12 zeigt einen Graphen, der eine exemplarische Aufbaustruktur abbildet.

Die Suche im Netzwerk erfolgt nach den Mechanismen einer beschränkten Breitensuche. Das Netzwerk wird mit Suchnachrichten geflutet (vgl. Yang

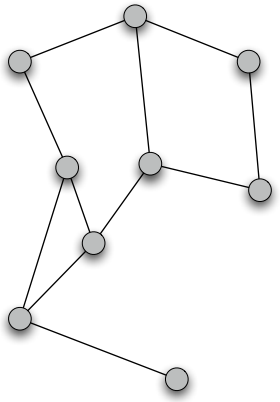
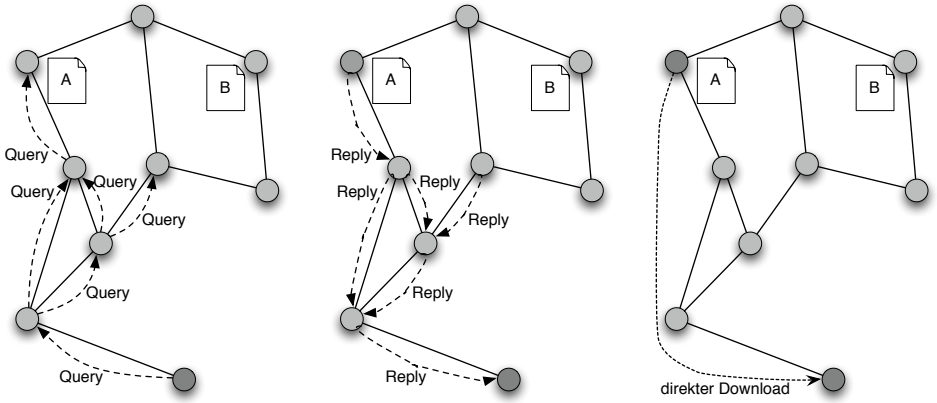


Abbildung 3.12: Exemplarische Aufbaustruktur eines Flooding-Netzwerks (nach Mahlmann und Schindelhauer, 2007, S. 59)

und Garcia-Molina (2002, S. 6) und Saake und Sattler (2004, S. 335)). In Abbildung 3.13 werden die Abläufe bei einer Suchanfrage verdeutlicht. Die Anfrage wird an alle direkt verbundenen Peers gesendet. Diese senden die Anfrage weiter bis die Limitierung der Suchtiefe erreicht ist (vgl. Abbildung 3.13(a)). Jeder Peer antwortet auf die Anfrage und leitet die Antworten anderer Peers weiter (vgl. Abbildung 3.13(b)). Erhält der suchende Peer eine positive Antwort, wird eine direkte Verbindung zwischen dem suchenden Peer und dem Peer, der den gesuchten Inhalt besitzt, aufgebaut und der Inhalt heruntergeladen (vgl. Abbildung 3.13(c)).

Die Suche per Flooding in Netzwerken (z. B. dem Gnutella-Netzwerk) erfolgt zuverlässig, wenn viele Peers den gesuchten Inhalt bereitstellen. Seltene Inhalte können aber nicht zuverlässig gefunden werden, da durch die tiefenbeschränkte Suche ein Inhalt nur gefunden wird, wenn er sich auf einem Knoten, der innerhalb der Tiefenbeschränkung erreicht wird, befindet. Der Inhalt B in Abbildung 3.13 kann vom suchenden Peer nicht gefunden werden, wenn die Suchtiefe kleiner oder gleich vier ist.

Flooding verursacht ein sehr hohes Kommunikationsaufkommen, da eine Nachricht nicht auf direktem Weg zum Ziel gelangt, sondern bis in eine vorgegebene Tiefe an alle Peers weitergegeben wird (vgl. Abbildung 3.13).



(a) Anfragen (b) Antworten (c) Download

Abbildung 3.13: Suche nach Inhalten im Gnutella-Netzwerk (nach Mahlmann und Schindelhauer, 2007, S. 61)

Dabei kann eine Nachricht einen Peer unnötigerweise mehrfach erreichen und wird dann mehrfach verarbeitet.

Suche über Document Routing

Im Document Routing werden alle Inhalte in einen gemeinsamen Schlüsselraum abgebildet. Dieser Schlüsselraum wird auf die vorhandenen Peers abgebildet, so dass jeder Peer für einen genau definierten Bereich des Schlüsselraums verantwortlich ist. Bei einer Anfrage für einen Inhalt wird die Anfrage immer an den Peer weitergegeben, dessen verwalteter Bereich im Schlüsslraum näher an der Position des gesuchten Inhalt liegt (vgl. Buchmann, 2006, S. 12).

Netzwerke, die Document Routing einsetzen, benötigen also eine strukturierte oder teilstrukturierte Datenhaltung, d. h. Inhalte werden nach festgelegten Regeln auf die Peers verteilt (vgl. Abbildung 3.14).

Zur Verdeutlichung des Document-Routing-Verfahrens wird hier das Content Addressable Network (CAN) vorgestellt (vgl. Ratnasamy et al., 2001, S. 1ff).

Merkmal	Merkmalsausprägung		
Netzwerk-topologie	zentralisiert	dezentralisiert	
		hybrid	vollständig dezentralisiert
Datenhaltung	strukturiert	teilstrukturiert	unstrukturiert
Suchmethode	zentrales Verzeichnis	Flooding	Document Routing

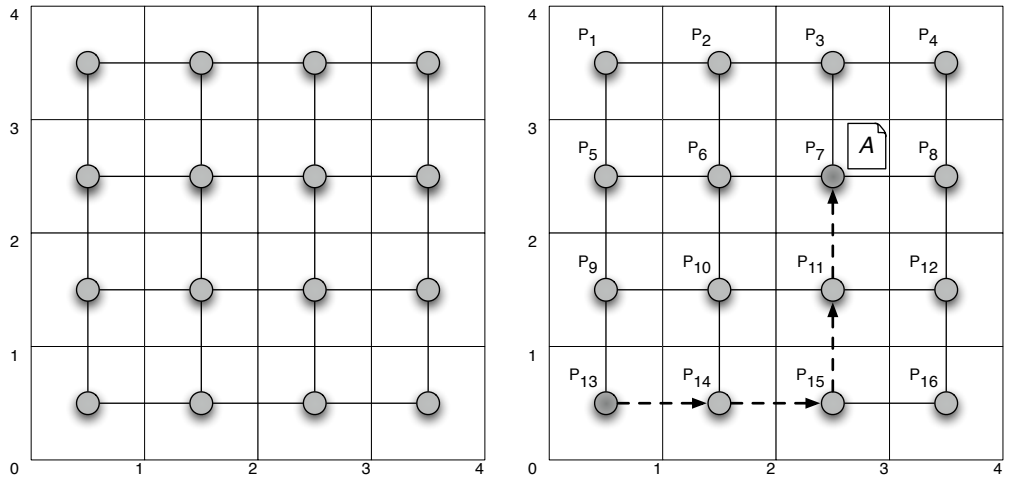
Abbildung 3.14: Merkmalsausprägungen im Morphologischen Kasten für das Content Addressable Network – dunkle Bereiche markieren Eigenschaften des Content-Addressable-Netzwerks

CAN basiert auf einem mehrdimensionalen Schlüsselraum S, der gleichmäßig partitioniert wird und dessen Partitionen jeweils einem Peer zugewiesen werden. In Abbildung 3.15(a) ist ein idealer Aufbau eines zweidimensionalen Schlüsselraums $S = [0, 4] \times [0, 4]$ dargestellt. Dabei stellt jedes Rechteck eine Partition des Schlüsselraums und jeder Kreis den verwaltenden Peer dar. Linien bilden Verbindungen zwischen den Peers ab.

Inhalte werden dabei für jede Dimension über eine eigene Hashfunktion $f : K \rightarrow S$ auf den Schlüsslraum abgebildet. K ist hierbei die Menge aller potentiellen Schlüssel für die Hashfunktion. Mögliche Schlüssel sind Dateinamen oder Meta-Daten wie Autor, Datum oder Titel (vgl. Mahlmann und Schindelhauer, 2007, S. 63).

Da zur Erstellung der Schlüssel Hashfunktionen verwendet werden und jeder Peer eine Tabelle der von seinen Nachbarn verwalteten Hashwerten besitzt, wird CAN zu den Peer-to-Peer-Netzwerken mit *Distributed Hash Tables* gezählt.

Als Beispiel sei in Abbildung 3.15(b) der Schlüsselraum aus Abbildung 3.15(a) und ein Dokument A angenommen. Durch zwei Hashfunktionen wurde das Dokument so auf den Schlüsslraum abgebildet, dass es von Peer P_7 verwaltet wird.



(a) Idealer Aufbau eines CAN (b) Suche im CAN

Abbildung 3.15: Content Adressable Network

Alle Inhalte einer Partition des Bildraums liegen auf dem zugehörigen Peer und werden von diesem verwaltet. Bei der Suche nach einem Inhalt bildet der suchende Peer den gesuchten Inhalt im Schlüsslraum ab und erfährt dadurch, welcher Peer den Inhalt verwaltet. Eine Suchanfrage nach Dokument A, ausgehend von Peer P_{13}, wird zum jeweils näherliegenden Peer weitergeleitet bis sie bei dem Peer angekommen ist, der den Inhalt verwaltet. Dieser Peer kann eine eindeutige Aussage über die Existenz des Inhalts im Netzwerk abgeben (vgl. Abbildung 3.15(b)).

Ist bei der Suche nicht jeder Schlüssel exakt bekannt, kann das Dokument nicht gefunden werden. Jede Abweichung vom Schlüssel führt zu einem anderen Hashwert und somit zu einer anderen Position im Schlüsselraum, an dem sich das Dokument nicht befindet. Daher ist in Peer-to-Peer-Netzwerken mit Distributed Hash Tables keine Ähnlichkeitssuche möglich.

Neben dem geometrischen Verfahren des n-dimensionalen Bildraums in CAN existieren ringbasierte Verfahren wie CHORD und PASTRY, welche die Peers in einem Ring anordnen (vgl. Stoica et al. (2001) und Rowstron und Druschel (2001)).

Peer-to-Peer-Netzwerke mit Document Routing, die Distributed Hash Tables nutzen, können das Auffinden eines sich im Netzwerk befindenden Inhalts im Gegensatz zu Peer-to-Peer-Netzwerken mit Flooding garantieren. Die errechnete Position im Schlüsselraum legt fest, welcher Peer für den Inhalt verantwortlich ist. Dieser Peer kann eine definitive Aussage treffen, ob der Inhalt im Netzwerk existiert oder nicht.

Im nachfolgen Abschnitt werden die Implikationen, die such aus den Vorangegangenen Abschnitten für die Konzeption der Data Source Layer dieser Arbeit ergeben zusammengefasst.

3.3.4 Implikationen für die Konzeption der Data Source Layer

Die Data Source Layer umfasst im in dieser Arbeit zu konzipierenden Konzept die Datenhaltung des Content Repository (vgl. Abschnitt 3.3.1). Eine Möglichkeit den Content-Bestand zu verteilen, wie es nach Krcmar (2005, S. 259) sinnvoll sein kann, bieten Peer-to-Peer-Netzwerke.

Peer-to-Peer-Netzwerke, die eine Content-Sharing-Funktion mit als Ressource bereitgestelltem Festplattenspeicher realisieren, können als Datenspeicher eines Content Repositories genutzt werden.

Das Auffinden des Content erfolgt in Peer-to-Peer-Netzwerken durch die *Suche über ein zentrales Verzeichnis*, *Flooding* oder *Document Routing* (vgl. Kapitel 3.3.3).

Flooding kann das Finden eines Inhalts im Netzwerk nicht garantieren. Daher sind Peer-to-Peer-Netzwerke, die Flooding zur Suche verwenden nicht als Datenspeicher für ein Content Repository geeignet. Ein zentrales Verzeichnis und Document Routing garantieren das Finden eines Inhalts. Peer-

to-Peer-Netzwerke, die diese Methoden zur Suche verwenden sind also zur Speicherung des Content eines Content Repository geeignet.

Um die in Kapitel 3.3.1 genannten Anforderungen an internetbasierte Anwendungen zu erfüllen und die Probleme einer serverbasierten Architektur zu vermeiden, muss ein Peer-to-Peer-Netzwerk mit vollständig dezentraler Netzwerktopologie verwendet werden. Dies schließt Netzwerke mit einem zentralen Verzeichnis aus. Daher wird in Kapitel 4 ein Peer-to-Peer-Netzwerk mit Document Routing als Suchmethode verwendet.

Ein Datenspeicher für ein Content Repository muss das sichere Speichern und Auffinden von gespeichertem Content ermöglichen. Dazu muss sichergestellt werden, dass der Content zu jeder Zeit verfügbar ist und sich in einem konsistenten Zustand befindet (vgl. Kapitel 2.4).

Wird zur Erhöhung der Verfügbarkeit und Steigerung der Performanz ein Peer-to-Peer-Netzwerk mit Replikation als Datenspeicher genutzt muss dazu insbesondere beim Beitritt eines Peers zum Netzwerk auf Konsistenz geachtet werden (vgl. *Performanz* in Kapitel 3.3.2).

Neben der Application Layer und der Data Source Layer besteht das Service Layer Pattern, dem das Konzept in dieser Arbeit folgt, aus der Service Layer. Im nachfolgenden Abschnitt wird die Service Layer und Webservices zur Umsetzung der Schicht vorgestellt.

3.4 Service Layer

Content Repositories sind geeignet, um von Enterprise-Content-Management-Systemen benötigte Funktionalität bereitzustellen (vgl. Kapitel 3.2). Dazu muss diese Funktionalität spezifiziert und in für die darauf zugreifenden Anwendungen nutzbar sein. Diese Definition der Systemgrenze des Content Repository geschieht in der Service Layer des Service Layer Pattern (vgl. Kapitel 2.4 und Abbildung 3.16).

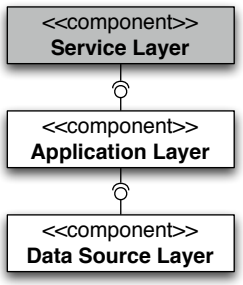


Abbildung 3.16: Service Layer Pattern - Service Layer

Die Anbindung an die ECM-Systeme muss *flexibel*, *offen*, *programmiersprachen-* und *plattformunabhängig* sein, um die Verwendung durch eine Vielzahl an Systemen zu erlauben (vgl. Kapitel 2.4):

Flexibel Content Repositories unterstützen eine inhaltszentrierte Anwendungslandschaft und damit ein modulares ECMS (vgl. Kapitel 2.3). Ziel eines modularen Systems ist die leichte Austauschbarkeit von Anwendungen. Damit dies möglich ist, darf keine Anwendung direkt auf die Funktionen des Content Repository zugreifen (lose Kopplung).

Offen Um eine Herstellerbindung (Vendor-Lock-In) bei der Wahl des Kommunikationsweges zwischen den Anwendungen und der Application Layer zu verhindern, muss die Kommunikation über offene Standards erfolgen. Offene Standards sind dadurch gekennzeichnet, dass sie von einer unabhängigen Organisation unter der Mitwirkung aller interessierten Personen, Firmen und Gesellschaften spezifiziert werden. Diese

Spezifikation ist frei verfügbar und unterliegt keinerlei Einschränkungen in ihrer Verwendung (vgl. IDABC, 2004, S. 9).

Plattformunabhängig Anwendungen laufen häufig auf unterschiedlichen Hardware- und Softwareplattformen (vgl. Kossmann und Leymann, 2004, S. 118). Um die Nutzung der Funktionalität des Content Repositories über Plattformgrenzen hinweg zu ermöglichen, muss die Anbindung an die Anwendungen einen plattformunabhängigen Zugriff bieten. Dadurch ist beispielsweise die Nutzung des Java Content Repository in einer J2EE-Umgebung durch eine .NET-Anwendung möglich.

Programmiersprachenunabhängig Mit der Plattformunabhängigkeit verknüpft ist die Programmiersprachenunabhängigkeit. Eine Plattform kann mehrere Programmiersprachen unterstützen. So lassen sich J2EE-Anwendungen in Java und Groovy (vgl. Judd et al., 2008, S.1ff) und .NET-Anwendungen u. a. in C#, VB.NET und Delphi.NET schreiben (vgl. Roden (2008, S. 5), Kettermann und Rohde (2005, S. 3f) und Vassbotn (2006, S. 221ff)). Eine übergreifende Anbindung muss also nicht nur plattformunabhängig sondern auch programmiersprachenunabhängig sein.

In der Service Layer werden die von der Application Layer bereitgestellten Funktionen und deren Rückgaben definiert und bereitgestellt. Eine Service Layer ist nach Fowler (vgl. 2002, S. 134) geeignet um eine Anwendung mit entfernten Funktionsaufrufen (engl. *remote function call*) zu konzipieren. So ist eine lose Kopplung zwischen Anwendungen und Content Repository erreichbar.

Zur Durchführung von entfernten Funktionsaufrufen existieren Konzepte wie RMI (Remote Method Invocation), DCOM (Distributed Component Object Model) und Webservices (vgl. Jäger, 2008, S. 205). Die beiden erstgenannten beiden Techniken sind nicht plattformunabhängig und erfüllen somit nicht die oben genannten Anforderungen (vgl. Melzer et al., 2007, S. 62). Daher

wird im Folgenden geprüft, ob Webservices diese Anforderungen erfüllen und sich somit zur Umsetzung der Service Layer eignen.

Im nächsten Abschnitt dieses Kapitels wird eine Definition des Begriffs Webservice gegeben. Danach werden Techniken, die im Webserviceumfeld eingesetzt werden, genannt und erläutert und schließlich die Vorgänge bei der Nutzung eines Webservice in Abschnitt 3.4.3 dargestellt.

3.4.1 Webservices

Der Begriff Webservice wird uneinheitlich verwendet und definiert. Auf einer sprachlichen Ebene wird unter einem Webservice ein Dienst verstanden, der über das Internet zugängig ist. Insbesondere für Dienste nach dem Self Service Pattern (vgl. Adams et al., 2002, S. 79)), bei denen eine direkte Interaktion zwischen einem Unternehmen und dem Kunden durch Internet-Technologien erfolgt, wird der Begriff Webservice verwendet (vgl. Moser et al., 2004, S. 4).

Da sich die Konzeption einer Service Layer auf technischer Ebene befindet, bedarf es für diese Arbeit einer technischeren Definition. So werden Webservices als *„wohldefinierte Funktionen, welche über standardisierte Protokolle zur entfernten Ausführung von Business-Funktionen oder Teilen davon in offenen Netzen angeboten werden“* beschrieben (vgl. Spahni und Meir, 2003, S. 187).

Diese abstrakte Definition wird von Austin et al. (2002) konkretisiert. In den *Web Services Architecture Requirements* des World Wide Web Consortium (W3C)[16] charakterisieren sie Webservices als eine Softwareanwendung, welche durch einen Uniform Resource Identifier (URI) identifizierbar ist, und dessen Schnittstellen als XML-Artefakte definiert und beschrieben werden können. Ein Webservice unterstützt die direkte Interaktion mit anderen Soft-

[16] http://www.w3c.org Stand: 16.03.2009

warekomponenten durch XML-basierte Nachrichten. Der Austausch dieser Nachrichten erfolgt über internetbasierte Protokolle.

Ein Webservice ist also eine durch einen eindeutigen Bezeichner adressierbare Softwareanwendung, die eindeutig beschriebene Funktionen anbietet und zu Maschine-zu-Maschine-Kommunikation fähig ist. Kommunikationsnachrichten und Funktionsbeschreibungen basieren dabei auf XML-Artefakten.

3.4.2 Webservicetechniken

Ein Webservice bedarf nach der Definition aus Abschnitt 3.4.1 einer Beschreibung und einer Möglichkeit, Nachrichten zu transportieren. Um einen Webservice nutzen zu können, muss es einem Konsumenten möglich sein den Webservice, der seine Anforderungen erfüllt, zu finden. Eine Softwarearchitektur für Webservices benötigt daher Komponenten für die *Kommunikation*, die *Dienstbeschreibung* und einen *Verzeichnisdienst* zum Auffinden von Webservices (vgl. Melzer et al., 2007, S. 51).

Nach Bettag (vgl. 2001, S. 304) bilden die Spezifikationen *SOAP*, *WSDL* und *UDDI* das Grundgerüst der Webservice-Architektur:

SOAP Mit der SOAP-Spezifikation wird ein XML-basiertes Nachrichtenformat und dessen Einbettung in ein Transportprotokoll beschrieben (vgl. Mitra und Lafon, 2007).

Eine SOAP-Nachricht besteht aus einem *SOAP Envelope*, einem *SOAP Header* und dem *SOAP Body*. Der SOAP Envelope ist das Wurzelelement der XML-Nachricht. Der Inhalt des SOAP Header ist nicht Bestandteil der SOAP-Spezifikation. Ein typischer Inhalt des Headers sind Authorisierungsinformationen des Absenders (vgl. Melzer et al., 2007, S. 73). Der SOAP Body enthält die eigentliche Nachricht. Diese muss selbst ein XML-Dokument sein (vgl. Kapitel 3.1.2).

Wird eine SOAP-Nachricht zur Ausführung eines entfernten Funktionsaufrufs (vgl. Abschnitt 3.4) genutzt, so muss der Aufbau des SOAP Header und SOAP Body einer speziellen Syntax folgen (vgl. Gudgin et al., 2007), damit der Anbieter der Funktion die Nachricht als Funktionsaufruf interpretiert.

Neben der SOAP-Nachricht beschreibt die SOAP-Spezifikation auch die Einbettung der Nachricht in Transportprotokolle. Dabei schreibt die Spezifikation kein Protokoll vor, so dass z. B. HTTP (Hypertext Transport Protocol), SMTP (Simple Mail Transfer Protocol) oder FTP (File Transfer Protocol) als Transportprotokoll möglich sind (vgl. Melzer et al., 2007, S. 85). HTTP bietet dabei den Vorteil, dass es auf fast jedem Rechner verfügbar ist und daher ohne Konfigurations- oder Installationsaufwand nutzbar ist (vgl. Quantz, 2003, S. 29).

Eine SOAP-basierte Kommunikation kann sowohl synchron, d. h. der Absender einer Nachricht wartet auf eine Antwort auf diese Nachricht, oder asynchron, d. h. der Absender einer Nachricht wartet nicht auf eine Antwort vor der Ausführung weiterer Arbeitsschritte, erfolgen (vgl. Gudgin et al., 2007, Abschnitt 6.).

WSDL Mit der *Web Service Definition Language* werden Webservices XML-basiert beschrieben (vgl. Chinnici et al., 2007). Die Beschreibung beantwortet drei Fragen (vgl. Kossmann und Leymann, 2004, S. 121):

Was? Welche Funktionen bietet der Webservice an? Welche Nachrichten versteht der Webservice und mit welchen Nachrichten antwortet er?

Wie? Welche Protokolle nutzt der Webservice zum Nachrichtenaustausch und wie werden Nachrichten kodiert?

Wo? Wie wird der Webservice genannt und unter welcher URI ist er zu erreichen?

Die WSDL-Beschreibung gliedert sich in einen abstrakten und einen konkreten Teil. Der abstrakte Teil beinhaltet die funktionale Beschreibung des Webservice und beantwortet die Fragen, welche Funktionen

der Webservice anbietet und welche Nachrichten er versteht und sendet. Der konkrete Teil beinhaltet Informationen über die Implementierung des Webservice. Damit wird die Frage der verwendeten Protokolle und der eindeutigen Bezeichnung des Webservice adressiert (vgl. Melzer et al., 2007, S. 102).

Durch diese Trennung kann der abstrakte Teil einer Webservice-Beschreibung von unterschiedlichen Webservices, die die gleiche Funktionalität anbieten, verwendet werden. Webservices mit gleicher abstrakter Beschreibung können gegeneinander ausgetauscht werden, da sie die gleiche Funktionalität anbieten.

Mit WSDL werden zustandsunabhängige Webservices beschrieben (vgl. Zhang et al., 2007, S 46f). Das bedeutet, dass ein Webservice kontextunabhängig ist und keine Kenntnis über zuvor erfolgte Aufrufe besitzt.

UDDI *Description, Discovery and Integration* stellt einen Verzeichnisdienst bei dem sich Webservices registrieren können. Dabei speichert das Verzeichnis die Webservicebeschreibungen. So kann UDDI genutzt werden, um durch eine Suchanfrage einen Webservice zu finden, der bestimmten gegebenen Anforderungen entspricht (vgl. Quantz, 2003, S. 35).

Inhalt des folgenden Abschnitts ist eine detailliertere Beschreibung der Abläufe beim Finden und Aufrufen eines Webservice.

3.4.3 Verwendung von Webservices

Bei der Verwendung von Webservices lassen sich nach Bettag (2001, S. 304) die drei Rollen *Konsument*, *Anbieter* und *Verzeichnis* identifizieren:

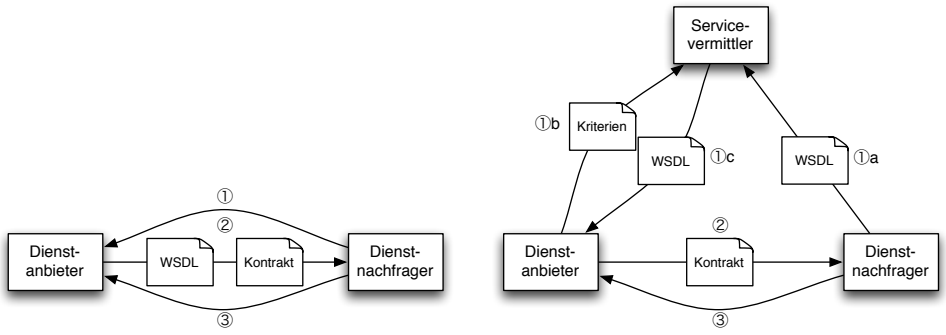
Konsument Der Konsument kommuniziert über Nachrichten mit dem Anbieter eines Webservice oder einem Webservice-Verzeichnis.

Anbieter Der Anbieter stellt den Webservice zur Verfügung und bietet die angebotene Funktionaität an.

Verzeichnis Das Verzeichnis enthält Beschreibungen von Webservices und deren Aufenthaltsort. Mit Hilfe des Verzeichnisses kann nach Webservices, die eine bestimmt Funktionalität erfüllen, gesucht werden.

Die Nutzung eines Webservice erfolgt nach Booth et al. (2004, Kapitel 1.4.5) in drei Schritten, dem *Finden des passenden Webservice*, dem *Schließen eines Abkommens* zwischen Konsument und Anbieter über die Nutzung und dem *Nutzen der Funktionen des Webservice*.

Für das Finden des passenden Webservice werden zwei Fälle unterschieden. Im ersten Fall sind sich Konsument und Anbieter eines Webservice bereits bekannt. In diesem Fall wird kein Verzeichnisdienst benötigt (vgl. Melzer et al., 2007, S. 51f). Ist dem Konsumenten kein entsprechender Anbieter bekannt, so erfolgt das Finden eines Anbieters über eine Suchanfrage an einen Verzeichnisdienst.



(a) ohne Verzeichnisdienst (b) mit Verzeichnisdienst

Abbildung 3.17: Aufruf eines Webservice
nach Booth et al. (2004, Kapitel 1.4.5, Kapitel 3.4)

Aufruf ohne Verzeichnisdienst (vgl. Abbildung 3.17(a)) Bei dem Aufruf eines Webservice sind sich Konsument und Anbieter bereits bekannt [1]. Konsument und Anbieter einigen sich auf eine Beschreibung

des Webservice und schließen ein Abkommen über dessen Nutzung [2]. Dieses Abkommen kann eine Nutzungserlaubnis seitens des Anbieters, aber auch ein Service Level Agreement mit detaillierten Absprachen über Qualität und Erreichbarkeit des Webservice sein. Danach erfolgt die Nutzung des Webservice durch den Konsumenten [3].

Aufruf mit Verzeichnisdienst (vgl. Abbildung 3.17(b)) Ist dem Konsumenten der Anbieter des passenden Webservice nicht bekannt, so kann er zur Suche einen Verzeichnisdienst nutzen. Dies setzt voraus, dass Anbieter ihre Webservices unter Angabe der jeweiligen Webservicebeschreibungen bei dem Verzeichnisdienst registrieren [1a] (vgl. Abschnitt 3.4.2). Für die Suche stellt der Konsument eine Anfrage mit den Kriterien des gesuchten Webservice an den Verzeichnisdienst [1b]. Kriterien können der Funktionsumfang sein, aber auch Qualitätsmerkmale wie Antwortzeiten oder Verfügbarkeit. Der Verzeichnisdienst liefert als Ergebnis die Webservicebeschreibungen der Webservices, welche den Kriterien entsprechen, an den Konsumenten zurück [1c]. Dieser wählt einen der Anbieter aus und schließt mit diesem ein Abkommen über die Nutzung des Webservice [2]. Danach erfolgt die Nutzung des Webservice durch den Konsumenten [3].

3.4.4 Implikationen für die Konzeption der Service Layer

Die Anforderung an eine Lösung zur Anbindung der Funktionalität des Content Repository an darauf zugreifende Anwendungen sollte nach Abschnitt 3.4 *flexibel*, *offen*, *plattform-* und *programmiersprachenunabhängig* sein. Webservices erreichen dies durch eine lose Kopplung und einen Zugriff auf Funktionen, der über Schnittstellen erfolgt. Ein Austausch der Implementierungen der Schnittstellen ist so möglich. Webservices sind durch die konsequente Nutzung XML-basierter Standards plattform- und programmiersprachenunabhängig (vgl. 3.1.2 und Großmann und Koschek (2005, S.

206)). Mit jeder Programmiersprache oder Plattform, für die eine Implementierung von SOAP verfügbar ist, können SOAP-Nachrichten gesendet und empfangen werden (vgl. Quantz, 2003, S. 28).

Die Nutzung eines Content Repository erfolgt über ein Session-Objekt (vgl. Kapitel 3.2.2). In ihm wird der aktuelle Zustand der Verbindung zum Content Repository gespeichert. Die Nutzung von Content Repositories ist also zustandsabhängig. Webservices sind zustandsunabhängig (vgl. Kapitel 3.4.2), daher muss das Konzept eine Möglichkeit bieten dennoch eine zustandsabhängige Verbindung zum Content Repository herzustellen.

Bei der Konzeption eines Webservice muss beachtet werden, dass die Nutzung von XML durch Webservices nach Tian et al. (2003, S. 6) einen nicht unbedeutenden Overhead bei der Größe der versendeten Nachrichten und der zur Verarbeitung benötigten Rechenleistung verursacht, so dass eine Reduzierung benötigter der Webservice-Aufrufe die Bandbreiten- und Rechenauslastung senkt.

Kapitel 4

Konzept eines Content Repository auf Basis eines P2P-Netzwerks und Webservices

Das in dieser Arbeit vorgestellte Konzept beschreibt ein verteiltes Informationssystem. Dabei befinden sich Teile des Informationssystems auf verschiedenen Hardwareeinheiten. Für das in dieser Arbeit vorgestellte Konzept wurden die drei Einheiten *Application*, *P2PContentRepository* und *PersistencePeer* definiert. Sie werden in Abbildung 4.1 in einem Verteilungsdiagramm als Knoten dargestellt und im Folgenden genauer beschrieben:

Abbildung 4.1: Die Verteilungsknoten des Konzepts

Knoten Application Auf dem Application-Knoten werden die Anwendungen ausgeführt, die auf das Content Repository zugreifen. Im Konzept ist dabei vorgesehen, dass null oder beliebig viele Application-Knoten mit einem Content Repository interagieren.

Knoten P2PContentRepository Auf dem P2PContentRepository-Knoten wird ein Content Repository ausgeführt, dass ein Peer-to-Peer-Netzwerk zur persistenten Speicherung verwendet. Dieser Knoten existiert genau einmal im konzipierten Informationssystem.

Knoten PersistencePeer Der Knoten P2PContentRepository ist mit null oder beliebig vielen PersistencePeer-Knoten verbunden. Auf jedem dieser Knoten läuft ein Peer des Peer-to-Peer-Netzwerks.

Nachdem die Knoten des Konzepts beschrieben wurden, werden im folgenden Abschnitt die Komponenten selbst vorgestellt. Die Anwendungen auf den Application-Knoten sind nicht Bestandteil dieser Arbeit, da sie über eine standardisierte Schnittstelle auf das hier konzipierte Informationssystem zugreifen. Daher beschränken sich die folgenden Ausführungen auf die Komponenten auf den Knoten P2PContentRepository und PersistencePeer.

Im Anschluss wird in Abschnitt 4.2 der Kommunikationsfluss zwischen den Komponenten bei der Benutzung des Content Repository dargestellt. Das Kapitel endet mit Abschnitt 4.3, in dem die Verteilung der einzelnen Softwareartefakte auf die in diesem Abschnitt beschriebenen Hardwareressourcen dargestellt ist.

4.1 Die Komponenten des Konzepts

Wie in Kapitel 2.4 beschrieben, wird zur Gliederung des Konzepts das Service Layer Pattern genutzt. Dabei wird das zu konzipierende Informationssystem in mehrere Schichten zerlegt, deren Funktionen durch Komponenten realisiert werden. Gemäß dem Service Layer Pattern sind dies die Schichten *Service Layer*, *Application Layer* und *Data Source Layer* (vgl. Kapitel 2.4).

Bei der Untersuchung der technischen Grundlagen konnte in Kapitel 3 gezeigt werden, dass zur Realisierung der Funktionalität der Service Layer ein Webservice, zur Realisierung der Funktionalität der Application Layer ein Content Repository und zur Realisierung der Funktionalität der Data Source Layer eine auf einem Peer-to-Peer-Netzwerk basierende Persistenzlösung geeignet ist.

Abbildung 4.2: Die Komponenten des Konzepts

In Abbildung 4.2 ist das Informationssystem durch drei Komponenten dargestellt, die jeweils die Funktionalität einer der identifizieren Schichten umfasst. Die Komponente Webservice bietet eine SOAP-basierte Schnittstelle, über die Anwendungen auf die Komponente zugreifen können. Sie nutzt die API des JSR 170 um auf die ContentRepository-Komponente zuzugreifen. Diese verwendet eine Schnittstelle Persistence zur Nutzung der Komponente P2PPersistence, welche in diesem Kapitel noch beschrieben wird.

In den nächsten Abschnitten werden die drei Komponenten detaillierter beschrieben und das abstrakte Komponentenmodell aus Abbildung 4.2 verfeinert.

4.1.1 Komponente ContentRepository

Die Komponente ContentRepository beinhaltet die Applikationslogik. Sie besteht in Anlehnung an Bernstein (vgl. Kapitel 3.2) aus einer Repository Engine, einem Repository Information Model, einer API zum Zugriff auf die Funktionalität der Komponente und einer Schnittstelle zur Anbindung einer Persistenzlösung. In Abbildung 4.3 ist der Aufbau der Komponente ContentRepository in einem Komponentendiagramm dargestellt.

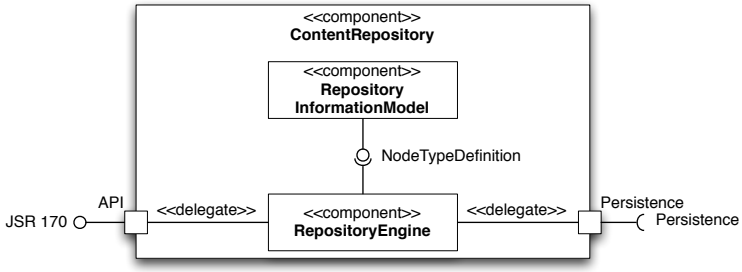
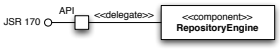


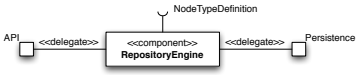
Abbildung 4.3: Komponentendiagramm: Content Repository

API (Port) 

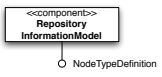
Die API dient zum Zugriff auf die Funktionalität der Komponente ContentRepository (vgl. Kapitel 3.2). Nach außen erscheint es, als würde die Funktionalität direkt von der Komponente ContentRepository angeboten. Daher ist die in Abbildung 4.3 als Port modelliert. Das in dieser Arbeit vorgestellte Konzept sieht ein Content Repository nach JSR 170 vor. Daher bietet die API den JSR 170 als Schnittstelle an. Die vom Port angebotene Schnittstelle wird intern durch die Komponente RepositoryEngine bereitgestellt.

Persistence (Port) 

Die Schnittstelle zur Persistenzlösung des Content Repository ist als Port modelliert, da nach außen nicht ersichtlich ist, welche der gekapselten Komponenten diese Schnittstelle verwendet. Die Komponente ContentRepository als Ganze wird als Verwender der Schnittstelle wahrgenommen (vgl. Abbildung 4.2). Intern wird die Schnittstelle von der Komponente RepositoryEngine verwendet. Die Schnittstelle zur Persistenzlösung ist nicht im JSR 170 Standard spezifiziert und daher abhängig von der jeweiligen Implementierung der Komponente RepositoryEngine.

RepositoryEngine (Komponente) 

Die gekapselte Komponente RepositoryEngine realisiert die Funktionalität der Komponente ContentRepository. Dazu nutzt es durch den Port Persistence die Schnittstelle zur Persistenzlösung des Content Repository und durch die Schnittstelle NodetypeDefinition die Nodetype-Definitionen der Komponente RepositoryInformationModel (vgl. 3.2.1). Der Zugriff auf die Komponente RepositoryEngine erfolgt über die bereitgestellten Funktionen des Ports API.

RepositoryInformationModel (Komponente) 

Die Komponente RepositoryInformationModel beinhaltet das Datenmodell des Content Repository. Die Nodetype-Definitionen werden über eine von der Implementierung der Komponente RepositoryEngine abhängige Schnittstelle bereitgestellt.

4.1.2 Komponente P2PPersistence

Die Komponente P2PPersistence stellt die Persistenzlösung des Konzepts bereit. Die dort zur Schaffung einer übersichtlicheren Darstellung in einer Komponente zusammengefasste Realisierung der Data Source Layer wird in diesem Abschnitt durch die drei Komponenten *PersistenceManager*, *P2PPersistenceManager* und *P2PNode* verfeinert (vgl. Abbildung 4.4).

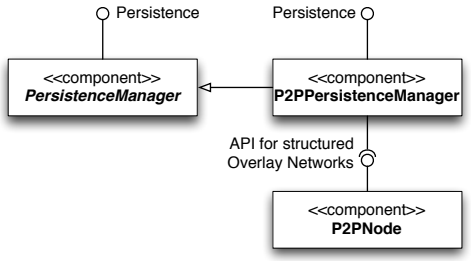


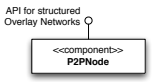
Abbildung 4.4: Komponentendiagramm: Peer-to-Peer-Persistenz

Im Folgenden werden die drei Komponenten erläutert. Dabei werden zuerst die Komponenten PersistenceManager und P2PNode sowie deren Schnittstellen betrachtet und im Anschluss die Komponente P2PPersistenceManager, die als Übersetzer zwischen diesen Schnittstellen dient.

PersistenceManager (abstrakte Komponente)

Die Persistenzlösung eines Content Repository muss dessen Persistenzschnittstelle realisieren, damit sie von diesem genutzt werden kann. Dies geschieht in der Komponente PersistenceManager. Um die Anbindung verschiedener Persistenzlösungen zu erlauben, ist diese Komponente abstrakt und muss für die jeweilige spezifische Persistenzlösung von einer weiteren Komponente konkretisiert werden (vgl. Kapitel 3.2.4).

P2PNode (Komponente)



Die Komponente P2PNode stellt einen Peer des Peer-to-Peer-Netzwerks, dass zur Speicherung des Content des Content Repository genutzt wird, dar. Daher ist bei der Betrachtung der Komponente zu klären, wie das Peer-to-Peer-Netzwerk des Konzepts aufgebaut ist und über welche Schnittstelle auf das Peer-to-Peer-Netzwerk zugegriffen werden kann.

Nach Abschnitt 3.3.4 muss ein Peer-to-Peer-Netzwerk als Persistenzlösung das *sichere Auffinden* des gespeicherten Content ermöglichen. Darüber hinaus muss sich der Content *jederzeit verfügbar* sein und sich in einem *konsistenten Zustand* befinden:

Sicheres Auffinden von Content Wie in Kapitel 3.3.4 gezeigt, bietet nur ein Peer-to-Peer-Netzwerk mit Document Routing ein sicheres Auffinden des gespeicherten Content mit einer vollständig dezentralen Netzwerktopologie. Daher wird für dieses Konzept ein Peer-to-Peer-Netzwerk, dass Document Routing als Suchmethode verwendet, genutzt.

Verfügbarkeit Die Ressourcen eines Peer-to-Peer-Netzwerks liegen auf Peers mit unsicherer Konnektivität (vgl. Kapitel 3.3.1). Das bedeutet, dass nicht davon ausgegangen werden kann, dass alle Peers des Netzwerks und damit die von ihnen verwalteten Ressourcen immer verfügbar sind. Für die Verwendung als Persistenzlösung eines Content Repository bedeutet dies, dass Content, der auf einem Peer liegt, der das Netzwerk verlässt, nicht erreichbar und somit nicht aufzufinden ist.

In diesem Konzept wird Replikation, also das Speichern einer festgelegten Anzahl von Kopien eines Content auf mehreren Peers, verwendet, so dass beim Ausfall eines Peers der Content von einem der anderen Peers abgerufen werden kann. Dabei ist insbesondere darauf zu achten, dass

trotz redundanter Datenhaltung ein konsistenter Zustand aller Kopien eingehalten wird (vgl. Kapitel 3.3.2).

Um die Replikate zu verwalten, wird ein weiterer Schlüsselraum mit anderen Hashfunktionen aufgebaut. Dadurch wird ein Replikat von einem anderen Peer verwaltet als das Original. Verlässt einer der Peers das Netzwerk ist der Content auf dem anderen Peer weiterhin verfügbar.

Konsistenz Bei der Verwendung von Replikation in einem Netzwerk mit unsicherer Konnektivität kann es zu einer inkonsistenten Datenhaltung kommen (vgl. Abbildung 4.5). In 4.5(a) wird ein Netzwerk aus drei Peers dargestellt. Peer P_1 und P_3 verwalten Kopien des Content A. In 4.5(b) hat P_3 das Netzwerk verlassen. P_2 ändert nun den Content A auf P_1 zu Content $A*$. Wenn sich danach in 4.5(c) P_3 mit seiner Kopie von Content A mit dem Netzwerk verbindet, befinden sich zwei unterschiedliche Versionen von Content A im Netzwerk. Die Daten im Netzwerk sind nicht mehr in einem konsistenten Zustand.

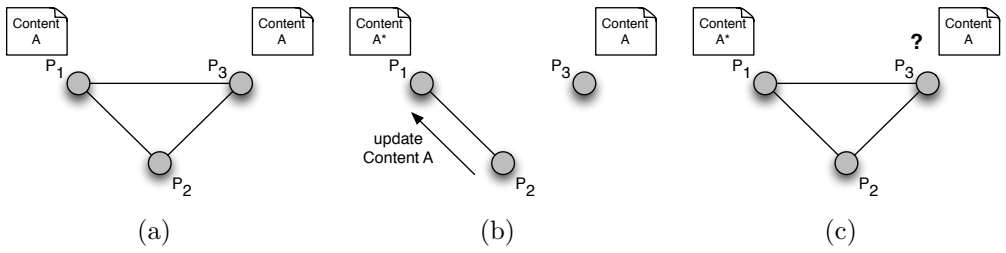


Abbildung 4.5: (a) Konsistente Datenhaltung, (b) Änderung eines Content A währen Abwesenheit von P_3 und (c) bei Rückkehr aller Peers inkonsistente Datenhaltung

Um dies zu verhindern sind zwei Ansätze möglich:

Supernode Ein Ansatz ist die Verwendung eines Peer-to-Peer-Netzwerks mit hybrider Netzwerktopologie (vgl. Kapitel 3.3.2). Supernodes speichern alle Änderungs- und Löschoperationen im Peer-to-Peer-Netzwerk. Wenn ein Peer sich mit dem Netzwerk verbindet, ruft er alle Änderungen seit seiner Abmeldung aus dem

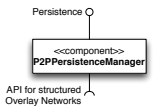
Netzwerk ab und führt diese auf seinem Datenbestand aus. So befinden sich seine Daten bei der Anmeldung im Netzwerk auf dem aktuellen Stand.

Löschen Ein weiterer Ansatz zur Verhinderung von Dateninkonsistenzen durch unsichere Konnektivität ist den gesamten von einem Peer verwalteten Content bei dessen Wiedereintritt in das Peer-to-Peer-Netzwerk zu löschen. Somit sind Inkonsistenzen beim Wiedereintritt eines Peers von vornherein ausgeschlossen. Da das Peer-to-Peer-Netzwerk Replikationen des Content enthält, ist trotz des Löschens von Content weiterhin der gesamte Content verfügbar.

Für das in dieser Arbeit vorgeschlagene Konzept wird der zweite Ansatz, das Löschen des Content vor dem Wiedereintritt in das Netzwerk, verwendet. Dieser Ansatz wird verfolgt, da ein vollständig dezentrales Peer-to-Peer-Netzwerk genutzt werden soll und daher die Verwendung von Supernodes nicht möglich ist (vgl. Kapitel 3.3.4).

Peer-to-Peer-Netzwerke mit Document Rouing gehören zu den strukturierten Overlay-Netzwerken (vgl. Kapitel 3.3.3). Es existiert der Versuch eine einheitliche API für den Zugriff auf strukturierte Overlay-Netzwerke zu schaffen (vgl. Dabek et al., 2003, S. 1). Eine solche API erlaubt eine Implementierung der Komponente P2PPersistenceManager, die unabhäng von der Realisierung des Peer-to-Peer-Netzwerks ist. Ein Wechsel auf eine andere, in einem relevanten Aspekt verbesserte, Implementierung ist somit leichter möglich. In diesem Konzept wird die in Dabek et al. (2003) vorgeschlagene Schnittstelle verwendet.

P2PPersistenceManager (Komponente)



Die Komponente P2PPersistenceManager übersetzt die von der Persistenzschnittstelle des Content Repository benötigten Operationen in die von der

Komponente P2PNode bereitgestellten Operationen. Darüber hinaus enthält die Komponente die Serialisierung der Content-Objekte des Content Repository, um diese persistent zu speichern. Die Komponente konkretisiert die abstrakte Komponente PersistenceManager, welche die Anbindung an das Content Repository beinhaltet, und realisiert so ebenfalls die Persistenzschnittstelle. Für den Zugriff auf die Komponente P2PNode wird die von der Komponente P2PNode angebotene Schnittstelle genutzt.

Um das Übersetzen der Operationen beider Schnittstellen zu spezifizieren, müssen deren Operationen bekannt sein. Da die Operationen der Persistenzschnittstelle implementierungsabhängig sind (vgl. Abschnitt 4.1.1) wird für diese Arbeit angenommen, dass für jedes der von Ehlers (2003, S. 116) identifizierten Rechte auf Content eine Operation existieren muss. Eine Betrachtung der Persistenzschnittstellen von Apache Jackrabbit und JBoss Portal bestätigen diese Vermutung[1]. Somit umfasst die Persistenzschnittstelle des Content Repository die Operationen *Erstellen*, *Lesen*, *Ändern* und *Löschen*. Die von der Komponente P2PNode verwendete Schnittstelle stellt die Operationen *Einfügen* (engl. *put*), *Entfernen* (engl. *remove*) und *Abrufen* (engl. *get*). Im Folgenden wird erläutert, wie die Übersetzung der Operationen der Persistenzschnittstelle erfolgt:

Erstellen Wird im Content Repository ein Content neu erstellt, so muss dieser persistent gespeichert werden. Im Peer-to-Peer-Netzwerk werden dazu entsprechende Hashwerte für den Content gebildet und dieser im Peer-to-Peer-Netzwerk abgelegt (vgl. Kapitel 3.3.3). Dies erfolgt über den Aufruf der *Einfügen*-Operation der von der Komponente P2PNode bereitgestellten API.

Lesen Um einen Content lesen zu können, muss dieser zuerst in der Persistenzlösung gefunden werden. Dazu werden für den zu findenden Con-

[1] *org.apache.jackrabbit.core.persistence.PersistenceManager* und *org.jboss.portal.portlet.state.producer.PortletStatePersistenceManager* im Verzeichnis *Resources* auf der beiliegenden CD

tent Hashwerte erstellt und dieser im Peer-to-Peer-Netzwerk gesucht (vgl. Kapitel 3.3.3). Dazu wird die API-Operation *Abrufen* verwendet.

Ändern Wird Content geändert, besteht die Möglichkeit, dass sich die Hashwerte des Content ändern. Dies geschieht dann, wenn Eigenschaften des Content, die zur Berechnung des Hashwertes verwendet werden, verändert wurden. Da dies eine Neupositionierung des Content im Schlüsselraum nach sich zieht, bietet die API für strukturierte Overlay-Netzwerke keine Operation zum Ändern von Content an. Bei der Änderung von Content, wird daher durch die API-Operationen *Entfernen* und *Einfügen* die alte Version des Content aus dem Peer-to-Peer-Netzwerk gelöscht und danach die neue Version eingefügt. Damit in der Zeit zwischen Löschen und neu Einfügen keine Zugriffe auf den Content erfolgen, wird dieser durhc ein Zwei-Phasen-Sperrprotokoll gesperrt und nach dem Einfügen wieder freigegeben (vgl. Saake et al., 2008, S. 385).

Löschen Soll Content aus dem Content Repository gelöscht werden, muss dieser auch aus der Persistenzlösung entfernt werden. Dazu wird der entsprechende Content gesucht und danach auf dem jeweiligen Peer gelöscht. Werden im Peer-to-Peer-Netzwerk Replikationen genutzt, so muss sichergestellt werden, dass jedes Replikat des Content gelöscht wird. Die Komponente P2PPersistenceManager verwendet dafür die Operation *Entfernen* der API.

4.1.3 Komponente Webservice

Die Service Layer wird durch die Komponente Webservice umgesetzt (vgl. Abschnitt 4.1). Dazu stellt sie eine Verbindung zur Komponente ContentRepository her, nutzt deren Funktionalität über die API des JSR 170 und bietet diese Anwendungen über eine SOAP-basierte Schnittstelle an, um so eine offene, flexible und programmiersprachenunabhängige Anbindung zu bieten (vgl. Abbildung 4.2).

In diesem Abschnitt wird eine detailliertere Darstellung der Umsetzung des Service Layer gegeben. Die Komponente Webservice aus Abschnitt 4.1 wird durch die beiden Komponenten *AbstractWebservice* und *SpecificWebservice* ersetzt (vgl. Abbildung 4.6). AbstractWebservice ist eine abstrakte Komponente, welche die grundlegende Anbindung an die Komponente ContentRepository herstellt. Sie wird durch die Komponente SpecificWebservice erweitert, welche Operationen für den Zugriff auf das Content Repository anbietet. Alle Webservice-Operationen erfolgen als synchroner Funktionsaufruf (vgl. Kapitel 3.4.2).

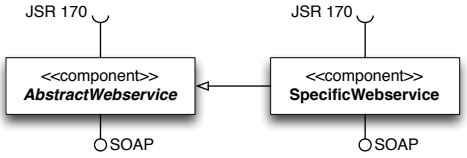
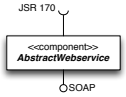


Abbildung 4.6: Komponentendiagramm: Webservice

Beide Komponenten werden im Folgenden näher betrachtet:

AbstractWebservice (Komponente) 

Die Komponente AbstractWebservice stellt die Anbindung an die Komponente ContentRepository zur Verfügung, so dass die Operationen der API des JSR 170 genutzt werden können. Dazu beinhaltet diese Komponente

auch Webservice-Operationen mit denen sich Benutzer von Anwendungen am Content Repository an- und abmelden können. Dabei ist bei einer Realisierung dieser Komponente zu beachten, dass die Nutzung eines Content Repository kontextabhängig ist, Webservices aber kontextunabhängig sind (vgl. Abschnitt 3.4.4).

Bei der Anmeldung am Content Repository wird ein Session-Objekt erzeugt (vgl. Kapitel 3.2.2), welches bei jedem Webserviceaufruf mit übertragen werden muss, damit der Webservice die Operationen des Content Repository nutzen kann. Zur Vereinfachung der Webservice-Nachrichtenübertragung und Senkung der Nachrichtengröße, wird in diesem Konzept ein *Ticket-System* verwendet. Die Session-Objekte von mit dem Content Repository verbundenen Benutzern werden im Webservice gespeichert und mit einem Ticket, repräsentiert durch eine Zeichenkette, versehen. Dadurch muss nicht das ganze Session-Objekt sondern nur eine Zeichenkette in den Webservice-Nachrichten übertragen werden.

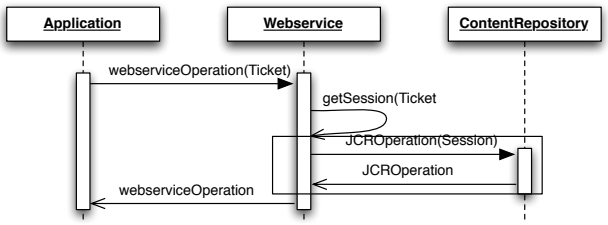
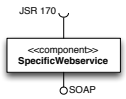


Abbildung 4.7: Sequenzdiagramm: Ticket-System

In Abbildung 4.7 ist der Aufruf einer Webservice-Operation mit dem Ticket-System dargestellt. Der Benutzer der Anwendung ruft eine Operation des Webservice auf und sendet das Ticket, dass er nach dem Einloggen am Content Repository erhalten hat, mit. Der Webservice sucht das dazugehörige Session-Objekt und führt die gewünschte Operation auf dem Content Repository aus und sendet den Rückgabewert an die Anwendung.

SpecificWebservice (Komponente)



Bei der Konzeption der Komponente Webservice wurde deutlich, dass eine direkte Abbildung der Operationen der JSR 170 API in vom Webservice angebotene Operationen zu einer hohen Anzahl an Webservice-Operationen und Webservice-Aufrufen und so zu einem relevanten Anstieg der Menge und Größe an zu übertragenden Nachrichten führt (vgl. Kapitel 3.4.4).

Als Beleg dafür sei als Anwendungsfall das Abrufen eines Content aus dem Content Repository angenommen. Um einen Content zu erhalten, muss eine Traversierung des Content-Baums erfolgen um an den Knoten des Content zu gelangen (vgl. Kapitel 3.2.1). Dazu bedarf es mindestens einer Operation. Da die Daten eines Content in den Eigenschaften eines Knotens gespeichert sind, muss eine Liste aller Eigenschaften des Knotens abgerufen und im Anschluss mit je einer Operation die Werte der Eigenschaften ausgelesen werden. Zum Abruf des Content f in Abbildung 3.4 wären so vier Aufrufe des Webservice nötig. Bei Content-Typen, die mehr Eigenschaften vorschreiben, erhöht sich die Anzahl der Aufrufe entsprechend.

Daher ist in diesem Konzept vorgesehen, dass Webservice-Operationen eine Reihe von Content-Repository-Operationen umfassen. Eine Webservice-Operation um einen Content abzurufen wäre daher z. B. *getContent(Identifier)*, welche die oben genannten Schritte ausführt. Diese Operation ist in Abschnitt 4.2.2 näher beschrieben.

Da nicht alle Anwendungen die gleichen Operationen des Content Repository nutzen, bietet das Konzept die Möglichkeit die Komponente AbstractWebservice durch eine konkrete Komponente SpecificWebservice zu erweitern, welche die gewünschten Operationen bereitstellt.

Zusammenfassend sind alle definierten Komponenten in Abbildung 4.8 dargestellt.

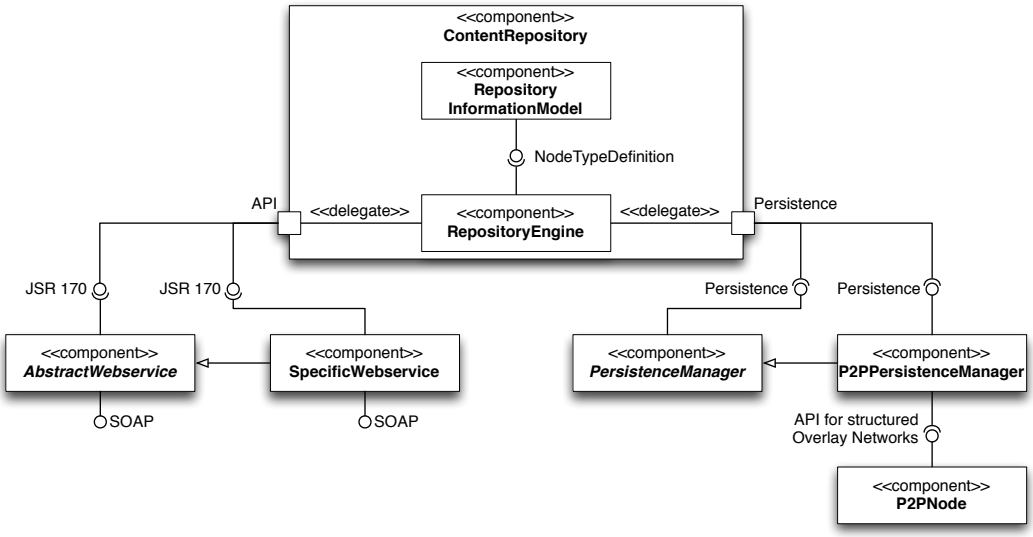


Abbildung 4.8: Komponentendiagramm: Überblick über alle Komponenten des Konzepts

Nachfolgend wird in Abschnitt 4.2 die Interaktion zwischen den Komponenten beschriebe.

4.2 Interaktion der Komponenten

Die Modellierung eines Informationssystems umfasst nicht nur die Abbildung der Struktur, sondern auch die Wiedergabe der Interaktionen der Struktureinheiten (vgl. Kapitel 3.1.1). Daher werden in diesem Abschnitt die zwischen den Komponenten des Konzepts ausgetauschten Nachrichten und ihre zeitliche Abfolge erläutert. Dies geschieht anhand von drei exemplarisch ausgesuchten Anwendungsfällen, dem *Speichern eines neuen Content*, dem *Abrufen eines bekannten Content* und der *Suche nach einem nicht exakt bekannten Content*. Die Abläufe der einzelnen Anwendungsfälle werden durch UML-Sequenzdiagramme abgebildet. Für eine Erläuterung der Diagrammsyntax sei auf Kapitel 3.1.1 verwiesen.

4.2.1 Speichern von Content im Content Repository

In Abbildung 4.9 sind die Abläufe bei der Speicherung eines neuen Content abgebildet.

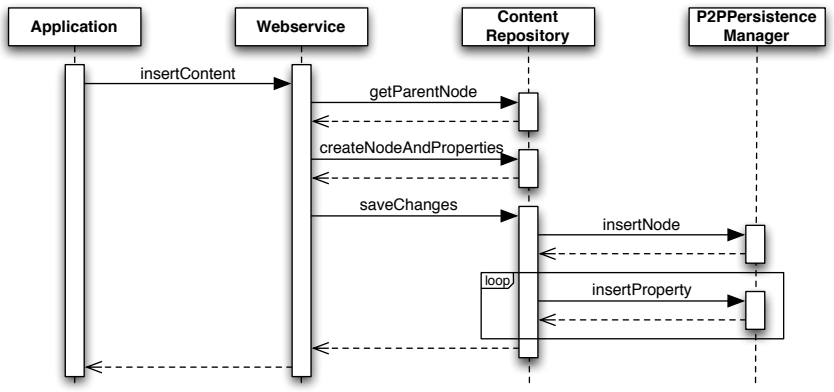


Abbildung 4.9: Sequenzdiagramm: Speichern von Content

Der Vorgang des Speicherns geht von einer Anwendung aus. Sie ruft die Webservice-Operation *insertContent* auf und übermittelt dabei alle zur Erstellung des Content benötigten Attribute. Daraufhin ruft der Webservice den

Knoten, unter dem der neue Content eingefügt werden soll, aus dem Content Repository ab (vgl. Kapitel 3.2.1 für Informationen über die Ablage von Content im Content Repository). Im Anschluss daran erstellt der Webservice einen neuen Knoten sowie dessen Eigenschaften um den zu speichernden Content im Content Repository abzulegen. Danach werden die Änderungen im Content Repository gespeichert. Dazu werden sowohl der Knoten, als auch jede Eigenschaft, als einzelne Objekte durch den P2PPersistenceManager im Peer-to-Peer-Netzwerk abgelegt. Die Nachricht, dass der Content gespeichert wurde, wird durch den Webservice an die Anwendung weitergegeben.

4.2.2 Aufrufen von Content aus dem Content Repository

In Abbildung 4.10 sind die Abläufe beim Abrufen eines Content abgebildet.

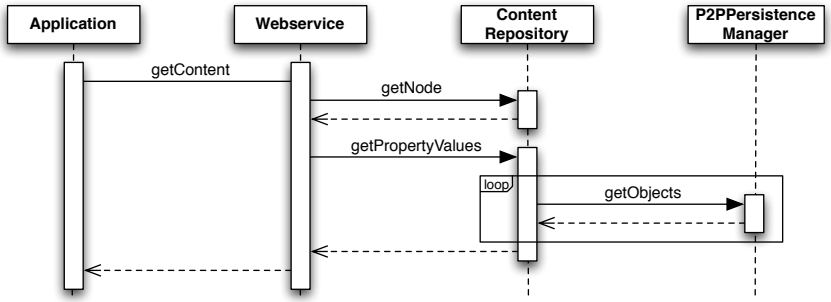


Abbildung 4.10: Sequenzdiagramm: Abrufen von Content

Wie der Vorgang des Speicherns wird auch das Abrufen von Content durch eine Anwendung ausgelöst. Nach dem Aufruf der *getContent* Operation des Webservice mit Attributen, die einen Content eindeutig identifizieren, ruft der Webservice den dazugehörigen Knoten aus dem Content Repository ab. Wie in Kapitel 3.2.1 beschrieben, sind die Daten eines Knotens in dessen Eigenschaften gespeichert. Um den Content zurückliefern zu können, muss der Webservice daher die Eigenschaften des Knotens abrufen. Diese ruft das Content Repository über den P2PPersistenceManager aus dem Peer-to-Peer-

Netzwerk ab. Den abgerufenen Content sendet der Webservice abschließend an die Anwendung.

4.2.3 Suchen nach Content im Content Repository

In Abbildung 4.11 sind die Abläufe bei der Suche nach einem Content abgebildet.

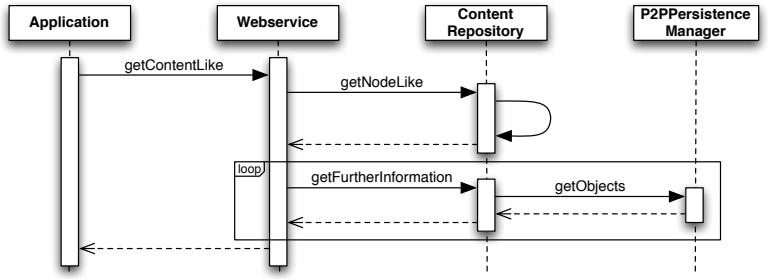


Abbildung 4.11: Sequenzdiagramm: Suche nach Content

Ist dem Benutzer einer Anwendung ein Content nicht exakt bekannt, kann er durch die Webservice-Operation *getContentLike* mit einigen Hinweisen auf den gewünschten Content nach passendem Content suchen. Dazu fragt der Webservice auf die Hinweise passende Knoten des Content Repository ab. Da in den in diesem Konzept verwendeten Peer-to-Peer-Netzwerken mit Document Routing die Suche nach Content, dessen Schlüssel nicht bekannt sind, nicht möglich ist (vgl. Kapitel 3.3.3), wird zur Suche nach dem passenden Content der eigene, nicht verteilte Index des Content Repository verwendet (vgl. Kapitel 3.2.2). Um dem Benutzer der Anwendung die Wahl des passenden Content zu ermöglichen, werden Informationen, beispielsweise Titel und Autor, zu den einzelnen Knoten durch den P2PPersistenceManager aus dem Peer-to-Peer-Netzwerk abgerufen und an die Anwendung gesendet. Daraufhin kann der Benutzer der Anwendung den für seine Anfrage passendsten Content abrufen (vgl. Abschnitt 4.2.2).

4.3 Verteilung der Artefakte des Konzepts

Eine Eigenschaft des in dieser Arbeit vorgestellten Konzepts ist, dass das konzipierte Informationssystem auf mehr als einem Knoten läuft. Daher ist es notwendig die Verteilung der einzelnen Komponenten des Konzepts auf die in Abschnitt 4 identifizierten Knoten zu modellieren.

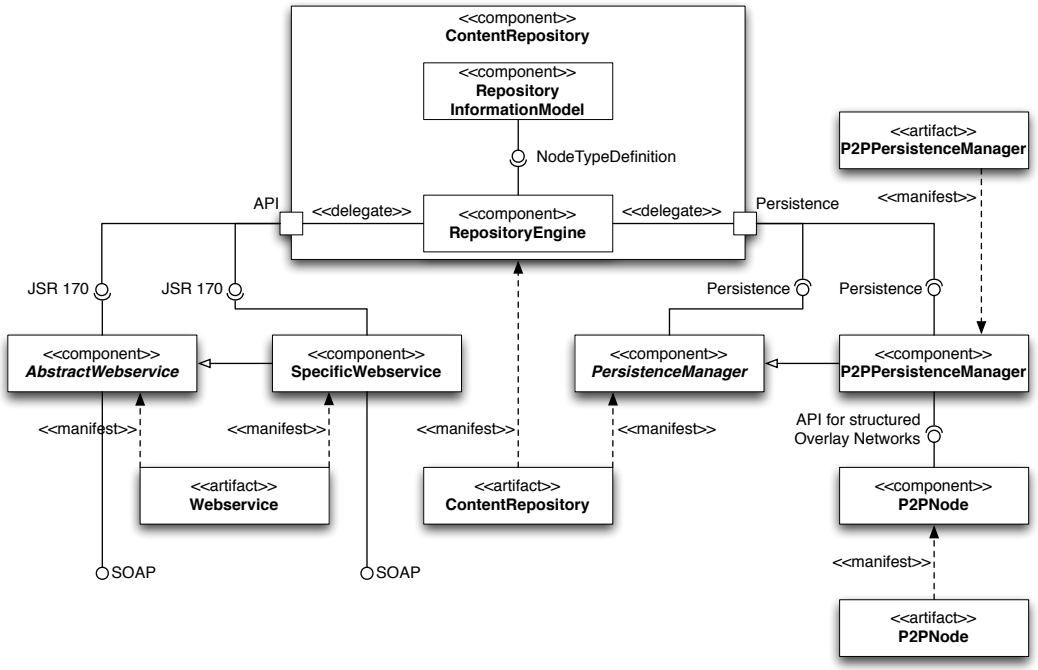


Abbildung 4.12: Komponentendiagramm: Zuordnung der Komponenten zu Artefakten

In einem Verteilungsdiagramm wird dargestellt, welche Knoten ein Softwaresystem besitzt und wie die Artefakte einer Softwareanwendung auf diesen Knoten verteilt sind (vgl. Rupp et al., 2007, S. 225). Dazu ist eine Zuordnung von Komponenten zu Artefakten notwendig. Die in diesem Konzept vorgesehenen Artefakte sind *Webservice*, *ContentRepository*, *P2PPersistenceManager* und *P2PNode*. Die Zuordnung der Komponenten aus Abschnitt 4.1 ist in Abbildung 4.12 dargestellt und wird im Folgenden erläutert:

Artefakt Webservice Das Artefakt Webservice beinhaltet Realisationen der Komponente AbstractWebservice und mindestens eine erweiternde Komponente SpecificWebservice.

Artefakt ContentRepository Das Artefakt ContentRepository beinhaltet die Komponente ContentRepository mit all ihren Subkomponenten sowie die abstrakte Komponente PersistenceManager. Diese Komponenten sind in einem Artefakt zusammengefasst, da die Komponente PersistenceManager abhängig von der Realisierung der Komponente ContentRepository ist.

Artefakt P2PPersistenceManager Dieses Artefakt beinhaltet die Komponente P2PPersistenceManager.

Artefakt P2PNode Das Artefakt P2PNode beinhaltet die Komponente P2PNode, die benötigt wird um einen Peer des Peer-to-Peer-Netzwerks zu betreiben.

In Abschnitt 4 wurden bereits die Knoten des Informationsystems beschrieben. Es besteht aus den Knoten *Application*, *P2PContentRepository* und *PersistencePeer*. In Abbildung 4.13 ist ein Verteilungsdiagramm dargestellt, das die Verteilung der eben beschriebenen Artefakte auf die Knoten des Informationssystems zeigt:

Knoten Application Auf dem Anwendungsknoten befindet sich die (ECM-) Anwendung, welche das Content Repository nutzt. Das sich darauf befindende Artefakt Application befindet sich außerhalb des für diese Arbeit gesteckten Rahmens und wurde daher nicht näher beschrieben.

Knoten P2PContentRepository Auf dem Knoten P2PContentRepository wird das Content Repository betrieben. Dazu beinhaltet der Knoten das Artefakt ContentRepository sowie das Artefakt Webservice. Zur persistenten Speicherung befindet sich das Artefakt P2PPersistence Manager und das Artefakt P2PNode ebenfalls auf dem Knoten.

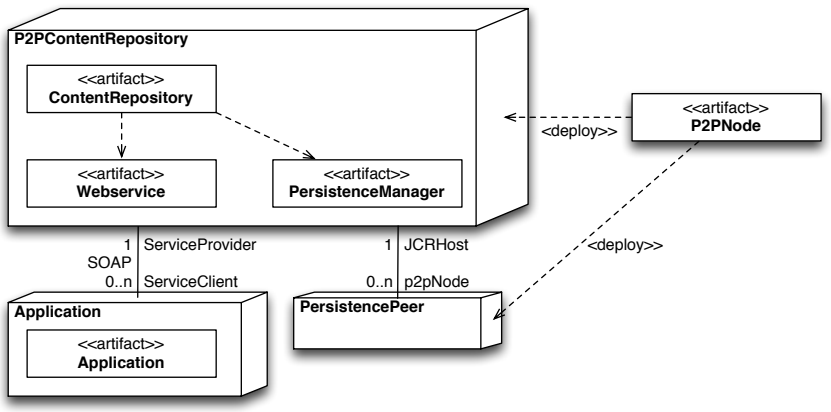


Abbildung 4.13: Verteilungsdiagramm der Knoten und Artefakte des Konzepts

Knoten PersistencePeer Jeder PersistencePeer-Knoten beinhaltet ebenfalls das Artefakt P2PNode um einen Peer des Peer-to-Peer-Netzwerks bilden zu können.

Um das in diesem Kapitel vorgestellte Konzept eines Informationssystems in einem lauffähigen Softwaresystem umzusetzen, werden die Artefakte durch Softwareprodukte oder Eigenentwicklungen realisiert werden und entsprechend dem Verteilungsdiagramm in Abbildung 4.13 verteilt werden. Dies wird im folgenden Kapitel anhand einer prototypischen Implementierung verdeutlicht.

Kapitel 5

Evaluierung des Konzepts

Im Rahmen dieser Arbeit wurde die technische Umsetzbarkeit des in Kapitel 4 vorgestellten Konzepts durch eine prototypische Implementierung evaluiert. Dazu wurde für jedes der in Kapitel 4.3 genannten Artefakte ein die Funktionalität der im Artefakt enthaltenen Komponenten realisierendes Softwareprodukt verwendet oder eine eigene Komponente implementiert.

Die folgenden Abschnitte erläutern je eines der Artefakte.

5.1 Artefakt ContentRepository

Dieses Artefakt beinhaltet die Komponenten ContentRepository und PersistenceManager. Das Konzept spezifiziert die API des JSR 170 als Schnittstelle zum Zugriff auf das Content Repository. Daher wurde als Realisierung der Komponente RepositoryEngine das Softwareprodukt Apache Jackrabbit, die Referenzimplemenatation des JSR 170 (vgl. Kapitel 3.2.3), verwendet.

Weiterer Bestandteil einer Komponente ContentRepository ist die Komponente RepositoryInformationModel (vgl. Kapitel 3.2). Für ein solches Modell müssen *Content-Typen*, *Ablagestruktur*, *Metdadatenattribute* und *Zugriffsrechte* eines Content Repository modelliert werden (vgl. 3.2.1). Im Folgenden werden dies für den Prototypen entwickelten Elemente des Repository Infor-

mation Model vorgestellt. Auf die Zugriffsrechte wurde dabei verzichtet, da der Prototyp für einen einzelnen Benutzer konzipiert wurde.

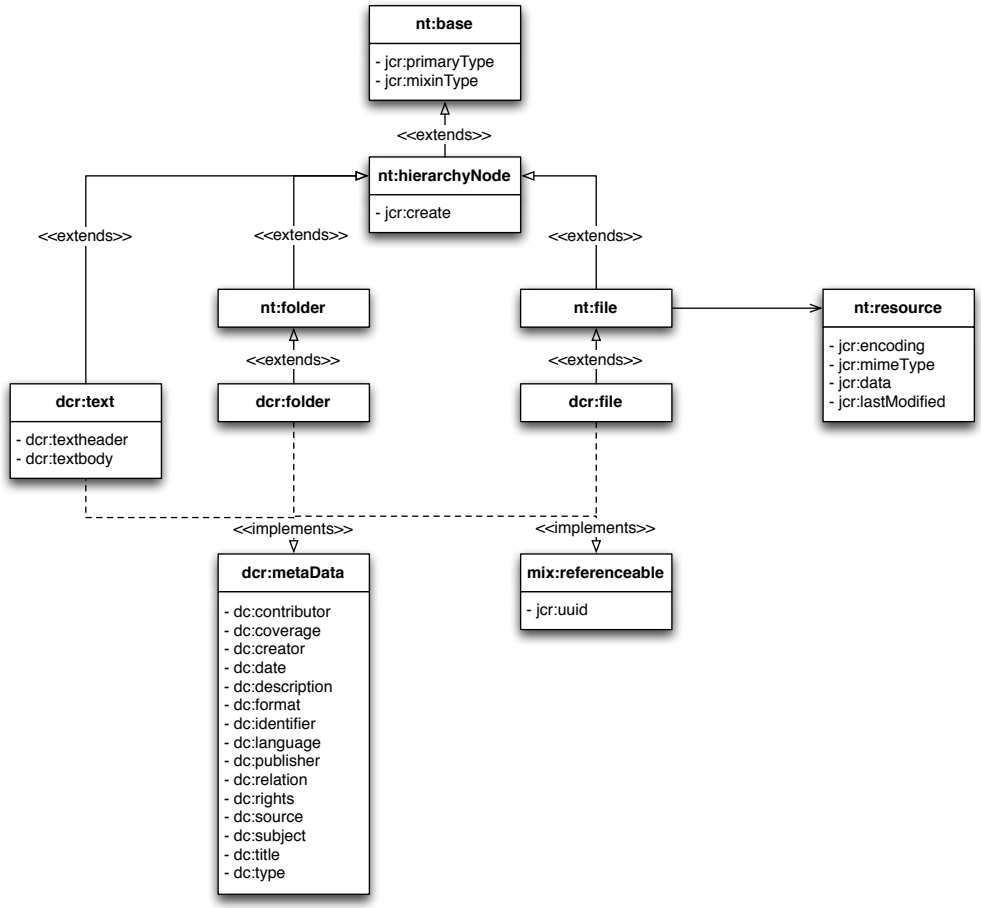


Abbildung 5.1: Klassendiagramm der Content-Typen des Prototypen

Content-Typen und Ablagestruktur Für den Prototypen wurden die Content-Typen *dcr:folder*, *dcr:file* und *dcr:text* definiert. Diese drei Content-Typen erben Eigenschaften von bereits im Apache Jackrabbit definierten Content-Typen (vgl. Abbildung 5.1). Für eine detailierte Beschreibung dieser Content-Typen sei auf Nüscheler und Piegaze (2005, S. 141ff) verwiesen. Die folgende Auflistung umfasst nur die für den Prototypen definierten Content-Typen:

dcr:folder Dieser Content-Typ repräsentiert ein Verzeichnis in einem Verzeichnisbaum. Unter einem Knoten dieses Content-Typen dürfen sich Knoten des Typs nt:hierarchyNode, also auch die Typen dcr:folder, dcr:file und dcr:text befinden.

dcr:file Dieser Content-Typ repräsentiert eine im Content Repository abgelegte Datei. Ein Knoten dieses Typs kann keine weiteren Kindknoten besitzen.

dcr:text Mit diesem Content-Typ Repräsentiert einen Content, der durch eine Überschrift und Text strukturiert ist. Wie der Content-Typ dcr:file kann auch dieser Content-Typ keine weiteren Kindknoten besitzen. Knoten dieses Content-Typs besitzen die Eigenschaften dcr:textheader, in dem die Überschrift des Texts gespeichert wird, und dcr:textbody, in dem der eigentliche Text gespeichert wird.

Metadatenattribute Für alle drei Content-Typen wurden die im Dublin Core Metadata Element Set vorgeschlagenen Metadatenattribute als Eigenschaften definiert (vgl. Kapitel 3.2.1 und Abbildung 5.1).

Die Komponente PersistenceManager wird durch die *org.apache.jackrabbit.core.persistence.PersistenceManager*-Schnittstelle des Apache Jackrabbit bereitgestellt. Sie ist auf der beiliegenden CD in der Datei *Resources/SRC_Jackrabbit-1.4.zip* enthalten.

5.2 Artefakt P2PNode

Nach Kapitel 4.1.2 sieht das Konzept ein Peer-to-Peer-Netzwerk mit Document Routing vor. Dazu sind CAN, CHORD oder PASTRY Peer-to-Peer-Netzwerke verwendbar (vgl. Kapitel 3.3.3).

Da eine eigene Implementierung eines Peer-to-Peer-Netzwerks im Bearbeitungszeitraum dieser Arbeit nicht realisierbar war, wurde für den Protoptypen eine existierende Implementierung eines Peer-to-Peer-Netzwerks an die Erfordernisse des Konzepts angepasst. Zum Zeitpunkt der Erstellung des Prototypen war PAST[1], eine Peer-to-Peer-Anwendung zur Speicherung von Daten in einem PASTRY-Netzwerk der Rice University, Houston, Texas, die einzige frei verfügbare, java-basierte und dokumentierte Implementierung eines Peer-to-Peer-Netzwerks mit Document Routing und wurde daher hier verwendet.

Die im Konzept vorgesehene Replikation von Content zur Erreichung einer durchgehenden Verfügbarkeit des Content ist in PAST bereits implementiert. Eine durch die Persistenzschnittstelle geforderte Möglichkeit Content zu entfernen ist in PAST nicht enthalten. Um dies zu ermöglichen wurde dem Peer-to-Peer-Netzwerk eine solche Operation hinzugefügt. Das Löschen des Content eines Peers bei dessen Eintreten in das Netzwerk zur Sicherstellung einer konsistenten Datenhaltung wurde ebenso implementiert.

Die hinzugefügten Bestandteile des Peer-to-Peer-Netzwerks befinden sich im Paket *de.vlbalab.dcr.persistence* im Verzeichnis *Source/DCR/src* auf der beigefügten CD.

5.3 Artefakt P2PPersistenceManager

Das Artefakt P2PPersistenceManager beinhaltet die Komponente P2PPersistenceManager. Sie dient als Übersetzer zwischen der Persistenzschnittstelle des Content Repository und der Schnittstelle des Peer-to-Peer-Netzwerks sowie zur Serialisierung der Content-Objekte des Content Repository (vgl. Kapitel 4.1.2).

[1] http://freepastry.org/PAST Stand: 16.03.2009

Durch die Verwendung des Apache Jackrabbit als Repository Engine und eines von PAST abgeleiteten Peer-to-Peer-Netzwerk (vgl. Kapitel 5.1 und 5.2) sind die beiden Schnittstellen, zwischen denen übersetzt wird, festgelegt.

Die Persistenzschnittstelle wird in *org.apache.jackrabbit.core.persistence.-PersistenceManager* definiert. Entgegen der im Konzept vorgesehenen Schnittstelle für strukturierte Overlay-Netzwerke verwendet PAST eine semantisch äquivalente aber syntaktisch abweichende in *rice.p2p.past.PAST* definierte Schnittstelle. Diese wurde, wie in Abschnitt 5.2 genannt, durch eine Operation zum Entfernen von Content erweitert. In Abbildung 5.2 ist dargestellt, wie die Operationen des Peer-to-Peer-Netzwerks genutzt werden um die von der Persistenzschnittstelle geforderten Operationen zu realisieren.

Operation der Persistenzschnittstelle	Beschreibung der Operation	Operation der Schnittstelle des P2P-Netzwerks
load( )	Lädt Content aus dem Datenspeicher	lookup( )
store( )	Speichert sowohl neuen Content als auch Änderungen an bestehende Content	insert( ) remove( ) + insert( )
remove( )	Löscht Content aus dem Datenspeicher	remove( )

Abbildung 5.2: Abbildung der Operationen der Persistenzschnittstelle auf die Operationen des Peer-to-Peer-Netzwerks

Zur Übergabe an die Schnittstelle des Peer-to-Peer-Netzwerk werden die Content-Objekte serialisiert und in einem XML-Dokument repräsentiert. Dieses wird im Peer-to-Peer-Netzwerk gespeichert.

Der implementierte P2P Persistence Manager befindet sich im Paket *de.vlbalab.dcr.persistence.manager* im Quelltextverzeichnis *Source/DCR/src* auf der beigefügten CD.

5.4 Artefakt Webservice

Für den Prototypen wurde ein Webservice entwickelt, der die Funktionalität der Komponente AbstractWebservice umfasst. Darüber hinaus verfügt er über prototypspezifische Webservice-Operationen für den Zugriff auf das Content Repository. Der Nachrichtenaustausch mit dem Webservice erfolgt über SOAP-Nachrichten, die HTTP als Transportprotokoll nutzen.

Die verfügbaren Operationen des Webservice sind *login()* und *logout()* um die Verbindung zum Content Repository herzustellen und verwendet dazu das in Kapitel 4.1.3 vorgestellte Ticket-System. Als prototypspezifische Operationen werden die Operationen *insertNewTextContent()*, *listTextContent()*, *getTextContent()*, *insertNewFileContent*, *listFileContent* und *getFileContent* bereitgestellt:

insertNewTextContent() Diese Operation fügt dem Content Repsitory einen neuen Content des Content-Typs *dcr:text* hinzu (vgl. Abschnitt 5.1). Dazu werden die benötigten Werte für die Eigenschaften mitgesendet. Zur Speicherung werden die Content-Repository-Operationen für die folgenden Schritte in einer Webservice-Operation zusammengefasst (vgl. Kapitel 4.1.3):

1. Finden des Knotens unterhalb dessen der neue Content-Knoten erstellt wird
2. Erstellen des *dcr:text*-Knotens
3. Erstellen der benötigten Eigenschaften
4. Speichern der Änderungen

listTextContent() Mit dieser Operation wird eine Liste des Content mit dem Content-Typ *dcr:text* zurückgegeben. Dazu werden die Content-Repository-Operationen für die folgenden Schritte in einer Webservice-Operation zusammengefasst (vgl. Kapitel 4.1.3):

1. Durchsuchen des gesamten Content Repository nach allen Knoten des Typs *dcr:text*
2. Speichern der Identifier dieser Knoten und zurücksenden dieser Liste an den Konsumenten des Webservice

getTextContent() Durch diese Operation kann ein Content des Typs *dcr:text* aus dem Content Repository angerufen werden. Dazu werden die Content-Repository-Operationen für die folgenden Schritte in einer Webservice-Operation zusammengefasst (vgl. Kapitel 4.1.3):

1. Das Content Repository wird nach dem aufzurufenden Knoten durchsucht
2. Abrufen des Knotens und all seiner Eigenschaften
3. Senden der Daten an den Konsumenten des Webservice

Für die Operationen *insertNewFileContent() listFileContent()* und *getFileContent()* erfolgen die Aufrufe analog, nur mit Verwendung des Content-Typs *dcr:file*.

5.5 Artefakt Anwendung

Um den Prototypen zu testen, wurde eine webbasierte Client-Anwendung implementiert, welche die in Abschnitt 5.4 aufgeführten Operationen des Webservice nutzt und die Ergebnisse anzeigt. In Abbildung 5.3 ist ein Bildschirmfoto der Client-Anwendung abgebildet.

Darauf wird dargestellt, dass die Client-Anwendung die Möglichkeit bietet Listen des Content anzuzeigen, einen einzelnen Content anzuzeigen sowie Content der beiden Content-Typen des Prototypen zu erstellen.

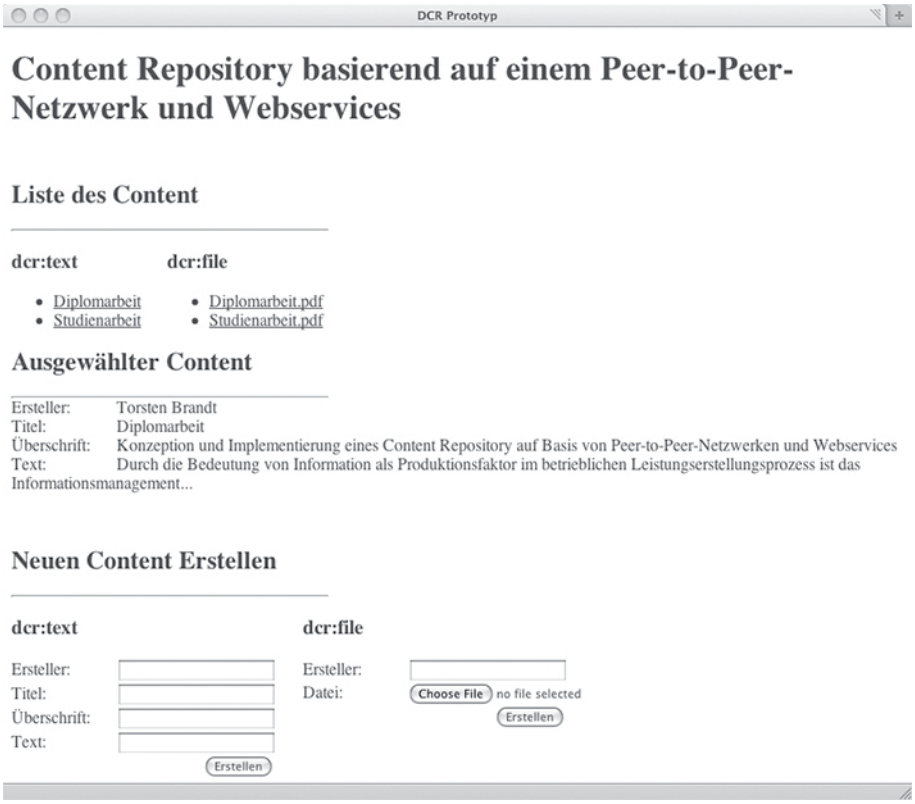


Abbildung 5.3: Bildschirmfoto der Client-Anwendung

Kapitel 6

Zusammenfassung und Ausblick

Das Content Management ist ein Teilbereich des Informationsmanagements und als solcher eine elementare Aufgabe der Unternehmensführung. Die dazu auf operativer Ebene eingesetzten Content-Management-Systeme nutzen Repositories zur Speicherung ihrer Daten. Werden zusätzliche Funktionalitäten wie ein Versionsmanagement, eine Sperrverwaltung oder das Management von Zugriffsberechtigungen durch ein Repository bereitgestellt, so wird von einem Content Repository gesprochen.

In dieser Arbeit wurde das Konzept eines Informationssystems, welches ein Content Repository mit verteilter Datenhaltung und Bereitstellung seiner Funktionalität über Webservices beinhaltet, vorgestellt. Dazu wurden, ausgehend vom Enterprise Content Management, die Anforderungen, die ein Content Repository an seine Datenhaltung sowie seine Anbindung an andere Anwendungen stellt, erarbeitet.

Für ein Konzept des Informationssystems basierend auf dem Service Layer Pattern wurde untersucht, ob ein Peer-to-Peer-Netzwerk zur Realisierung der Data Source Layer und somit einer verteilten Datenhaltung geeignet ist. Für die Service Layer wurde die Eignung von Webservices zur Realisierung der Schicht geprüft.

Zur Erfüllung der erarbeiteten Anforderungen an die Datenhaltung, Verfügbarkeit sowie Konsistenz des verwalteten Content zu jeder Zeit, wird im vorgestellten Konzept ein Peer-to-Peer-Netzwerk mit Document Routing, Re-

plikation und einem Ansatz, der auch im Umfeld der unsicheren Konnektivität eines Peer-to-Peer-Netzwerks eine konsistente Datenhaltung erlaubt, verwendet. Für die Anbindung an andere Anwendungen wurde eine flexible, offene sowie programmiersprachen- und plattformunabhängige Lösung benötigt. Diese kann durch Webservices angeboten werden und erlaubt so die Verwendung eines Content Repository in heterogenen Umgebungen.

Das erarbeitete Konzept wurde durch eine prototypische Implementierung ergänzt. Dadurch konnte die Umsetzbarkeit des Konzepts evaluiert werden. Somit kann festgehalten werden, dass sich ein Peer-to-Peer-Netzwerk sowie Webservices eignen, um die Datenhaltung eines Content Repository zu verteilen und eine flexible und plattformübergreifende Zugriffsmöglichkeit darauf bereitzustellen.

In dieser Arbeit konnte durch die prototypische Implementierung die Umsetzbarkeit des Konzepts verifiziert werden. Eine qualitative Untersuchung hinsichtlich der Performanz und praktischen Einsetzbarkeit eines solchen Informationssystems konnte in der gegebenen Bearbeitungsdauer nicht erfolgen. In einer solchen Untersuchung ist insbesondere zu prüfen, wie sich der Einsatz unterschiedlicher Transportprotokolle für Webservice-Nachrichten und die Art der Serialisierung von Content auf die Performanz auswirkt.

Ein weiteres Forschungsgebiet im Bereich von Content Repositories ist die Überlegung, wie über die Datenhaltung hinaus auch die Funktionalität eines Content Repository verteilt werden kann um die Verwendung eines zentralen Servers auch für diesen Bereich zu vermeiden.

Literaturverzeichnis

Adams, J., Koushik, S., Vasudeva, G. und Galambos, G. (2002): Patterns for E-Business: A Strategy for Reuse. IBM Press, Double Oak, USA.

Alexander, C. (1979): The Timeless Way of Building. Oxford University Press.

Andersen, D. (2001): *SETI@home*. In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Arbeitsgemeinschaft Online Forschung (2005): *Internet Facts 2005-I*. http://www.agof.de/studienarchiv.587.html. Stand: 16.03.2009.

Arbeitsgemeinschaft Online Forschung (2008): *Internet Facts 2008-III*. http://www.agof.de/studie.583.html. Stand: 16.03.2009.

Austin, D., Barbir, A., Ferris, C. und Garg, S. (2002): *Web Services Architecture Requirements*. http://www.w3.org/TR/2002/WD-wsa-reqs-20021011. Stand: 16.03.2009.

Baset, S. A. und Schulzrinne, H. (2006): *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications.

Becker, J. (1999): *Informationsmanagement*. In H. Corsten (Hrsg.), Betriebswirtschaftslehre. Oldenbourg, München.

Bernstein, P. A. (1997): *Repositories and Object Oriented Databases*. In Proceedings of BTW Conference, March 1997, Ulm, Germany.

Berthel, J. (1975): *Informations*. In E. Grochla (Hrsg.), Handwörterbuch der Betriebswirtschaft. C.E. Poeschel Stuttgart.

Bettag, U. (2001): *Web-Services*. Informatik-Spektrum, Band 24, S. 302–304.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. und Orchard, D. (2004): *Web Services Architecture*. http://www.w3.org/TR/ws-arch/. Stand: 16.03.2009.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. und Yergeau, F. (2008): *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/REC-xml/. Stand: 16.03.2009.

Buchmann, E. (2006): Erkennung und Vermeidung von unkooperativem Verhalten in Peer-to-Peer-Datenstrukturen. Dissertation, Otto-von-Guericke Universität Magdeburg.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. und Stal, M. (1998): Pattern-orientierte Software Architektur. Addison-Wesley-Logman, Bonn, Paris.

Chinnici, R., Moreau, J.-J., Ryman, A. und Weerawarana, S. (2007): *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. http://www.w3.org/TR/wsdl20/. Stand: 16.03.2009.

Comer, D. E. (2008): TCP/IP - Principles, Protocols and Architecture. Pearson Prentice Hall, Upper Saddle River, NJ, USA.

Curran, P. (2004): *JSR 215: Java Community Process version 2.6*. http://jcp.org/en/jsr/detail?id=215. Stand: 16.03.2009.

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R. und Stoica, I. (2001): *Wide-area cooperative storage with CFS*. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01).

Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. und Stoica, I. (2003): *Towards a Common API for Structured Peer-to-Peer Overlays*. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03).

Dambekalns, K. (2007): *A Content Repository for TYPO3 5.0*. In TYPO3 Developer Days 25.-29.04.2007, Dietikon, Switzerland.

Darlagiannis, V., Mauthe, A. und Steinmetz, R. (2004): *Overlay Design Mechanisms for Heterogeneous, Large-Scale, Dynamic P2P Systems.* Journal of Network and Systems Management, Band 12, S. 371–395.

de Carvalho, R. A. (2008): *An Enterprise Content Management Solution Based on Open Source.* In L. D. Xu, A. M. Tjoa und S. S. Chaudhry (Hrsg.), Research and Practical Issues of Enterprise Information Systems II Volume 1. Springer Boston.

Dingledine, R., Freedman, M. J. und Molnar, D. (2001a): *Accountability.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Dingledine, R., Freedman, M. J. und Molnar, D. (2001b): *Free Haven.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Dixit, A. K. und Nalebuff, B. J. (1997): Spieltheorie für Einsteiger: Strategisches Know-how für Gewinner. Schäffer-Poeschel Verlag Stuttgart.

Dornfest, R. und Brickley, D. (2001): *Metadata.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Dublin Core Metadata Initiative (2008): *Dublin Core Metadata Element Set, Version 1.1.* http://dublincore.org/documents/dces/. Stand: 16.03.2009.

Duc, B. M. (2007): Real-Time Object Uniform Design Methodology with UML. Springer, Dordrecht, The Netherlands.

Ehlers, L. H. (2003): Content Management Anwendungen - Spezifikation von Intenet-Anwendungen auf Basis von Content Management Systemen. Logos Verlag Berlin.

Ferraiolo, J., Jun, F. und Jackson, D. (2003): *Scalable Vector Graphics (SVG) 1.1 Specification.* http://www.w3.org/TR/SVG11/. Stand: 20.03.2009.

Fielding, R. T. (2005): *JSR 170 Overview - Standardizing the Content Repository Interface*. Technischer Bericht, Day Software.

Fischer, P. und Hofer, P. (2008): Lexikon der Informatik. Springer-Verlag, Berlin Heidelberg.

Fowler, M. (2002): Patterns of Enterprise Application Architecture. Addison-Wesley Longman, Amsterdam.

Goik, M., Hedler, M. und Westbomke, J. (2007): *Einsatz von Markup-Languages*. In R. Schmitz (Hrsg.), Kompendium Medieninformatik. Springer Berlin Heidelberg.

Großmann, M. und Koschek, H. (2005): Unternehmensportale. Springer.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A. und Lafon, Y. (2007): *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. http://www.w3.org/TR/2007/REC-soap12-part2-20070427/. Stand: 16.03.2009.

Hansen, H. R. (1996): Wirtschaftsinformatik 1. Lucius & Lucius, Stuttgart.

Heinrich, L. J. (2005): Informationsmanagement. R. Oldenbourg Verlag München.

Herden, S., Gómez, J. M., Rautenstrauch, C. und Zwanziger, A. (2006): Software-Architekturen für das E-Business - Enterprise-Application-Integration mit verteilten Systemen. Springer Berlin.

Herden, S. und Zwanziger, A. (2004): *Modeling Business Applications with Patterns*. In Proceedings of IV International Conference on Applied Enterprise Science (International Symposium on Business Informatics) (CICE'2004), Santa Clara (Cuba) 2004.

Hesse, W. und Mayr, H. C. (2008): *Modellierung in der Softwaretechnik: eine Bestandsaufnahme*. Informatik-Spektrum, Band 31, S. 377–393.

Hohpe, G. und Woolf, B. (2003): Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, Boston.

Huang, C.-T. und Gouda, M. G. (2006): Hop Integrety in the Internet. Springer New York.

Huemer, C. (2001): *Electronic Business XML - Grundlagen und Nutzen.* In K. Turowski und K. J. Fellner (Hrsg.), XML in der betrieblichen Praxis - Standards, Möglichkeiten, Praxisbeispiele. dpunkt-Verlag, Heidelberg.

IDABC (Hrsg.) (2004): European Interoperability Framework for Pan-European eGovernment Services. Office for Official Publications of the European Communities, Luxembourg.

ISO/IEC (1994): Open Systems Interconnection - Basic Reference Model: The Basic Model.

Judd, C. M., Nusairat, J. F. und Shingler, J. (2008): Beginning Groovy and Grails - From Novice to Professional. Springer-Verlag New York.

Jäger, K. (2008): Ajax in der Praxis - Grundlagen, Konzepte, Lösungen. Springer-Verlag Berlin Heidelberg.

Kampffmeyer, U. (2003): Enterprise Content Management - zwischen Vision und Realität. Whitepaper, Project Consult.

Kettermann, U. und Rohde, A. (2005): Spiele effektiv programmieren mit VB.net und DirectX. Springer Berlin Heidelberg New York.

Kossmann, D. und Leymann, F. (2004): *Web Services.* Informatik-Spektrum, Band 27, S. 117–128.

Krcmar, H. (2005): Informationsmanagement. Springer Berlin Heidelberg.

Kretzschmar, O. (2007): *Content-Related-Technologien.* In R. Schmitz (Hrsg.), Kompendium Medieninformatik. Springer Berlin Heidelberg.

Kuhlen, R. (1999): Die Konsequenzen von Informationsassistenten. Suhrkamp Frankfurt.

Langley, A. (2001): *Freenet.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Lassmann, W. (Hrsg.) (2006): Wirtschaftsinformatik. Gabler.

Maass, W. und Stahl, F. (2008): *Content Management als Teil des Kommunikationsmanagement.* In M. Meckel und B. F. Schmid (Hrsg.), Kommunikationsmanagement im Wandel. GWV Fachverlage GmbH, Wiesbaden.

Mahlmann, P. und Schindelhauer, C. (2007): Peer-to-Peer-Netzwerke. Springer-Verlag Berlin Heidelberg.

Melzer, I., Werner, S., Sauter, P., Hilliger von Thile, A., Flehmig, M., Zengler, B., Dostal, W., Tröger, P., Stumm, B., Lipp, M. und Jeckle, M. (2007): Service-orientierte Architekturen mit Web Services. Spektrum Akademischer Verlag, München.

Mertens, P. (Hrsg.) (2003): XML-Komponenten in der Praxis. Springer-Verlag Berlin, Heidelberg, New York.

Mertens, P., Bodendorf, F., König, W., Picot, A., Schumann, M. und Hess, T. (2005): Grundzüge der Wirtschaftsinformatik. Springer, Berlin, Heidelberg, New York.

Meszaros, G. und Doble, J. (1996): *Metapatterns: A pattern language for pattern writing.* In 3rd Pattern Languages of Programming conference, Monticello, Illinois.

Milleg, D. und Wagner, B. (2001): *Effizientes Content Management - ein strategischer Hebel für das Medienhaus der Zukunft.* Information Managament & Consulting, Band 16, S. 36–41.

Miller, J. (2001): *Jabber - Conversational Technologies.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S. und Xu, Z. (2003): *Peer-to-Peer Computing.* Technischer Bericht, HP Laboratories Palo Alto.

Mitra, N. und Lafon, Y. (2007): *SOAP Version 1.2 Part 0: Primer (Second Edition).* http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Stand: 16.03.2009.

Morris, C. W. (1988): Grundlagen der Zeichentheorie : Ästhetik und Zeichentheorie. Fischer Taschenbuchverlag Frankfurt/M.

Moser, J., Neugebauer, M., Ott, F., Stark, M., Sundermann, M., Vollmar, F. und Heinz, B. (2004): Leitfaden Web Services - Architektur, Integration, Nutzen. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., Berlin.

Ngo, T., Venkat, J., Stolarz, D. und Barkai, D. (2001): *Taxonomy of Peer-to-Peer Architectures.* Technischer Bericht, Peer-To-Peer Working Group.

Nüscheler, D. (2003): *Original Java Specification Request (JSR).* http://jcp.org/en/jsr/detail?id=170. Stand: 16.03.2009.

Nüscheler, D. und Piegaze, P. (2005): *JSR 170: Content Repository for Java technology API.* http://jcp.org/en/jsr/detail?id=170. Stand: 16.03.2009.

OMG (2007a): Unified Modeling Language: Infrastructure.

OMG (2007b): Unified Modeling Language: Superstructure.

Oram, A. (2000): *Gnutella and Freenet Represent True Technological Innovation.* http://www.oreillynet.com/lpt/a/208. Stand: 16.03.2009.

Pemberton, S., Austin, D., Axelsson, J., Çelik, T., Dominiak, D., Elenbaas, H., Epperson, B., Ishikawa, M., Matsui, S., McCarron, S., Navarro, A., Peruvemba, S., Relyea, R., Schnitzenbaumer, S. und Stark, P. (2002): *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition).* http://www.w3.org/TR/xhtml1/. Stand: 09.12.2008.

Pietsch, T., Martiny, L. und Klotz, M. (2004): Strategisches Informationsmanagement: Bedeutung, Konzeption und Umsetzung. Schmidt, Berlin.

Quantz, J. (2003): Basisreport Integration mit Web Services - Konzept, Fallstudien und Bewertung. Berlecon Research GmbH, Berlin.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. und Shenker, S. (2001): *A Scalable Content-Addressable Network*. In ACM SIGCOMM.

Rautenstrauch, C. und Schulze, T. (2003): Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker. Springer Berlin Heidelberg.

Richards, R. (2006): Pro PHP XML and Web Services. Springer New York.

Roden, G. (2008): Auf der Fährte von C# - Einführung und Referenz. Springer-Verlag Berlin Heidelberg.

Rowstron, A. und Druschel, P. (2001): *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms. Heidelberg.

Rupp, C., Queins, S. und Zengler, B. (2007): UML 2 glasklar. Praxiswissen für die UML-Modellierung. Hanser Fachbuch.

Röttgers, J. (2003): Mix, Burn & R.I.P. Heise-Verlag, Hannover.

Röwekamp, L. (2001): *Prinzipien und Aufbau eines Content Management Systems*. Information Managament & Consulting, Band 16, S. 12–17.

Rückel, D. C., Steininger, K., Riedl, R. und Roithmayr, F. (2007): *Fallstudie: Einführung eines Enterprise-Content-Management-Systems*. HMD - Praxis der Wirtschaftsinformatik, Band 258, S. 78–88.

Saake, G. und Sattler, K.-U. (2004): Algorithmen & Datenstrukturen. dpunkt Verlag.

Saake, G., Sattler, K.-U. und Heuer, A. (2008): Datenbanken - Konzepte und Sprachen. Mitp-Verlag, Heidelberg.

Seufert, S. E. (2002): *Der Entwurf strukturierter rollenbasierter Zugriffskontrollmodelle.* Informatik - Forschung und Entwicklung, Band 17, S. 1–11.

Shirky, C. (2001): *Listening to Napster.* In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Smolnik, S. (2007): *Nutzenpotentiale und Performancemessung.* HMD - Praxis der Wirtschaftsinformatik, Band 258, S. 25–34.

Spahni, D. und Meir, J. (2003): *Web Services im eGovernment: Vision und Konzept veränderter Wertschöpfungsketten der staatlichen Leistungserbringung.* In Zwischen Rechtstheorie und e-Government, Aktuelle Fragen der Rechtsinformatik 2003. Tagungsband des 6. Internationalen Rechtsinformatik Symposium (IRIS), Salzburg, 2003.

Stachowiak, H. (1973): Allgemeine Modelltheorie. Springer-Verlag.

Steinmetz, R. und Wehrle, K. (2004): *Peer-to-Peer-Networking & - Computing.* Informatik-Spektrum, Band 20. Februar, S. 51–54.

Steinmüller, W. (1993): Informationstechnologie und Gesellschaft. Wissenschaftliche Buchgesellschaft, Darmstadt.

Stoica, I., Adkins, D., Zhuang, S., Shenker, S. und Surana, S. (2002): *Internet Indirection Infrastructure.* In Proceedings of ACM SIGCOMM.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. und Balakrishnan, H. (2001): *Chord: A Scalable Peer-to-Peer Lookup Service for Intenet Applications.* In SIGCOM.

Szyperski, N. (1980): *Informationsbedarf.* In E. Grochla (Hrsg.), Handwörterbuch der Organisiation. C.E. Poeschel Stuttgart.

Tian, M., Voigt, T., Naumowicz, T., Ritter, H. und Schiller, J. (2003): *Performance impact of web services on Internet servers.* In Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems. Marina Del Rey, California, USA.

Turowski, K. (2003): Fachkomponenten : Komponentenbasierte betriebliche Anwendungssysteme. Shaker Verlag, Aachen.

Vassbotn, H. (2006): *Delphi for .NET*. In J. Shemitz (Hrsg.), .NET 2.0 for Delphi Programmers. Springer-Verlag New York.

Völker, R., Sauer, S. und Simon, M. (2007): Wissensmanagement im Innovationsprozess. Physica-Verlag Heidelberg.

Waldman, M. und Rubin, A. (2001): *Trust*. In A. Oram (Hrsg.), Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc.

Wieczorrek, H. W. und Mertens, P. (2008): Management von IT-Projekten. Springer-Verlag Berlin Heidelberg.

Wilde, T. und Hess, T. (2007): *Forschungsmethoden der Wirtschaftsinformatik - Eine empirische Untersuchung*. Wirtschaftsinformaik, Band 49, S. 280–287.

Wollnik, M. (1988): *Ein Referenzmodell des Informationsmanagements*. Information Management, Band 3, S. 34–43.

Xu, Z. und Hu, Y. (2003): *SBARC: A Supernode Based Peer-to-Peer File Sharing System*. In Proceedings of the Eighth IEEE International Symposium on Computers and Communications.

Yang, B. und Garcia-Molina, H. (2002): *Efficient Search in Peer-to-peer Networks*. In ICDCS.

Yang, B. und Garcia-Molina, H. (2003): *Designing a Super-Peer Network*. In Proceedings of the 19th IEEE International Conference on Data Engineering, S. 49 – 62.

Zhang, L.-J., Zhang, J. und Cai, H. (2007): Services Computing. Springer-Verlag, Berlin Heidelberg.

Zhuang, S., Lai, K., Stoica, I., Katz, R. und Shenker, S. (2003): *Host Mobility Using an Internet Indirection Infrastructure.* In First International Conference on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys).

Abschließende Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Magdeburg, den 30.03.2009