Verteilte Suche in Digitalen Bibliotheken

Diplomarbeit

UNIVERSITÄT ROSTOCK
FACHBEREICH INFORMATIK
LEHRSTUHL DBIS
ALBERT-EINSTEIN-STRASSE 21
18059 ROSTOCK

vorgelegt von Mathias Zarick
geboren am 23.02.1977 in Schwerin

Erstgutachter:	Prof. Andreas Heuer
Zweitgutachter:	Prof. Peter Forbrig
Betreuer:	Dr. Holger Meyer, Mathias Bietz

Abgabedatum: 30. Juni 2002

Zusammenfassung

Die vorliegende Diplomarbeit untersucht die Thematik der verteilten Suche auf Digitalen Bibliotheken, die durch eine dezentrale Bereitstellung von Informationen in verschiedenen meist heterogenen Datenquellen motiviert ist.

Dazu wird eine Komponente in die Architektur Digitaler Bibliotheken eingegliedert, welche eine verteilte Suche über mehrere Archive ermöglicht. Für diese werden Anforderungen sowie Realisierungsansätze genannt und diskutiert.

Zwei Implementierungen, die in Zusammenhang mit dieser Arbeit entstanden sind, realisieren verteilte Suchen über mehrere entfernte Repositories. Zum einen wurden diese für MyCoRe umgesetzt. Die zweite Implementierung realisiert einen Cross Archive Searching Service, der auf dem OAI-Protokoll basiert.

Abstract

This diploma thesis investigates the subject of distributed search on Digital Libraries that is motivated by a decentral provision of information. This information is mostly located in different heterogeneous data sources.

Therefore a component for distributed search on several archives is integrated in the architecture of Digital Libraries. The requirements and ways of realization for this component are shown and discussed.

Two implementations provided with this thesis realize distributed search on several remote repositories. One is for MyCoRe. The second is an implementation of a cross archive searching service based on the OAI protocol.

CR-Klassifikation

E.2. Data Storage Representations
H.2.5. Heterogeneous Databases
H.2.8. Database Applications
H.3.3. Information Search And Retrieval
H.3.7. Digital Libraries

Keywords

Digitale Bibliothek, verteilte Systeme, verteilte Suche, Information Retrieval, XML, Content Management, OAI, Z39.50

Inhalt

1 Einleitung **1**

1.1 Ziel und Inhalt der Diplomarbeit 1
1.2 Digitale Bibliotheken am Lehrstuhl DBIS 2
1.3 Gliederung der Arbeit 2

2 Grundlagen **5**

2.1 Digitale Bibliotheken 5
2.1.1 Definition 5
2.1.2 Dokumente und Metadaten 5
2.1.3 Arten von Anfragen 6
2.1.4 Realisierungen 7
2.2 Begriffe 8
2.3 Dublin Core 8
2.3.1 Charakteristika von Dublin Core 8
2.3.2 Die Dublin Core Elemente 9
2.4 Gegenüberstellung mit verteilten Datenbanken 11
2.5 MILESS und MyCoRe 14
2.6 IBM Content Management 16
2.6.1 Allgemeines 16
2.6.2 Architektur CM V8 17
2.6.3 Datenmodell und Anfragesprache 19

3 Ein verteilter Such- / Integrationsdienst **23**

3.1 Architektur einer Digitalen Bibliothek nach [EF00] 23
3.2 Broker / Trader 25
3.3 Föderationsdienst 26
3.4 VDS - LDS Architektur in BlueView 27
3.5 Ausprägungen des Dienstes 28
3.5.1 Replizierung 28
3.5.2 “On-the-Fly”-Verarbeitungen 29
3.6 Theoretische Betrachtungen zur Integration 30

4 Anforderungen an den verteilten Suchdienst 33
4.1 Überblick . . . 33
4.1.1 Parallele Suche . . . 33
4.1.2 Ergebnismischung . . . 34
4.1.3 Doublettenerkennung . . . 34
4.1.4 Kein Informationsverlust . . . 34
4.1.5 Kapselung der einzelnen Digitalen Bibliothekssysteme . . . 34
4.1.6 Vollständige Suche . . . 35
4.1.7 Adäquate Recherchemöglichkeiten . . . 35
4.1.8 Globales Ranking . . . 35
4.1.9 Sicherheit bei der Kommunikation . . . 36
4.1.10 Kontrolle und Schutz von Urheberrechten . . . 36
4.1.11 Personalisierung . . . 36
4.1.12 Rechtemanagement . . . 38
4.1.13 Lokale Autonomie . . . 38
4.1.14 Hohe Verfügbarkeit . . . 38
4.2 Parallele Suche . . . 38
4.2.1 Probleme . . . 38
4.2.2 Realisierungsmöglichkeiten/Problemlösungen . . . 39
4.3 Ergebnismischung . . . 39
4.3.1 Probleme . . . 39
4.3.2 Realisierungsmöglichkeiten/Problemlösungen . . . 40
4.4 Doublettenerkennung . . . 41
4.4.1 Probleme . . . 41
4.4.2 Realisierungsmöglichkeiten/Problemlösungen . . . 42
4.5 Globales Ranking . . . 43
4.5.1 Ranking allgemein . . . 43
4.5.2 Übertragung auf die verteilte Suche . . . 47
4.5.3 Probleme . . . 53
4.5.4 Realisierungsmöglichkeiten/Problemlösungen . . . 53
4.5.5 Zusammenfassung . . . 55
4.6 Zusammenfassung . . . 56

5 Existierende Ansätze für verteilte Suchdienste 57
5.1 Kommunikation in verteilten Anwendungen . . . 57
5.1.1 Low-Level-Netzwerk-Programmierung . . . 57
5.1.2 HTTP-HTTPS-Kommunikation . . . 58
5.1.3 CORBA . . . 58
5.1.4 RMI . . . 60
5.1.5 SOAP . . . 60
5.2 Metadatenzugriff über OAI . . . 61
5.2.1 OAI Konzepte . . . 62
5.2.2 Das OAI Protokoll . . . 64
5.2.3 Realisierungen eines Cross Archive Search Services . . . 65

5.3 Z39.50 66
5.4 Federated Search des Enterprise Information Portals 69
5.5 Zusammenfassung 73

6 Konzeption **75**
6.1 Ein verteilter Suchdienst für MyCoRe 75
6.1.1 MyCoRe Konzepte 75
6.1.2 MyCoRe-Architektur 78
6.1.3 verteilte Suche in MyCoRe 80
6.1.4 Implementierung und Konfiguration 82
6.1.5 Bewertung und Erweiterbarkeit 83
6.2 Cross Archive Searching Service 85
6.2.1 Architektur und Funktionsweise 85
6.2.2 Auswahl der Programmiermittel/Implementierungsdetails 87
6.2.3 Bewertung und Erweiterbarkeit 88

7 Zusammenfassung und Ausblick **91**

A MyCoRe Beispiele **93**
A.1 XML-Repräsentation eines multimedialen Objektes 93
A.2 XML-Schema für eine Klasse multimedialer Objekte 94
A.3 XML-Repräsentation eines Suchergebnisses 96
A.4 XSLT-Stylesheet zur Darstellung 96
A.5 Ausgabe nach XSLT-Transformation 97

B Installation des Cross Archive Searching Service **99**

C Statistiken zum Harvesting des Cross Archive Searching Service **103**

Kapitel 1

Einleitung

> WE SHALL NOT CEASE FROM EXPLORATION
> AND THE END OF ALL OUR EXPLORING
> WILL BE TO ARRIVE WHERE WE STARTED
> AND KNOW THE PLACE FOR THE FIRST TIME.
>
> T.S. ELIOT [ELI44]

Eine der wichtigsten Funktionen einer Digitalen Bibliothek ist zweifelsohne eine adäquate Suche auf den Inhalten. Denn jede sehr gut umgesetzte Archivierung von Wissen findet erst durch die Möglichkeit einer Recherche auf diesem Wissen einen erfüllenden Zweck.

Die Entwicklung von kommerziellen Digitalen Bibliothekssystemen steht noch ziemlich am Anfang. Jedoch werden Digitale Bibliotheken immer mehr Objekte der aktuellen Forschungsarbeit. Dadurch werden jetzt auch viele neue Anforderungen an Digitale Bibliotheken aufgedeckt.

1.1 Ziel und Inhalt der Diplomarbeit

Durch die dezentrale Bereitstellung von Informationen in immer mehr werdenden digitalen Archiven ergibt sich eine Anforderung, die in dieser Diplomarbeit näher beleuchtet wird. Es gilt zu untersuchen, wie eine Integration fremder Digitaler Bibliothekssysteme in ein verteiltes System realisiert werden kann. Unter einem verteilten System soll hier ein Digitales Bibliothekssystem verstanden werden, dass in der Lage ist, andere digitale Archive[1] einzubinden. Für eine solche Einbindung ist vor allem zu klären, wie auf der entstehenden Einheit mit verteilt vorliegenden Daten übergreifend und adäquat gesucht werden kann.

In dieser Arbeit werden Anforderungen an eine entsprechende Softwarearchitektur und ihre Dienste formuliert. Für die Realisierung eines integrierten, verteilten Suchdienstes wurden verschiedene Ansätze für eine Umsetzung untersucht.

[1] in der Literatur findet man oft auch die Bezeichnung *Repository*

Zum einen wurden die Standardprotokolle “Z39.50” und “OAI Harvesting Protocol” untersucht. Für letzteres wurde ein Suchdienst implementiert, der in der Lage ist auf verteilt vorliegenden Archiven zu suchen.

Desweiteren bietet ein kommerzielles Softwaresystem von IBM einen Ansatz. Durch das Produkt “Enterprise Information Portal” aus dem “Content Manager”-Produktportfolio wird mit dem “Federated Search”-Feature eine föderative Suchmöglichkeit für verschiedene digitale Archive bereitgestellt.

Zuguterletzt wurden auch Protokolle für die Kommunikation in verteilten Anwendungen als Ansatz für proprietäre Lösungen untersucht. Für MyCoRe (siehe Kapitel 2.5) wurde ein verteilter Suchdienst implementiert. MyCoRe stand dabei als grundlegende Architektur zur Verfügung.

1.2 Digitale Bibliotheken am Lehrstuhl DBIS

Im Rahmen des Förderkonzepts Global-Info des Bundesministeriums für Bildung und Forschung (BMBF) ist BlueRose[2] als eine Sonderfördermaßnahme entstanden. Ein von der Universität Rostock beantragtes Teilprojekt an dieser Maßnahme wurde leider als nicht förderungswürdig erachtet und abgelehnt. Da der Lehrstuhl DBIS aber zum Zeitpunkt der Ablehnung bereits umfangreiche Vorarbeiten geleistet hat, wurde das Haushaltsprojekt BlueView[3] ins Leben gerufen.

In BlueRose und BlueView wurden weitreichende Konzepte für die Realisierung Digitaler Bibliothekssysteme geschaffen. In mehreren Arbeiten wurden die Dienste einer Digitalen Bibliothek implementiert. Diese Implementierungen stützen sich in der Regel auf die Nutzung von Datenbankmanagementsystemen.

Die in diesen Arbeiten entstandenen Konzepte und Realisierungsvorschläge stellen die Grundlage dieser Arbeit dar. Die Implementierung eines verteilten Suchdienstes als ein Modul in der MyCoRe-Architektur bedeutet einen weiteren Schritt für das Vorhaben des Lehrstuhls, die Forschungsarbeit aus BlueRose und BlueView in MyCoRe einfließen zu lassen.

1.3 Gliederung der Arbeit

Die vorliegende Diplomarbeit gliedert sich wie folgt:

Kapitel 2 enthält grundlegende Betrachtungen für diese Arbeit. Dazu gehören die Vorstellung des Begriffes Digitale Bibliothek und von anderen Begriffen im Zusammenhang mit der verteilten Suche. Außerdem wird der Metadatensatz des Dublin Core vorgestellt. Die verteilte Suche wird mit den Konzepten und Anforderungen bezüglich verteilter Datenbanken gegenübergestellt. Eine Vorstellung der MILESS/MyCoRe-Systeme und des IBM Content Managements schließen dieses Kapitel ab.

[2] *B*uilding *L*ibraries *U*nifying *E*nhanced *R*etrieval-*O*riented User *Se*rvices

[3] *Blue*Rose *V*irtual *I*ntegration of *E*lectonic *W*eb Services

Kapitel 3 stellt die Architektur einer Digitalen Bibliothek mit seinen Komponenten vor. In diese Architektur wird ein Dienst zur verteilten Suche und Integration eingegliedert, der es ermöglichen wird, verteilte Suchen über mehrere Digitale Bibliotheken durchzuführen. Außerdem werden in diesem Kapitel zwei unterschiedliche Ausprägungsarten dieses Dienstes beschrieben. Einige theoretische Betrachtungen zur Integration fremder Digitaler Bibliotheken runden dieses Kapitel ab.

Kapitel 4 umfasst einen Anforderungskatalog an den verteilten Such- / Integrationsdienst. Es werden 14 Anforderungen genannt. Auf vier wichtige wird detaillierter eingegangen. Die Realisierungsmöglichkeiten einiger Anforderungen werden diskutiert.

Kapitel 5 stellt vier bestehende Ansätze für die Umsetzung eines Dienstes für die verteilte Suche über mehrere Digitale Bibliotheken vor. Dazu gehören zuerst allgemeine Betrachtungen von Kommunikationsmöglichkeiten in verteilten Anwendungsszenarien. Weiterhin werden in diesem Kapitel das OAI- und das Z39.50-Standardprotokoll als Lösungsansätze vorgestellt. Das “Federated Search”-Feature aus dem Enterprise Information Portal bietet einen weiteren Ansatz und wird ebenfalls in Kapitel 5 beschrieben.

Kapitel 6 beschreibt die in Zusammenhang mit dieser Arbeit entstandenen Implementierungen. Zum einen wurde eine verteilte Suche für die MyCoRe-Plattform realisiert. Die andere Implementierung nutzt das OAI-Protokoll und sucht auf replizierten Metadaten von entfernten Repositories.

Kapitel 7 fasst die vorliegende Arbeit zusammen und nennt weitere Forschungsaspekte als Ausblick des Themas der verteilten Suche in Digitalen Bibliotheken.

Kapitel 2

Grundlagen

In diesem Kapitel werden für die Diplomarbeit grundlegende Konzepte und Begriffe diskutiert. Desweiteren wird das MILESS/MyCoRe-Projekt vorgestellt, welches die Basisplattform für die Implementierungsarbeit im Rahmen dieser Diplomarbeit bildet. In einem weiteren Abschnitt dieses Kapitels wird ein konkretes System von IBM vorgestellt, der IBM Content Manager mit dem Enterprise Information Portal.

2.1 Digitale Bibliotheken

2.1.1 Definition

Man findet in der Literatur eine Vielzahl von Definitionen für eine Digitale Bibliothek. Eine umfassende Definition findet man in [HM01]. Demnach gibt es zwei Betrachtungsweisen, wenn man von dem Begriff Digitale Bibliothek spricht.

Zum einen wird eine Digitale Bibliothek als das eigentliche Archiv aufgefasst, dass Dokumente von bleibendem Wert sammelt. Diese werden in ihrem Katalog mit Metadaten erschlossen. Es wird verlangt, dass die Dokumente langfristig zitierbar bleiben. Sie können einer Versionierung unterliegen.

Eine Digitale Bibliothek wird aber auch als ein Softwaresystem verstanden, das in der Lage ist, eine Archivierung von Dokumenten zu bewerkstelligen. Außerdem muss es Funktionen für die Erzeugung, Bereitstellung, Erschließung, Speicherung, Verteilung, Suche, Darstellung und Nutzung von Dokumenten anbieten. Solch ein System kann weltweit verteilt sein. Es hat verschiedene Nutzergruppen mit unterschiedlichen Interessen: Es gibt Anbieter (Verlage, Labels), Vermittler (Bibliotheken) und Benutzer.

2.1.2 Dokumente und Metadaten

Der Hauptdatenbestandteil einer Digitalen Bibliothek sind also die Dokumente und die dazu erschlossenen Metadaten. Die Dokumente als “digitale Ware” sind elektronisch speicherbar. Es sind Texte, Bilder, Audio- und Videoaufzeichnungen etc..

Metadaten beschreiben die eigentliche Informationsressourcen, also die Dokumente, Bücher, Schriftstücke, etc. die in einer konventionellen Bibliothek (und natürlich auch in einer Digitalen Bibliothek) archiviert werden. Metadaten sind also Daten über Daten.

Ein Satz von Metadaten besteht aus einer Menge von Attributen oder anderen Elementen, die benutzt werden, eine Ressource zu beschreiben. Die Menge aller Metadatensätze einer Bibliothek ist ihr Katalog. Ein Metadatensatz enthält zum Beispiel folgende Elemente, die das zugehörige Dokument beschreiben: den Namen des Autors, den Titel, die Daten der Aufnahme in den Bestand und der Veröffentlichung, eine Kurzbeschreibung, etc..

Die Aufbewahrung der Metadaten kann auf zwei verschiedenen Arten erfolgen. Zum einen kann man die Metadaten in die beschriebene Ressource selbst einbetten. Der bessere Weg ist es jedoch, die Metadaten gesondert in einem Katalogsatz zu speichern.

2.1.3 Arten von Anfragen

Anfragen auf die Metadaten werden auch als parametrische Anfragen bezeichnet. Der Nutzer sucht nach einem bestimmten Inhalt in den Metadaten. Dabei kann man zwischen einer exakten Suche, wo ein bestimmtes Metadatenfeld einen bestimmten Wert haben muss, und einer untypisierten Suche unterscheiden, bei der der Wert auf ein beliebiges Metadatenfeld zutreffend sein muss.

Ein Beispiel, das hier in SQL-MM/Text [sql00] formuliert ist, wäre:

```
SELECT doi
FROM documentstore
WHERE dc_creator.contains(' "Gauss" ') = 1
```

Das Metadatenfeld `dc_creator`, auf das sich diese Anfrage richtet, soll hierbei das *CREATOR*-Feld aus dem Dublin Core Metadatensatz (siehe 2.3) darstellen. Durch diese Anfrage werden also alle *Document Identifier (DOI)* zurückgegeben, die von `Gauss` verfasst worden sind.

Anfragen auf die Dokumente zielen auf den Inhalt der gespeicherten Dokumente ab. Dazu zählen Volltextsuchen und inhaltsbasierte Anfragen auf Bilder sowie Audio- und Videosequenzen.

Ein Beispiel dafür in SQL-MM/Text wäre:

```
SELECT doi
FROM documentstore
WHERE document_content.contains(' "Software Engineering"
          IN SAME PARAGRAPH AS "Database Management System" ') = 1
```

Hier wird nach allen Dokumenten gefragt, die die Wortgruppen ‘Software Engineering’ und ‘Database Management System’ in einem Absatz enthalten. Es

werden dabei ebenfalls die *Document Identifier* dieser Dokumente zurückgegeben. `document_content` steht in der Anfrage für den eigentlichen Volltext der Dokumente.

Ein anderes Beispiel, in dem der Inhalt von unbewegten Bilddokumenten (*Still Images*) abgefragt wird und in SQL-MM/Still Image [sql01] verfasst ist, wäre:

```
SELECT doi
FROM imagestore
WHERE SI_ScoreByAvgClr(SI_mkAvgClr(SI_mkRGBClr(255,0,255)),image) < 1.2
```

Hier wird nach allen Bilddokumenten gefragt, dessen Durchschnittsfarbe ungefähr *Violett (RGB-Farbe (255,0,255))* ist. Die Funktion `SI_ScoreByAvgClr` liefert ein Ähnlichkeitsmaß, das größer oder gleich 0 ist. Je kleiner der Wert ist, desto ähnlicher sind sich die beiden Farben. Die Skalierung dieses Ähnlichkeitsmaßes ist implementierungsabhängig und außerhalb des Fokus von SQL-MM/Still Image. `Image` steht in der Anfrage für das eigentliche Bild. Dieses wird durch einen Wert des SQL-Typs `SI_StillImage` repräsentiert.

Hybride Anfragen Natürlich lassen sich die oben beschriebenen atomare Anfragen mittels Boolescher Operationen zu komplexeren Anfragen verknüpfen. Vermischt man parametrische Anfragen mit Dokument-Anfragen so nennt man diese hybride Anfragen.

Je nach dem, was das Ergebnis einer Anfrage sein soll, unterscheidet man zusätzlich noch zwischen Metadaten-Retrieval und Dokument-Retrieval. In den oberen Beispielen wurden jeweils *Document Identifier* also Metadaten zurückgegeben.

Für den Zugriff auf die Metadaten und die Dokumente müssen eventuell Lizensierungs- und Autorisierungsdienste angesprochen werden, um zu überprüfen, ob ein Zugriff auf die entsprechenden Daten erlaubt ist.

2.1.4 Realisierungen

Mittlerweile gibt es eine Vielzahl von Realisierungen Digitaler Bibliotheken. Hier sollen 2 kurz vorgestellt werden, die eine verteilte Suche realisieren.

Die Bayrische Staatsbibliothek München archiviert historische Schriften verschiedener Epochen, diese sind in verschiedenen Sammlungen gruppiert.Über ein Web-Interface[1] kann auf den Datenbeständen verteilt recherchiert werden.

Die Deutsche Bibliothek bietet einen Z39.50-Gateway[2], über den es möglich ist verteilt über verschiedene Digitale Bibliotheken zu suchen. Für diese verteilte Suche kommt die Kommunikation über Z39.50 zum Einsatz. Z39.50 wird in 5.3 näher vorgestellt.

[1] http://mdz.bib-bvb.de

[2] http://z3950gw.dbf.ddb.de

In 5.2.3 werden 2 weitere Dienste vorgestellt, die in der Lage sind, über verteilte Daten verschiedener Archive zu suchen. Dazu nutzen sie das OAI-Protokoll (siehe 5.2) um die Metadaten in eine lokale Datenbank zu replizieren.

2.2 Begriffe

Archiv, bzw. Repository Das Archiv oder das Repository ist die Gesamtheit der Dokumente und der dazugehörigen Metadaten, die in einer Digitalen Bibliothek gespeichert sind. Sie sind in ihren Lokalen Dokumentenservern abgelegt.

Integration Unter Integration wird die Einbindung des Repositories einer Digitalen Bibliothek in ein anderes System verstanden. Die Integration hat immer eine Richtung. Es sind aber auch bidirektionale Beziehungen denkbar, bei denen sich zwei Systeme gegenseitig integrieren.

Metasystem Als ein Metasystem bezeichne ich in dieser Diplomarbeit genau jene betrachtete Systeme, die in der Lage sind, andere Digitale Bibliothekssysteme bzw. deren Repositories zu integrieren und ihre Inhalte für eine verteilte Recherche zu nutzen.

Verbund Ein Verbund Digitaler Bibliothekssysteme ist eine Menge von Systemen, die allesamt eine Integrationsbeziehung zu mindestens einem der anderen Systeme dieses Verbundes haben. Die Richtung der Integrationsbeziehungen spielt dabei keine Rolle.

2.3 Dublin Core

Für die Beschreibung einer breiten Masse von Ressourcen hat sich der Standard des Dublin Core Metadatensatzes etabliert. Er stellt 15 Attribute bereit.

2.3.1 Charakteristika von Dublin Core

Die Stärke von Dublin Core ist seine Einfachheit und Intuitivität. Im folgenden sind die wichtigsten Charakteristika der Dublin Core Metadatensätze aufgelistet:

Einfachheit der Anlage und Aufrechterhaltung Die Menge der Elemente ist mit Bedacht klein und einfach gehalten worden. Dadurch ist es sehr einfach und mit wenig Kosten möglich, beschreibende Datensätze für Dokumente zu schaffen. Die Datensätze reichen aber trotzdem aus, ein effektives Retrieval zu ermöglichen.

Allgemein verständliche Semantik Eine unklare Semantik von Metadatenfeldern kann hinderlich für eine Recherche sein. Die Semantik der Dublin Core Elemente ist daher einfach und intuitiv erkennbar.

Internationale Ausrichtung Für die Entwicklung des Standards wird die multilinguale und multikulturelle Natur des elektronischen Informationsuniversums berücksichtigt.

Erweiterbarkeit Der Dublin Core Metadatensatz ist leicht um andere Metadatenfelder erweiterbar. Dadurch können eventuell notwendige präzisere Retrievalanforderungen befriedigt werden.

2.3.2 Die Dublin Core Elemente

Die Metadaten Elemente von Dublin Core werden in 3 Gruppen klassifiziert:

1. Elemente, die auf den Inhalt der Ressource bezogen sind,
2. Elemente, die auf die Ressource betrachtet als intellektuelles Eigentum bezogen sind,
3. Elemente, die auf die Instanzierung der Ressource bezogen sind.

Die einzelnen Elemente dieser Gruppen sind in Tabelle 2.1 dargestellt:

Inhalt	intellektuelles Eigentum	Instanzierung
Title	Creator	Date
Subject	Publisher	Type
Description	Contributor	Format
Source	Rights	Identifier
Language		
Relation		
Coverage		

Tabelle 2.1: Die 15 Dublin Core Elemente

Die Bedeutung der einzelnen Elemente sind in der Tabelle 2.2 kurz zusammengefasst:

Alle diese 15 Elemente können bei einer Beschreibung einer Ressource gar nicht oder beliebig oft auftauchen. So ist es möglich, dass einem Dokument mehrere Autoren also CREATOR-Elemente zugewiesen werden können.

Durch das Dublin Core XML-Namespace[3] lassen sich Dublin Core Metadaten einfach in XML-Dokumente einbetten. Ein selbsterklärendes Beispiel ist mit dem nachfolgenden XML-Dokument gegeben. Der Metadatensatz ist in diesem Beispiel jedoch sehr spärlich gehalten.

[3] http://purl.org/dc/elements/1.1/

TITLE	Der Name der Ressource zugeteilt vom CREATOR oder PUBLISHER.
CREATOR	Die Person oder Organisation, die für den intellektuellen Inhalt der Ressource verantwortlich ist, der Autor.
SUBJECT	Das Thema der Ressource, Schlüsselwörter, Klassifikationen.
DESCRIPTION	Eine textuelle Beschreibung des Inhalts der Ressource, etwa Abstracts bei Dokumenten oder Kurzbeschreibungen bei Bildern.
PUBLISHER	Der Herausgeber, der dafür verantwortlich ist, dass die Ressource in der entsprechenden Form verfügbar ist, der Veröffentlicher.
CONTRIBUTOR	Zusätzliche Person oder Organisation, die entscheidend jedoch auf sekundärer Basis bei der Entstehung der Ressource beigetragen hat.
DATE	Das Datum an welchem die Ressource in der vorliegenden Form verfügbar gemacht wurde.
TYPE	Der Typ der Ressource, wie zum Beispiel Homepage, Roman, Arbeitspapier etc.
FORMAT	Die Datenrepräsentation der Ressource, zum Beispiel der MIME-Type [FB96].
IDENTIFIER	Ein String oder ein numerischer Wert zur eindeutigen Identifikation der Resource, z.B. URLs, DOIs (siehe 4.4), ISBNs.
SOURCE	Eine Quelle, aus denen die vorliegende Ressource erarbeitet worden ist.
LANGUAGE	Die Sprache, in der der Inhalt der Ressource verfasst ist.
RELATION	Eine Beziehung zu einer anderen Ressource.
COVERAGE	Ein Charakteristikum zum Bereich des Inhalts der Ressource. Typisch sind hierfür ein Aufbewahrungsort und/oder eine Zeitperiode der Gültigkeit.
RIGHTS	Informationen über Rechte bezüglich der Ressource.

Tabelle 2.2: Bedeutung der Dublin Core Elemente

```
<?xml version="1.0"?>
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>
    Objektorientierte Datenbanken
  </dc:title>
  <dc:creator>
    Andreas Heuer
  </dc:creator>
  <dc:publisher>
    Addison-Wesley
  </dc:publisher>
  <dc:identifier>
    ISBN 3-89319-800-8
  </dc:identifier>
</metadata>
```

Dublin Core kommt in vielen Bibliotheksszenarien zum Einsatz. So wurde in MILESS (siehe 2.5) Dublin Core als Metadatensatz für ein Dokument verwendet. Ein OAI Data Provider, also ein System, das eine OAI-Schnittstelle anbietet (siehe 5.2), muss laut Standard als Metadatensatz mindestens Dublin Core anbieten.

2.4 Gegenüberstellung mit verteilten Datenbanken

Bei verteilten Datenbanken werden die Datenbestände auf verschiedenen Rechnern repliziert oder verteilt gehalten. Dafür wird der Datenbestand kontrolliert auf die einzelnen Rechner fragmentiert. Unter Replikation von Daten versteht man das redundante Speichern von Daten auf mehreren Servern. Bei einer Verteilung werden bestimmte Teile der Daten auf bestimmten Rechnern gehalten. Diese Datenverteilung bringt den Vorteil der Steigerung von Performance und der Ausfallsicherheit.

Im Unterschied dazu werden bei einem Metasystem die Daten zu einer Einheit “zusammengeschweißt”. Dadurch entsteht eine größere Wissensbasis, die aber genauso verteilt vorliegt. Das verteilte Vorliegen der einzelnen Ressourcen bedeutet aber auch für die Metasysteme einen Vorteil bezüglich Lastverteilung und dadurch der Steigerung der Performance.

Die Richtung des Flusses der Datenbestände ist bei beiden betrachteten Systemen jedoch gegensätzlich. Bei verteilten Datenbanksystemen liegen die Daten in der Gesamtheit vor und werden dann kontrolliert auf verschiedenen Servern verteilt bzw. repliziert gehalten. Metasysteme integrieren dagegen Daten zu einer Einheit.

Abbildung 2.1 zeigt ein Beispiel für ein Verteiltes Datenbanksystem. Hier wird ein Datenbankobjekt, zum Beispiel eine Relation, fragmentiert und auf drei Server verteilt.

Benutzt eine Digitale Bibliothek für die Realisierung ihres Archivs, also ihres Lokalen Dokumentenservers, eine verteilte Architektur wie zum Beispiel ein verteiltes Datenbanksystem, so nennt man solch ein System verteilte Digitale Biblio-

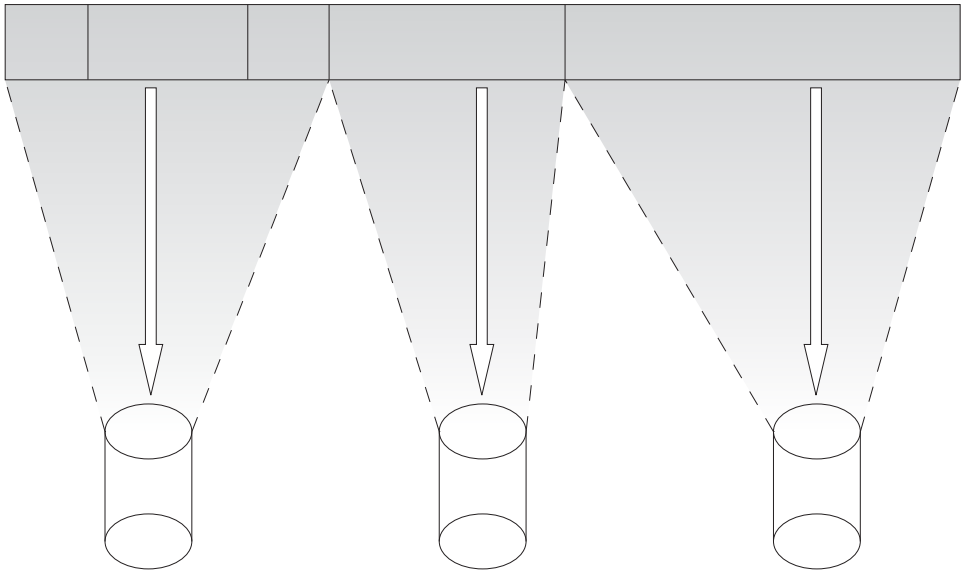

Abbildung 2.1: Fragmentierung eines logischen Datenbankobjektes in Verteilten Datenbanksystemen

thek. Solch eine Bibliothek zeichnet sich dadurch aus, dass ihr Archiv verteilt vorliegt. Es integriert jedoch nicht zwingend fremde Archive Digitaler Bibliotheken. Eine Suche auf solch einer Digitalen Bibliothek ist also eine Suche auf dem verteilt vorliegenden Archiv, jedoch keine verteilte Suche in mehreren Digitalen Bibliotheken.

Verteilte Datenbanksysteme zeichnen sich dadurch aus, dass die Verteilung nach außen hin nicht erkennbar ist (*Verteilungstransparenz*), das heißt, dass Benutzer, die Anfragen an das System stellen, von der Partitionierung nichts bemerken. Dieses Konzept ist in gewissem Maße auch für die Metasysteme wünschenswert. Eine Suche sollte für einen Benutzer beispielsweise Ergebnisse liefern, die in vielen verschiedenen Digitalen Bibliotheken vorhanden sind. Ein Hinweis, in welchem Archiv ein gefundenes Dokument zu finden ist, sollte aber der einzige sein, der auf eine entfernte Speicherung hindeutet. Dagegen sollten der Zugriff auf ein Dokument und die Bedienungslogik eines Benutzerinterfaces niemals abhängig von einem bestimmten digitalen Archiv sein.

Für verteilte Datenbanksysteme wurden Anforderungsregeln für eine Realisierungsform formuliert. Bekannt ist dabei insbesondere ein Anforderungskatalog mit 12 Regeln von Date [Dat90]. Diese werden auch in dem Lehrbuch von Saake und Heuer [SH99] genannt. Ausgehend von diesen Regeln möchte ich ein ähnliches Regelwerk für die betrachteten Digitalen Bibliothekssysteme ableiten.

Zuerst nenne ich noch einmal die 12 Anforderungsregeln nach Date. Sie sind aus [SH99] übernommen:

1. *Lokale Autonomie*
Die einzelnen Knotenrechner sollten ein Maximum an Kontrolle über die auf ihnen gespeicherten Daten behalten, so sollte der Datenzugriff nicht von anderen Rechnern abhängen.
2. *Unabhängigkeit von zentralen Systemfunktionen*
Zentrale Systemfunktionen und zentrale Datenstrukturen sind ein Flaschenhals in verteilten Systemen. Hohe Verfügbarkeit und Knotenautonomie sind neben dem Leistungsengpass weitere Gründe, die gegen zentrale Dienste sprechen.
3. *Hohe Verfügbarkeit*
Der Datenbankbetrieb sollte auch bei Rechnerausfällen und Rekonfiguration des Netzes ununterbrochen gewährleistet sein.
4. *Ortstransparenz*
Der Ort der Speicherung bestimmter Daten sollte vor dem Benutzer verborgen sein.
5. *Fragmentierungstransparenz*
Die interne Aufteilung von Datenbankobjekten (etwa die Partitionierung einer Relation) sollte vor dem Benutzer verborgen bleiben.
6. *Replikationstransparenz*
Die Replikation von Daten zur Leistungssteigerung und zur Erhöhung der Ausfallsicherheit betrifft die interne Realisierung und sollte daher im Sinne der Datenunabhängigkeit ebenfalls unsichtbar bleiben.
7. *Verteilte Anfragebearbeitung*
Die Anfragebearbeitung sollte verteilt erfolgen. Die Realisierung der verteilten Bearbeitung sollte durch einen Optimierer festgelegt und nicht dem Benutzer aufgebürdet werden.
8. *Verteilte Transaktionsverwaltung*
Auch im verteilten Szenario muss eine korrekte Transaktionsverarbeitung garantiert werden.
9. *Hardware-Unabhängigkeit*
Die Datenbankverarbeitung sollte auf unterschiedlicher Hardware möglich sein.
10. *Betriebssystemunabhängigkeit*
Die Datenbankverarbeitung soll auch auf unterschiedlichen Betriebssystemen möglich sein.
11. *Netzwerkunabhängigkeit*
Die Datenbankverarbeitung soll auch auf unterschiedlichen Netztechnologien möglich sein.
12. *Datenbanksystemunabhängigkeit*
Die Datenbankverarbeitung soll unter Nutzung unterschiedlicher Datenbank-Management-Systeme (DBMS) als Knoten-DBMS möglich sein.

Die genannten Regeln kann man leicht auf Digitale Bibliothekssysteme in einem Integrationsverbund übertragen:

Die *Lokale Autonomie* bewirkt, dass die einzelnen Digitalen Bibliotheken ein Maximum an Kontrolle über die auf ihnen gespeicherten Daten behalten. Der Datenzugriff hängt nicht von anderen Systemen im Verbund ab.

Die *Unabhängigkeit von zentralen Systemfunktionen* sind auch für Digitale Bibliotheksverbunde von Wichtigkeit. Zentrale Systemfunktionen und zentrale Datenstrukturen stören auch in Verbunden von Digitalen Bibliotheken als ein verteiltes System.

Die *Ortstransparenz* sollte etwas anders gestaltet werden. Der Ort der Speicherung bestimmter Daten sollte dem Benutzer nicht ganz transparent sein. Ein Hinweis, wo die entsprechenden Daten lokalisiert sind, wäre zweckmäßig. Die Art und Weise des Zugriffs soll aber vom Ort unabhängig sein.

Die *Replikationstransparenz* spielt in Digitalen Bibliotheken ebenfalls eine wichtige Rolle. Digitale Bibliotheken sollten Replikations- und Caching-Mechanismen haben, um Zugriffe gerade bei Wiederholungen performanter zu gestalten. Wenn ein Zugriff auf lokal replizierte Daten erfolgt, soll es dem Benutzer absolut transparent sein. Für ihn "kommt" ein Dokument von dem entsprechenden entfernten System.

Die anderen Anforderungen *Hohe Verfügbarkeit, Fragmentierungstransparenz, Verteilte Anfragebearbeitung, Verteilte Transaktionsverwaltung, Hardware-Unabhängigkeit, Betriebssystemunabhängigkeit, Netzwerkunabhängigkeit, Datenbanksystemunabhängigkeit* lassen sich analog übernehmen.

Die Anforderungen an einen verteilten Such- / Integrationsdienst in Digitalen Bibliotheken werden in Kapitel 4 diskutiert.

2.5 MILESS und MyCoRe

In diesem Abschnitt wird die Entstehung von MyCoRe und dafür vor allem sein Vorgängersystem MILESS beleuchtet.

Das Projekt MILESS (Multimedialer Lehr- und Lernserver Essen) wurde 1998 an der Universität Essen ins Leben gerufen. Die aus dem Projekt entstandene Plattform für das Dokumentenmanagement wurde bis heute ständig weiterentwickelt.

Diese Digitale Bibliothekslösung wurde ursprünglich auf die Anforderungen der Universität Essen zugeschnitten. Dadurch, dass aber von Anfang an für die Realisierung Standards benutzt wurden, war es möglich, das System auch für die Nutzung in einer anderen Umgebung zu flexibilisieren. So wurden beispielsweise für die Metadaten Dublin Core und für die Dokumentenrepräsentation XML als Standards gewählt.

Andere Universitäten bekundeten Interesse an dem System und nutzten es erfolgreich für eigene Projekte nach. So entstanden beispielsweise das System "Bach Digital" an der Universität Leipzig und das System "Urmel" an der Universität Jena. Heute gibt es ca. ein Dutzend Installationen von MILESS bzw. MILESS-Derivaten an verschiedenen Standorten. So gibt es zum Beispiel auch eine Installation an der Universität Rostock, die in Zusammenarbeit zwischen dem Lehrstuhl Datenbanken und Informationssyteme, dem Rechenzentrum und der Universitätsbibliothek aufgebaut wurde.

Das MILESS-System benutzt den IBM Content Manager und das Datenbankma-

nagementsystem DB2 für die Persistenzierung der zu archivierenden Dokumente. Für die Recherche wird die Text Search Engine des Enterprise Information Portals genutzt.

MILESS nutzt für seine Implementierung die Java-APIs des Content Managers und des Enterprise Information Portals. Als Schnittstelle zu dem Benutzer wurden Servlets und Applets implementiert. Der Java-Code umfasst mittlerweile ca. 45000 Quellcode.

Durch die immense Größe des Quellcodes, die MILESS angenommen hat, wurde eine langfristige Weiterentwicklung und Pflege problematisch. Eine erste Idee war es, MILESS von IBM übernehmen zu lassen. IBM sollte dann MILESS als eine Beispiel-Anwendung einer IBM Content Manager Distribution mitausliefern. Dieses Vorhaben ist aber aufgrund von rechtlichen Problemen gescheitert. Das neue Konzept war es dann nicht etwa, MILESS zu kommerzialisieren, sondern MILESS-Code als Open Source unter der GNU General Public License für jedermann nachnutzbar zu gestalten. Unterstützt wurde das von der "CampusSource Initiative Nordrhein-Westfalen". Das Ziel war die Errichtung einer Gemeinschaft zur Weiterentwicklung und Pflege. Es entstand die MILESS-Community in der ein Erfahrungsaustausch zwischen den Benutzern von MILESS möglich war.

Aus der MILESS-Community der Nachnutzer erwuchsen neue zusätzliche Anforderungen und Flexibilisierungswünsche für das System. Diese sind z.B. eine flexiblere Datenmodellierungsmöglichkeit der Metadaten, die über die Möglichkeiten von Dublin Core hinausgeht oder die verteilte Suche in anderen Digitalen Archiven. Aus diesen neuen Anforderungen entstand schließlich MyCoRe.

MyCoRe [4] steht für "**M**ILESS Communit**y** **Co**ntent **Re**pository". MyCoRe ist ein Open Source Projekt zur Entwicklung eines Systems für Digitale Bibliotheken und Archivlösungen (oder allgemeiner "Content Repositories" $>>$ CoRe). Im MyCoRe Projekt arbeitet eine Gruppe von Universitäten daran, für derartige Anwendungen einen gemeinsamen flexiblen Software-Kern ("core") zu erstellen, der sich an die eigenen Bedürfnisse anpassen und erweitern lässt (daher das "My" für die lokale Adaption). Das System wird so konzipiert, dass sich verschiedene Backends für die Archivierung der Dokumente und Metadaten anbinden lassen werden. Unter anderem wird auch wieder der IBM Content Manager anbindbar sein, für den schon erste Module für MyCoRe implementiert worden sind.

Im Folgenden sind die Projektbeteiligten von MyCoRe aufgelistet:

- Universität Essen: Zentrale Einrichtungen
- Universität Jena: Rechenzentrum, Bibliothek
- Universität Leipzig: Rechenzentrum
- Universität Münster: Rechenzentrum, Bibliothek
- Universität Halle: Rechenzentrum, Bibliothek

[4] www.MyCoRe.de

- Universität Freiburg: Rechenzentrum
- Universität Rostock: Bibliothek, Rechenzentrum, Lehrstuhl Datenbanken und Informationssysteme
- Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
- Universität Uppsala, Schweden mit Louisiana Tech University
- IBM Deutschland: Content Manager Gruppe, Forschung und Lehre

Im Folgenden sind weitere Interessenten an MILESS/MyCoRe aufgelistet, die zu weiteren Nutzern eventuell auch Entwicklern von MILESS/MyCoRe werden könnten:

- Universität Greifswald
- Universität Düsseldorf
- Universität Bochum
- BASF Ludwigshafen

Jeder Projektteilnehmer übernimmt einen Teil der Implementierungsarbeit für MyCoRe. Die Universität Rostock, speziell der Lehrstuhl Datenbanken und Informationssysteme, trägt zu dem Projekt mit der Entwicklung einer Basis für die verteilte Suche in MyCoRe bei. Diese Diplomarbeit und vor allem die in dieser Diplomarbeit erarbeitete Implementierung trägt einen Teil zur dieser Funktionalität von MyCoRe bei.

2.6 IBM Content Management

2.6.1 Allgemeines

Der Content Manager (kurz CM) und das Enterprise Information Portal (EIP) sind zwei Produkte des Content Management Portfolios aus der Data Management Produktgruppe von IBM. Diese Produkte richten sich an die Entwickler von Digitalen Bibliotheken[5]. Sie bieten eine grundlegende Architektur für eine Implementierung. Sie ist jedoch keine komplett fertige Lösung für Digitale Bibliotheken. Die Anwendung “On-Top” muss entsprechend den speziellen Anforderungen unter Benutzung der objektorientierten Programmierschnittstellen selbst implementiert werden.

Der Content Manager ist in der Lage, elektronische Dokumente wie Texte, Images, Audio- und Videodaten zusammen mit seinen Metadaten zu persistenzieren. Das System speichert Metadaten auf Library Servern und Dokumente auf eventuell mehreren Object Servern. Durch diese Architektur ist das System nahezu beliebig

[5]Frühere Versionen des Produktes hießen auch *Digital Library*.

skalierbar und an diverse Ansprüche anpassbar. Das System bietet eine parametrische Suche, die mit den Suchmöglichkeiten auf relationalen Datenbanken mittels SQL vergleichbar ist.

Das Enterprise Information Portal bestehend aus vielen Komponenten bietet nun zusätzlich erweiterte Suchmöglichkeiten wie z.B. eine Volltextsuche mit linguistischen Features. “Federated search” ist ebenfalls eine Komponente des Enterprise Information Portals. Durch sie wird es ermöglicht auf verschiedenen Backends wie z.B. relationalen Datenbanken, Lotus Domino Server, entfernten Dateisystemen oder natürlich auch Content Manager verteilt zu suchen.

Die aktuelle Version des Content Manager und des Enterprise Information Portals ist zur Zeit Version 7.1.3. IBM wird demnächst[6] die sich im Beta-Test befindliche Version 8.1 auf den Markt bringen. Der Lehrstuhl Datenbanken und Informationssysteme nimmt an dem Beta-Test teil.

Die Content Manager setzt auf eine relationale Datenbank auf. Derjenige, der Content Manager erwirbt, erhält zusätzlich eine Lizenz für die Nutzung von IBM DB2. Es ist aber auch möglich, den Content Manager zusammen mit einer Oracle-Datenbank zu installieren. Der Content Manager lässt sich direkt an ein Tivoli Storage Management System anbinden. Dieses System ermöglicht problemlos automatisierte und gut kontrollierbare Backups der archivierten Daten.

2.6.2 Architektur CM V8

An dieser Stelle möchte ich kurz die wesentlichen Punkte der Architektur des Content Managers vorstellen. IBM wird wesentliche Neuerungen in der Version 8 einbringen. Die Präsentationen zum Beta-Programm [cmv8] standen für die Zusammenstellung der hier beschriebenen Informationen zur Verfügung.

In Abbildung 2.2 ist die Architektur des Content Manager Version 8 dargestellt.

Die Architektur besteht aus 4 wesentlichen Grundkomponenten.

- Der **Library Server** bildet im Wesentlichen die Persistenzeinheit für die Metadaten einer Anwendung. Der Library Server basiert dabei auf einer DB2-Datenbank. In ihr werden zum Beispiel die ACL-Listen[7] zum Rechtemanagement im System gespeichert. Die gesamte vorgenommene Datenmodellierung der Klassen von zu speichernden Items wird im Library Server gehalten. Desweiteren beinhaltet er Informationen zur Versionierung von gespeicherten Items und deren Strukturierung. Auf die Datenmodellierung mit Content Manager 8 wird im weiteren Verlauf noch näher eingegangen. Neu in Version 8 ist, dass ein Großteil der Applikationslogik in den Library Server in Form von Stored Procedures gerutscht ist. Der Library Server

[6] Am 11. Juni 2002 gab es seitens IBM eine Vorabankündigung zum Content Manager 8.1. In der Veröffentlichung heißt es, dass eine Auslieferung als Electronic Software Delivery (also als Download) zum 30. September 2002 geplant ist.

[7] Access Control List, speichert Informationen über Zugriffsrechte zu gewissen Objekten.

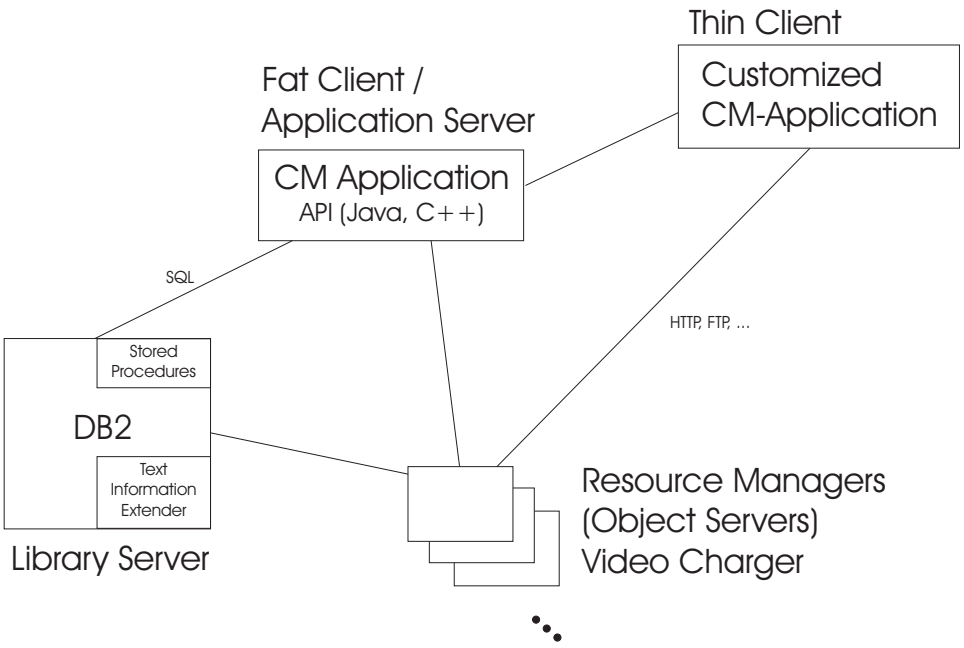

Abbildung 2.2: Die Architektur des Content Manager Version 8

nimmt Anfragen über die CM-API (Fat-Client) entgegen und gibt die Ergebnisse an diese zurück. Außerdem wird es jetzt auch möglich sein, direkt an die Metadaten im Library Server Volltextanfragen zu stellen. Das musste in früheren Versionen umständlicher realisiert werden. Der Vorteil dieser Architektur des Library Servers ist die volle Ausnutzung der Fähigkeiten der DB2-Datenbank inklusive ihres Transaktionsmanagements.

- Die **Resource Manager**, die früher Object Server hießen, speichern die Dokumente im Sinne der Definition einer Digitalen Bibliothek. Speziell für Videos unterstützt der CM das Video-Streaming-System von IBM, den IBM Video Charger, der ebenfalls mit Auslieferung des Content Managers in einer neuen Version 8 erscheinen wird. Der Resource Manager nutzt DB2 für die Persistenzierung. Der Resource Manager ist eine webbasierte Anwendung. Es wird also ein Web Application Server[8] gebraucht. Die Anwendungslogik wird in der Datenbank und in Java-Servlets gesteuert. Der Resource Manager besitzt also eine HTTP- bzw. HTTPS-Schnittstelle, über welche die CM APIs auf den Resource Manager kontaktieren können. Außerdem können die Thin Client Anwendungen über diese Schnittstelle die gespeicherten Objekte für eine Präsentation, etc. abrufen. Ein einziger Library Server kann mehrere

[8] Mit dem Content Manager Software Package wird der IBM WebSphere Application Server ausgeliefert.

Resource Manager benutzen und der Inhalt kann auf beliebigen von diesen gespeichert werden. Der Content Manager erlaubt das Speichern von mehreren Kopien von Objekten auf einem oder auf mehreren Resource Managern sowie das automatische Replizieren von Objekten bei Einfügen von Objekten.

- Der **Fat Client** oder der CM Application Server bietet dem Benutzer eine API für den Zugriff auf den Library Server und auf den oder die Resource Manager. Die wesentlichen Funktionen sind dabei die sogenannten *CRUD*-Services. Die 4 Buchstaben des Akronyms stehen dabei für *create*, *retrieve*, *update* und *delete*. CRUD sind also die Persistenzoperationen. Eine weitere wichtige Aufgabe ist die Entgegennahme von Anfragen. Die Anfrage ist eine Teilmenge von XQuery [BCF+02] und kann dadurch adäquat die neue hierarchische Struktur des CM V8 abfragen.

- Der **Thin Client** ist nun eine spezielle Anwendung, die die API des CM nutzt. Sie greift also auf den Fat Client und dadurch indirekt auf den Library Server sowie über HTTP, FTP, etc. auch auf die Inhalte der Resource Manager bzw. Video Charger zu.

2.6.3 Datenmodell und Anfragesprache

Das Datenmodell in CM V8 bietet wesentliche Novationen.
Zunächst sollen vier Begriffe aus der IBM-Begriffswelt erläutert werden:

- **Objects** sind die Dokumente im Sinne der Definition Digitaler Bibliotheken aus 2.1.1.

- **Items** halten die Daten, die ein oder mehrere Objects beschreiben und identifizieren. Es sind also die Metadatenrecords zu den Objects.

- **Item Types** beschreiben die spezifische Menge von Informationen, die gebraucht werden, um Objects zu beschreiben, zu identifizieren und zu lokalisieren. Es sind die Container für einzufügende Items. Die Item Types können, wie wir gleich sehen, hierarchisch gegliedert werden. Ein Item Type kann zum Beispiel Komponente eines anderen sein. Das Pendant zu den Item Types in Version 8 sind die Indexklassen in den früheren Versionen. Dort war jedoch keine hierarchische Gliederung möglich.

- Jedes Item Type hat eine Menge von **Attributes**. Name und Typ dieser Attribute werden dabei spezifisch für jedes Item Type festgelegt. Die Items haben eine konkrete Belegung der Attribute. Jedes Attribut hat einen gewissen Wert, das zugehörige Objects beschreibt.

Die wesentlichen Features bezüglich des Datenmodells sind:

- **Item Hierarchie:** Die Item Types können hierarchisch gegliedert werden. Die Gliederungsmöglichkeiten sind in Abbildung 2.3 grafisch dargestellt. Root Components können aus mehreren Child Components bestehen. Child Components selbst können wiederum aus mehreren Child Components bestehen. Dadurch entsteht die Hierarchie. Eine Component (Root oder Child Component) kann Root Components referenzieren. Eine Referenz wird dabei in einem Attribut gehalten. Es ist eine ID zu einer anderen Root Component. Es ist also eine Art Fremdschlüssel. Im Vergleich dazu gibt es die Links, die jedoch nur zwischen Root Components existieren können, und benutzt werden können, um Item Types miteinander in Beziehung setzen zu können. Spezielle Root Components sind Resource Items, die genau ein Resource Object, also z.B. Textdokumente, Bilder, Videos, haben.

- **Versioning:** Die Metadaten und die Dokumente bzw. Objects können versioniert im System vorliegen. Wenn die Daten modifiziert werden, dann entstehen solche neue Versionen. Der Systemverwalter kann die Art und Weise der Versionierung pro Item Type festlegen. Er legt dazu den Modus *always*, *sometimes* oder *never* sowie die maximale Anzahl an Versionen, die aufrechterhalten werden sollen, fest. *always* bedeutet, dass immer eine Version angelegt werden soll, *sometimes* verlangt die Steuerung über die Applikation, und *never* legt fest, dass keine Versionen entstehen sollen.

- **Attribute Groups:** Es lässt sich eine Menge von Attributen zu einer Attribute Group zusammenfassen. Dieses ist von besonderem Vorteil, wenn die gleichen Attribute immer wieder zusammen im gleichen Zusammenhang auftauchen. Zum Beispiel wäre es sinnvoll, die Attribute *Straße*, *Postleitzahl* und *Stadt* zu *Adresse* zu gruppieren.

- **Multi Valued Attributes:** Ein Attribut kann mehrere Werte haben. Es werden also mengenwertige Attribute unterstützt.

Die Anfragesprache für den Content Manager ist nun in der Lage, die beschriebene hierarchische Struktur auszunutzen. Sie basiert auf XQuery [BCF+02] und XPath [CD99]. Es ist eine Untermenge von XQuery. Für jedes Item gibt es eine XML Repräsentation, auf welche die Anfrage abzielt.

Die Anfragesprache erlaubt weiterhin das Verfolgen von Links und Referenzen, eine Volltextsuche auf den Attributen und Objects, eine Sortierung, die Recherche auf den Versionen und die Verwendung von Mengenoperatoren.

Ein Beispiel für eine Anfrage in der Content Manager Version 8 Query Language sei hier gegeben:

```
//Journal_Article[Journal_Author/@LastName ="Richardt"
AND contains-text(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

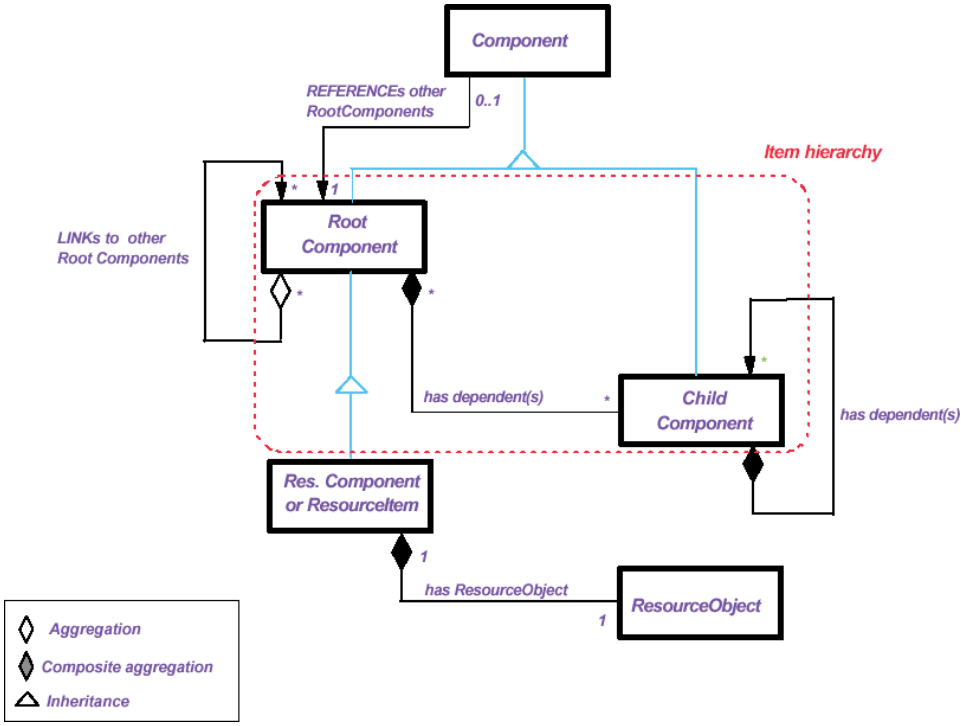

Abbildung 2.3: Das Content Manager Version 8 Meta-Modell

Die Anfrage sucht nach allen Artikeln aus Journalen mit dem Autor *Richardt*, die den Text *Java* und *XML* enthalten. Die Ergebnisse werden sortiert nach den Score-Werten des Text Search ausgegeben. Sie werden also nach ihrer Relevanz gerankt. Die Syntax und dessen Semantik sind dabei aus XQuery bzw. XPath übernommen. `/` steht für ein Kindkomponente, während `//` für eine Komponente in beliebiger Tiefe steht. Die eckigen Klammern `[]` stehen für eine Bedingung, die zu einer Komponente erklärt wird. Das `@`-Zeichen kennzeichnet Attribute.

Der neue Content Manager erlaubt also eine sehr flexible Datenmodellierung und eine adäquate Anfragesprache, die auf Standards aufbaut. Da die MyCoRe-Konzepte (siehe Kapitel 6.1.1) auf einer sehr ähnlichen Datenstrukturierung und Anfragesprache basieren, wird er einen wesentlichen Beitrag für die Umsetzung der MyCoRe-Plattform leisten. MyCoRe ist grundsätzlich so konzipiert, dass es mehrere Backends unterstützen kann. CM V8 wird jedoch das wichtigste MyCoRe-Backend.

Das System ist skalierbar. Die Verteilung auf mehrere Resource Manager kann netzweit erfolgen. Eine Möglichkeit, mehrere verteilte heterogene Datenquellen (CM oder andere) simultan abzufragen, bietet das ‘Federated Search’-Feature des EIP, welches in 5.4 näher beschrieben wird.

Kapitel 3

Ein verteilter Such- / Integrationsdienst

In diesem Kapitel wird eine Basisarchitektur, die für eine verteilte Suche notwendig ist, betrachtet. Um verteilt zu suchen bzw. fremde Digitale Bibliotheken für eine Suche zu integrieren, braucht eine Digitale Bibliothek einen speziellen Dienst, der in der Lage ist, solche Aufgaben zu übernehmen.

Zunächst wird dieser Dienst in die Gesamtarchitektur einer Digitalen Bibliothek nach [EF00] eingeordnet. Dabei spielen gerade die Randkomponenten und das Zusammenspiel mit ihnen eine wichtige Rolle. Die BlueView-Architektur mit dem Virtuellen Dokumenten Server ist ebenfalls eine Architektur für Digitale Bibliotheken und wird in diesem Kapitel vorgestellt. Abschließend werde ich einige allgemeine theoretische Betrachtungen zur Integration Digitaler Bibliotheken machen.

3.1 Architektur einer Digitalen Bibliothek nach [EF00]

Nach [EF00] bietet eine Digitale Bibliothek folgende Dienste:

Nutzer-Verwaltung und Zugangsdienste dienen dem Management von Zugangskontrollen und Profildaten der einzelnen Nutzer. Ein Nutzer authentifiziert sich über solch einen Dienst. Solch ein Dienst ist auch Voraussetzung für die Personalisierung einer Digitalen Bibliothek. Für jeden Nutzer werden Konten geführt, die den aktuellen Stand ihrer Verbindlichkeiten bzw. Forderungen widerspiegeln. Diese entstehen bei Einkauf eines kostenpflichtigen Zugriffs auf ein Dokument.

Suchdienste und Metadatenverwaltung dienen der Management der Metadaten und bieten eine Recherchemöglichkeit auf lokaler Ebene. Sie sollten zusätzlich eine von außen erreichbare Schnittstelle anbieten, um das Repository für verteilte Suchen von globalen Metasuchdiensten nutzen zu können.

Dokument-Verwaltung stellt die Speicherungseinrichtung der lokalen Dokument-Ressourcen dar. Es sollten Dokumentenmanagementsysteme oder MultimediaDatenbankmanagementsysteme zum Einsatz kommen, um deren Vorteile in Bezug auf Datensicherung, Integritätssicherungsmechanismen, Performance und Transaktionsverwaltung zu nutzen.

Bestelldienste dienen dem Management bei Zugriffen auf geschützte, zu bezahlende Dokumente. Ein Nutzer interessiert sich für ein Dokument und bestellt dieses über den Bestelldienst. Sind die Abrechnungen erfolgt, so wird ihm der Zugriff gewährt. Dadurch werden dann evtl. Lieferdienst und Rechnungs- oder Zahlungsdienst angestoßen.

Lieferdienste dienen der Realisierung des Zugriffs auf ein Dokument. Einem Benutzer wird also der Zugriff auf ein Dokument gewährt. Dieses wird für Rechnungs- / Zahlungsdienste protokolliert.

Rechnungsdienste können vor oder nach der Lieferung angestoßen werden. Wenn ein Dokumentzugriff kostenpflichtig ist, wird ein Inrechnungstellen veranlasst. Der Rechnungsbetrag wird ermittelt und dem Benutzer mitgeteilt. Die entstandenen Verbindlichkeiten werden außerdem in der Nutzer-Verwaltung vermerkt.

Zahlungsdienste realisieren das Inkasso der Digitalen Bibliothek. Der Nutzer gleicht durch Zahlungen wie z.B. über Kreditkarte, per Überweisung oder durch elektronische Zahlungsmittel seine Verbindlichkeiten aus, die durch den kostenpflichtigen Zugriff auf Dokumente entstanden sind.

Lieferanten-Verwaltung und Dokument-Beschaffung Die Lieferanten einer Digitalen Bibliothek sind zum Beispiel Verlage, Label oder Hochschulen. Sie stellen der Bibliothek die Dokumente also das intellektuelle Gut zur Verfügung. Eine Lieferung kann einmal (z.B. Monographien) oder auch fortlaufend (z.B. Zeitschriften) erfolgen. Die Verwaltung der Lieferanten und die Beschaffung der Dokumente muss von einer Anwendung aus steuerbar sein.

Alle Komponenten sind in Abbildung 3.1 dargestellt:

Die beschrieben Dienste realisieren die Anforderungen jedoch nur aus lokaler Sichtweise, also für eine autonome Digitale Bibliothek. Die Architektur muss also noch um eine Komponente erweitert werden, die eine Integration zur Recherche in anderen Digitalen Bibliotheken ermöglicht. Es sollten also zusätzlich Dienste angeboten werden, die einen entfernten Zugriff auf das Repository ermöglichen.

Dabei sind zwei Integrationswege zu betrachten. Zum einen kann sie Dienste anbieten, um von entfernten Klienten genutzt zu werden. Sie bietet dann also eine Exportmöglichkeit zum Beispiel über OAI oder Z39.50. Zum anderen kann sie

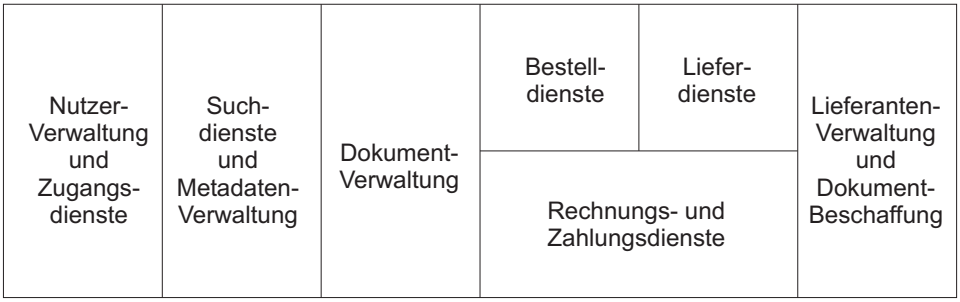

Abbildung 3.1: Dienste einer Digitalen Bibliothek nach [EF00]

selbst andere Digitale Bibliotheken integrieren. Sie hat dann also selbst einen verteilten Such- / Integrationsdienst.

Also kommen zwei Dienste hinzu:

Exportschnittstellen bieten einen entfernten Zugriff auf Inhalte des Repositories. Als Protokolle bieten sich sicher auf XML basierende Datenflüsse sehr gut an.

Verteilter Such- / Integrationsdienst realisiert einen Zugriff und eine Recherche auf entfernte Datenquellen. Diese können alle verschiedene Zugriffsmethoden besitzen. Bei Suchen / Recherchen müssen die einzelnen Anfrageergebnisse der heterogenen entfernten Datenquellen homogenisiert und konsolidiert werden. Die Anforderungen dieses Dienstes sind im nächsten Kapitel geschrieben und werden dort diskutiert.

Somit entsteht eine Gesamtarchitektur einer Digitalen Bibliothek, in der auch die Integrationsmöglichkeiten berücksichtigt sind, die in Abbildung 3.2 dargestellt ist:

3.2 Broker / Trader

Eine Digitale Bibliothek kann als Informationsvermittlungskomponenten Broker und / oder Trader nutzen.

Ein Broker hat Informationen über die integrierten entfernten Datenquellen, eventuell sogar sämtliche Metadaten. Der verteilte Such- / Integrationsdienst kann bei einer verteilten Anfrage direkt den Broker statt der entfernten Quelle befragen. Das kann Performanceverbesserungen mit sich ziehen. Der Broker kennt vielversprechende Quellen für eine Anfrage und kann diese in einem ersten Schritt selektieren. Im zweiten Schritt wird dann die Anfrage zu den ausgewählten entfernten Quellen delegiert.

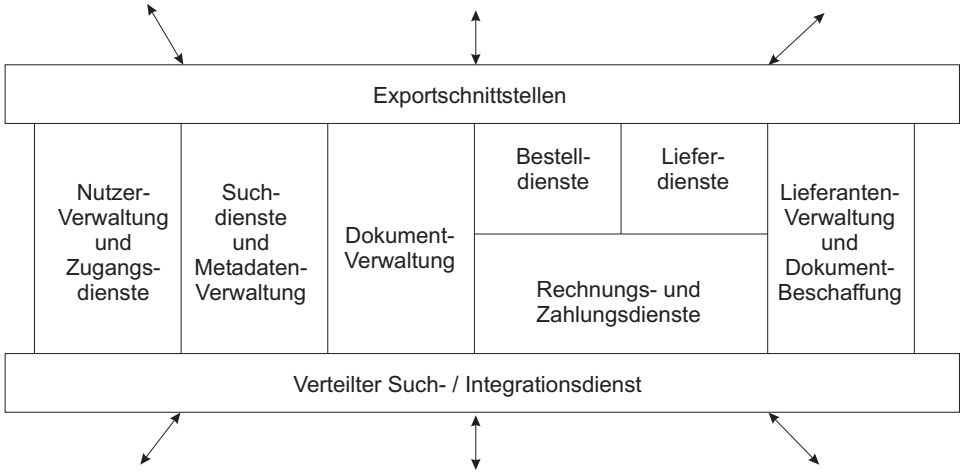

Abbildung 3.2: Eine Digitale Bibliothek mit Integrationsfähigkeit

Ein Trader verwaltet die Dienste eines verteilten Systems. Ihm werden neue Dienste mitgeteilt. Dazu gehören auch Informationen, wie ein Zugriff auf den Dienst zu erfolgen hat. Wenn ein Client Dienste in Anspruch nehmen möchte, befragt er den Trader, welche Dienste vorliegen und wie diese zu benutzen sind. Ihm werden dann die notwendigen Informationen mitgeteilt und er kann danach den Dienst in Anspruch nehmen. Das Trading ist ausführlicher in [Gri98] beschrieben. Durch Trading kann eine dynamische Anpassung eines verteilten Systems realisiert werden. Es können Dienste ausgetauscht und verändert werden und das System kann sich über den Trader rekonfigurieren.

3.3 Föderationsdienst

Der Föderationsdienst in einer Digitalen Bibliothek ermöglicht einen Zugriff auf die verschiedenen heterogenen Quellen. Jede entfernte Quelle bietet mitunter absolut verschiedene Zugriffsmethoden und Datenaufbereitungen.

Der Föderationsdienst kennt jede entfernte Quelle und ihre speziellen Charakteristika. Dazu gehören z.B. das Datenmodell und die Anfragesprache für ein Metadaten- oder Dokumentenretrieval. Die einzelnen entfernten Datenquellen werden zu einem föderierten System vereinheitlicht.

Ein verteilter Such- / Integrationsdienst kann direkt den Föderationsdienst abfragen. Dies geschieht in einer für jede entfernte Datenquelle homogenisierten Art und Weise. Dazu geschehen Transformationen auf die Anfragemethoden und Datenmodelle der heterogenen entfernten Quellen.

Diese Homogenisierung der entfernten lokalen Dokumentenbestände geschieht durch Wrapper. Sie sorgen für die Transformation auf homogene Datenmodelle

und Anfragesprachen.

3.4 VDS - LDS Architektur in BlueView

Im BlueRose / BlueView-Projekt [HMPT00] und der darauf aufbauenden Diplomarbeit von Patrick Titzler [Tit99] sind weitreichende grundlegende Arbeiten für die Architekturen von Digitalen Bibliotheken gemacht worden. Sie gelten als wesentliche Grundlage für meine Arbeit.

Die wichtigsten Dienste einer Digitalen Bibliothek sind nach BlueRose die folgenden:

Ein Anfrage- und Retrievaldienst realisiert die Recherche auf Metadaten oder Dokumenten der verteilt liegenden Datenquellen.

Ein Föderationsdienst ist in der Lage, verschiedene heterogene Systeme an das eigene System anzubinden, indem es verschiedene Datenformate und/oder Protokolle aufeinander abbildet.

Ein Profildienst hat die Aufgabe, sämtliche Profildaten eines Nutzers zu speichern. Dazu gehören beispielsweise Interessengebiete in Form von Schlagwörtern für den Benachrichtigungsdienst, Konfigurationseinstellungen für die Nutzung des Anfragedienstes oder auch fertige Anfragen oder Anfrageergebnisse (bookmarked queries).

Ein Benachrichtigungsdienst (Alerting) benachrichtigt den Nutzer, wenn es ein für ihn interessantes Thema gibt. Beispielsweise, wenn ein neues Dokument mit einem Stichwort eingegangen ist, welches in den Interessen des Nutzers auftaucht. Der Nutzer konfiguriert dieses Alerting über sein Profil mit Hilfe des Profildienstes.

Ein Broker- / Trader Dienst hat Kenntnis darüber welche entfernten Datenquellen existieren und wie sie ansprechbar sind. Er kann sogar die Metadaten von entfernten Systemen speichern.

VDS / LDS In BlueView/BlueRose [HMPT00] verbindet ein Virtueller Dokumentenserver (VDS) verschiedene heterogene Lokale Dokumentenserver (LDS) und integriert diese so zu einer Einheit. Dieser Dienst nimmt Anfragen vom Nutzer entgegen und verteilt sie auf die integrierten Systeme. Dabei werden je nach Integrationsart die Lokalen Dokumentenserver oder bei entfernten Systemen Föderationsdienst bzw. Broker/Trader angesprochen. Die einzelnen Rechercheergebnisse werden im VDS konsolidiert und an den Nutzer in entsprechender Form zurückgegeben. Der VDS ist ein verteilter Such- / Integrationsdienst.

Die Architektur des VDS und seine Einbettung in die gesamte BlueRose-Architektur wurden in [Tit99] entworfen.

Abbildung 3.3 zeigt den grundlegenden Aufbau einer Digitalen Bibliothek nach BlueRose.

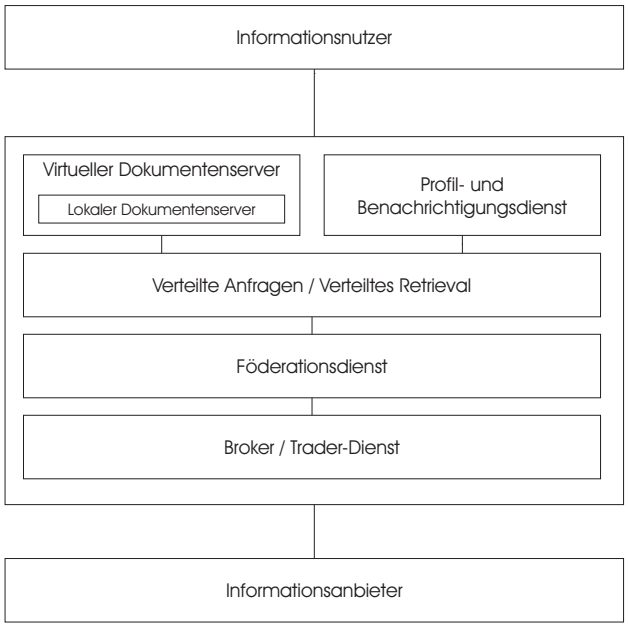

Abbildung 3.3: Aufbau einer Digitalen Bibliothek nach BlueRose

3.5 Ausprägungen des Dienstes

Die Realisierung eines verteilten Such- / Integrationsdienstes kann in 2 grundlegenden Ausprägungen geschehen.

3.5.1 Replizierung

Zum einen kann eine Replizierung von Metadaten und Dokumenten auf eine lokale Instanz erfolgen. Die Recherche wird dann auf diesen replizierten Daten realisiert. Ein Beispiel dafür war die Vorgehensweise in einem *Komplexe Softwaresysteme*-Projekt von Zeitz, Zarick, Zimmermann, Romberg und Below [ZZZ+01], bei dem die Metadaten eines entfernten Repositories (Springer) in einer lokalen Datenbank vorgehalten wurden und dem Suchdienst zur Verfügung standen.

Das Problem bei dieser Vorgehensweise ist, dass ein Speicherbedarf für die Metadaten entsteht, die eigentlich ja auf dem entfernten System zu finden sind. Es

entsteht also eine Redundanz. Damit kommt auch gleich ein weiteres Problem. Wie kann die Gültigkeit der replizierten redundanten Daten sichergestellt werden? Die Metadaten können sich auf dem entfernten System verändern, sie können eine neue Version bekommen. Die Zugriffsrechte auf die Metadaten oder auch auf die zugeordneten Dokumente können sich ändern. Sie können verschärft oder abgeschwächt werden. Ist der Zugriff auf die Metadaten bei einem entfernten Repository plötzlich kostenpflichtig oder ganz verboten, könnte trotzdem noch auf die entsprechenden replizierten Daten zugegriffen werden. Das könnte dann aber eventuell zu einem Urheberrechtsverstoß führen.

Bei der Vorgehensweise der Replizierung bedarf es also einer Konsistenthaltungskomponente. Alerting-Dienste des entfernten Repositories, die darauf hinweisen, wenn sich im Datenbestand etwas geändert hat, bieten einen Ansatzpunkt, um die Nachteile der replizierten Datenhaltung zu lindern.

Der entscheidende Vorteil der Replizierung von Metadaten und Dokumenten ist jedoch ein Performance-Gewinn bei Anfragen, die jetzt nicht mehr an entfernte Systeme delegiert werden müssen. Der erhöhte Kommunikationsaufwand entfällt, bzw. verschiebt sich jetzt in eine Replikationskomponente, die für die Aktualisierung der replizierten Daten sorgt.

Ein weiterer Vorteil der Replizierung ist, dass eine Homogenisierung der entfernten Daten direkt in der Replikationskomponente geschehen kann. Man hält in der Regel ein lokal gültiges Datenmodell, auf das die entfernten zu replizierenden Daten abgebildet werden. Durch die Replizierung entsteht so eine Datenbasis, auf der vollkommen homogen recherchiert werden kann.

3.5.2 “On-the-Fly”-Verarbeitungen

Bei Verarbeitungen “On-the-Fly” werden die Anfragen an die entfernten Repositories direkt weiterdelegiert. Es stellt also eine echte verteilte Suche dar. Die Transformationen zwischen Anfragesprachen und Datenmodellen werden “On-the-Fly” vom Föderationsdienst und seinen Wrappern bewältigt.

Jedes einzelne Repository liefert auf die Anfrage ein Resultat in einem bestimmten Darstellungsformat. Diese Ergebnisse müssen an eine globale Ergebnisaufbereitungskomponente des verteilten Such-/ Integrationsdienstes weitergeleitet werden und ebenfalls sofort verarbeitet werden.

Der verteilte Such-/ Integrationsdienst sichert dann die einheitliche Ergebnisdarstellung. Er kümmert sich außerdem um die Verwaltung der einzelnen Resultatlisten. Ein Resultat als Ergebnis einer Suche lässt sich als ein Resultat-Handle abstrahieren, über das ein Zugriff auf eine Resultatliste gesteuert werden kann. Der Zugriff kann durch diese Vorgehensweise angepasst erfolgen, was bei der Verarbeitung von sehr großen Ergebnismengen von Vorteil ist.

Der Nachteil der On-the-Fly Verarbeitung ist natürlich der erhöhte Aufwand während des Absetzens einer verteilten Anfrage. Der verteilte Such- / Integrationsdienst hat wesentlich mehr Aufgaben als bei einer Replizierung, wie zum Beispiel das Ergebnismischen, adäquate Ergebnisaufbereitung und eine kompliziertere

Steuerung von Ergebnislisten.

Der Vorteil ist natürlich, dass die Ergebnisse stets aktuell sind und keinerlei Konsistenzprobleme auftauchen.

Beide beschriebenen Ausprägungen eines verteilten Such- / Integrationsdienstes, also die Replizierung und die “On-the-Fly”-Verarbeitung schließen sich gegenseitig nicht aus. Es ist durchaus denkbar, dass man beide Vorgehensweisen kombiniert.

3.6 Theoretische Betrachtungen zur Integration

In diesem Abschnitt werde ich einige theoretische Überlegungen zu Verbunden von Digitalen Bibliothekssystem darlegen.

Ich untersuche hier ein Digitales Bibliothekssystem, welches andere Digitale Bibliothekssysteme integriert und so eine verteilte Suche auf der gesamten Einheit ermöglicht. Ein System, was in der Lage ist auf andere zuzugreifen, werde ich von diesem Punkt an *metarisiertes Digitales Bibliothekssystem* oder kurz *Metasystem* nennen.

Innerhalb eines Verbundes von Digitalen Bibliothekssystemen kann es mehrere solche Metasysteme geben, das heißt, das eine System integriert ein anderes und umgekehrt. Diese Integration lässt sich formal als Integrationskorrespondenz $I \subseteq S \times S$ ausdrücken. S steht dabei für eine Menge von Digitalen Bibliothekssystemen.

Mit Hilfe dieser Korrespondenz lässt sich nun ein Integrationsgraph für einen Verbund Digitaler Bibliothekssysteme aufstellen: Die Knoten des Graphen entsprechen den Digitalen Bibliothekssystemen. Gerichtete Kanten bedeuteten eine Integration eines digitalen Archivs in eine bestimmte Richtung. Sie repräsentieren die Integrationskorrespondenz. Knoten, von denen Kanten abgehen, integrieren andere Systeme, sind also Metasysteme. Eine verteilte Suche an ein System, das selbst in der Lage ist, andere Systeme zu durchsuchen, also ein Metasystem ist, pflanzt sich immer anhand des Integrationsgraphen fort.

Natürlich ist es wünschenswert, den Integrationsgraphen, der dabei entsteht, sehr einfach zu halten, um Performance und Konsistenz zu wahren. Es gilt zu vermeiden, eine verteilte Suche zu einer verketteten Suche entarten zu lassen. Ein System sollte also möglichst immer nur die Daten eines anderen integrieren, die wirklich in dem Archiv des zu integrierenden Systems vorhanden sind. Eine mehrstufige indirekte Integration ist nicht wünschenswert.

In der Abbildung 3.4 sind 3 Integrationsgraphen dargestellt. Alle 3 Graphen veranschaulichen einen Verbund aus jeweils 8 Digitalen Bibliothekssystemen.

a) zeigt einen schlechten, zu vermeidenden Integrationsgraphen für einen Verbund von Digitalen Bibliothekssystemen. Wird eine Suche auf dem Metasystem F abgesetzt, so werden beispielsweise auch alle Daten auf B mitdurchsucht. Das geschieht allerdings auf indirektem Wege über E und H. Es wäre aus Sicht von F erstrebenswerter, die Daten von B direkt zu integrieren.

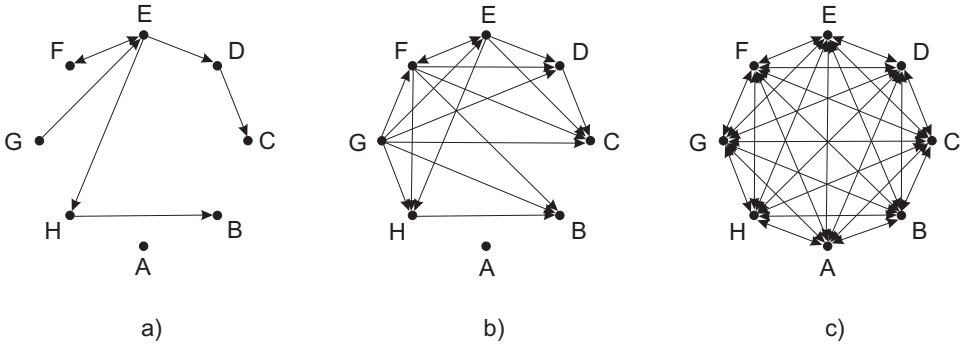

Abbildung 3.4: Integrationsgraphen von Verbunden Digitaler Bibliothekssysteme

Die Menge der Systeme ist $S = \{A, B, C, D, E, F, G, H\}$. Die Integrationskorrespondenz dieses Verbundes a) ist $I = \{(D, C), (E, D), (E, F), (E, H), (F, E), (G, E), (H, B)\}$.

Analysiert man den Wissenspool für jedes einzelne System, kommt man zu folgender Erkenntnis: Jedes Archiv kennt natürlich trivialerweise seinen eigenen Inhalt. A, B und C kennen nur ihre eigenen Inhalte. D kennt neben seinem eigenen Inhalt auch den von C. E kennt zusätzlich den Inhalt von D und indirekt auch den von C. Das ganze lässt sich für jedes System ableiten. Man erhält dann folgende *funktionale Integrationsabhängigkeiten* zwischen den einzelnen Systemen:
$A \rightarrow A$
$B \rightarrow B$
$C \rightarrow C$
$D \rightarrow C, D$
$E \rightarrow B, C, D, E, F, H$
$F \rightarrow B, C, D, E, F, H$
$G \rightarrow B, C, D, E, F, G, H$
$H \rightarrow B, H$

$\rightarrow$ ist eine binäre Relation auf S. Diese funktionalen Integrationsabhängigkeiten lassen sich einfach aus der Integrationskorrespondenz bilden.
DEFINITION: S sei eine Menge von Digitalen Bibliothekssystemen, $I \subseteq S \times S$ sei die Integrationskorrespondenz der Systeme.
$\mathcal{D} \subseteq S \times S$ ist die Menge der *funktionalen Integrationsabhängigkeiten*, wenn gilt:
$X \rightarrow Y \in \mathcal{D} :\Leftrightarrow$
$X = Y \vee$
$(X, Y) \in I \vee$
$\exists n \geq 1 \exists Z_1, \ldots, Z_n : \{(X, Z_1), (Z_1, Z_2), \ldots, (Z_{n-1}, Z_n), (Z_n, Y)\} \subseteq I;$
$X, Y, Z_1, \ldots, Z_n \in S, n \in \mathbb{N}$

□

$\mathcal{D}$ ist per Definition reflexiv und transitiv. Es ist die transitive Hülle von $\{(X, X) : X \in S\} \cup I$.

Auf b) findet man einen besseren Verbund von diesen Digitalen Bibliothekssystemen. Der Integrationsgraph hat sich geändert. Die funktionalen Integrationsabhängigkeiten sind jedoch unverändert geblieben. Der entstandene Verbund ist in Performance verbessert worden. Eine Suche auf dem System F kann jetzt direkt die Archive B, C, D, E und H ansprechen und muss nicht mehr über die indirekten Wege gehen, wie es in dem ersten Verbund beispielsweise für das Suchen von Dokumenten in B war.

Den optimierten Graphen von b) kann man mit einem einfachen Algorithmus leicht bestimmen.

Der optimale Verbund ist ein Verbund von Systemen, in dem jedes System jedes kennt und auch integriert. Dieses ist in c) dargestellt. Für c) ist $I = S \times S \setminus \{(X, X) : X \in S\}$.

Haben die Integrationsgraphen Zyklen oder gibt es mehrere Wege von einem Knoten zu einem anderen, so haben die Systeme darauf entsprechend zu reagieren. Solche Situationen müssen von Ihnen erkannt werden. So kann es verhindert werden, dass keine Suchen unendlich im Kreis laufen bzw. gewisse Ergebnisse mehrmals gefunden werden. Wenn man jedem Element der Suchdomäne von Digitalen Bibliothekssystemen, also jedem Dokument mit seinen Metadaten, eindeutig eine Identifikation (globale ID) zuordnet, ist es leicht möglich, adäquat in oben beschriebenen Situationen reagieren zu können. Dieses wäre auch vorteilhaft, um Doubletten auf verschiedenen Datenquellen erkennen zu können (siehe auch Abschnitt 4.4). Außerdem sollte natürlich auch jedes Archiv eines Verbundes eindeutig identifizierbar sein.

Die in diesem Abschnitt erarbeiteten Formalismen, vor allem der des Integrationsgraphen, sollen Entwicklern und Administratoren von Digitalen Bibliotheken bei der Planung und der Implementierung von Metasystemen zu Hilfe stehen. Sie eignen sich ferner dazu, bestehende verteilte Szenarios zu optimieren.

Kapitel 4

Anforderungen an den verteilten Suchdienst

4.1 Überblick

In diesem Kapitel werden Anforderungen an den verteilten Suchdienst/ Integrationsdienst formuliert. Die Realisierungsmöglichkeiten und die Probleme für eine Realisierung einiger dieser Anforderungen werden anschließend abschnittsweise diskutiert.

Meta-Suchmaschinen sind nach [EF00] Suchwerkzeuge, die nicht wie herkömmliche Suchmaschinen nur suchen können, sondern in der Lage sind, die Anfragen zu analysieren und geeignete Suchalgorithmen auf geeigneten Suchmaschinen anzustoßen. Sander-Beuermann [SB98], der Erfinder der deutschen Meta-Suchmaschine *MetaGer*, hat Kriterien für die Beurteilung von Meta-Suchmaschinen aufgestellt. Aus diesen Kriterien lassen sich auch Anforderungen an metarisierte Digitale Bibliothekssysteme ableiten.

Im Kapitel 2 wurden schon Anforderungen an Digitale Bibliotheken mit Integrationsfähigkeit genannt, die aus dem Vergleich mit verteilten Datenbanken heraus formuliert wurden (siehe 2.4).

Hier sind die Anforderungen im Überblick genannt. Auf einige wichtige Anforderungen wird in den anschließenden Abschnitten detaillierter eingegangen.

4.1.1 Parallele Suche

Ein Metasystem soll in der Lage sein, mehrere Recherchen auf den integrierten Digitalen Bibliothekssystemen parallel zu starten und die Ergebnisse dieser Recherchen auszuwerten. Nur so kann gewährleistet werden, dass das System für den Nutzer akzeptable Antwortzeiten hat. Diese Anforderung wird detaillierter in Abschnitt 4.2 diskutiert.

4.1.2 Ergebnismischung

Die Ergebnisse der einzelnen Recherchen sollen zu einem globalen Endergebnis zusammengeführt werden. Dieses soll durchweg ein einheitliches homogenes Format haben. Dazu müssen verschieden geformte und ausgeprägte Ergebnislisten bzw. Ergebnisobjekte in ein anderes Format überführt werden. Diese Anforderung wird detaillierter in Abschnitt 4.3 diskutiert.

4.1.3 Doublettenerkennung

Wenn ein Dokument mit seinen Metadaten in mehreren Repositories zu finden ist, so nennt man diese *Doublette* bzw. *Duplikat*.

Die Präsentation von doppelten Anfrageergebnissen ist für den Benutzer ein Ärgernis. Sie vergrößern das Suchergebnis unnötig und liefern dem Benutzer dabei keinen Nutzen, sondern erschweren seine Arbeit nur. Eine automatische Erkennung gleicher Objekte in den Ergebnislisten wäre wünschenswert und ist anzustreben. Die Umsetzung dieser Anforderung ist eine große Herausforderung. Die Doublettenerkennung wird detaillierter in Abschnitt 4.4 diskutiert.

4.1.4 Kein Informationsverlust

Jedes Detail eines Ergebnisses, welches von einem einzelnen integrierten Digitalen Bibliothekssystem geliefert worden wäre, soll in dem globalen Ergebnis nicht verloren gehen.

Ein Nutzer, der das Gefühl hat, dass ihm einige Details vorenthalten werden, ist in der Regel mehr als unzufrieden. Deshalb darf das Metasystem also keine Details unterschlagen. Diese Anforderung, die auf den ersten Blick als selbstverständlich gilt, ist jedoch nicht absolut trivial umzusetzen.

Diese Anforderung umzusetzen heißt, dass das Metasystem in der Lage sein muss, genauso auf das ausgefallenste wie auf ein typisches Datenmodell zu reagieren, und die Ergebnislisten zu verarbeiten. Es muss also sehr flexibel sein und das globale Datenmodell sogar bei Bedarf dynamisch anpassen können.

4.1.5 Kapselung der einzelnen Digitalen Bibliothekssysteme

Diese Anforderung fordert, dass der Nutzer die konkrete Handhabung der einzelnen integrierten Bibliothekssysteme nicht zu beherrschen braucht. Er soll seine Rechercheanfragen an das Metasystem stellen, das eine einheitliche Oberfläche haben soll.

Diese Anforderung bringt eine verbesserte Qualität des Metasystems. Der Benutzer würde sich weigern, verschiedene Oberflächen für die Suchergebnisse von verschiedenen Datenquellen zu nutzen.

4.1.6 Vollständige Suche

Die verteilte Suche soll nicht eher beendet werden, bis das letzte integrierte System seine Teilergebnisse an das Metasystem abgeliefert hat.

Hier geht es wieder darum, dass dem Benutzer nichts unterschlagen werden darf. Das gilt sowohl für die Details zu einzelnen Suchergebnissen wie für die gesamte Liste von Suchergebnissen.

Schwierig wird diese Anforderung natürlich umzusetzen zu sein, wenn ein System nicht innerhalb einer akzeptablen Zeit reagiert. Bei Metasystemen, die ihre Funktionalität über Replizierung umsetzen, wird es dieses Problem nicht geben. Die Daten liegen dann ja als Replikat bei dem Metasystem.

4.1.7 Adäquate Recherchemöglichkeiten

Ein Metasystem soll adäquate Möglichkeiten anbieten, um zu recherchieren. Es sollen folgende Anfragearten unterstützt werden:

- parametrische Anfragen auf den Metadaten,
- untypisierte Anfragen auf den Metadaten,
- liegen die Metadaten in einer wohldefinierten Strukturierung vor, so muss eine Anfrage auch gegen diese Struktur formulierbar sein,
- Metadaten und auch Dokumente können evtl. versioniert vorliegen, diese Versionierung muss ebenfalls recherchierbar sein,
- inhaltsbasierte Anfragen auf Dokumente, sofern möglich, (z.B. Volltextanfragen auf Textdokumente oder Anfragen auf den Inhalt von Bilddokumenten)
- hybride Anfragen auf Metadaten und zugehörige Dokumente.

Diese strukturierten und inhaltsbasierten Anfragearten sollen sich durch Boolesche Operatoren verknüpfen lassen. Für die Formulierung der Rechercheanfrage sollen Boolesche Operatoren, wie `NOT`, `AND`, `OR` zur Verfügung stehen.

4.1.8 Globales Ranking

Da bei Anfragen auf sehr große Datenbasen sehr viele Ergebnisse zurückgeliefert werden können, ist es notwendig, die erhaltenen Ergebnisse zu bewerten, das heißt, dem Ergebnis einen Wert zuzuordnen, wie gut es auf die Anfrage passt. Diese Bewertung nennt man *Ranking* oder *Rangieren*. Die gefundenen Dokumente sollen in der Reihenfolge ihrer Relevanz ausgegeben werden. Diese wichtige Anforderung wird in Abschnitt 4.5 näher diskutiert.

4.1.9 Sicherheit bei der Kommunikation

Wenn in verteilten Anwendungen auf entfernte Ressourcen zugegriffen wird, so dürfen bei der notwendigen Kommunikation und beim Datenaustausch über ein Netzwerk keine sensiblen Daten durch Dritte “mitgehört” werden können. Solche sensiblen Daten sind zum Beispiel User-IDs und Passwörter, wie sie bei einer Anmeldung an einem System oftmals über das Netz übertragen werden. Andere sensible Daten sind die Metadaten und die Dokumente selbst, wenn sie Urheberrechtsschutzen zugrundeliegen.

Es ist die Nutzung von verschlüsselten Kommunikationsmethoden anzustreben. Die Secure Socket Layer Methode ist eine solche Methode. Die kryptographischen Überlegungen, die von den Entwicklern der Methode angestellt worden sind, garantieren höchste Datensicherheit und nahezu unmögliche Entschlüsselung ohne den Besitz eines geheimen privaten Schlüssels.

4.1.10 Kontrolle und Schutz von Urheberrechten

Die Inhalte der integrierten Digitalen Bibliothekssysteme können urheberrechtlich geschützt sein. Verstöße gegen Urheberrechte können strafrechtliche Konsequenzen nach sich ziehen. Daher ist es eine wichtige Aufgabe jedes einzelnen Digitalen Bibliothekssystems aus einem Verbund, stets zu überprüfen, ob der Nutzer das Recht hat, auf ein bestimmtes Dokument zuzugreifen. Die Überprüfung muss auch sichergestellt werden, wenn der Zugriff über ein Metasystem erfolgt, das das entsprechende Digitale Archiv integriert.

Es gibt Dokumente, für die ein Zugriff von ihrem Anbieter kostenpflichtig angeboten wird. Es ist Aufgabe der Digitalen Bibliothekssysteme, von einem Nutzer erkaufte Lizenzen für einen Zugriff auf solche Dokumente zu verwalten. Jedes System verwaltet die Zugriffsrechte und überprüft bei einem Zugriffswunsch auf ein bestimmtes geschütztes Dokument, ob dieser Zugriff erlaubt ist. Auch ein Metasystem soll die Zugriffsrechte eines Nutzers für entfernte, auf fremden integrierten Digitalen Bibliothekssystemen vorliegende Dokumente kennen.

Um diese Anforderung zu realisieren, muss das Metasystem mit den Lieferdiensten, Rechnungsdiensten und Zahlungsdiensten der integrierten Systeme zusammenarbeiten.

4.1.11 Personalisierung

Aus der letzten Anforderung erwächst sofort diese nächste. Ein Digitales Bibliothekssystem muss für jeden individuellen Nutzer personalisiert werden können. Ein Nutzer muss sich am System autorisieren, wenn er erweiterte Privilegien nutzen möchte. Benutzer, die sich nicht am System autorisieren, werden als anonyme Nutzer mit minimalen Rechten behandelt.

Für jeden registrierten Benutzer wird ein persönliches Profil archiviert. Das Profil beinhaltet Zugangsberechtigungen und Autorisierungsinformationen zu entfernten Systemen, persönliche Konfigurationen des Nutzers sowie die einzelnen vom

Nutzer erworbenen Lizenzen für den Zugriff auf urheberrechtlich geschützte Dokumente. Außerdem enthält es die vom Benutzer ausgewählten Interessengebiete für einen Alerting-Dienst.

[Zei01] beschreibt Konzepte für eine Personalisierung Digitaler Bibliotheken tiefgehend.

Ein Beispiel für ein Profil, das im XML-Format beschrieben ist, sei hiermit gegeben. Es soll demonstrieren, welche Informationen in einem Metasystem zu jedem Benutzer sinnvollerweise gespeichert werden könnten bzw. sollten:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Profiles SYSTEM "Profile.dtd">
<Profiles>
  <Profile>
    <User>
      <Username>valikovacs</Username>
      <Fullname>Valéria Kovács</Fullname>
      <Email>valikovacs@gmx.net</Email>
    </User>
    <Interests>
      <Interest NotificationPolicy="weekly"
                Type="title">
        Friedrich Nietzsche
      </Interest>
      <Interest NotificationPolicy="daily"
                Type="abstracttitle">
        german literature
      </Interest>
    </Interests>
    <Property>
      <ResultProperty>
        <MaxResultTuples>25</MaxResultTuples>
      </ResultProperty>
    </Property>
    <Rights>
       <AccessInformation>
         <Repository Alias="MyCoRe Freiburg">
           <ID>valikovacs</ID>
           <Password>digBibRulez</Password>
         </Repository>
         <Repository Alias="Digbib Hong Kong">
           <ID>valikovacs</ID>
           <Password>iLoveHongkong</Password>
         </Repository>
       </AccessInformation>
       <BoughtItems>
         <Repository Alias="Digbib Hong Kong">
           <ItemID>985xy32</ItemID>
         </Repository>
       </BoughtItems>
```

```
    </Rights>
  </Profile>
</Profiles>
```

Es kann natürlich weitaus mehr Informationen geben, die ein Metasystem zu jedem Benutzer speichert.

4.1.12 Rechtemanagement

Jeder Nutzer hat am Metasystem gewisse Privilegien. Es gibt beispielsweise gewöhnliche Nutzer, die “nur” suchen dürfen, und solche, die auch neue Dokumente einstellen oder bestehende ändern bzw. versionieren können.

Die Digitale Bibliothek hat eine eigene Komponente für das Rechtemanagement, mit der der verteilte Such- / Integrationsdienst zusammenarbeitet.

Für Privilegien für Dokumente auf entfernten integrierten Repositories muss das Rechtemanagement mit denen der entfernten Repositories synchronisiert werden. Dazu wird auf das Profil eines Nutzers zugegriffen, das diese Informationen bezüglich Rechten und Zugang enthält.

4.1.13 Lokale Autonomie

Die einzelnen digitalen Archive eines Integrationsverbundes sollten die gesamte Kontrolle über die auf ihnen gespeicherten Dokument- und Metadaten behalten. Der Datenzugriff darf nicht von anderen Archiven abhängen.

Jedes einzelne Bibliothekssystem ist nicht von einem zentralen Metasystem abhängig. Das heißt, alle Aufgaben kann jedes Archiv auch bei einem eventuellen Ausfall des Metasystems alleine bewältigen.

4.1.14 Hohe Verfügbarkeit

Der Betrieb des verteilten Systems soll auch bei Rechnerausfällen und Rekonfiguration des Netzes ununterbrochen gewährleistet sein. Dieses kann z.B. durch eine automatische Zuschaltung eines Mirror-Servers bei Ausfall eines Rechners erfolgen. Fällt das Metasystem komplett aus, so kann es natürlich keine Anfragen beantworten. Fallen entfernte Systeme aus, so soll eine verteilte Suche zumindest auf den erreichbaren Systemen möglich sein.

4.2 Parallele Suche

Durch Threadprogrammierung lässt sich eine Parallelität realisieren.

4.2.1 Probleme

Ein typisches Problem bei einer Parallelverarbeitung ist die Synchronisation. Wie kann man sicherstellen, dass der Zugriff von mehreren parallelen Prozessen oder

Threads auf ein und dasselbe Objekt so erfolgt, so dass das erwartete Ergebnis nicht verfälscht wird? Das Objekt, auf das von mehreren parallel laufenden Threads in diesem Anwendungsszenario zugegriffen wird, ist in der Regel ein Anfrageergebnisobjekt. In ihm werden die einzelnen Ergebnisse der Anfragen gesammelt.

4.2.2 Realisierungsmöglichkeiten/Problemlösungen

Eine Parallelverarbeitung ist heute in nahezu allen modernen Programmiersprachen möglich. Diese bieten in der Regel auch Möglichkeiten der Synchronisationskontrolle. So gibt es in objektorientierten Programmiersprachen Sprachkonstrukte, die bewirken, dass ein Teil eines Programmes isoliert abläuft. Wenn dieser Teil des Programmes bearbeitet wird, so darf kein anderer Programmteil parallel dazu arbeiten. Diese Prozesse sind dann blockiert. Dieses wird durch das Setzen und Aufheben von Sperren realisiert. So kann sichergestellt werden, dass nur ein Prozess bzw. Thread bzw. Programmteil auf auf Parallelzugriff sensible Objekte bzw. Datenstrukturen zugreifen kann.

Java bietet mit seiner Threading-Technologie genau jene Parallelverarbeitungsmöglichkeit. Threads sind quasi parallel ablaufende Steuerflüsse innerhalb eines Programms. Das “quasi” gilt insbesondere für Rechner mit nur einem einzigen Prozessor. Echte Parallelität ist dort nicht möglich. In solchen Fällen simuliert ein Scheduler die Parallelität, indem er den verschiedenen parallelen Threads in gewissen Zeitabständen die Rechenleistung des Prozessors zuteilt und gegebenenfalls wieder entzieht.

Innerhalb einer Java Virtual Machine können also mehrere Threads parallel zueinander ablaufen. Für jedes Objekt, welches einen kritischen Bereich enthält (gekennzeichnet durch das Schlüsselwort `synchronized`), legt Java zur Laufzeit eine Kontrollliste an. Betritt nun ein Thread während seiner Abarbeitung solch einen kritischen Bereich, so hat kein anderer Thread mehr Zugriff auf das Objekt. Er wird beim Versuch des Zugriffs vielmehr blockiert, bis der frühere Thread den kritischen Bereich wieder freigibt.

Diese synchronisierte Threading-Technik habe ich in meiner Implementierung für MyCoRe eingesetzt. Die einzelnen entfernten MyCoRe-Instanzen werden bei einer verteilten Suche jeweils in einem autarken Thread bearbeitet. Die Ergebnisse werden in einem Result-Objekt gemischt. Das Eintragen in dieses Objekt geschieht synchronisiert.

4.3 Ergebnismischung

4.3.1 Probleme

Das Problem für diese Anforderung ist, dass sich die Ergebnisdarstellungen in verschiedenen Digitalen Bibliothekssystemen maßgeblich unterscheiden können. Beispielsweise kann ein System eine relativ unstrukturierte Liste von Ergebnissen liefern, die gewisser wohldefinierter Interpretationen bedarf. Ein zweites liefert

eine Ergebnisliste eventuell fortschrittlich als XML-Datenstrom, aber die XML-Schemastruktur unterscheidet sich von der eines dritten Systems.

4.3.2 Realisierungsmöglichkeiten/Problemlösungen

Hier ist es erforderlich, Transformationen für die einzelnen Formate durchzuführen. Abbildung 4.1 veranschaulicht eine mögliche Vorgehensweise. Als Format für ein strukturiertes Endergebnis bietet sich XML geradezu an. Transformationen von Daten, die bereits in XML vorliegen, können mit der Transformationssprache XSLT vollzogen werden.

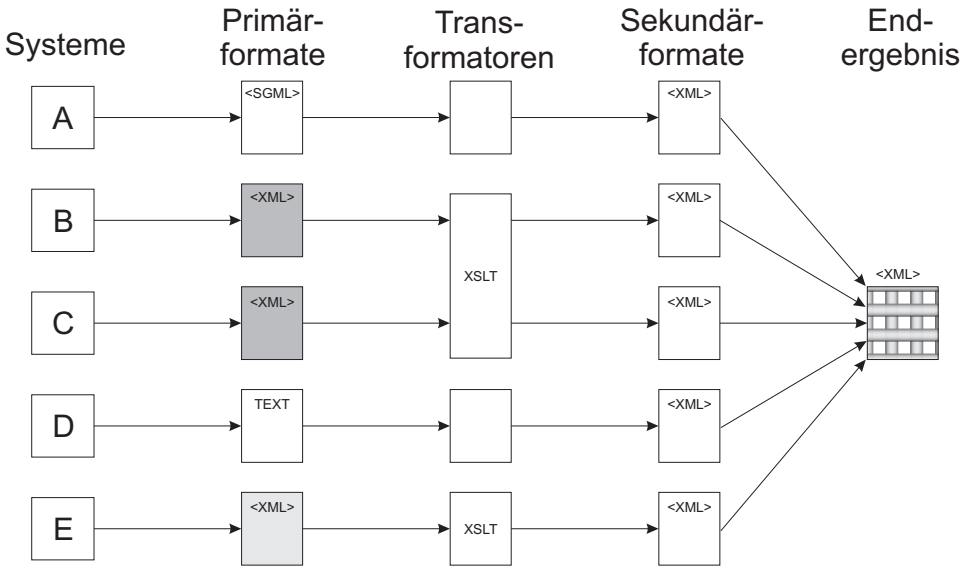

Abbildung 4.1: Mischung der Ergebnisse

Verschiedene Füllmuster der XML-Formate in der Abbildung sollen als eine XML-Dokumentklasse interpretiert werden. Das heißt, die Klassen der entsprechenden XML-Dokumente gleicher Füllmuster sind untereinander gleich. Diese Klassen können sinnvollerweise durch eine DTD [BPSM00] oder besser durch ein XML-Schema [Fal01, TBMM01, BM01] bestimmt worden sein.

Bei der dargestellten Vorgehensweise werden die von den integrierten Digitalen Bibliothekssystemen gemeldeten Ergebnisse von ihren Primärformaten über Transformatoren in ein einheitliches Sekundärformat konvertiert. Die Ergebnislisten in diesem Sekundärformat dienen dann als Eingabe für einen Merger, der diese Ergebnisse zu dem globalen Endergebnis zusammenmischt.

Die Wahl des Formates für die globale Ergebnisliste spielt eine entscheidene Rolle. Für jedes einzelne Ergebnisformat der einzelnen entfernten Repositories muss ein Mapping für eine Umwandlung in das globale Ergebnisformat durch die

Transformatoren definiert werden. Die Realisierung der Transformatoren ist also für jedes entfernte Repository individuell und von der Art und Weise der Auslieferung ihrer Ergebnisse abhängig.

4.4 Doublettenerkennung

Mehrfach gefundene Objekte sollen als identisch erkannt werden. Das System soll adäquat reagieren, indem es diese Objekte nur einmal im globalen Ergebnis präsentiert. Jedoch präsentiert das System dem Nutzer zusätzlich eine Liste der Lokationen des Objekts, wenn es auf verschiedenen entfernten Digitalen Bibliotheken zu finden ist.

4.4.1 Probleme

Die Erkennung von Doubletten von elektronischen Dokumenten ist problematisch. Es ist schwierig bis unmöglich zu entscheiden, dass es sich bei zwei Dokumenten um ein und dasselbe logische Dokument handelt.

Ein Grund ist beispielsweise die verschiedenartige Erfassung der Metadaten. Eines der Dokumente kann ausführlicher erfasst worden sein als das andere. Es kann mehrere Schreibweisen von ein und demselben erfassten Metadatum, wie zum Beispiel einem Autor geben. Ein Beispiel dafür ist der russische Autor *Maxim Gorki*, für den man außerdem die Schreibweise *Maksim Gorkii* findet. Aber sein Name lautet eigentlich *Aleksei Maksimovich Peshkov*, den man wahrscheinlich auch noch anders schreiben könnte. Es ist also möglich, dass unzählig viele Datensätze entstehen, die aber eigentlich alle dasselbe meinen.

Die Gleichheit könnte ein Mensch sicher einfacher beurteilen als eine Maschine. Daraus ergibt sich eine Möglichkeit der Realisierung. Ein Mensch könnte die Informationen über Duplikate manuell erfassen. Das könnte so passieren, dass das von einem Systemverwalter vorgenommen wird. Ein weiterer Ansatz ist das Einholen eines Feedbacks von den Benutzern des Systems.

Die automatische Erkennung von Doubletten elektronischer Dokumente ist in oben beschriebenen Fällen nahezu unmöglich. Es müssten die einzelnen Metadaten auf Identität überprüft werden. Ein Weg, Identitäten aufzuspüren, ist das Führen von Synonymwörterbüchern, die für eine Entscheidung der Objektidentität herangezogen werden. Ein wichtiger Anhaltspunkt für die Entscheidung darüber, ob zwei gefundene Dokumente dasselbe sind, ist eine Nachbarschaft in der globalen Rankingliste. Idealerweise wurden die Dokumente mit den gleichen Werten gerankt. Auf die Anforderung des globalen Rankings wird in 4.5 näher eingegangen.

Wünschenswert wäre ein globaler Identifizierungsmechanismus ähnlich der *ISBN (International Standard Book Number)* für Bücher und der *ISSN (International Standard Serial Number* für Zeitschriften. Wenn man elektronische Dokumente eindeutig identifizieren möchte, gibt es einiges zu beachten. Dokumente können in unterschiedlichen Formaten (z.B. HTML, PDF, PS) vorliegen, die die Benutzung

nur einer ID für alle Formate bedeuteten. Sie können außerdem in verschiedenen Versionen vorliegen, diese verschiedenen Versionen erfordern aber schon das Nutzen verschiedener IDs. Eine URL als Dokument-Identifizierung zu benutzen, ist mehr als ungenügend. Sie gibt an, auf welchem Rechner und in welcher Datei man ein Dokument zu einem bestimmten Zeitpunkt gefunden hat. Es ist nicht sichergestellt, dass diese Datei dort ewig zu finden ist *(dead links)*. Die Datei könnte ebenso verändert werden, ohne dass sich die URL ändert.

4.4.2 Realisierungsmöglichkeiten/Problemlösungen

Die International DOI Foundation[1], entstanden aus einer internationalen Initiative von Verlagen, hat ein Identifizierungsschema mit der Bezeichnung *DOI (Digital Object Identifier)* entwickelt. DOIs werden von zentralen Registrierungsagenturen vergeben. Die DOI setzt sich aus 2 Komponenten zusammen. Das Präfix wird von der Registrierungsagentur einem Content-Anbieter, wie einem Verlag oder einem Label zugewiesen. Alle Präfixe beginnen mit 10, gefolgt von einer Nummer, die den Content-Anbieter ausweist. Das Suffix wird von dem Content-Anbieter bestimmt. Hier sind dem Content-Anbieter jegliche Freiheiten überlassen, die zu identifizierende Einheit zu bezeichnen. Er kann also ohne Probleme Informationen wie die Auflagen- bzw. Versionsnummer oder das Format in der DOI codieren. Außerdem kann er die Granularität der Identifizierung festlegen. Will er nicht nur ein Buch sondern auch die darin enthaltenen Artikel granular identifizieren, so kann er das mit dem DOI-Identifizierungsmechanismus problemlos tun. Für Anwendungen ist die DOI lediglich eine Zeichenkette ohne jegliche sinngebende Bedeutung. Aber sie identifiziert die zugehörigen Dokumente eindeutig.

Jedem DOI ist ein Eintrag in einem verteilten zentralen Verzeichnis zugeordnet. Dort ist unter anderem die aktuelle Adresse des Dokumentes gespeichert. Wählt ein Nutzer den Zugriff auf ein DOI, so wird der Zugriff über das DOI-System mit dem Verzeichnis aufgelöst. Es wird dann die aktuelle Adresse des Dokumentes zurückgegeben. Ändert sich der Speicherort eines Dokumentes oder geht ein Dokument in einen anderen Besitz über wird das in dem zentralen Verzeichnis vermerkt und die Nutzer kommen automatisch zu der neuen Lokation.

Es ist Aufgabe des Content-Anbieters, die Metadatensätze zu den mit DOI registrierten Dokumenten zu erfassen, zu strukturieren und öffentlich bereitzustellen. Diese Metadatensätze können in einzelnen Digitalen Bibliothekssystemen erweitert werden.

Mit DOI ist ein wohldurchdachter Mechanismus entstanden, elektronische Dokumente eindeutig zu identifizieren. Eine Nutzung in Digitalen Bibliothekssystemen ist erstrebenswert. So kann eine Duplikaterkennung effektiv durchgeführt werden.

Ein ähnlicher Mechanismus ist mit dem *National Bibliographic Name* (kurz NBN) gegeben. Er ist von der Deutschen Bibliothek aus dem Arbeitspaket 4 des

[1] http://www.doi.org

Carmen-Projekts[2] heraus vorgeschlagen worden.

4.5 Globales Ranking

4.5.1 Ranking allgemein

Eine Ranking-Funktion, die einen Ranking-Wert berechnet, kann natürlich nie genau "wissen", was für einen Anfragesteller relevant ist und was nicht. Daher kann die Ermittlung der Relevanz nur nach heuristischen Methoden erfolgen.

Eine Bewertung der Güte Ranking-basierter Anfragemethoden und der Ranking-Funktion ist mit den Maßen *Recall*, *Precision* und *Fallout* [SH99] gegeben.

$$Recall = \frac{\text{Anzahl gefundener relevanter Dokumente}}{\text{Gesamtanzahl relevanter Dokumente}}$$

Je höher der Recall ist, desto besser ist die Suchmethode hinsichtlich des Findens der Dokumente, die tatsächlich durch die Anfrage gesucht worden sind.

$$Precision = \frac{\text{Anzahl gefundener relevanter Dokumente}}{\text{Gesamtanzahl gefundener Dokumente}}$$

Je höher die Precision ist, desto weniger Dokumente wurden gefunden, die gar nicht gesucht waren, die also irrtümlicherweise in das Suchergebnis mitaufgenommen wurden.

$$Fallout = \frac{\text{Anzahl gefundener irrelevanter Dokumente}}{\text{Gesamtamzahl irrelevanter Dokumente}}$$

Je niedriger der Fallout ist, desto besser ist die Methode im Herausfiltern der irrelevanten Dokumente.

Die einzelnen Mengen, die für die Ermittlung der eben genannten Werte eine Rolle spielen, sind in einem Euler-Diagramm in Abbildung 4.2 dargestellt. Innerhalb eines Repositories gibt es relevante und gefundene Dokumente. Je besser beide Mengen aufeinander fallen, je größer also die Schnittmenge ist, desto besser war die Suchmethode.

Die Anfragemethoden und Ranking-Funktionen sollten optimiert werden, so dass Recall und Fallout möglichst nahe an 1 und Fallout möglichst nahe an 0 liegen. Problematisch ist nur, dass sich diese Werte nur schwer oder gar nicht bestimmen lassen, da der Begriff "Relevanz" eines Dokumentes nur subjektiv beurteilbar ist und eine semantische Eigenschaft ist.

Sucht man verteilt auf mehreren Repositories, so kommen wir zu der Situation wie sie in Abbildung 4.3 für den Fall der Suche auf 2 Repositories A und B dargestellt ist. Es gibt evtl. Dokumente, die in beiden Repositories auftauchen. Diese gilt es zu erkennen, gleich zu bewerten und auf einmaliges Vorkommen im Ergebnis zu reduzieren.

[2] http://www.mathematik.uni-osnabrueck.de/projects/carmen

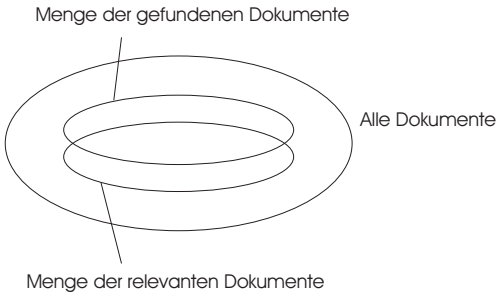

Abbildung 4.2: Euler-Diagramm für das Szenario der Suche auf einem Repository

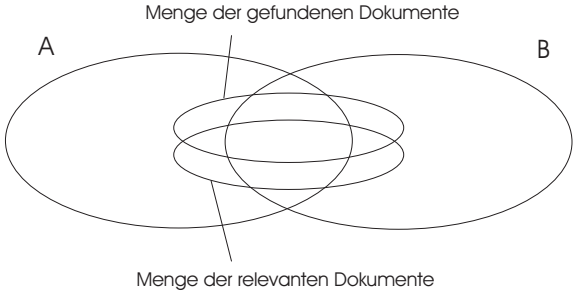

Abbildung 4.3: Euler-Diagramm für das Szenario der Suche auf zwei Repositories

Die Beurteilung der Relevanz durch eine Ranking-Funktion basiert wie bereits erwähnt auf Heuristiken. Der Ranking-Funktion liegen dafür gewisse Modelle zugrunde. Für Anfragen auf Volltexte sind das Vektorraummodell und das probabilistische Modell populär, die beide auf statistischen Erhebungen über das Vorkommen von Deskriptoren in einem Dokument basieren.

Ein Deskriptor ist ein Attribut eines Dokumentes, welches durch eine Deskribierung gewonnen wird. Je nach Verfahren sind es einfach nur die Worte aus einem Text (*precise index*), in Beziehung gesetzte Worte, z.B. vorzugsweise zu benutzende Synonyme eines Wortes, entsprechend einem Thesaurus, Grundformen von Flexionsformen[3] von Wörtern (Morphologische Reduktion, *linguistic index*) oder andere sinnvolle Attribute. Durch Stoppwortlisten werden häufig in Sprachen vorkommende Wörter von der Aufnahme als ein Deskriptor ausgeschlossen, da man davon ausgeht, dass solche Worte keinen Beitrag zur Charakterisierung eines Dokumentes leisten.

[3] finite Verben, deklinierte Substantive, etc.

Vektorraummodell

Beim Vektorraummodell [Har92] werden für die Anfrage und für die Dokumente Vektoren im multidimensionalen Raum bestimmt. Die Dimension M dieses Vektorraumes ist die Anzahl von verschiedenen Deskriptoren in der Datenbasis. Es sind in der Regel also Vektorräume mit sehr hohen Dimensionen.

Ein Vektor kann nun für Anfragen und Dokumente bestimmt werden. Jede Komponente des Vektors ist einem Deskriptor zugeordnet, der zumindest einmal in der gesamten Datenbasis über alle Dokumente vorkommt. Sie kann je nach Verfahren beispielsweise 1 bei Vorkommen oder 0 bei Nichtvorkommen (*Simple Match*) oder die Häufigkeit des Vorkommens im Dokument (*Weighted Match*) sein.

Nun gilt es die Ähnlichkeit zwischen einem Anfragevektor $\vec{q}$ und einem Dokumentvektor $\vec{d}$ zu bestimmen, um dadurch Rückschlüsse auf die Relevanz des Dokumentes hinsichtlich der Anfrage ziehen zu können. Typischerweise bestimmt man dafür den Kosinus des Winkels zwischen den beiden Vektoren. Man geht also davon aus, dass je kleiner der Winkel zwischen den beiden Vektoren ist, desto ähnlicher sind sie sich. Der Kosinus des Winkels ergibt sich aus der Definition des Skalarproduktes in Vektorräumen. Es gilt:

$$\cos \measuredangle(\, \vec{d}, \vec{q}\,) = \frac{\vec{d} \cdot \vec{q}}{|\, \vec{d}\,| \cdot |\, \vec{q}\,|}$$

Wenn man den Vektorraum als einen Euklidischen Vektorraum mit orthonormierter Basis und Standardskalarprodukt versteht, so gilt:

$$\cos \measuredangle(\, \vec{d}, \vec{q}\,) = \frac{\sum_{i=1}^{M} (\, d_i \cdot q_i\,)}{\sqrt{\sum_{i=1}^{M} d_i^2 \cdot \sum_{i=1}^{M} q_i^2}} \quad \text{, wobei } \vec{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_M \end{pmatrix} \text{ und } \vec{q} = \begin{pmatrix} q_1 \\ \vdots \\ q_M \end{pmatrix}$$

Diesen Wert nimmt man dann als Ähnlichkeitsmaß. Je geringer der Winkel zwischen den beiden Vektoren ist, bzw. je ähnlicher sich Anfrage und Dokument sind, desto höher ist der Wert. Anders ausgedrückt: Je höher der Wert, desto relevanter ist das Dokument bezüglich der Anfrage. Er liegt immer zwischen -1 und 1.

Probabilistisches Modell

Beim probabilistischen Modell [MK60] werden Wahrscheinlichkeiten dafür berechnet, wie sehr ein Dokument bezüglich einer Anfrage relevant ist.

Diese Berechnungen dafür beruhen ebenfalls auf statistischen Informationen, zum Beispiel auf einer Gewichtung der Deskriptoren nach ihrer Vorkommenshäufigkeit in der Datenbasis und im einzelnen Dokument. Bei vielen probabilistischen Modellen kommt auch zusätzlich der Einsatz von *Relevance Feedback* zum Einsatz, bei denen eine Relevanz-Bewertung der Benutzer von früheren Anfrageergebnissen mit in die Berechnung der Wahrscheinlichkeit der Relevanz eines Dokumentes eingeht.

[LM78] haben ein einfaches heuristisches Berechnungsverfahren für die Ermittlung der Relevanz vorgeschlagen. Dieses Verfahren nutzt allerdings kein Relevance Feedback. Empirische Untersuchungen der Ergebnisse haben eine hohe Güte des Verfahrens nachgewiesen.

Es seien

- N die Anzahl der Dokumente in der Datenbasis,
- M die Anzahl der verschiedenen Deskriptoren in der Datenbasis,
- f_{in} die Häufigkeit des Deskriptors T_{in} im Dokument D_n,
- t_n die Anzahl der verschiedenen Deskriptoren im Dokument D_n,
- d_m die Anzahl der Dokumente in der Datenbasis, in denen der Deskriptor T_m vorkommt,
- F_m die Häufigkeit des Deskriptors T_m in allen Dokumenten der Datenbasis,
- $Sf_n = \sum_{i=1}^{t_n} f_{in}$ die Summe aller Deskriptorauftreten im Dokument D_n,
- w_{nm} die Bewertung des Deskriptors T_m im Dokument D_n,

wobei $n \in \{1, \ldots, N\}$, $i \in \{1, \ldots, t_n\}$ und $m \in \{1, \ldots, M\}$. Zwischen m, i und n besteht eine funktionelle Abhängigkeit: $m = \psi(i, n)$. ($\exists m \exists n \exists i : T_m = T_{in}$)

Dann sind durch

- $F_1 = \frac{1}{d_m}$,
- $F_2 = \ln \frac{N}{d_m}$,
- $F_3 = \frac{1}{t_n}$,
- $F_4 = f_{in}$,
- $F_5 = \frac{f_{in}}{Sf_n}$ und
- $F_6 = \frac{f_{in}}{F_m}$

mit $w_{nm} = \begin{cases} F_k, \text{ falls } T_m \text{ in } D_n \text{ vorkommt} \\ 0, \text{ sonst} \end{cases}$, $k \in \{1, \ldots, 6\}$

Bewertungsfunktionen des Deskriptors T_m im Dokument D_n gegeben.

Bei F_1 und F_2 liegt die Annahme zugrunde, dass über die gesamte Datenbasis häufiger vorkommende Begriffe ein Dokument schlechter charakterisieren als selten vorkommende Begriffe. Diese Bewertungen sind datenbasisspezifisch und dokumentunabhängig.

Bei F_3 wird angenommen, dass ein einzelner Deskriptor umso weniger aussagekräftig ist, je mehr Deskriptoren das Dokument hat. Diese Bewertung ist datenbasisunabhängig und dokumentspezifisch.

Bei F_4 und F_5 ist die Häufigkeit eines Deskriptors in einem Dokument entscheidend. F_5 normiert diese zusätzlich über die Länge des Dokumentes. Diese Bewertungen sind datenbasisunabhängig und dokumentspezifisch.

Bei F_6 wird die Ausschließlichkeit des Deskriptors im Dokument bewertet. Diese Bewertung ist datenbasisspezifisch und dokumentspezifisch.

Nun kann man daraus den Relevanzgrad eines Dokumentes D_n bezüglich einer Anfrage ermitteln. Eine Anfrage kann jetzt wieder aus mehreren Deskriptoren bestehen, die in dem Dokument vorkommen sollen. Diese können gewichtet sein. Eine Anfrage hat dann die Gewichte v_m mit $m \in \{1, \ldots, M\}$. Deskriptoren, die nicht in der Frage vorkommen, haben das Gewicht 0.

Eine typische Berechnung der Relevanz des Dokumentes bezüglich der Anfrage ist:

$$R_n = \frac{1}{M} \sum_{m=1}^{M} v_m \cdot w_{nm}$$

4.5.2 Übertragung auf die verteilte Suche

Weil man bei einer Suche über verteilt vorliegende Daten nicht auf eine Relevanzbewertung und Anzeige in einer Reihenfolge entsprechend der Relevanz verzichten möchte, ist es notwendig das Ranking auch auf die verteilte Suche zu übertragen. Man strebt also ein *Globales Ranking* an.

Gehen wir davon aus, dass die einzelnen Repositories die Dokumente in ihrer Datenbasis adäquat und ausreichend gut bewerten können. Der Ranking-Wert, den die Repositories liefern, hat jedoch nur lokalen Charakter. Er gilt nur in seiner Umgebung auf seiner lokalen Instanz.

Erhält man jetzt also auf eine verteilte Anfrage von verschiedenen Repositories Ergebnisse, etwa Listen von Dokumenten oder Metadatenrecords, und zu jedem Ergebnis einen Relevanz-Wert, so reicht es nicht aus, die Ergebnisse einfach zusammenzumischen und die Rankingwerte zu übernehmen.

Ein Beispiel soll den Effekt verdeutlichen, der bei dieser ‘naiven’ Vorgehensweise entstehen würde.

Beispiel: Ich betrachte 2 Repositories, die ihre Volltextdokumente nach dem probabilistisches Modell ranken.

Das Repository A hat 1200 verschiedene Deskriptoren: $M_A = 1200$. B hat 2300: $M_B = 2300$. Die Menge der Deskriptoren in A ist $\mathcal{T}_A = \{T_1^A, \ldots, T_{M_A}^A\}$, die in B entsprechend $\mathcal{T}_B = \{T_1^B, \ldots, T_{M_B}^B\}$. $M_{A \cup B} = M_{\mathcal{T}_A \cup \mathcal{T}_B} = |\mathcal{T}_A \cup \mathcal{T}_B|$ ist die Anzahl der verschiedenen Deskriptoren über beide Repositories und sei 2800. Die Anzahl der Deskriptoren, die es in A und B gibt, ist damit $|\mathcal{T}_A \cap \mathcal{T}_B| = 700$.

Die Anfrage sei nun "`Information` : 0.5 **and** `Retrieval` : 0.8 ", `Information` soll mit der Wichtung 0.5 und `Retrieval` mit der Wichtung 0.8 in die Berechnung eines Ranking-Wertes eingehen. Alle anderen Deskriptoren sind mit 0 gewichtet ($v_m = 0$).

Die Ranking-Funktion sei $R_n = \frac{1}{M} \sum_{m=1}^{M} v_m \cdot w_{nm}$ und w_{nm} sei $\frac{f_{in}}{F_m}$.

Tabelle 4.1 führt die bisher getroffenen Annahmen weiter.

	`Information` : 0.5	`Retrieval` : 0.8	$R_n \cdot 10^6$	Reihenfolge
D_1^A	$3/130$	$6/240$	26.28	3
D_2^A	$1/130$	$3/240$	11.54	4
D_3^A	$15/130$	$28/240$	125.85	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
D_1^B	$10/720$	$15/840$	9.23	5
D_2^B	$7/720$	$3/840$	3.36	6
D_3^B	$45/720$	$70/840$	42.57	2
D_4^B	$2/720$	$2/840$	1.43	8
D_5^B	$1/720$	$4/840$	1.96	7
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Tabelle 4.1: Beispiel: 'Naives' globales Ranking

Zur Erklärung: Die zweite und dritte Spalte beinhalten im Tabellenkörper die w_{nm}-Werte, sie sind abhängig von Dokument und Deskriptor. Er berechnet sich aus $F_6 = \frac{f_{in}}{F_m}$. Zum Beispiel enthält das Dokument D_1^A den Deskriptor `Information` 3 mal, der seinerseits 130 mal in der gesamten Datenbasis von A auftaucht. Also ist $w^A_{1,\,\texttt{Information}} = 3/130$. Die vierte Spalte enthält den Ranking-Wert des Dokumentes bezüglich der Anfrage und ist zur besseren Lesbarkeit mit dem Faktor 10^6 skaliert. Er berechnet sich für D_1^A mit

$$R_1^A = \frac{1}{M_A} \sum_{m=1}^{M_A} v_m \cdot w_{1m}^A = \frac{1}{1200}(0.5 \cdot \frac{3}{130} + 0.8 \cdot \frac{6}{240} + 0 + \ldots + 0) \approx 26.28 \cdot 10^{-6}.$$

Die letzte Spalte zeigt die sich ergebende Sortierreihenfolge entsprechend der Relevanz der Dokumente nach der 'naiven' Vorgehensweise für das globale Ranking. Diese Reihenfolge gilt natürlich nur für die betrachteten 8 Dokumente. Es ist durchaus denkbar, dass sich andere Dokumente aus A oder B dazwischen drängen.

In Tabelle 4.2 werden die einzelnen w_{nm}-Werte und der R_n-Wert nun entsprechend den statistischen Parametern der neuen globalen Datenbasis $A \cup B$ berechnet. Es entsteht ein besseres Ergebnis.

$R_1^{A \cup B}$ berechnet sich nun so:

$$R_1^{A \cup B} = \frac{1}{M_{A \cup B}} \sum_{m=1}^{M_{A \cup B}} v_m \cdot w_{1m}^{A \cup B}$$
$$= \frac{1}{2800}(0.5 \cdot \frac{3}{850} + 0.8 \cdot \frac{6}{1080} + 0 + \ldots + 0) \approx 2.22 \cdot 10^{-6}$$

	`Information` : 0.5	`Retrieval` : 0.8	$R_n \cdot 10^6$	Reihenfolge
D_1^A	$3/850$	$6/1080$	2.22	5
D_2^A	$1/850$	$3/1080$	1.00	7
D_3^A	$15/850$	$28/1080$	10.56	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
D_1^B	$10/850$	$15/1080$	6.07	3
D_2^B	$7/850$	$3/1080$	2.26	4
D_3^B	$45/850$	$70/1080$	27.97	1
D_4^B	$2/850$	$2/1080$	0.95	8
D_5^B	$1/850$	$4/1080$	1.27	6
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Tabelle 4.2: Beispiel: Verbessertes globales Ranking

Deutlich sieht man, dass das ‘naive’ globale Ranking die richtige Relevanz-Reihenfolge aus Tabelle 4.2 stark verfälscht. Offenbar schneiden die Dokumente aus kleineren Datenbasen bei dieser Relevanzermittlungsmethode besser ab, als solche aus größeren Datenbasen. Die relative Ordnung der Dokumente aus A und B bleibt unverändert.

Man kann sich vorstellen, dass das Ergebnis sogar noch mehr verfälscht wird, wenn sich die Ranking-Funktionen oder gar die Ranking-Modelle unterscheiden.

Das ‘naive’ globale Ranking führt im Falle der Wahl von dokumentspezifischen und datenbasisunabhängigen Funktionen, z.B. F_3, F_4 und F_5, für w_{nm} durchaus zu unverfälschten Ergebnissen, wenn die gleiche Berechnungsformel auf allen Repositories gilt. Aber wenn man nicht weiß, wie R_n jeweils berechnet wurde, dann ist das Zufall.

Die beiden vorgestellten Modelle sollen jetzt auf das Szenario der verteilten Suche übertragen werden.

Vektorraummodell

Es seien

- $\mathcal{R} = \{R_1, \ldots, R_k\}$ die Menge der einzelnen Repositories,
- $k = |\mathcal{R}|$ die Anzahl der Repositories,
- $r \in \mathcal{R}$ ein Index über die Repositories,
- $\mathcal{T}_r = \{T_1^r, \ldots, T_{M_r}^r\}$ die Menge der verschiedenen Deskriptoren in r,
- $M_r = |\mathcal{T}_r|$ die Anzahl der verschiedenen Deskriptoren in r,
- $m \in \{1, \ldots, M_r\}$ ein Index über diese Deskriptoren,

- T_m^r der m-te Deskriptor in der Deskriptorliste der Datenbasis von r.

In folgendem Beispiel sind die Deskriptorlisten zweier Repositories angegeben.

Beispiel

$k = 2$

$$\begin{array}{ll} T_1^{R_1} & = \texttt{"abbey"} \\ T_2^{R_1} & = \texttt{"ability"} \\ T_3^{R_1} & = \texttt{"academy"} \\ \vdots & \vdots \\ T_{M_{R_1}-2}^{R_1} & = \texttt{"wrapper"} \\ T_{M_{R_1}-1}^{R_1} & = \texttt{"zenith"} \\ T_{M_{R_1}}^{R_1} & = \texttt{"zoom"} \end{array} \qquad \begin{array}{ll} T_1^{R_2} & = \texttt{"abandon"} \\ T_2^{R_2} & = \texttt{"abbreviation"} \\ T_3^{R_2} & = \texttt{"ability"} \\ \vdots & \vdots \\ T_{M_{R_2}-2}^{R_2} & = \texttt{"wire"} \\ T_{M_{R_2}-1}^{R_2} & = \texttt{"zodiac"} \\ T_{M_{R_2}}^{R_2} & = \texttt{"zone"} \end{array}$$

Daraus resultiert die global gültige Deskriptorliste
$\mathcal{T}^{\texttt{global}} = \mathcal{T}_{R_1} \cup \ldots \cup \mathcal{T}_{R_k} = \mathcal{T}_{R_1} \cup \mathcal{T}_{R_2}$:

$$\begin{array}{ll} T_1^{\texttt{global}} & = \texttt{"abbandon"} \\ T_2^{\texttt{global}} & = \texttt{"abbey"} \\ T_3^{\texttt{global}} & = \texttt{"abbreviation"} \\ T_4^{\texttt{global}} & = \texttt{"ability"} \\ \vdots & \vdots \\ T_{M_{\texttt{global}}-2}^{R_1} & = \texttt{"zodiac"} \\ T_{M_{\texttt{global}}-1}^{R_1} & = \texttt{"zone"} \\ T_{M_{\texttt{global}}}^{R_1} & = \texttt{"zoom"} \end{array}$$

$|\mathcal{T}_{\texttt{global}}| = M_{\texttt{global}}$
Für den Fall $k = 2$ gilt:
$|\mathcal{T}_{\texttt{global}}| = |\mathcal{T}_{R_1 \cup R_2}| = |\mathcal{T}_{R_1} \cup \mathcal{T}_{R_2}| = |\mathcal{T}_{R_1}| + |\mathcal{T}_{R_2}| - |\mathcal{T}_{R_1} \cap \mathcal{T}_{R_2}|$
Für den allgemeinen Fall gilt:

$$\begin{aligned} &|\mathcal{T}_{\texttt{global}}| = |\mathcal{T}_{R_1} \cup \ldots \cup \mathcal{T}_{R_k}| \\ &= |\mathcal{T}_{R_1}| + \ldots + |\mathcal{T}_{R_k}| \\ &- (|\mathcal{T}_{R_1} \cap \mathcal{T}_{R_2}| + \ldots + |\mathcal{T}_{R_1} \cap \mathcal{T}_{R_k}| + \ldots + |\mathcal{T}_{R_i} \cap \mathcal{T}_{R_{i+1}}| + \ldots + |\mathcal{T}_{R_i} \cap \mathcal{T}_{R_k}| \\ &+ \ldots + |\mathcal{T}_{R_{k-1}} \cap \mathcal{T}_{R_k}|) \\ &+ \ldots \\ &+ (-1)^{s-1}(|\underbrace{\mathcal{T}_{R_1} \cap \ldots \cap \mathcal{T}_{R_{k-1}}}_{s \text{ Mengen } (s=k-1)}| + \ldots + |\underbrace{\mathcal{T}_{R_2} \cap \ldots \cap \mathcal{T}_{R_k}}_{s \text{ Mengen } (s=k-1)}|) \\ &+ (-1)^{s-1}(|\underbrace{\mathcal{T}_{R_1} \cap \ldots \cap \mathcal{T}_{R_k}}_{s \text{ Mengen } (s=k)}|) \end{aligned}$$

Im Beispiel gelten z.B.:
$\texttt{"ability"} = T_2^{R_1} = T_3^{R_2} = T_4^{R_1 \cup R_2} = T_4^{\texttt{global}}$ und
$\texttt{"abandon"} = T_1^{R_2} T_1^{R_1 \cup R_2} = T_1^{\texttt{global}}$

Es gibt also eine Funktion $\rho(m, r)$, die eine eindeutige Nummer der Deskriptors T_m^r in einer globalen Deskriptorliste $\mathcal{T}_{\texttt{global}}$ berechnet:

$$\forall r \in \mathcal{R} \forall m \in \{1, \ldots, M_r\} : T_m^r = T_{\rho(m,r)}^{\texttt{global}}$$

Es gilt also eine globale Deskriptorliste zu bestimmen und damit den Vektorraum für die globale Betrachtung aufzuspannen. Außerdem muss die Funktion ρ berechnet werden. Man kann sich ρ als eine Menge von Tabellen für jedes Repository vorstellen:

m	$\rho(m, R_1)$
1	2
2	4
3	5
$\vdots$	$\vdots$
$M_{R_1} - 2$	$M_{\texttt{global}} - 4$
$M_{R_1} - 1$	$M_{\texttt{global}} - 3$
M_{R_1}	$M_{\texttt{global}}$

m	$\rho(m, R_2)$
1	1
2	3
3	4
$\vdots$	$\vdots$
$M_{R_2} - 2$	$M_{\texttt{global}} - 5$
$M_{R_2} - 1$	$M_{\texttt{global}} - 2$
M_{R_2}	$M_{\texttt{global}} - 1$

Tabelle 4.3: Die Abbildung von lokalen auf globale Deskriptoren mittels ρ

Wie groß die Deskriptorlisten, also M_{R_1}, M_{R_2} und $M_{\texttt{global}}$, sind, bleibt in diesem Beispiel offen.

Um das Vektorraummodell für eine verteilte Suche anzuwenden, bedarf es demnach folgender Schritte:

Eingabe: $\mathcal{T}_1, \ldots, \mathcal{T}_k, \forall r \in \mathcal{R} \forall n \in \{1, \ldots, N_r\} \vec{d}_n^{\,r}$
N_r ist die Anzahl der Dokumente in Repository r, $\vec{d}^{\,r}$ ist der Dokumentvektor des Dokumentes n in Repository r.

1. Vereine die Deskriptorlisten $\mathcal{T}_1, \ldots, \mathcal{T}_k$ zu einer globalen Deskriptorliste $\mathcal{T}_{\texttt{global}}$, baue dabei gleichzeitig die ρ-Tabellen auf.
2. Ermittle den Fragevektor $\vec{q}^{\,\texttt{global}}$ im globalen Vektorraum.
3. Für jedes Repository r: und für jedes Dokument n mit seinem Vektor $\vec{d}^{\,r}$:
   (a) Transformiere den Dokumentenvektor $\vec{d}^{\,r}$ in den globalen Vektorraum. Ermittle also: $\vec{d}^{\,\texttt{global}}$.
   (b) Ermittle die Ähnlichkeit zwischen $\vec{d}^{\,\texttt{global}}$ und $\vec{q}^{\,\texttt{global}}$ als Relevanzgrad des Dokumentes n:

$$R_n^{\texttt{global}} = \frac{\sum_{m=1}^{M_{\texttt{global}}} (\, d_m \cdot q_m \,)}{\sqrt{\sum_{m=1}^{M_{\texttt{global}}} d_m^2 \cdot \sum_{m=1}^{M_{\texttt{global}}} q_m^2}} \quad ,$$

$$\text{wobei } \vec{d}^{\,\texttt{global}} = \begin{pmatrix} d_1 \\ \vdots \\ d_{M_{\texttt{global}}} \end{pmatrix} \text{ und } \vec{q}^{\,\texttt{global}} = \begin{pmatrix} q_1 \\ \vdots \\ q_{M_{\texttt{global}}} \end{pmatrix}$$

Der entscheidene Schritt ist dabei Schritt 3(a). Hier muss ein lokal gültiger Dokumentenvektor in einen Vektor des globalen Vektorraumes mit der Dimension $M_{\texttt{global}}$ transformiert werden. Die Vorgehensweise dafür ist die folgende:

Eingabe: $\vec{d}^{\,r} = \begin{pmatrix} d_1 \\ \vdots \\ d_{M_r} \end{pmatrix}, \rho$

1. Initialisiere $\vec{d}^{\,\texttt{global}} = \begin{pmatrix} d_1^{\texttt{global}} \\ \vdots \\ d_{M_{\texttt{global}}}^{\texttt{global}} \end{pmatrix}$ mit dem Nullvektor $\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$,
2. Für jede Komponente d_m mit $m \in \{1 \ldots M_r\}$ von $\vec{d}^{\,r}$: $d_{\rho(m,r)}^{\texttt{global}} = d_i$

Probabilistisches Modell

Auch bei einem globalen Ranking, das nach probabilistischem Modell gestaltet ist, müssen viele statistische Informationen von den einzelnen Repositories übertragen werden. Geht man von der typischen Relevanzberechnungsformel

$$R_n = \frac{1}{M} \sum_{m=1}^{M} v_m \cdot w_{nm}$$

aus, so ist klar, dass die Parameter M und w_{nm} für den globalen Fall neu bestimmt werden müssen. Welche statistischen Informationen übermittelt werden müssen, hängt von der Berechnungsvorschrift für w_{nm} (Vgl. Seite 46) ab.

Im Folgenden sind die Vorgehensweisen für die Ermittlung der global gültigen statistischen Parameter aus den lokal gültigen beschrieben. Die Semantiken der einzelnen Parameter sind dieselben, wie sie auf Seite 46 beschrieben sind.

- f_{in}, t_n und Sf_n sind dokumentspezifisch und ändern sich daher nicht.
- M: Es ist wiederum notwendig eine Vereinigung der Deskriptorlisten zu einer globalen Deskriptorliste genau wie beim Vektorraummodell vorzunehmen, dadurch lässt sich der Parameter M einfach als Anzahl der Einträge der globalen Liste bestimmen.
- d_m und F_m werden ebenfalls beim Aufbau der globalen Deskriptorliste bestimmt. Ist der Eintrag in der globalen Liste bei Abarbeitung eines bestimmten Deskriptors noch nicht vorhanden, so wird er dort hinzugefügt und seine

d_m- und F_m-Werte werden mit denen initialisiert, die sie in dem gerade betrachteten Repository hatten. Ist der Eintrag schon vorhanden, so werden die Werte einfach addiert. Die Werte sind also die Summen der entsprechenden Parameter, wie sie in den einzelnen lokalen Instanzen Gültigkeit haben.

- N ergibt sich einfach aus der Summe der Anzahl der Dokumente in den einzelnen Repositories: $N_{\texttt{global}} = \sum_{r \in \mathcal{R}} N_r$.

4.5.3 Probleme

Das Problem beim Ranking für eine verteilte Suche ist, dass ein sehr hoher Aufwand durch den Austausch von statistischen Informationen entsteht. Der Kommunikationsaufwand ist also immens hoch. Es entsteht außerdem ein hoher Rechenaufwand mit sehr großen Datenmengen. Daraus resultiert auch gleich das nächste Problem: Wie will man überhaupt an solche Informationen kommen? Nicht jeder Anbieter ist sicherlich gewillt oder einfach nicht in der Lage, so umfangreiche statistische Informationen bereitzustellen.

4.5.4 Realisierungsmöglichkeiten/Problemlösungen

'Naives' globales Ranking

Das 'naive' globale Ranking ist wesentlich einfacher zu realisieren, auch wenn es nicht so korrekt ist, wie die beschriebenen exakteren Methoden. Das 'naive' globale Ranking lässt sich jedoch verbessern, wenn man auf datenbasisspezifische Ranking-Funktionen verzichtet.

Reranking

Ein anderer Weg wäre es, wenn man auf der globalen Seite Teile der Ergebnislisten in einem zweiten Ranking-Prozess neu bewertet. Dazu muss man sich die gesamten Dokumente der ausgewählten dort hin replizieren und neu analysieren, also indizieren bzw. deskribieren. Dieses zweite Ranking, also das *Reranking*, geschieht wiederum nach den bekannten bewährten Methoden, also zum Beispiel nach dem Vektorraummodell oder nach dem probabilistischem Modell.

Man könnte zum Beispiel das 'naive' Ranking dazu benutzen, um eine erste Vorselektion vorzunehmen. Für den zweiten Schritt würden sich dann die ersten n Dokumente der 'naiven' Rankingliste qualifizieren. Ein anderer besserer Weg für die Vorauswahl wäre es, die jeweils ersten n Treffer der Ergebnislisten zu replizieren und als Ausgangspunkt für das zweite Ranking auf globaler Ebene zu nutzen. Die statistischen Informationen, die für die Relevanzberechnungen benötigt werden, müssen nun nicht mehr übermittelt werden, sondern sie werden nunmehr durch Analyse der komplett übermittelten Dokumente neu bestimmt.

Der Vorteil dieser Vorgehensweise ist der Wegfall der Notwendigkeit der Übertragung der statistischen Informationen, jedoch auf Kosten gravierender Nachteile:

Es müssen zwar nicht alle aber komplette Dokumente übertragen werden. Daraus resultiert ein hohes Datenübertragungsvolumenpotential, was sich stark auf die Netzlast auswirkt. Desweiteren müssen die Dokumente neu analysiert, also indiziert bzw. deskribiert, werden. Dafür ist immens starke Rechenleistung notwendig, um dem Benutzer zumutbare Antwortzeiten anbieten zu können.

Finden gemeinsamer Eckpunkte

Eine andere vielversprechendere Methode basiert auf dem Finden gemeinsamer Eckpunkte. Die Vorgehensweise ist dabei folgendermaßen:

Zuerst wird die globale Anfrage wieder an die einzelnen Repositories delegiert und dazu eventuell durch einen Wrapper entsprechend umgeformt. Dann werden die einzelnen Ergebnislisten übertragen. Sie enthalten typischerweise Metadaten und Relevanzwerte. Nun werden in den einzelnen Trefferlisten Doubletten (Vgl. Abschnitt 4.4) erkannt, diese sollen dann natürlich wieder auf ein Resultat reduziert werden. Das Erkennen von Duplikaten bietet aber außerdem einen Ansatzpunkt für das globale Ranking. Gleiche Dokumente in den Trefferlisten haben natürlich auch die gleiche Relevanz und sind damit die Eckpunkte in dieser Methodik. Die Relevanz dieser in mehreren verschiedenen Repositories vorkommenden Dokumente kann also gleichgesetzt werden. Dadurch entsteht eine Abbildung von Relevanzwerten der einzelnen Repositories aufeinander. Diese Abbildung kann man nun als eine gute Grundlage für das Bilden globaler Ranking-Werte genutzt werden. Ein Beispiel soll die Methodik besser veranschaulichen.

Beispiel Es gilt 3 Repositories abzufragen und die Ergebnisse global zu ranken. Die einzelnen Ergebnislisten sind in Tabelle 4.4 dargestellt:

R_1		R_2		R_3	
α	80%	ϵ	90%	θ	95%
β	75%	γ	85%	β	80%
γ	72%	ζ	71%	κ	69%
δ	70%	η	65%	λ	67%
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabelle 4.4: Beispiel: Finden gemeinsamer Eckpunkte – Ausgangslage

Mit α bis λ sind Dokumente bezeichnet. Gleiche griechische Buchstaben in den verschiedenen Ergebnislisten deuten dabei auf erkannte Doubletten hin. Nun gelten in diesem Beispiel: 75% in R_1 entsprechen 80% in R_3, da β bei beiden identifiziert wurde. Ebenso entsprechen 72% in R_1 85% in R_2 wegen der Duplizität von γ. Damit sind also Eckpunkte (β und γ) gefunden, die es erlauben, die 3 Listen miteinander zu verbinden.

Einer dieser Eckpunkte wird mit einem beliebigen Relevanzwert für das globale Ergebnis festgesetzt. Anschließend lassen sich die anderen Relevanzwerte über Verhältnisgleichungen und die anderen Eckpunkte ermitteln.

In dem Beispiel setze man beispielsweise den Wert für die Relevanz von β auf 1. Dann ergibt sich eine global gerankte Liste, die in Tabelle 4.5 dargestellt ist.

θ	$1,19$
α	$1,07$
ϵ	$1,02$
β	1
γ	$0,96$
δ	$0,93$
κ	$0,86$
λ	$0,84$
ζ	$0,80$
η	$0,73$
$\vdots$	$\vdots$

Tabelle 4.5: Beispiel: Finden gemeinsamer Eckpunkte – Endergebnis

Problematisch wird dieses Verfahren, wenn zu wenig Eckpunkte gefunden werden. Wenn also eine Verbindung zweier Listen unmöglich ist, weil keine gemeinsamen Dokumente erkannt wurden. In diesem Falle muss wieder auf Strategien zurückgegriffen werden, wie sie im Vorhinein beschrieben worden sind.

Eine andere Problematik tut sich allerdings auf, wenn zu viele Eckpunkte vorhanden sind und diese widersprüchliche Informationen liefern. Der Extremfall wäre, wenn Paare von Dokumenten in einem Repository in umgekehrter Reihenfolge wie ihre Doubletten in einem zweiten Repository gerankt worden sind.

4.5.5 Zusammenfassung

Das globale Ranking ist eine wichtige Anforderung an eine verteilte Suche in mehreren Digitalen Bibliotheken. Jedoch ist eine Realisierung dieser Anforderung eine ziemlich komplizierte Herausforderung.

In diesem Abschnitt wurden Strategien für ein globales Ranking aufgezeigt. Gute Methoden erfordern einen Austausch von statistischen Informationen von den einzelnen entfernten Repositories hin zum integrierenden System. Diese Informationen stehen sicherlich nicht in jedem Falle zur Verfügung. Deshalb wurden auch Realisierungsmöglichkeiten aufgezeigt, die ohne diese Übermittlung auskommen, dadurch aber auch Nachteile in Kauf nehmen müssen. Die Ergebnisse werden dann ungenauer bzw. es wachsen die Netzlast- und Rechenleistungsanforderungen sehr stark an.

Eine weitere interessante Strategie zum globalen Ranking ist in [GG97] beschrieben.

4.6 Zusammenfassung

In diesem Kapitel wurden 14 Anforderungen an einen verteilten Such- / Integrationsdienst formuliert. Für diese wurden weiterhin Realisierungsmöglichkeiten vorgestellt und beleuchtet. Die Anforderungen der parallelen Suche, der Ergebnismischung der Doublettenerkennung und des globalen Rankings wurden ausführlicher diskutiert.

Dieser Anforderungskatalog kann als Ausgangspunkt für die Konzipierung eines verteilten Such- / Integrationsdienstes auf Digitalen Bibliotheken genutzt werden. Jedoch wird es wahrscheinlich schwierig sein, alle diese Anforderungen in hohem Grad zu erfüllen.

Die Güte eines Systems, das eine verteilte Suche anbietet, kann mit Hilfe dieses Anforderungskataloges bewertet werden. Für jede einzelne Anforderung würde dazu vermerkt werden, wie gut sie von dem entsprechenden System erfüllt wird. Solche Bewertungen sind für die beiden implementierten Systeme MyCoRe und den Cross Archive Searching Service in 6.1.5 und 6.2.3 vorgenommen worden.

Kapitel 5

Existierende Ansätze für verteilte Suchdienste

Es gibt verschiedene Ansätze, um einen verteilten Such- / Integrationsdienst umzusetzen. In diesem Kapitel werden einige wichtige Ansätze näher beleuchtet.

Zunächst werden Aspekte der Kommunikation in verteilten Anwendungen diskutiert. Dazu gehören verschiedene existierende Ansätze und Protokolle, sowie Fragen der Sicherheit beim Austausch von Informationen.

Anschließend wird auf die Möglichkeit einer Metadatenabfrage eines Repositories eingegangen. Durch den Zugriff auf die Metadaten eines entfernten Repositories lassen sich verteilte Suchdienste in der Ausprägung realisieren, die in 3.5.1 beschrieben ist. Hier wird speziell der Zugriff auf die Metadaten mittels des OAI-Protokolls betrachtet.

Danach werden zwei Lösungsansätze beschrieben, die eine echte verteilte Suche mit On-The-Fly-Verarbeitung bieten (3.5.2). Dazu gehören zum einen das Bibliotheksprotokoll Z39.50 und das Feature des “Federated Search” des Enterprise Information Portals.

5.1 Kommunikation in verteilten Anwendungen

Um eine Kommunikation in verteilten Anwendungen zu realisieren, gibt es eine Reihe von Ansatzpunkten. Die Art und Weise, wie diese Kommunikation vonstatten geht, ist in der Regel anwendungsneutral. In diesem Abschnitt sind nun einige wichtige Kommunikationsmöglichkeiten beschrieben.

5.1.1 Low-Level-Netzwerk-Programmierung

Unter Low Level Netzwerk Programmierung versteht man das Programmieren auf einem relativ gering abstrahierten Niveau. Über sogenannte Sockets können Informationen zwischen Server- und Client-Prozessen ausgetauscht werden.

Diese Programmierung basiert direkt auf dem TCP/IP Protokoll. Java bietet mit dem `java.net`-Package Klassen für die Socket-Programmierung. Für andere Programmiersprachen gibt es ebenfalls APIs zur Socket-Programmierung.

Der Vorteil der Socket-Programmierung ist, dass eine hohe Performance erzielt werden kann. Jedoch ist der Aufwand für eine Programmierung einer kontrollierten und fehlerfreien Kommunikation relativ hoch. Diese Art der Programmierung ist außerdem sehr fehleranfällig. Der Programmierer steht quasi vor der Herausforderung ein eigenes Kommunikationsprotokoll zu entwickeln. Hinzu kommen die Fragen der Sicherheit, der Austausch der Informationen erfolgt in der Regel im Klartext und kann von Dritten belauscht werden. Es ist für eine Übertragung von sensiblen Daten also notwendig, Verschlüsselungskonzepte miteinzubeziehen. Dafür bietet die Java Secure Sockets Extension (JSSE) von Sun hilfreiche Programmierwerkzeuge für Socket-Kommunikation mittels Java.

5.1.2 HTTP-HTTPS-Kommunikation

Ein auf TCP/IP aufsetzendes Protokoll ist das *Hyper Text Transfer Protocol*, kurz HTTP-Protokoll, es ist heute eines der Standardinternetprotokolle und wird tagtäglich zur Übertragung in Web-Anwendungen benutzt. HTTP realisiert selbst eine Socket-Kommunikation und kapselt den Informationsaustausch in HTTP-requests und HTTP-responses.

Der Vorteil der Benutzung von HTTP ist die relativ einfache Programmierung. Java bietet mit den Java Servlets eine Möglichkeit, um dynamisch auf HTTP-Requests reagieren zu können. Andere Programmiersprachen nutzen dazu das *Common Gateway Interface* (CGI). Die Client-Seite kann auch einfach aus Programmen in beliebigen Programmiersprachen realisiert werden.

Ein Nachteil ist immer noch das relativ geringe Abstraktionsniveau dieser Kommunikation. Es werden immer noch Datenströme und keine Objekte im objektorientierten Sinne ausgetauscht, die erst wieder interpretiert werden müssen.

Ein anderer Nachteil ist genau wie bei der Socket-Programmierung mangelnde Datensicherheit bei der Übertragung. Dieser kann aber durch den Einsatz von HTTPS als die verschlüsselte Variante von HTTP erreicht werden.

5.1.3 CORBA

Die *Common Object Request Broker Architecture* kurz CORBA [corba] ist ein etablierter OMG[1]-Standard für die Entwicklung von verteilten Anwendungen. CORBA erlaubt den Aufruf von Methoden auf entfernten Objekten, bekannt als *Remote Procedure Calls* (RPC).

CORBA-basierende Programme können über die RPCs miteinander interoperieren, und das ganze über nahezu beliebige Programme in beliebigen Programmiersprachen, Betriebssystemen und Netzwerktechnologien.

[1] Object Management Group - http://www.omg.org

CORBA richtet sich gerade an die objektorientierten Programmiersprachen. Um den Vorteil dieser Programmierweise zu erhalten, sollen Objekte in einer CORBA-Anwendung global handhabbar sein. Das ist bei den bisher behandelten Kommunikationskonzepten ausgeschlossen. Dort wird jeweils nur ein Datenstrom ausgetauscht, der von beiden Seiten entsprechend interpretiert werden muss. CORBA bietet nun global gültige Objekte und Klassen nach objektorientierten Programmierkonzepten.

Für jedes zu benutzende Objekt definiert man ein CORBA Interface in OMG IDL (Interface Definition Language). Über dieses Interface bietet ein Server-Objekt den Clients, die dieses Objekt benutzen wollen, Zugriff. Jeder Client muss sich an das IDL Interface halten, um das Objekt zu nutzen. Das Interface beschreibt außerdem die Rückgabewerte von Methoden und öffentliche Attribute von Objekten in gewohnter Manier objektorientierter Technologie.

Die IDL Interface Definition ist unabhängig von der Programmiersprache der eigentlichen Implementierung, sie wird auf die entsprechende Programmiersprache gemappt. Es existieren standardisierte Mappings für nahezu alle populären objektorientierten Programmiersprachen.

Im folgenden wird beschrieben, wie nun der Aufruf ein entfernter Methodenaufruf vonstatten geht:

1. Auf dem Client wird ein sogenanntes Stub-Objekt erstellt, dieses Objekt ist auf die Schnittstellen des entsprechenden Server-Objektes zugeschnitten. Auf dem Server gibt es einen Skeleton als Gegenstück dazu, das von der IDL auf die entsprechende Programmiersprache des eigentlichen Objektes auf dem Server umsetzt. Das Stub-Objekt passt genau auf die Schnittstellenbeschreibung, die mittels IDL definiert wurde.

2. Nun wird mit dem Stub-Objekt “normal” gearbeitet. Es werden Attributfelder abgefragt, Methoden aufgerufen etc..

3. Da das entsprechende Objekt aber auf einem entfernten Server liegt, werden diese Aufrufe über den ORB (*Object Request Broker*) an das entsprechende Objekt weiterdelegiert.

4. Nachdem gewisse Methodenaufrufe auf dem Server abgelaufen sind, können Ergebnisse auf demselben Wege wieder an den Client zurückgegeben werden.

Ein Stub-Objekt bildet also eine Abstrahierung eines Objektes, das auf einem entfernten Server implementiert ist. Das Skeleton-Objekt mappt einen Request von IDL auf die Programmiersprache, in der das Objekt implementiert ist.

Das ganze Abblauf ist in Abbildung 5.1 grafisch dargestellt:

CORBA ist eine sehr umfangreiche Architektur. Sie ist weitestgehend sprachunabhängig. Der Aufwand für eine Umsetzung von CORBA-Anwendungen ist jedoch relativ hoch.

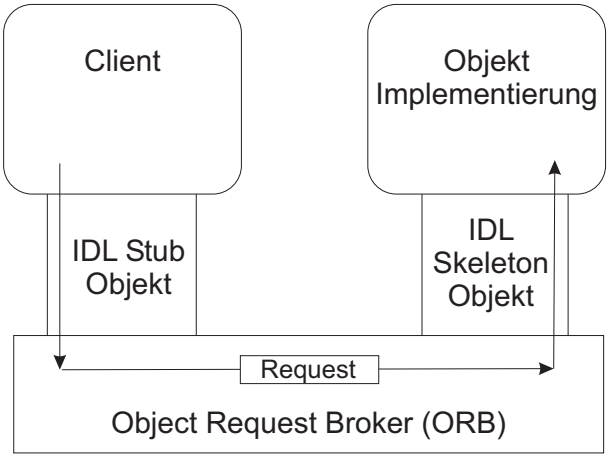

Abbildung 5.1: Aufruf einer entfernten Methode mit CORBA

5.1.4 RMI

Remote Method Invocation (RMI) ist eine RPC-Variante, die direkt auf Java zugeschnitten wurde. Das heißt das Server- und die Client-Anwendung müssen beide in Java geschrieben sein. Die Funktionsweise ist vergleichbar mit der von CORBA.

Auch bei RMI gibt es Stub-Objekte, die Objekte auf entfernten Systemen abstrahieren, dessen eigentliche Implementierung auf dem Server wiederum über einem Skeleton-Objekt angesprochen wird. Ein RMI-Aufruf wird über den sogenannten Remote Reference Layer an das Skeleton-Objekt auf dem entsprechenden Server übergeben.

Das Interface des Server-Objektes ist aber im Gegensatz zu CORBA nicht in IDL sondern direkt in Java. Eine aufwendige Umwandlung durch das Skeleton-Objekt entfällt dadurch.

Java RMI vereinfacht die relativ komplizierte Infrastruktur von CORBA, jedoch geht dabei die Sprachunabhängigkeit verloren. Der Programmierer ist bei RMI auf Java eingeschränkt.

5.1.5 SOAP

Das *Simple Object Access Protocol* [BEK+00, BTN00] ist ein weiteres Protokoll zur Client-Server-Kommunikation basierend auf RPCs. Wie der Name schon sagt sollen Programme über dieses Protokoll mittels Objektzugriffen miteinander kommunizieren und das möglichst einfach. Diese Objekte liegen wieder auf einem entfernten Server.

SOAP setzt auf andere Protokolle wie zum Beispiel HTTP, HTTPS oder SMTP (*Simple Mail Transfer Protocol*) auf. Die übertragenen Daten zum Aufruf von entfernten Methoden werden in XML gekapselt und dann übertragen. Dank XML

kann eine beliebige Schachtelung und Schachtelungstiefe verwendet werden. Dem Programmierer sind hier keine Grenzen gesetzt.

Der Aufruf einer Methode auf einer entfernten Softwarekomponente erfolgt nun folgendermaßen:

Zuerst wird ein SOAP-Request in Form einer SOAP Message erzeugt. Diese beinhaltet den Namen der entsprechenden Methode auf dem entfernten System und entsprechende Parameter. Diese Message ist nach einem gewissen Schema in XML-codiert. Es gibt zwei Modelle für das Erstellen von SOAP Messages. Eines erlaubt das Anbringen von Attachments, das andere nicht. Attachments können genutzt werden, um beliebige Binärdaten mit der SOAP Message zu übertragen.

Die SOAP Message wird dann übertragen und auf dem entfernten System ausgewertet. Dieses ruft dann die entsprechenden Methoden auf. Es gibt die Möglichkeiten der Anfrage, bei der die aufrufende Komponente ein Ergebnis als Antwort erwartet, und des Updates, bei der als einzige Antwort eine Quittung bzw. ein Acknowledgement erwartet wird. Beides kann synchron (Der Sender wartet aktiv auf das Ergebnis. Er ist blockiert, solange dieses nicht vorliegt.) oder asynchron (Der Sender wartet nicht aktiv auf das Ergebnis, sondern bearbeitet währenddessen andere Dinge.) Die Ergebnisse werden wieder in Form von SOAP messages, also XML-codiert, zurückgesandt. Eine andere Möglichkeit des Informationsaustausches ist mit *Fire and Forget* gegeben, bei der der Sender die Nachricht einfach absendet, ohne eine Antwort darauf zu erwarten.

Der Vorteil von SOAP ist die Unabhängigkeit von den verwendeten Betriebssystemen und vor allem Programmiersprachen sowie vom verwendeten Übertragungsprotokoll. Sicherheitsanforderungen können durch das Nutzen von sicheren Übertragungsprotokollen wie z.B. HTTPS leicht realisiert werden.

Die Java API for XML-Processing (JAXM) von Sun bietet Klassen für das Benutzen von SOAP mit Java. SOAP ist aber auch für die Interoperabilität zwischen Programmen in beliebigen Programmiersprachen geeignet.

5.2 Metadatenzugriff über OAI

Ein weiterer Ansatz, dem ich in dieser Diplomarbeit nachgehe, ist die Betrachtung eines anerkannten Standardprotokolls zum Zugriff auf die Metadaten eines entfernten Repositories. Diese Metadaten können genutzt werden, um eine Suche auf verteilt vorliegenden Daten mittels Replikation und Indizierung umzusetzen (siehe 3.5.1).

Die Open Archives Initiative hat mit ihrem OAI Protokoll[2] eine standardisierte Möglichkeit entwickelt, auf Repositories Digitaler Bibliotheken zuzugreifen. Dabei werden allerdings lediglich die Metadaten der Repositories angeboten, ein Bezug zu den eigentlichen Dokumentdaten ist aber in aller Regel in den Metadaten

[2] Open Archives Initiative Protocol for Metadata Harvesting [VL01]

enthalten[3]. OAI wird immer mehr akzeptiert und es gibt viele Archive, die jetzt oder in naher Zukunft OAI unterstützen.

Anbietern von Inhalten im World Wide Web ist durch das Protokoll eine standardisierte Möglichkeit gegeben, ihre Inhalte zu veröffentlichen und für eine Recherche bereitzustellen. Sie öffnen damit ihr intellektuelles Eigentum für die Allgemeinheit – was ja schon aus dem Namen "Open Archives" hervorgeht – und erlauben so ein *metadata harvesting*, welches zur Errichtung von Suchdiensten benutzt werden kann. Über das Protokoll selbst ist es nicht möglich in einem Repository, welches einen OAI Service anbietet, 'intelligent" zu suchen. Das Protokoll bietet lediglich Browsing und Navigation in den Inhalten des Repositories.

Trotzdem bietet das Protokoll eine Möglichkeit, Inhalte verschiedener Digitaler Bibliotheken zu durchsuchen. Eine Suche kann bewerkstelligt werden, indem Metadaten der verschiedenen Repositories "geerntet" (harvest) werden und anschließend für einen Suchdienst[4] persistent gehalten werden. Es entsteht damit eine ähnliche Architektur wie bei Suchmaschinen, wo in zwei Stufen Daten gesammelt und indiziert werden. Jedoch ist nur eine Suche auf verteilt vorliegenden Daten möglich, keine echte Verteilte Suche. Ein System, welches über OAI Metadaten "erntet" und für eine Suche bereitstellt kann also auch als OAI Suchmaschine bezeichnet werden. Man findet für ein solches System üblicherweise die Bezeichnung Cross Archive Searching Service.

Man unterscheidet zwischen zwei Klassen von Systemen bezüglich ihrer Beziehung zu einem OAI Service. Es gibt Systeme, die selbst einen OAI Service anbieten, sie werden *Data Provider* genannt. Sie sind also in der Lage, ihre Metadaten zu exportieren. Die Systeme, die OAI Services von Data Providern nutzen und die erhaltenen Metadaten für ihre Services verarbeiten, werden *Service Provider* genannt. Ein Cross Archive Searching Service agiert demnach als OAI Service Provider.

5.2.1 OAI Konzepte

Jedes OAI konforme Repository bietet einen Zugriff auf die Metadaten ihrer Dokumente. Diese Dokumente werden *Items* genannt.

Wenn die Metadaten solcher Items über einen OAI Request abgefragt werden, werden diese in der Form eines OAI *Records* als Response geliefert. Diese Metadaten-Records sind in XML [BPSM00] codiert und dadurch strukturiert.

Für jedes Item eines Repository gibt es einen eindeutigen Identifikator (*unique identifier*), um die Metadaten des Items zu extrahieren. Außerdem besitzt es einen *datestamp*, der den Zeitpunkt der Erstellung bzw. Modifikation des Items beinhaltet.

Ein Repository kann mehrere Metadatenformate unterstützen. Es muss für jedes Item jedoch mindestens Dublin Core (siehe 2.3) unterstützen. Dabei ist auch

[3] Oftmals enthält das Dublin Core Feld *Identifier* eine URL auf das beschriebene Dokument (siehe 2.3).

[4] So ein Suchdienst wird häufig Cross Archive Search Service genannt.

möglich, dass sich das Angebot der vom Repository für die einzelnen Items unterstützten Metadatenformate unterscheidet. Zum Beispiel ist es denkbar, dass die Metadaten eines gewissen Items in Dublin Core und in RFC1807 [LC95], einem anderen bibliographischen Metadatenformat, vorliegen, während dasselbe Repository für ein anderes Item nur Dublin Core anbietet. Die unterstützten Metadatenformate eines Repositories, auch in Verbindung mit einem bestimmten Item, lassen sich über einen OAI-ListMetadataFormats-Request abfragen.

Das OAI Protokoll bietet desweiteren ein optionales Konstrukt, dass es erlaubt, die Items eines Repositories in einer Mengenhierarchie zu gruppieren. Dadurch ist möglich, selektiv auf die Metadaten von Dokumenten nur einer bestimmten Kategorie zuzugreifen. Jedes Repository kann eine hierarchische Organisation ihrer Items anbieten. Jeder Knoten der Hierarchie ist ein *Set*. Die Hierarchie kann mehrere Einstiegspunkte auf oberster Ebene haben (*top level sets*), und ihre darunter liegenden Sets[5] können sich in beliebig vielen Stufen nach unten verfeinern. Ein Item kann einem Set, mehreren Sets oder gar keinem Set zugehörig sein.

Die Set-Hierarchie eines Repositories lässt sich mittels OAI-ListSets-Request abfragen. Beispielsweise erhält man folgende Set-Hierarchie des OAI Repositories von ArXiv [6] als Antwort auf den in HTTP eingebetteten OAI-Request *http://arXiv.org/oai1?verb=ListSets*:

```
<?xml version="1.0" encoding="UTF-8"?>
 <ListSets xmlns="http://www.openarchives.org/OAI/1.1/OAI_ListSets"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.1/OAI_ListSets
  http://www.openarchives.org/OAI/1.1/OAI_ListSets.xsd">
  <responseDate>2002-04-21T08:56:37-04:00</responseDate>
  <requestURL>http://arXiv.org/oai1?verb=ListSets</requestURL>
  <set>
   <setSpec>nlin</setSpec>
   <setName>Nonlinear Sciences</setName>
  </set>
  <set>
   <setSpec>math</setSpec>
   <setName>Mathematics</setName>
  </set>
  <set>
   <setSpec>physics</setSpec>
   <setName>Physics</setName>
  </set>
  <set>
   <setSpec>cs</setSpec>
   <setName>Computer Science</setName>
  </set>
 </ListSets>
```

ArXiv bietet demnach 4 top level sets, die nicht weiter verfeinert sind.

[5] Bewusst habe ich diese nicht Subsets oder Untermengen genannt, denn diese Beziehung muss bei OAI nicht zwingend gelten.

[6] http://arxiv.org

5.2.2 Das OAI Protokoll

Das OAI Protokoll ist komplett in HTTP [FGM+97] eingebettet. Jedem OAI Repository wird eine Basis-URL zugeordnet, welche die Grundlage für den Aufbau eines OAI Requests bildet. Ein Repository nimmt einen HTTP GET oder POST Request auf und antwortet mit einer entsprechenden Response. Der Content-Type der Response ist *text/xml*. Sie ist also in XML codiert.

Ein OAI Request besteht aus der Basis-URL und einer Liste von Argumenten in der Form `Schlüssel=Wert`. Jeder Request muss mindestens das Argument `verb=`*OAIRequest* enthalten, wobei *OAIRequest* eines der unterstützten Requests (siehe unten) ist. Weitere Argumente hängen von dem entsprechenden OAI Request ab.

Ein OAI Repository unterstützt folgende Requests:

GetRecord wird benutzt, um einen einzelnen Record (also die Metadaten eines Items) von dem Repository zu extrahieren. Dafür werden ein *identifier* und ein *metadataPrefix* benötigt.

Identify wird benutzt, um Informationen über das Repository selbst zu erhalten. So wird beispielsweise ein Name für das Repository geliefert.

ListIdentifiers wird benutzt, um die Identifier aufzulisten, dessen Metadaten-Records aus dem Repository extrahiert werden können.

ListMetadataFormats wird benutzt, um die Metadatenformate aufzulisten, die das Repository unterstützt.

ListRecords wird benutzt, um mehrere Records von dem Repository zu extrahieren. Dafür wird ein *metadataPrefix* benötigt.

ListSets wird benutzt, um die Set-Hierarchie eines Repositories abzufragen.

Ein gültiger OAI Request, der in HTTP GET eingebettet ist, ist zum Beispiel *http://dissertationen.hu-berlin.de/OAI-script?*
verb=ListRecords&metadataPrefix=oai_dc.
Er liefert alle Records (bis zu einem gewissen `resumptionToken`[7]) der Humboldt Universität Berlin im Dublin Core Format.

[7] Wenn zum Beispiel sehr große Listen von Records bei `ListRecords` oder Identifiern bei `ListIdentifiers` entstehen, können bei OAI `resumptionToken` für den Datenfluss genutzt werden. Dadurch entsteht eine Kette von OAI-Requests für das stückweise Holen von Metadaten.

5.2.3 Realisierungen eines Cross Archive Search Services

Ein Cross Archive Search Service ist ein Service, der OAI nutzt, um eine Suche auf den OAI Data Providern zu ermöglichen. Dazu repliziert sich der Service in einem Harvesting-Prozess sämtliche oder ausgewählte Metadaten von OAI Data Providern. Diese replizierten Metadaten werden dann auf einer lokalen Instanz gespeichert und für eine Suche indiziert.

Dieser Service stellt also einen typischen Vertreter eines Suchdienstes nach 3.5.1 dar. Es ist also keine echte verteilte Suche, sondern nur eine Suche auf verteilt vorliegenden Daten.

Eigene Implementierung

Im Rahmen dieser Diplomarbeit wurde ein eigener Cross Archive Searching Service konzipiert und implementiert. Der Aufbau, die Funktionsweise und die Implementierung sind im nächsten Kapitel unter 6.2 beschrieben.

my.OAI

my.OAI[8] ist ein Cross Archive Searching Service von FS Consulting Incorporated. Er bietet eine Nutzerverwaltung. Den Nutzern ist es möglich, Systemeigenschaften nach persönlichen Vorstellungen sowie Anfrageergebnisse und Suchanfragen für spätere Nutzung in einem persönlichen Profil abzuspeichern.

Die Volltextindizierung der Metadatenfelder geschieht mit dem MPS Information Server, einem Produkt aus dem eigenen Hause von FS Consulting. Er bietet linguistische Indizes mit Stammwortreduktion.

ARC

ARC[9] ist ein Cross Archive Searching Service, der von der Bibliothek der Old Dominion University[10] in Norfolk, Virginia, USA betrieben wird.

Die Architektur wird in [LMZN01] näher vorgestellt. Auch dieser Service bietet eine Volltextindizierung. ARC nutzt den Oracle InterMedia Server und MySQL full-text search. Als ein zusätzliches Feature bietet dieser Service eine intelligente Fehlerbehandlung. Nicht jeder OAI Data Provider hält sich immer korrekt an den Standard. ARC versucht bei Fehlerfällen die Datensätze durch das Absetzen von OAI-Requests für jeden einzelnen Datensatz einzusammeln. Dadurch entsteht dann aber ein erhöhter Datenverkehr.

[8] http://www.myoai.com
[9] http://arc.cs.odu.edu
[10] http://www.odu.edu

Vergleich

In der folgenden Tabelle sind die 3 kurz vorgestellten Cross Archive Searching Services in Bezug auf die Erfüllung von 4 Eigenschaften gegenübergestellt:

Eigenschaft	own	my.OAI	ARC
intelligente Fehlerbehandlung		?	×
Volltextindizierung	×	×	×
Personalisierung		×	
Ranking		×	×

Tabelle 5.1: Vergleich von Cross Archive Searching Services

Über das Verhalten von my.OAI bei Fehlerfällen beim Harvesting kann ich keine Angaben machen, weil mir diese Informationen nicht zugänglich sind. Wahrscheinlich ist es aber, dass dort eine Fehlerbehandlung erfolgt, da das System zum Beispiel sämtliche Records der Universität Bremen enthielt. Ein Harvesting der Records dieses Repositories brach bei der eigenen Implementierung immer wieder aufgrund des Nutzens nicht erlaubter Character im XML-Datenstrom[11] ab.

Der im Rahmen dieser Arbeit implementierte Cross Archive Searching Service hat nur prototypischen Charakter und kann sich deshalb nicht mit den anderen beiden Vertretern messen. Von den 3 vorgestellten Services ist my.OAI der "Winner". Er bietet die meisten Vorzüge und verdankt das vor allem seiner gut umgesetzten Personalisierungskomponente.

5.3 Z39.50

In diesem Abschnitt soll ein Standardprotokoll für das Information Retrieval vorgestellt werden, welches auf dem Gebiet der Digitalen Bibliotheken bedeutsame Relevanz erlangt hat. Das Z39.50[12] Protokoll ist sowohl bei der ANSI/NISO [z3950] als auch bei der ISO standardisiert.

Der Standard Z39.50 wird durch die Library of Congress[13] vertreten. Die Weiterentwicklung des Standards geschieht innerhalb der Z39.50 Implementors Group (ZIG). Innerhalb der ZIG gibt es jährlich öffentliche Meetings.

Z39.50 ist ein Protokoll für den Zugriff und die Recherche auf Bibliotheksprotokollen. Das Protokoll basiert auf einer typischen Client-Server-Kommunikation, bei der Anwender (Clients) einen Z39.50-Service (Server) nutzen, um in entfernten Bibliotheks-Katalogen zu recherchieren. Der Z39.50 Server nimmt die Anfragen entgegen und gibt diese in passender Form an seine Persistenzschicht weiter.

[11] Man teste beispielsweise *http://elib.suub.uni-bremen.de/cgi-bin/oai?verb=ListRecords&resumptionToken=30sDISS-moai_dc* und stelle fest, dass ein XML-Parser diesen Datenstrom nicht akzeptiert.

[12] Z39.50 ist die Nummer der ANSI-Norm.

[13] http://lcweb.loc.gov

Es muss also eine Umwandlung von Z39.50-Anfragen in solche erfolgen, die von dem Lokalen Dokumentenserver verstanden werden.

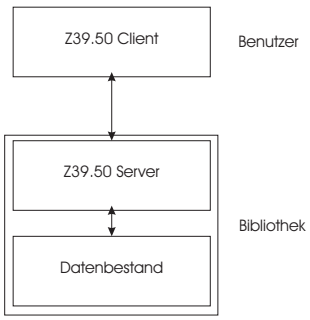

Abbildung 5.2: Client-Server-Kommunikation mit Z39.50

Z39.50 ist konzipiert worden um Interoperabilität zwischen verschiedenen heterogenen Datenquellen zu schaffen: Die Datenbanksysteme und ihre Benutzerschnittstellen können sich unterscheiden. Die benutzten Bibliotheksformate für die Metadatenpersistierung können sich unterscheiden. Z39.50 versucht deshalb darüber zu abstrahieren und die Datenquellen zu homogenisieren.

Z39.50 definiert standardisierte Zugriffsmethoden über festgelegte Attribute Sets, Anfragen und dessen Syntax und die Möglichkeiten auf Datensätze zuzugreifen. Jeder Z39.50 Server hat ein Profil, das beschreibt, welche Funktionen des Standards umgesetzt sind, wie der Zugriff zu erfolgen hat, welche Attribute verfügbar sind, welche bibliothekarischen Metadatenformate werden unterstützt etc.. Die wenigsten Z39.50 Server erfüllen jegliche Funktionalitäten, die im Standard vorgesehen sind. Einige von ihnen veröffentlichen ihr Profil, z.B. die Deutsche Bibliothek auf ihrer Webseite, die meisten jedoch nicht.

Das Protokoll ist verbindungs- und sitzungsorientiert. Ein Client erstellt eine Verbindung zum Server. Diese bleibt solange erhalten, bis sie wieder explizit beendet wird oder vom Server getrennt wird. Dies kann zum Beispiel bei Time Outs erfolgen.

Z39.50 erlaubt vom Standard her mehrere Anfragetypen und auch Arten von Attribute Sets. Die verbreitetsten sind jedoch *Query Type-1* und das *Bib-1* Attribute Set[14], die im Folgenden kurz vorgestellt werden sollen:

Eine Query hat einen oder mehrere Operanden, die mit Booleschen Operatoren verknüpft sind (`AND, OR, AND-NOT`). Jeder Operand ist ein Suchausdruck, dieser muss aus 7 Bestandteilen aufgebaut sein. Alle Bestandteile außer 0. sind Nummern, dessen Semantiken in Bib-1 festgelegt sind:

- **0. Term**, gibt an, wonach man sucht;

[14]Beide sind in der Z39.50 Spezifikation [z3950] definiert.

- **1. Use Attributes**, gibt an, auf welchen Objekten man Anfragen stellen möchte, z.B. Personenname = 1, Name einer Körperschaft = 2, ... ;
- **2. Relation Attribute**, gibt die Beziehung zwischen dem Term und dem Objekt an, die bestehen soll, z.B. gleich = 3, größer als = 5, phonetisch ähnlich = 100, ... ;
- **3. Position Attributes**, gibt an, wo der Term in dem Objekt zu finden sein soll, z.B. an erster Stelle = 1, an beliebiger Stelle = 3, ... ;
- **4. Structure Attributes**, gibt an, wie der Term für die Anfrage betrachtet werden soll, z.B. als Wortfolge = 1, als Wörter = 2, als Datum = 100, ... ;
- **5. Truncation Attributes**, gibt an, ob und wenn ja wie eine Wortstutzung angewandt werden soll, z.B. Stutzung an der rechten Seite = 1, keine Stutzung = 100, ... ;
- **6. Completeness Attributes**, gibt an, inwieweit der Term das im Use Attribute angegebene Objekt ausfüllen soll, z.B. Teil von Unterfeldern = 1, ganze Unterfelder = 2, ganze Felder = 3.

Eine Beispielanfrage auf einen Z39.50-Server wäre z.B.
`("Mozart", 1:1003, 2:3, 3:1, 4:1, 5:100, 6:1)`
`("Mozart, Leopold", 1:1003, 2:3, 3:3, 4:101, 5:100, 6:2)`
`AND-NOT`.

Sie liefert die Records von Autoren *Mozart*, die aber nicht *Leopold Mozart* sein dürfen.

Z39.50 ist ein sehr komplexer Standard, der zwar eine Interoperabilität auf mehreren Ebenen zwischen heterogenen Datenquellen ermöglicht. Jedoch ist es aufgrund der Komplexität von Z39.50 sehr aufwendig, alle vorgesehen Features von Z39.50 umzusetzen. Einige Anbieter verzichten deshalb wie bereits erwähnt auf die Implementierung bestimmter Z39.50 Features.

Über Z39.50 Services lassen sich natürlich auch verteilte Suchen aufbauen. Ein Beispiel für eine Realisierung einer verteilten Suche über Z39.50 ist das Gateway der Deutschen Bibliothek[15].

Abbildung 5.3 zeigt die Funktionsweise solcher Gateways.

Diese Gateways sind Web-Anwendungen, die Z39.50-Clients (Z-Clients) integriert haben, und über diese parallele Suchen auf mehreren Datenquellen realisieren. Der Benutzer arbeitet an einem Web-Browser und formuliert seine Anfragen, dafür nutzt er eine grafische Nutzerschnittstelle. Er muss also keine Z39.50-Spezifika kennen. Er kann ferner auswählen, welche Datenquellen mit in eine Anfrage einbezogen werden sollen. Dann bearbeitet das Gateway die Anfrage und wandelt sie in Z39.50-Anfragen um. Diese werden dann parallel an die betreffenden Z39.50-Server geschickt. Die Ergebnisse werden wieder in dem Gateway

[15] http://z3950gw.dbf.ddb.de

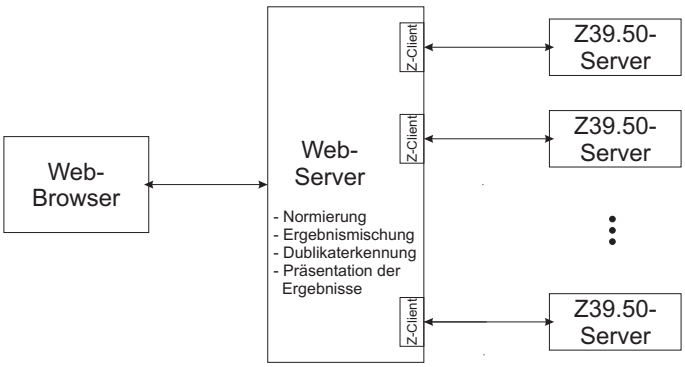

Abbildung 5.3: Verteilte Suche über einen Z39.50 Gateway

gesammelt. Jetzt kann eine Dublikatserkennung und eine Mischung der Ergebnisse vorgenommen werden, bevor dem Benutzer ein globales Ergebnis präsentiert wird.

5.4 Federated Search des Enterprise Information Portals

Das *Federated Search* des IBM Enterprise Information Portals (EIP) bietet einen weiteren Ansatzpunkt für eine verteilte Suche. Das EIP Framework besitzt dafür eine spezielle Komponente, den sogenannten Federated Datastore. Außerdem besitzt er Connectoren zu anderen Datenquellen (Native Datastores).

In der Version EIP Version 8[16] werden als wichtigste Datenquellen die folgenden unterstützt:

- Content Manager Version 8 (mit seinen Library Servern und Resource Managern);
- Content Manager Version 7 (mit seinen Library Servern und Object Servern);
- DB2;
- JDBC und ODBC-Quellen, dadurch auch andere Datenbanksysteme;
- Inhalte im Internet, Extranet, Intranet, in Lotus Notes Datenbanken, Filesystemen, etc. mit Hilfe des Web Crawlers.

Der Web Crawler ist ein Feature des EIP's, welches einen Zugriff auf die Inhalte von Webservern, FTP-Server, Lotus Notes Datenbanken, lokalen Filesystemen, etc. ermöglicht. In einem Crawling Prozess wird ein Data Mining der entsprechenden Datenquelle initiiert. Das heißt, dass sein Inhalt ergründet wird. Beispielsweise können bei HTML-Inhalten Informationen wie URL, Titel, Body, Zeit der letzten

[16] Die Version befindet sich zur Zeit noch in der Beta-Phase.

Modifizierung und die Inhalte der Metatags erschlossen werden. Desweiteren kann der Web Crawler Links in HTML-Seiten verfolgen und die dortigen Ressourcen mit in das Data Mining einbeziehen.

Man kann außerdem zusätzliche Connectoren für weitere Datenquellen programmieren. Dazu helfen dem Programmierer das EIP Connector Toolkit und einige Beispiele.

In seinem EIP Framework kann der Nutzer nun solche entfernten Datenquellen registrieren. Dazu wird ihnen eine eindeutige Serverkennung gegeben. Außerdem müssen gewisse Parameter angegeben werden, die für einen entfernten Zugriff nötig sind. Das sind in der Regel Angaben über Lokation der entfernten Datenquelle sowie zur Autorisierung.

Hat man eine entfernte Datenquelle im System registriert, so startet man einen *Explore*-Request für diese Datenquelle. Über den Connector wird nun eine Verbindung aufgebaut und es werden sämtliche verfügbaren Datenkatalog-Informationen der entfernten Quelle ausgelesen. Das sind die Informationen dazu, welche Daten es dort gibt, in welchen Containern sie gebündelt sind und welche Attribute gespeichert sind. Es wird also das Data Dictionary der Quelle empfangen und gespeichert. Konkret sind das:

- bei Content Manager Version 8 Datenquellen: die Namen der existierenden Item Types, ihre Struktur untereinander und ihre Attribute
- bei Content Manager Version 7 Datenquellen: die Namen der existierenden Indexklassen und ihre Attribute
- bei DB2: die Tabellendefinitionen der in der Datenbank gespeicherten Tabellen
- bei JDBC und ODBC-Quellen: analog
- bei Webressourcen: Inhalte von FTP-, HTTP-, HTTPS-Servern etc.
- bei Lotus Notes: Informationen über auf dem Server gespeicherte Lotus-Notes-Datenbanken
- bei Filesystemen: Inhalte der Filesysteme

Innerhalb des Federated Datastores können nun Federated Entities mit ihren Attributen modelliert werden. Durch diese Modellierung entsteht das konzeptuelle Schema des Federated Datastores. Dieses bildet dann die Grundlage eines zu durchsuchenden Archivs. Der große Gewinn dabei ist nun, dass die Daten, die in diesen Federated Entities zugeordnet werden, nicht lokal gespeichert sind, sondern auf den vorher registrierten Datenquellen verstreut liegen können. Der Web Crawler legt die gesammelten Informationen in einer DB2-Datenbank ab.

Dazu wird das konzeptuelle Schema des Federated Datastore auf die der Native Datastores gemappt. Zum Beispiel kann ein Federated Entity auf eine bestimmte Indexklasse auf einem CMV7-Server und auf einen bestimmten Item Type auf

einem CMV8-Server und auf eine Tabelle in einer Oracle Datenbank gemappt werden. Seine Attribute werden dementsprechend auf gewisse Attribute im Native Datastore gemappt. Abbildung 5.4 zeigt ein Beispiel für ein solches Mapping:

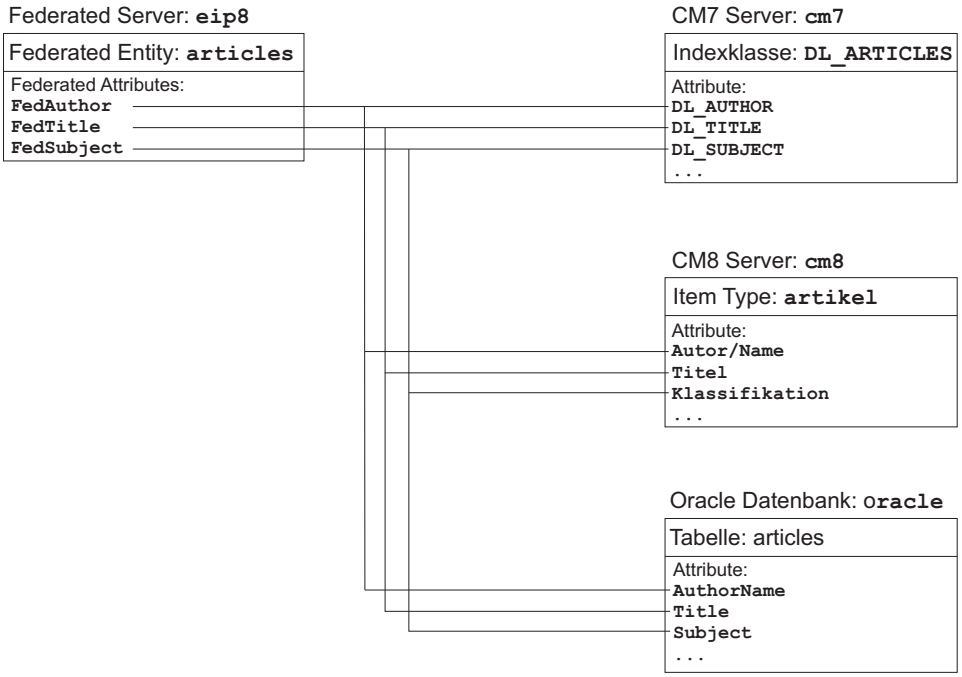

Abbildung 5.4: Verteilte Suche mit Federated Search

Hat der Nutzer nun einmal solche Federated Entities angelegt, so kann er für diese statische Such-Templates anlegen. So ein Such-Template bewerkstelligt bei einem Zugriff eine Suche auf den gemappten Entities und liefert das Ergebnis für eine Verarbeitung zurück. Wenn man ein Such-Template anlegt, so muss man festlegen, wonach man suchen möchte, indem man einen Such-String angibt und wer Zugriff auf das Template hat. Man kann nur ein Entity pro Template nutzen, aber man kann ein Entity in mehreren Templates benutzen.

Für das Erstellen des Templates werden die Attribute des Entities entsprechend eines Suchkriteriums kombiniert. Darüberhinaus können die Inhalte des Entities, sofern es ein Textdokument ist, mit in die Suche einbezogen werden. Für die Formulierung des Suchkriteriums stehen verschiedene Operatoren zur Verfügung, wie sie beispielsweise aus relationalen Datenbanken bekannt sind. Für den Bezug auf den Textinhalt eines Textdokument-Entities gibt es besondere Operatoren. Die unterstützten Operatoren hängen von den Native Datastores und den gemappten Attributen ab.

Hier ist ein Beispiel für einen Suchstring in einem Such-Template:

```
(FedAuthor equals "Gauss") or
(FedTitle like "%algebra%") or
(FedSubject like "%mathematics%")
```

Solche Such-Templates bieten einen guten Ansatzpunkt für immer wiederkehrende Anfragen statischer Natur. Es ist vergleichbar mit Views in relationalen Datenbanken. Wenn man jedoch aus einem Anwendungsprogramm heraus eine Anfrage dynamisch aufbauen möchte, so kommt man mit den Such-Templates nicht weit. Für dynamische Anfragen auf Federated Entities stehen dem Anwendungsentwickler daher APIs[17] bereit, die eine dynamische Anfrageverarbeitung erlauben.

Durch die APIs stehen dem Programmierer vielseitige Funktionen für eine Suche in dem Federated Datastore zur Verfügung. Die wichtigsten am Beispiel der Java-API sind im Folgenden aufgelistet:

- Aufbau einer Verbindung zu einem Federated Datastore mit
  `DKDataStoreFed.connect(String datastoreName, String userName, String authentication, String connectString)`
- Auflisten der definierten Entities und Such-Templates mit
  `DKDataStoreFed.listEntities()` und `listSearchTemplates()`
- Erzeugen parametrischer Anfragen und Textsearch-Anfragen auf den Datastore (Klasse `dkFederatedQuery`) unter Verwendung von
  `DKDataStoreFed.createQuery(String command, short commandLangType, DKNVPair[] params)`
- Absetzen dieser Queries mittels
  `dkFederatedQuery.execute(DKNVPair[] params)`
- Auffangen der Anfrageergebnisse in `DKResults` (Alle Anfrageergebnisse werden auf einmal geholt.) oder `dkResultSetCursor` (Anfrageergebnisse werden nach dem Cursorprinzip nach und nach geholt.[18]) mittels
  `dkFederatedQuery.result()` oder `dkFederatedQuery.resultSetCursor()`
- Casten des Anfrageergebnisses auf eine `dkFederatedCollection`, um auf dem Ergebnis iterieren zu können. `dkFederatedCollection` enthält eine Liste von `dkCollection`-Objekten.
- Jedes `dkCollection`-Objekt der Liste enthält die Anfrageergebnisse eines bestimmten Native Datastores.
- Iterieren über `dkFederatedCollection` mittels `dkFederatedIterator`
- Iterieren über `dkCollection` mittels `dkIterator`

[17] In Version 7 waren es Java, C++, ActiveX und Dynamic Page Builder. Für Version 8 sind bisher nur C++ und Java vorgesehen.

[18] Nur wenn der zugehörige Native Datastore das unterstützt.

- Schließen der Verbindung mit `DKDataStoreFed.disconnect()`

Im Folgenden ist die Query-Syntax für eine parametrische Anfrage an den Federated Datastore nach Dokumenten des Autors *Gauss* angegeben:

```
PARAMETRIC_SEARCH=(ENTITY=articles,
                   MAX_RESULTS=10,
                   COND=(FedAuthor == "Gauss")
                   );
                   OPTION=(CONTENT=YES)
```

Federated Search bietet also einen Ansatzpunkt für eine echte verteilte Suche, dessen Verarbeitung wirklich On-The-Fly erfolgt.

5.5 Zusammenfassung

Es wurden hier 4 Ansätze vorgestellt, die eine Realisierung eines verteilten Such- / Integrationsdienstes ermöglichen. Zum einen waren das grundlegende Kommunikationsarten, wie sie in verteilten Anwendungen vorkommen. Dafür wurden die Low-Level-Netzwerk-Kommunikation, die HTTP- und HTTPS-Kommunikation, CORBA, RMI und SOAP aufgeführt. Als nächstes wurde das Standardprotokoll OAI vorgestellt, welches ein Metadata-Harvesting ermöglicht. Es erlaubt dadurch eine Suche auf replizierten Metadaten. Z39.50 ist ein Standardprotokoll, das für die Suche auf Bibliothekskatalogen entwickelt wurde. Es bietet Interoperabilität und ermöglicht dadurch eine einfache Homogenisierung der entfernten Datenquellen. Es erlaubt die Entwicklung echter verteiler Anfragen “On-the-Fly”. Das “Federated Search”-Feature des EIPs bietet ebenfalls eine echte “On-the-Fly”-Verarbeitung und die Kopplung verschiedener heterogener Backends an einen virtuellen Datastore.

Die hier vorgestellten Ansätze für eine Realisierung einer verteilten Suche schließen sich gegenseitig nicht aus. Es wäre sogar mehr als wünschenswert viele verschiedene Systeme mit unterschiedlichen Zugriffsarten zu einem globalen Wissensverbund zu integrieren. Ein verteilter Such- / Integrationsdienst sollte mehrere entfernte heterogene Digitale Bibliotheken mit verschiedenen Zugriffsarten integrieren und sollte sich nicht auf bestimmte Zuriffsarten beschränken.

Kapitel 6

Konzeption

Nachdem im letzten Kapitel grundlegende Ansätze für eine Realisierung eines verteilten Such- / Integrationsdienstes diskutiert wurden, werden nun in diesem Kapitel zwei Ansätze konkreter verfolgt.

Für das MyCoRe-System (siehe 2.5) wurde ein verteilter Suchdienst konzipiert und implementiert. Ebenso wurde ein Cross Archive Searching Service, der auf dem OAI-Protokoll basiert, konzipiert und implementiert, der schon im letzten Kapitel mit anderen ähnlichen Lösungen verglichen wurde.

6.1 Ein verteilter Suchdienst für MyCoRe

Das MyCoRe-Projekt ist schon in 2.5 vorgestellt worden. Für die MyCoRe-Plattform habe ich Klassen für eine verteilte Suche über mehrere MyCoRe-Installationen implementiert. Die Quellcodes sind Open Source. Die Dokumentationen zum Projekt sind ebenfalls öffentlich. Über die Projekt-Homepage[1] kann man auf das CVS-Repository des Projektes zugreifen und Dokumentation sowie Quellcodes einsehen.

6.1.1 MyCoRe Konzepte

Zunächst möchte ich ein paar MyCoRe-Konzepte näher vorstellen. Dabei werde ich insbesondere auf die Datenmodellierung eingehen. Es werden die Möglichkeiten und die Art und Weise ihrer Umsetzung beschrieben.

MyCoRe speichert *Multimediale Objekte*, und zwar versteht man im Rahmen von MyCoRe unter solchen multimedialen Objekten beliebige Objekte wie Texte, Bilder, Videos, Audio, etc. – also die eigentlichen Dokumente im Sinne der Digitalen Bibliothek – inklusive ihrer beschreibenden Daten. Diese beschreibenden Daten und zugehörige Dokumente, die im Wortgebrauch von MyCoRe Derivate genannt werden, werden also naheliegenderweise als eine Einheit, ein multime-

[1] http://www.mycore.de

diales Objekt, betrachtet. Einem multimedialen Objekt können 0 bis n Derivate zugeordnet werden.

Ein Beispiel für ein multimediales Objekt ohne Derivate wäre zum Beispiel ein Metadatensatz zu einer Person. Ein Film, der aus zwei Datei-Fragmenten zusammengesetzt ist, ist ein Beispiel für einen Anwendungsfall für ein multimediales Objekt mit 2 Derivaten. Wenn Dokumente in mehreren verschiedenen Formaten (z.B. Postscript, Latex, PDF) abgespeichert werden sollen, dann ist das ein weiteres Szenario für den Einsatz von multimedialen Objekten mit mehr als einem Derivat.

Die multimedialen Objekte werden in ihrer Gesamtheit in MyCoRe persistent gespeichert. Ein multimediales Objekt kann auch Beziehungen zu anderen besitzen.

Die beschreibenden Daten eines multimedialen Objektes werden auch Objektdaten genannt und werden in 3 Gruppen eingeteilt:

- Die Strukturdaten enthalten alle Informationen über Beziehungen des multimedialen Objektes zu anderen. Über solche Beziehungen lassen sich in MyCoRe beispielsweise komplexe zusammengesetzte Objekte bilden, wie z.B. ein Buch, das aus mehreren Kapiteln besteht, wobei ein autonomer Charakter jedes einzelnen Kapitels zum Ausdruck gebracht werden soll. Desweiteren sind die Verweise auf die einzelnen Derivate des multimedialen Objektes in den Strukturdaten vermerkt.

- Die eigentlichen Metadaten, die verschiedenste Parameter mit verschiedensten Datentypen haben können. Verweise auf andere multimediale Objekte können ebenfalls in den Metadaten auftauchen.

- Die Managementdaten sind rein administrativer Natur. Sie ermöglichen das Ablegen von beliebigen Informationen in einem Flag-Teil und können beispielsweise für die Steuerung von Workflow-Prozessen genutzt werden. Diese werden in der Regel von der Programmlogik oder den Systemadministratoren verwendet und sind für die eigentlichen Benutzer uninteressant. Desweiteren sind Datumsangaben Inhalt der Managementdaten. Dazu gehören zum Beispiel das Datum der Kreierung (`createdate`), der letzten Modifizierung (`modifydate`) oder der Gültigkeit des Objektes (`validfromdate, validtodate`).

Für jedes multimediale Objekt (MyCoRe-Objekt) gibt es eine XML-Repräsentation seiner Objektdaten. Die Struktur dieses XML-Dokumentes wird über ein XML-Schema festgelegt und ist für jede MyCoRe-Installation anpassbar. Beim Aufbau der Objektdaten stehen dabei durch MyCoRe gewisse Grundtypen, wie z.B. `Boolean`, `Date` oder `Number` zur Verfügung. Die Beziehungen zu anderen MyCoRe-Objekten bzw. zu den eigentlichen Dokumenten werden über den XLink-Mechanismus [DMO01] abgebildet. Im Anhang A.1 und A.2 sind ein komplettes

MyCoRe-Objekt und das zugehörige Schema als ein Beispiel angegeben. Es ist ein Beispieldokument aus dem Papyri-Projekt[2].

Die Datenmodellierung für die multimedialen Objekte in MyCoRe ist detaillierter in [KHK02] nachzulesen.

Ergebnisse von Anfragen sind Listen von multimedialen Objekten. Für die Resultatslisten gibt es wiederum eine XML-Repräsentation. In ein gewisses Grundgerüst werden die XML-Repräsentationen der zum Resultat gehörenden multimedialen Objekte eingebettet. Ein Beispiel für eine solche Resultatsliste ist im Anhang A.3 angegeben.

Die Nutzung von etablierten Standards aus der XML-"Familie" ist natürlich von großem Vorteil. Für die Darstellung von solchen Objekten kann jetzt einfach XSLT [Cla99] und für eine Anfrage die Query Language XQuery [BCF+02] genutzt werden.

MyCoRe nutzt für die Darstellung von multimedialen Objekten bzw. von Listen mehrerer solcher Objekte bei Resultatslisten XSLT. Die Funktionsweise ist in Abbildung 6.1 dargestellt.

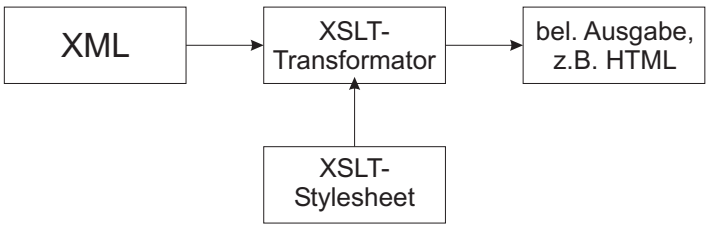

Abbildung 6.1: Transformation mittels XSLT

Ausgangspunkt ist ein XML-Dokument. Dieses wird zusammen mit einem XSLT-Stylesheet von einem XSLT-Transformer geparst. Der Transformer wandelt den Inhalt des XML-Dokumentes entsprechend den Regeln im Stylesheet in ein anderes Format um. Die Ausgabe kann nahezu beliebig sein. Typisch ist die Umwandlung in HTML zur Darstellung des Dokumentes in einem Browser. Denkbar sind aber auch andere Formate, wie z.B. ein anderes XML-Dokument oder ein ASCII-Text. Beispiele für ein XSLT-Stylesheet und für die über dieses Stylesheet konfigurierte Ausgabe sind ebenfalls im Anhang in den Abschnitten A.4 und A.5 angegeben.

Der Vorteil in der Nutzung von XSLT liegt in der sehr guten Anpassbarkeit. Es ist möglich die Darstellungsweise einer großen Menge von XML-Dokumenten über ein einziges Stylesheet zu definieren.

Anfragen innerhalb von MyCoRe werden in XQuery formuliert. Diese zielen auf die XML-Repräsentationen der gespeicherten multimedialen Objekte ab. XQuery

[2] Das Papyri-Projekt (http://papyri.dl.uni-leipzig.de) ist ein Verbundprojekt der Universitäten Halle, Jena und Leipzig. Es befasst sich mit der digitalen bibliothekarischen Archivierung von Papyrusschriften und nutzt dabei MyCoRe als Implementierungskern.

ist eine mittlerweile etablierte einfache und präzise XML-Anfragesprache. Ein Beispiel für eine XQuery-Anfrage in MyCoRe sei hier gegeben:

```
metadata[.="*XML*"]
```

Diese Anfrage gibt eine Liste aller multimedialen Objekte zurück, bei denen irgendwo in den Metadaten der Term `XML` auftaucht. Die Syntax, die XQuery für Pfadausdrücke nutzt, ist an den XPath-Standard [CD99] angelehnt. Der . steht dabei beispielsweise für den aktuellen Knoten.

MyCoRe implementiert XQuery im Moment leider noch nicht komplett. Es sind bisher lediglich Anfragen über Pfadausdrücke möglich. Die Implementierungen der anderen Anfragemöglichkeiten, die XQuery vorsieht, soll in naher Zukunft durch die MyCoRe-Community vorgenommen werden.

6.1.2 MyCoRe-Architektur

Abbildung 6.2 zeigt die Architektur des MyCoRe-Systems.

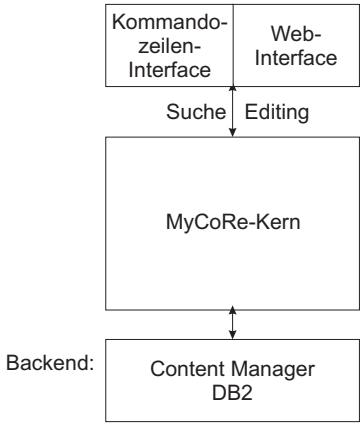

Abbildung 6.2: Die MyCoRe-Architektur

Die Persistierung übernimmt ein Backend. In MyCoRe ist es konzeptionell vorgesehen, mehrere Backends zu unterstützen. Für Content Manager Version 7 sind bereits die Persistenzierungsklassen implementiert. Die Klassen für Version 8 befinden sich gerade in der Entwicklung.

Der MyCoRe-Kern ist in Java implementiert. Er hat Schnittstellen zur Persistenzschicht über ein Java-Interface[3], für welches es mehrere Implementierungen je nach Backend geben wird, und Schnittstellen zu den Benutzern nach außen. Es gibt ein Kommandozeilen-Interface und ein Web-Interface, das aus einer Menge von Java-Servlets besteht. Diese Interfaces erlauben den Benutzern das Editing nach

[3] mycore.datamodel.MCRObjectPersistenceInterface

erfolgter Authentifizierung. Sie können damit neue Objekte erstellen, vorhandene modifizieren oder löschen sowie in den MyCoRe-Repositories recherchieren.

Abbildung 6.3 zeigt den Teil des MyCoRe-Kerns, der für die Persistierung der multimedialen Objekte verantwortlich ist.

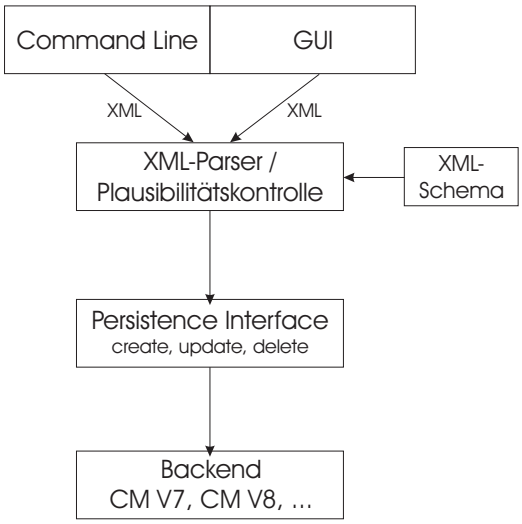

Abbildung 6.3: Teil des MyCoRe-Kerns für die Persistierung der multimedialen Objekte

Über das Kommandozeilen-Interface und über das grafische Interface muss sich zuerst angemeldet werden. Ein Benutzermanagement prüft die Zugangsrechte von den einzelnen Nutzern. Danach können gewisse Editing-Befehle, zum Beispiel `create`, `update` und `delete`, ausgelöst werden, welche sich auf die Persistenzschicht von MyCoRe auswirken sollen. Außerdem werden über die Interfaces die XML-Datenströme, die die zugehörigen multimedialen Objekte repräsentieren, als Parameter geladen. Sie werden dann von einem XML-Parser (Xerces[4]) geparst. Dabei werden sie auf ihren korrekten Aufbau geprüft. Das XML-Dokument wird unter Zuhilfenahme des zugehörigen XML-Schemas validiert. Es entsteht dabei ein DOM-Objekt[5], welches die Inhalte und die Struktur des XML-Dokumentes im Hauptspeicher der Java Virtual Engine repräsentiert. Dieses wird dann nochmals auf Plausibilität geprüft. Die wichtigste Überprüfung ist dabei, ob die Rechte für das Einfügen oder eine Manipulation von multimedialen Objekten vorliegen. Nachdem die Plausibilität erfolgreich kontrolliert wurde, wird die entsprechende Operation letztendlich über das Persistence Interface im Backend vollzogen.

Abbildung 6.4 zeigt den Teil des MyCoRe-Kerns, der für die Anfragebearbei-

[4] *Xerces* ist ein XML-Parser-Paket der Apache Software Foundation. (siehe http://xml.apache.org)

[5] *Document Object Model* [W3C98]: Eine Programmierschnittstelle für HTML- und XML-Dokumente.

tung verantwortlich ist.

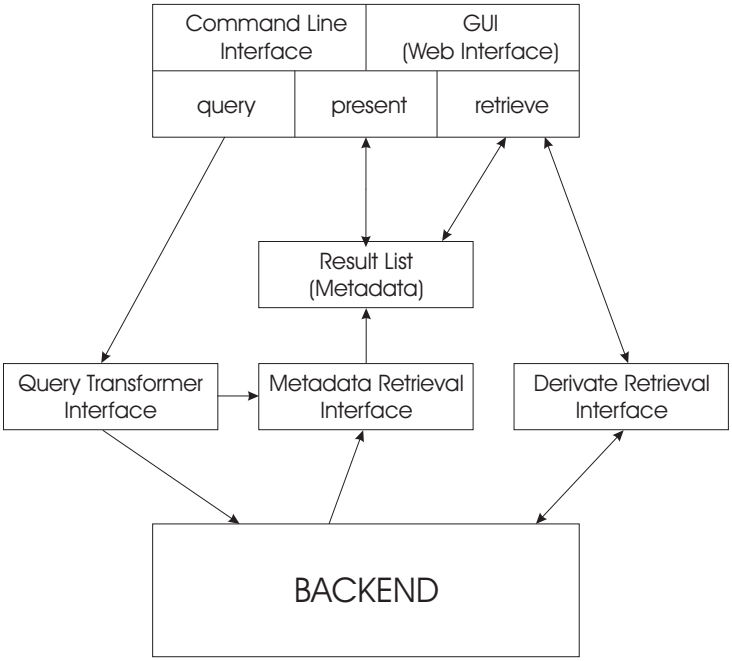

Abbildung 6.4: Teil des MyCoRe-Kerns für die Anfragebearbeitung

Über die Interfaces werden bezüglich Anfragebearbeitung 3 wesentliche Aufgaben gesteuert, das Anfragen (*query*), die Darstellung (*present*) und das Retrieval der Derivate (*retrieve*). Ein *Query Transformer Interface* nimmt die XQuery-Anfragen entgegen und wandelt diese auf die passende Anfragesprache des Backends um. Die Ergebnisse werden dann an über das *Metadata Retrieval Interface* in einer Ergebnisliste (*Result List*) gesammelt. Diese Ergebnisliste kann dem Benutzer nun präsentiert werden. Dabei kommt XSLT zum Einsatz. Der Nutzer kann sich für das Retrieval eines bestimmten Derivates entscheiden. Die entsprechenden Requests werden über das *Derivate Retrieval Interface* an das Backend geleitet.

6.1.3 verteilte Suche in MyCoRe

Die eben beschriebene Anfragebearbeitung muss nun um die Komponenten für eine verteilte Suche erweitert werden. Jede MyCoRe-Instanz braucht dazu eine Komponente, die als Schnittstelle nach außen fungiert, einen *Remote Query Server (RQS)*, und eine Komponente, die auf einen entfernten RQS zugreifen kann, einen *Remote Query Client* (RQC).

Abbildung 6.5 zeigt die entstandene Architektur am Beispiel einer verteilten Suche über 4 entfernte MyCoRe-Instanzen.

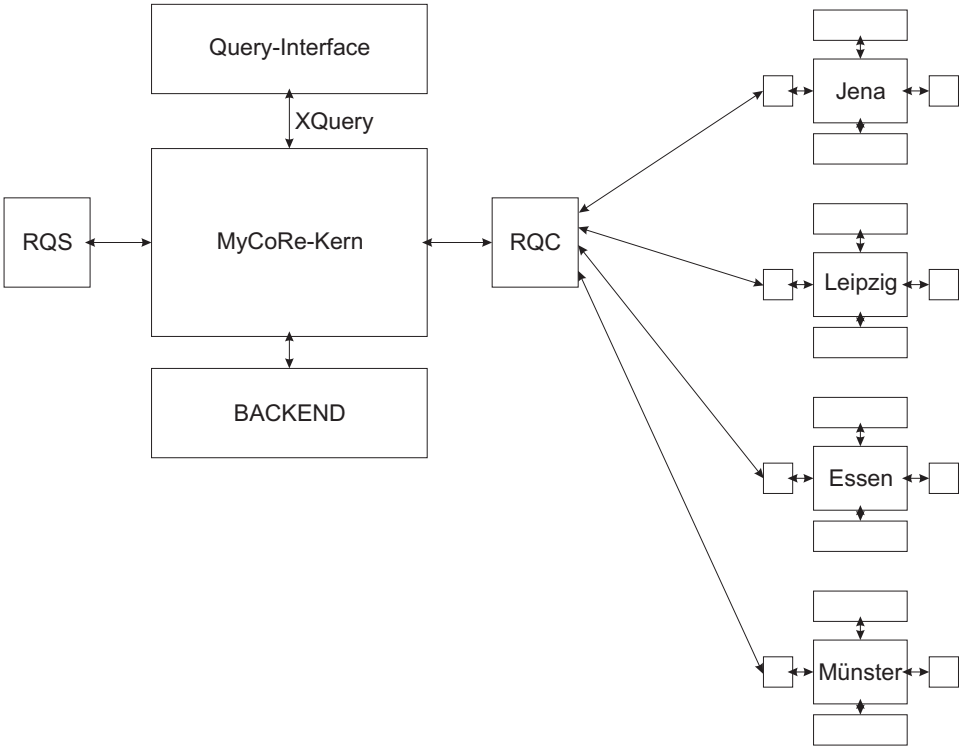

Abbildung 6.5: verteilte Suche in MyCoRe

Der RQS und der RQC sind mit dem MyCoRe-Kern verbunden. Über den RQS kann eine MyCoRe-Instanz Anfragen von entfernten Systemen entgegen nehmen. Diese werden dann lokal ausgewertet und die Ergebnisse werden, wenn die Zugriffsrechte das erlauben, an die Instanz zurückgesandt, die die Anfrage gestartet hat.

Der RQC hat die Aufgabe, mit entfernten Systemen Kontakt aufzunehmen, die Anfragen zu delegieren und die Ergebnisse an den MyCoRe-Kern weiterzugeben, wo dann letztendlich die Ergebnisse der einzelnen Instanzen gemischt werden. Eigentlich gibt es nicht nur einen RQC pro MyCoRe-Instanz. Für jedes entfernte MyCoRe-System wird ein RQC in Form eines Threads instanziert.

Über die Kommunikation zwischen RQC und RQS sollen nicht nur Anfragen und Ergebnisse sondern auch Daten zur Steuerung der Anwendungslogik übertragen werden. Das können zum einen Rechteinformationen bei Authorisationsprozessen sein. Sinnvoll wäre auch eine Steuerung von stückweiser Übertragung von Resultatslisten, wenn sie sehr groß sind. Dazu werden die Listen in Form von Handles gehalten, es kann eine bestimmte Zeit immer wieder darauf zugegriffen werden, ohne die Anfrage erneut am jeweiligen Backend zu vollziehen.

6.1.4 Implementierung und Konfiguration

Die im Rahmen dieser Arbeit entstandene Implementierung realisiert die eben vorgestellte Anfragebearbeitung im verteilten Szenario. Rechtemanagement und Handle-Processing sind dabei aber noch nicht realisiert.

Die Implementierung geschah nach Vorgabe von MyCoRe in Java. Es wurden drei Arten der Kommunikation zwischen zwei entfernten Systemen implementiert:

- Low-Level-Netzwerk-Kommunikation über TCP-IP-Sockets
- HTTP-Kommunikation
- HTTPS-Kommunikation

Für die erste Art der Kommunikation über Sockets muss auf der RQS-Seite ein Server-Prozess gestartet werden. Die Klasse `MCRRemoteQueryServer` aus dem `mycore.communication`-Package verkörpert diesen Server-Prozess. Die Startparameter seien hier gegeben:

```
start - start RQS
stop  - stop RQS
restart - restart RQS or start if not running
status - tests whether RQS is running
help  - this screen
```

Ein Aufruf von

```
java mycore.communication.MCRRemoteQueryServer start
```

startet also den RQS. Wichtig ist dabei nur, dass die `CLASSPATH`-Variable richtig gesetzt ist (siehe MyCoRe-Dokumentation).

Der RQS für Socket-Kommunikation braucht noch einige Parameter, um zu starten. Diese werden in der `mycore.properties`-Datei abgelegt. Ein Ausschnitt aus der Konfiguration für den RQS sei hier gegeben:

```
# Local Server Connections
MCR.communication_rqs_socket_port=4444
MCR.communication_rqs_socket_workdir=/MyCoRe/workdir
```

Der Server hört nach Starten auf Port 4444 und benutzt als Arbeitsverzeichnis `/MyCoRe/workdir`. Dort wird eine Semaphore gespeichert, die angibt, ob der Server läuft, und wenn ja, auf welchem Port.

Auf der anderen Seite muss der RQC konfiguriert werden, der auf einen entfernten RQS per Socket-Verbindung zugreifen soll. Er benutzt beispielsweise folgende Parameter in der `mycore.properties`:

```
# Configuration for the host with alias corfu (Socket Connection)
MCR.communication_corfu_class=mycore.communication.MCRSocketCommunication
MCR.communication_corfu_host=corfu.informatik.uni-rostock.de
MCR.communication_corfu_port=4444
```

Der erste Parameter gibt die Kommunikationsart an (hier Socket Verbindung). `mycore.communication.MCRSocketCommunication` ist dabei eine spezielle Implementierung eines Java-Interfaces, für welches die Implementierung dynamisch zur Laufzeit durch die Konfiguration gewählt wird. Der zweite Parameter

steht für den Host-Namen und der dritte für den Port, auf dem der Server hört.Über diese drei Parameter ist das Alias `corfu` konfiguriert. Es werden normalerweise mehrere Aliases konfiguriert, und zwar für jede entfernte MyCoRe-Instanz, die in eine verteilte Suche eingebunden werden soll.

Für die HTTP und HTTPS Kommunikation fungiert ein Java-Servlet als RQS. `mycore.communication.MCRRemoteQueryServlet` muss dafür von einem Web Application Server (z.B. Tomcat oder Websphere) eingebunden werden. Wird HTTPS-Support gewünscht, so muss der Web-Server dafür konfiguriert werden.

Die RQC-Seite bildet die Klasse `MCRServletCommunication`. Hier ist eine Konfiguration für zwei Aliase, eines mit HTTP-Kommunikation und das andere mit HTTPS-Kommunikation gegeben:

```
# List of the Remote Server aliases
MCR.communication_hostaliases=leipzig,essen

# Configuration for the host with alias leipzig (HTTP Connection)
MCR.communication_leipzig_class=mycore.communication.MCRServletCommunication
MCR.communication_leipzig_protocol=http
MCR.communication_leipzig_host=leilib.uni-leipzig.de
MCR.communication_leipzig_port=80
MCR.communication_leipzig_servlet_location=/rqs/rqs

# Configuration for the host with alias essen (HTTPS Connection)
MCR.communication_essen_class=mycore.communication.MCRServletCommunication
MCR.communication_essen_protocol=https
MCR.communication_essen_host=ceylon.uni-essen.de
MCR.communication_essen_port=443
MCR.communication_essen_servlet_location=/rqs/rqs

MCR.communication_https_support=true
MCR.communication_keystore=/MyCoRe/keystore
```

Die ... `servlet_location` gibt den Pfad zum entfernten RQS auf dem Web-Server an. Die URL für den Zugriff wird aus den Parametern 2 bis 5 zusammengesetzt. Für den Leipziger Server wäre das *http://leilib.uni-leipzig.de:80/rqs/rqs*. Das ist natürlich nur eine Basis-URL. Der MyCoRe-Kern nutzt HTTP-Parameter, um Informationen zum Server zu senden.

Um HTTPS nutzen zu können ist es erforderlich, der JVE Zugriff auf das *Java Secure Sockets Extension*-Paket zu gewähren. Diese Klassen sind im neuen Java 1.4 bereits integriert. Falls der entfernte Web-Server, der per HTTPS eingebunden werden soll, für die Verschlüsselung ein Zertifikat nutzt, das nicht als allgemein vertrauenswürdig gilt, weil es nicht von einer Certification Authority (CA) zum Beispiel Verisign ausgestellt wurde, so muss sein Zertifikat in einen Java-Keystore importiert werden. Dazu nutzt man das `keytool` von Java. Dem MyCoRe-System teilt man nach dem Import mit, wo der Java-Keystore lokalisiert ist. Danach ist der RQC in der Lage auf den entfernten HTTPS-Server zuzugreifen, auch ohne allgemein vertrauenswürdiges Zertifikat.

6.1.5 Bewertung und Erweiterbarkeit

Die vorgestellte MyCoRe-Architektur mit verteilter Suche realisiert eine Digitale Bibliothek mit Such- /Integrationsdienst in der Ausprägung einer echten On-The-Fly-Verarbeitung.

Im Folgenden sind noch einmal die Anforderungen aus Kapitel 4 zusammen mit ihren Realisierungsgraden in MyCoRe aufgelistet.

Parallele Suche Vollständig realisiert.

Ergebnismischung Vollständig realisiert. Es war nicht ganz so problematisch, da die einzelnen System homogen sind, es entartete eher zur Verkettung der Ergebnislisten.

Dublettenerkennung Noch nicht realisiert. Es ist aber bereits ein Mechanismus zur globalen eindeutigen Identifizierung von multimedialen Objekten umgesetzt: Der schon im Abschnitt 4.4 erwähnte NBN ist ein persistenter Identifikator für elektronische Dokumente. Multimedialen Objekten in MyCoRe kann jetzt solch ein NBN zugeordnet werden. Damit hat man einen Ansatzpunkt für eine Dublettenerkennung über mehrere MyCoRe-Instanzen.

Kein Informationsverlust Vollständig realisiert.

Kapselung Vollständig realisiert.

Vollständige Suche Vollständig realisiert.

Adäquate Recherchemöglichkeiten Nicht vollständig realisiert. Die XQuery-Implementierung in MyCoRe ist noch nicht komplett realisiert, aber grundsätzlich sollen alle Anfragearten unterstützt werden.

Globales Ranking Noch nicht realisiert. Die jetzige MyCoRe-Architektur mit einem CM V7-Backend liefert aufgrund der MyCoRe-spezifischen CM V7-Persistierung nur ungenügende Ranking-Werte. Sie eignen sich nicht, ernsthafte Aussagen über die Relevanz der Suchergebnisse zu machen.

Sicherheit bei der Kommunikation Vollständig realisiert, durch Nutzung der HTTPS-Kommunikation.

Kontrolle und Schutz von Urheberrechten, Personalisierung und Rechtemanagement Diese 3 Anforderungen müssen mit den Komponenten des Benutzermanagements (Universität Freiburg) abgestimmt werden. Diese Komponenten befanden sich während der Implementierung der Kommunikationsklassen noch in einer frühen Entwicklungsphase. Ein globales User- und Rights-Management war daher bisher noch nicht umsetzbar.

Lokale Autonomie Vollständig realisiert, durch die MyCoRe-Architektur impliziert.

Hohe Verfügbarkeit Eine Fehlerbehandlung bei einem Ausfall eines entfernten Rechners ist realisiert. Die Vermeidung von Rechner- und Netzausfällen, z.B. durch Mirror-Server oder alternatives Routing ist möglich, ist aber von der Tatsache abhängig, wie tragisch ein Ausfall des Systems wäre.

Die grundlegenden Anforderungen sind also bereits realisiert. MyCoRe steckt aber noch in den “Kinderschuhen”. Restliche Anforderungen gerade zum globalen Benutzermanagement und eine vollständige XQuery-Implementierung werden demnächst folgen. Dann bleiben nur noch Dublettenerkennung und Globales Ranking. Die Dublettenerkennung kann jetzt mit Hilfe eines NBN für die Dokumente vorgenommen werden. Das Globale Ranking wird sicher einfacher, wenn als Backend der Content Manager Version 8 zur Verfügung steht. Mit seinem Einsatz wird es einfacher sein, gute Ranking-Werte zu den Anfragen zu ermitteln. Außerdem sollte die Realisierung von XQuery als Anfragesprache sehr leicht fallen, denn CM V8 bietet schon eine Teilmenge von XQuery als Anfragesprache (siehe 2.6.3).

6.2 Cross Archive Searching Service

Im Rahmen dieser Diplomarbeit wurde desweiteren ein Service implementiert, der es möglich macht, über mehrere OAI-Repositories zu suchen. Dieser Cross Archive Searching Service nutzt den OAI-Service der einzelnen Repositories für ein Sammeln der Metadaten (*harvest*). Diese Metadaten werden in einer lokalen DB2-Datenbank gespeichert. So kann eine Suche nach Items in den Repositories bewerkstelligt werden.

Der implementierte Cross Archive Searching Service ist am Fachbereich Informatik der Universität Rostock installiert. Einen Einstieg auf diesen Dienst findet man über die URL *http://corfu.informatik.uni-rostock.de/OAISearch*.

6.2.1 Architektur und Funktionsweise

Die Architektur des Services ist in Abbildung 6.6 dargestellt:

Das User Interface erlaubt eine Bedienung der anderen Teilkomponenten des Services. Die bedienbaren Komponenten sind *Search Component*, *Manage Database Component* und *Harvest Component*.

Die Search Component nimmt parametrische Anfragen von dem User Interface entgegen und wandelt diese in entsprechende SQL-Statements um. Diese SQL-Anfragen werden an die Datenbank abgesetzt. Das Ergebnis der Anfrage wird dem Benutzer anschaulich präsentiert.

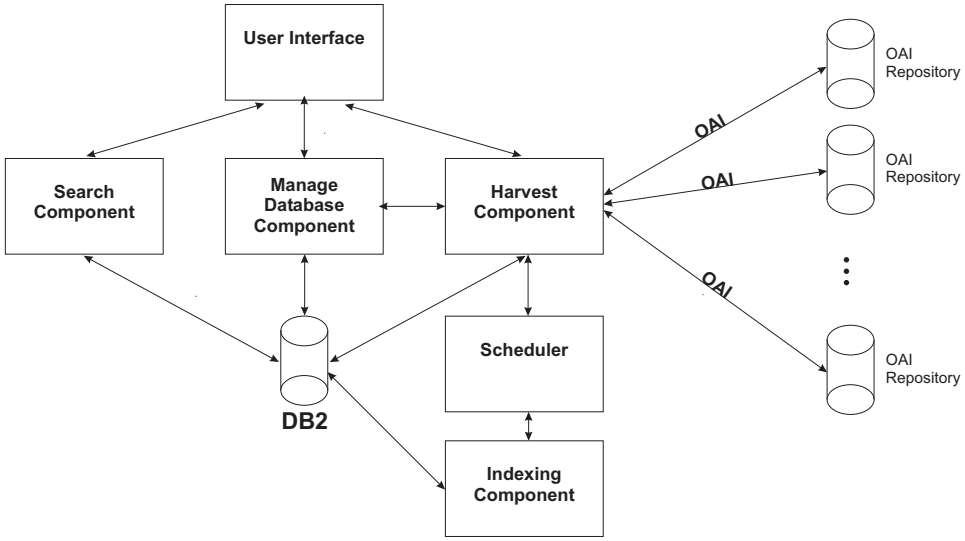

Abbildung 6.6: Architektur des implementierten Cross Archive Searching Services

Die Manage Database Component erledigt administrative Aufgaben des Services. Sie präsentiert die Repositories, von denen bereits Records in die Datenbank gesammelt wurden. Außerdem kann sie auch den Inhalt beeinflussen. Sie kann die gesammelten Records von einzelnen Repositories löschen (*delete*) oder erneut sammeln (*reharvest*). Desweiteren kann sie bisher nicht eingebundene Repositories zu der Datenbank hinzufügen. Dafür arbeitet sie mit der *Harvest Component* zusammen.

Die Harvest Component ist in der Lage, OAI-Requests an OAI Repositories abzusetzen, und dadurch die Metadaten-Records[6] von den Repositories einzusammeln. Diese werden der Datenbank hinzugefügt bzw. dort aktualisiert. Solch ein Harvest-Prozess kann von dem Benutzer über das User Interface oder über die Manage Database Component ausgelöst werden.

Die Indexing Component ist für das Indizieren der Records in der Datenbank verantwortlich. Dafür wird der Text Information Extender eingesetzt. Dadurch wird eine Volltext-Anfrage auf die gesammelten Metadaten möglich. Es entsteht ein Peformance-Gewinn gegenüber den alternativen Anfragen, wo langsame `LIKE`-Anfragen über den gesamten Datenbestand abgesetzt werden müssten. Hier sind zwei Beispiele:

[6]Es werden Metadaten-Records im Dublin-Core-Format (siehe 2.3) eingesammelt. OAI-Repositories müssen mindestens Dublin Core unterstützen.

```
select id from records where creator like '%Gauss%'
select id from records where contains(creator, '"Gauss"') = 1
```

Beide Anfragen richten sich an die Dokumente, deren Autor *Gauss* sein soll. In dem Attribut `CREATOR` ist eine Liste aller Autoren gespeichert. *Gauss* könnte auch irgendwo innerhalb dieses Textfeldes auftauchen, deshalb kann man nicht `creator = 'Gauss'`-Bedingungen und in Verbindung damit normale Datenbankindizes über die `CREATOR`-Attribute nutzen. Die erste Anfrage nutzt `LIKE` für die Formulierung der Anfrage, die zweite nutzt den Text-Index des Text Information Extenders. Der Performance-Gewinn ist beachtlich. Bei einer Datenbank mit 64449 Metadaten-Records dauerte die erste Anfragemethode 224 Sekunden, die zweite ganze 6 Sekunden.

Der Scheduler spielt eine sehr wichtige Rolle für die Aktualität der Suchergebnisse. Damit dem Benutzer, der Suchen auf dem System absetzt, immer ein aktueller Stand der gesammelten Inhalte der jeweiligen Data Providers präsentiert wird, ist es enorm wichtig regelmäßig einen Harvesting Prozess durch die Harvest Component anzustoßen. Diese Aufgabe übernimmt der Scheduler. Er kann in regelmäßigen wohldefinierten Zeitabständen ein *reharvest* für die abgesammelten Archive auslösen, sowie neu entdeckte Data Provider erstmalig in den Datenbestand aufnehmen. (Die Open Archives Initiative bietet eine Liste von registrierten Data Providern, die für solch eine Funktionalität genutzt wird.) Eine weitere Aufgabe ist das regelmäßige Up-to-Date-Halten des Text-Indexes in der DB2-Datenbank.

6.2.2 Auswahl der Programmiermittel/Implementierungsdetails

Für die Implementierung des Services wurde in Java bewerkstelligt. Mit Java-Servlets werden dynamische HTML-Seiten erzeugt, welche User Interfaces für Search und Manage Database Component bilden. Sie sind also über einen herkömmlichen Web-Browser bedienbar.

Die Harvest Component ist über die Kommandozeile und natürlich auch über die Manage Database Component, also über einen Browser bedienbar.

Um die von den OAI Repositories empfangenen XML-Daten auszuwerten, nutzt die Harvest Component den Java-SAX-Parser von Xerces.

Die Harvest Component verfügt über eine Logging-Einrichtung, die es ermöglicht, den Erfolg von beauftragten Harvest Requests zu überprüfen. Das Log ist über den Web-Browser einsehbar.

Die Indexing Component wurde durch ein Skript umgesetzt, welches die Text-Indizes in der Datenbank aktualisiert.

Der Scheduler wurde durch den Taskplaner von Windows 2000 verwirklicht. Der Taskplaner erlaubt es, Programme auf dem Windows 2000 System regelmäßig zu starten. Auf dem entsprechenden System wurden wöchentlich auszuführende Jobs erstellt, welche ein Anstoßen der Harvest Component und der Indexing Component bewirken.

Das Datenbankdesign ist im Anhang B in Form eines SQL-Skriptes beschrieben.

Die Search Component ist durch ein Java-Servlet implementiert worden. Sie nimmt die Parameter für eine Suche über den Web-Server entgegen und produziert die SQL-Anfragen zur Laufzeit. Die Aufbereitung der Suchergebnisse und die Präsentation wird ebenfalls durch das Servlet realisiert.

6.2.3 Bewertung und Erweiterbarkeit

Der Service könnte um die wenigen folgenden Dinge erweitert werden, um sich mit kommerziell entwickelten vergleichbaren Services (siehe 5.2.3) messen zu können:

Es bietet sich an, erweiterte Suchmasken zu erstellen, in denen mehr Details angegeben werden können.

Die Manage Database Component ist für jedermann bedienbar, das heißt es ist keine Authentifizierung notwendig, um die Inhalte der Datenbank zu beeinflussen. Jedermann könnte Datensätze beliebig löschen. (Hört sich kritisch an! Ist aber gar nicht so schlimm, denn die Metadaten können ja auch jederzeit wieder über die Harvest Component zurück bekommen werden.) Es sollte dem Service also ein Rechtemanagement hinzugefügt werden. Die Manage Database Component sollte nicht über öffentlichen Zugriff bedienbar sein.

Die Search Component könnte verbessert werden. Derzeit liefert sie das gesamte Suchergebnis in einem HTML-Stream, also auf einer Browser-Seite. Es sollte eine seitenweise Präsentation des Ergebnisses angeboten werden. Dafür sollten die erhalten Resultate zwischengespeichert werden. Wünschenswert wären auch ein Speichern von Anfragen und Ergebnissen für späteren Gebrauch. Dafür braucht das System eine Personalisierungskomponente.

Ein Ranking ist bisher noch nicht umgesetzt. Dieses wäre ein wünschenswertes Feature. Der Text Information Extender bietet bereits eine Ranking-Funktion, welche bisher aber noch nicht ausgenutzt worden ist.

Der vorgestellte, implementierte Cross Archive Searching Service mit einer Suche auf verteilten Daten realisiert einen Such- /Integrationsdienst für eine Digitale Bibliothek in der Ausprägung einer Replizierung der Metadaten nach 3.5.1.

Im Folgenden sind noch einmal die Anforderungen aus Kapitel 4 zusammen mit ihren Realisierungsgraden in dem Service aufgelistet.

Parallele Suche Vollständig realisiert. Bei der Replizierung der Metadaten wird diese Anforderung trivial.

Ergebnismischung Vollständig realisiert. Ebenfalls trivial.

Dublettenerkennung Nicht realisiert.

Kein Informationsverlust Vollständig realisiert, zumindest, was die Dublin-Core-Metadaten zu den einzelnen Dokumenten angeht. Es ist aber denkbar, dass für einzelne Dokumente noch mehr Metadaten in anderen Formaten über die entsprechenden OAI-Repositories abrufbar wären. Hier könnte man ansetzen, um den Service zu erweitern.

Kapselung Vollständig realisiert. Impliziert OAI.

Vollständige Suche Vollständig realisiert.

Adäquate Recherchemöglichkeiten Teilweise realisiert. Wünschenswert wäre eine Suchmöglichkeit auf den Volltexten der eigentlichen Dokumente. Dieses geht aber weit über den Fokus von OAI hinaus. OAI erlaubt keine Volltextsuche.

Globales Ranking Noch nicht realisiert. Ein Ranking für die Anfragen auf die Metadaten ist mit Hilfe des Text Information Extenders aber relativ leicht umsetzbar.

Sicherheit bei der Kommunikation Teilweise realisiert. Diese Anforderung ist nicht zweckmäßig bei OAI. Open Archives sind – wie der Name schon sagt – offen. Lediglich die Kommunikation mit dem Benutzer über das Interface sollte eventuell gesichert erfolgen, wenn zum Beispiel sensible Nutzerdaten übertragen werden. Dafür wurde das Web-Interface zusätzlich über HTTPS[7] verfügbar gemacht.

Kontrolle und Schutz von Urheberrechten, Personalisierung und Rechtemanagement Nicht realisiert.

Lokale Autonomie Realisiert. Die einzelnen Systeme bieten durchweg eigene Interfaces zur Suche und sind natürlich nicht von dem Cross Archive Searching Service in Rostock abhängig.

Hohe Verfügbarkeit Bei Nichterreichen eines entfernten Systems während des Harvesting, ist der Fehler und die Art des Fehlers durch das Log-File ersichtlich. Eine automatische Fehlerbehandlung erfolgt nicht. Sie geschieht bisher manuell oder gar nicht.

Der implementierte Cross Archive Searching Service liefert trotz seiner prototypischen Umsetzung ein zufriedenstellendes Ergebnis. Suchen auf dem Inhalt sind ausreichend performant. Die Funktionalität ist vielseitig und nutzt viele Features von OAI.

[7] https://corfu.informatik.uni-rostock.de/OAISearch

Kapitel 7

Zusammenfassung und Ausblick

Ziel dieser vorliegenden Arbeit war die Konzeption eines integrierten, verteilten Suchdienstes für eine Digitale Bibliothek. Durch die Integration fremder Digitaler Bibliotheken soll ein größerer Wissensverbund geschaffen werden.

In Kapitel 3 wurde dazu die Architektur Digitaler Bibliotheken vorgestellt, in diese wurden Komponenten für die verteilte Suche eingegliedert. Es entstand das Konzept des verteilten Such- / Integrationsdienstes. Dieser Dienst kann in zwei Ausprägungen umgesetzt werden. Zum einen kann er eine echte verteilte Suche “On-the-Fly” durchführen oder er kann auf replizierten Daten suchen.

Für den verteilten Such- / Integrationsdienst wurden in Kapitel 4 Anforderungen definiert und diskutiert. Eine wichtige Anforderung ist die des globalen Rankings, deshalb wurde diese Anforderung ausführlicher beleuchtet. Der entstandene Anforderungskatalog bietet einen Ansatzpunkt für die Umsetzung einer verteilten Suche auf Digitalen Bibliotheken. Außerdem eignet er sich als Anhaltspunkt für eine Bewertung der Güte von Systemen mit verteilter Suche.

In Kapitel 5 wurden vier verschiedene Ansätze für die Realisierung eines solchen Dienstes vorgestellt und untersucht. Zuerst wurden typische Kommunikationsarten in verteilten Systemen vorgestellt. Das waren die Kommunikation über Sockets, die HTTP- und die HTTPS-Kommunikation, CORBA, RMI und SOAP. Als nächstes wurde OAI vorgestellt, welches sich nur für das Harvesting von Metadaten offener Archive eignet. Daher kann eine verteilte Suche über OAI-Repositories auch nur über replizierte Metadaten realisiert werden. Die anderen beiden Ansätze waren das Federated Search aus dem IBM Enterprise Information Portal und das Bibliotheksprotokoll Z39.50. Alle 4 Ansätze können auch parallel in einem verteilten Such- / Integrationsdienst umgesetzt werden. Sie schließen sich nicht gegenseitig aus.

In Kapitel 6 wurden zwei Implementierungen für die verteilte Suche in Digitalen Bibliotheken vorgestellt, die im Rahmen dieser Diplomarbeit getätigt wurden: Zum einen war das die Implementierung für das MyCoRe-System. Es wurden 3 Kommunikationsarten für MyCoRe umgesetzt, Kommunikation über Sockets, HTTP- und HTTPS-Kommunikation, Das System befindet sich noch in einem frühen Ent-

wicklungsstadium. Die Implementierung für die verteilte Suche im Rahmen dieser Diplomarbeit hat einen Teil zur Funktionalität von MyCoRe beigetragen. Die andere vorgenommene Implementierung realisiert eine Suche über mehrere OAI-Repositories und hat nur prototypischen Charakter.

Die in dieser Arbeit vorgestellte MyCoRe-Plattform realisiert die verteilte Suche bereits, sie wird sich in Zukunft aber noch mehr weiterentwickeln. Sie wird dabei vor allem vom Release des Content Managers in seiner neuen Version 8 profitieren. Die Benutzung dieses Systems als Backend erlaubt die einfachere Umsetzung der bestehenden MyCoRe-Konzepte, insbesondere der hierarchischen Gliederung und der Anfragen in XQuery.

Ziel zukünftiger Forschungen sollte es sein, ein modernes Standardprotokoll zu entwickeln, um auf entfernte Repositories zuzugreifen. Die Open Archives Initiative hat zwar gerade eine neue Version ihres OAI-Protokolls verabschiedet, es findet auch immer mehr Akzeptanz weltweit, es eignet sich aber trotzdem immer noch nicht aus den in dieser Arbeit angegeben Gründen, eine echte verteilte Suche auf Digitalen Bibliotheken zu realisieren. Das Z39.50-Protokoll schafft zwar Interoperabilität, ist aber nicht mehr modern. Die Realisierung dieses Protokolls als Exportschnittstelle nach außen stößt bei vielen Anbietern aufgrund der Komplexität von Z39.50 auf Abneigung. Viele implementieren auch nur Teile des Standards. Dennoch bietet dieses Protokoll die derzeit beste Möglichkeit standardisiert auf ein entferntes Repository zuzugreifen und darauf zu suchen. Bei einer Neuentwicklung eines Standardprotokolls sollte ein Hauptaugenmerk auf die Ideen und Funktionsweisen von Z39.50 gelegt werden. XML und die verwandten Standards des W3C sollten in die Entwicklung miteinfließen.

Ein weiterer Aspekt, der in eine zukünftige Betrachtung einfließen könnte, wäre zum Beispiel der der Kostenoptimierung bezüglich den Preisen von elektronischen Dokumenten. Ein Benutzer, der ein bestimmtes Dokument sucht, möchte dafür natürlich so wenig Geld wie möglich ausgeben. Eventuell ist er auch mit einem ähnlichen Dokument zufrieden, wenn er es viel billiger erwerben kann. Wenn diese Ähnlichkeit von Dokumenten vom Metasystem beurteilt werden kann, wäre das sicher ein entscheidender innovativer Schritt in diese Richtung.

Das Relevanz Feedback bietet außerdem einen interessanten Ansatz für die verteilte Suche auf Digitalen Bibliotheken. Von seinem Einsatz könnten das globale Ranking und die Doublettenerkennung profitieren. Für letzteres ist die Entwicklung eines globalen Identifizierungsmechanismusses für elektronische Dokumente von großer Wichtigkeit.

Das Phänomen der Digitalen Bibliotheken ist zweifelsohne noch in der Entwicklungsphase, es werden weitere kommerzielle Lösungen auf den Markt kommen und weitere Forschungsaspekte entstehen. In der weiteren Entwicklung sind sicher auch Innovationen auf dem Gebiet der verteilten Suche über mehrere Repositories zu erwarten.

Anhang A

MyCoRe Beispiele

A.1 XML-Repräsentation eines multimedialen Objektes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mycoreobject
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="LeiLibPapyri_Document.xsd"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 ID="LeiLibPapyri_Document_3"
 label="Lips. Inv. 482">
<structure>
 <linksTo>
  <linkTo
   xlink:type="locator"
   xlink:href="LeiLibPapyri_PhysFrag_3"
   xlink:title="PhysFrag" />
 </linksTo>
 <linksFrom>
  <linkFrom
   xlink:type="locator"
   xlink:href="LeiLibPapyri_Text_3"
   xlink:title="Text" />
  <linkFrom
   xlink:type="locator"
   xlink:href="LeiLibPapyri_Adaption_3"
   xlink:title="Adaption" />
 </linksFrom>
 <derivates>
  <derivate
   xlink:type="locator"
   xlink:href="http://papyri.dl.uni-leipzig.de/Bilder/physfrag_482_r_1.html"
   xlink:label="recto" />
 </derivates>
</structure>

<metadata xml:lang="de">
 <!--01 Titel -->
 <dok_titels class="MCRMetaLangText">
  <dok_titel>
   Papyrus 482
  </dok_titel>
 </dok_titels>

 <!--02 Inventarnummer (<inventarnummer> kann mehrfach auftreten )-->
 <inventarnummers class="MCRPapyriInventar" heritable="true">
  <inventarnummer>
   <katalog>
    Lips
   </katalog>
   <nummer>
    482
   </nummer>
  </inventarnummer>
 </inventarnummers>

 <!--03 Abmessungen des vorhandenen Gesamtdokumentes -->
 <abmessungs class="MCRMetaNumber">
  <abmessung dimension="Breite" measurement="cm">34</abmessung>
```

```
  <abmessung dimension="Hoehe" measurement="cm">13</abmessung>
 </abmessungs>

 <!--04 Zustand des Gesamtdokumentes-->
 <zustand_docs class="MCRMetaLangText">
  <zustand_doc>komplett</zustand_doc>
 </zustand_docs>

 <!--05 Form/Ausfuehrung -->
 <ausfuehrungs class="MCRMetaLangText">
  <ausfuehrung>
   Blatt
  </ausfuehrung>
 </ausfuehrungs>
</metadata>

<service>
 <dates>
  <date type="validfromdate">13.12.2001</date>
 </dates>
 <flags>
  <flag>Cu</flag>
 </flags>
</service>

</mycoreobject>
```

A.2 XML-Schema für eine Klasse multimedialer Objekte

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'
         xmlns:xml='http://www.w3.org/XML/1998/namespace'
         xmlns:xlink='http://www.w3.org/1999/xlink'
         elementFormDefault="unqualified">

 <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
          schemaLocation="xml-2001.xsd" />

 <xsd:import namespace="http://www.w3.org/1999/xlink"
          schemaLocation="xlinks-2001.xsd" />

 <xsd:include schemaLocation='MCRObjectStructure.xsd'/>
 <xsd:include schemaLocation='MCRObjectService.xsd'/>
 <xsd:include schemaLocation='MCRMetaDate.xsd'/>
 <xsd:include schemaLocation='MCRMetaLangText.xsd'/>
 <xsd:include schemaLocation='MCRMetaNumber.xsd'/>

 <xsd:element name="mycoreobject" type="MCRObject"/>

 <xsd:complexType name="MCRObject">
  <xsd:sequence>
  <xsd:element name="structure" type="MCRObjectStructure" minOccurs='1'
   maxOccurs='1' />
  <xsd:element name="metadata" type="MCRObjectMetadata" minOccurs='1'
   maxOccurs='1' />
  <xsd:element name="service" type="MCRObjectService" minOccurs='1'
   maxOccurs='1' />
  </xsd:sequence>
  <xsd:attribute name="ID" type="xsd:string" use="required" />
  <xsd:attribute name="label" type="xsd:string" use="required" />
 </xsd:complexType>

 <xsd:element name="structure" type="xsd:string"/>

 <xsd:complexType name="MCRObjectMetadata">
  <xsd:sequence>

<!--01=====================================================================-->
   <xsd:element name="dok_titels" minOccurs="1" maxOccurs="1">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="dok_titel"
         type="MCRMetaLangText" minOccurs="1" maxOccurs="2"/>
     </xsd:sequence>
     <xsd:attribute name="class" type="xsd:string" use="required" />
     <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
    </xsd:complexType>
   </xsd:element>

<!--02=====================================================================-->
   <xsd:element name="inventarnummers" minOccurs="1" maxOccurs="1">
    <xsd:complexType>
```

```
    <xsd:sequence>
     <xsd:element name="inventarnummer" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="katalog" type="xsd:string"
          minOccurs="1" maxOccurs="1"/>
        <xsd:element name="nummer" type="xsd:integer"
          minOccurs="1" maxOccurs="1"/>
        <xsd:element name="rectoverso" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="teil" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
       </xsd:sequence>
       <xsd:attribute ref="xml:lang"/>
       <xsd:attribute name="type" use="optional"/>
      </xsd:complexType>
     </xsd:element> <!-- /inventarnummer -->
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element> <!-- /inventarnummers -->
<!--03====================================================================-->
  <xsd:element name="abmessungs" minOccurs="1" maxOccurs="1">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="abmessung"
       type="MCRMetaNumber" minOccurs="1" maxOccurs="2"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element>
<!--04====================================================================-->
  <xsd:element name="zustand_docs" minOccurs="1" maxOccurs="1">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="zustand_doc"
       type="MCRMetaLangText" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element>
<!--05====================================================================-->
  <xsd:element name="ausfuehrungs" minOccurs="0" maxOccurs="1">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="ausfuehrung"
       type="MCRMetaLangText" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element>
<!--06====================================================================-->
  <xsd:element name="bemerk_exts" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="bemerk_ext"
       type="MCRMetaLangText" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element>
<!--07====================================================================-->
  <xsd:element name="bemerk_ints" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="bemerk_int"
       type="MCRMetaLangText" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string" use="required" />
    <xsd:attribute name="heritable" type="xsd:boolean" use="optional" />
   </xsd:complexType>
  </xsd:element>
<!--====================================================================-->

 </xsd:sequence>
 <xsd:attribute ref="xml:lang" />
</xsd:complexType>

</xsd:schema>
```

A.3 XML-Repräsentation eines Suchergebnisses

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mcr_results>
<mcr_result host="corfu.informatik.uni-rostock.de" id="MyCoReDemoDC_Document_1" rank="99" >
<mycoreobject
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xsi:noNamespaceSchemaLocation="MyCoReDemoDC_Document.xsd"
   ID="MyCoReDemoDC_Document_1"
   label="Studienarbeit Mathias Zarick">
<structure/>
<metadata xml:lang="de">
  <titles class="MCRMetaLangText" heritable="true">
    <title xml:lang="de" type="Haupttitel">
      Konzeption einer Suchmaschine f&#252;r XML-Dokumente
    </title>
    <title xml:lang="en" type="Maintitle">
      Conception of a Search Engine for XML Documents
    </title>
  </titles>
  <creators class="MCRMetaLink" heritable="false">
    <creator xlink:type="locator" xlink:href="MyCoReDemoDC_LegalEntity_1" xlink:label="Author"/>
  </creators>
  <descriptions class="MCRMetaLangText" heritable="false">
    <description xml:lang="de">
      Diese Studienarbeit untersucht die M&#246;glichkeiten der Suchbarkeit auf XML Dokumenten.
    </description>
  </descriptions>
  <publishers class="MCRMetaLink" heritable="false">
    <publisher xlink:type="locator" xlink:href="MyCoReDemoDC_LegalEntity_1" xlink:label="Publisher"/>
  </publishers>
  <contributors class="MCRMetaLink" heritable="false">
    <contributor xlink:type="locator" xlink:href="MyCoReDemoDC_LegalEntity_1" xlink:label="Contributor"/>
  </contributors>
  <dates class="MCRMetaDate" heritable="false">
    <date xml:lang="de">21.05.2000</date>
  </dates>
  <rights class="MCRMetaLangText" heritable="false">
    <right xml:lang="de">
      Alle Urheberrechte obliegen dem Autor.
    </right>
    <right xml:lang="en">
      All rights keeps the author.
    </right>
  </rights>
</metadata>
<service>
  <dates>
     <date xml:lang="de" type="createdate">09.03.2002</date>
     <date xml:lang="de" type="modifydate">09.03.2002</date>
     <date xml:lang="de" type="validfromdate">21.09.2001</date>
  </dates>
  <flags>
     <flag xml:lang="de">F1</flag>
     <flag xml:lang="de">F2</flag>
  </flags>
</service>
</mycoreobject>
</mcr_result>
<mcr_result host="corfu.informatik.uni-rostock.de" id="MyCoReDemoDC_Document_2" rank="95">
<mycoreobject
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:noNamespaceSchemaLocation="MyCoReDemoDC_Document.xsd"
ID="MyCoReDemoDC_Document_2"
label="XML Guide">
.
.
.
</mycoreobject>
</mcr_result>
</mcr_results>
```

A.4 XSLT-Stylesheet zur Darstellung

Folgendes Stylesheet produziert reinen ASCII-Text und passt zu dem Beispiel aus A.3. Die Ausgabe nach der Transformation ist in A.5 angegeben.

```
<?xml version="1.0"?>
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="metadata/titles">
Titel:
<xsl:value-of select="title"/>
</xsl:template>

<xsl:template match="metadata/descriptions">
Beschreibung:
<xsl:value-of select="description"/>
</xsl:template>

<xsl:template match="metadata/dates">
Datum:
<xsl:value-of select="date"/>
</xsl:template>

<xsl:template match="metadata/coverages">
Beteiligte:
<xsl:value-of select="coverage"/>
</xsl:template>

<xsl:template match="metadata/rights">
Rechte:
<xsl:value-of select="right"/>
</xsl:template>

<xsl:template match="mycoreobject">
Ein MyCoRe Dokument im Dublin-Core-Format
========================================
<xsl:apply-templates select="metadata/titles"/>
<xsl:apply-templates select="metadata/descriptions"/>
<xsl:apply-templates select="metadata/dates"/>
<xsl:apply-templates select="metadata/coverages"/>
<xsl:apply-templates select="metadata/rights"/>
</xsl:template>
</xsl:stylesheet>
```

A.5 Ausgabe nach XSLT-Transformation

Hier ist das Ergebnis einer XSLT-Transformation des Beispieles aus A.3 unter Verwendung des Stylesheet aus A.4 angegeben.

```
Ein MyCoRe Dokument im Dublin-Core-Format
========================================

Titel: Konzeption einer Suchmaschine für XML-Dokumente

Beschreibung: Diese Studienarbeit untersucht die Möglichkeiten der Suchbarkeit auf XML Dokumenten.

Datum: 21.05.2000
Rechte: Alle Urheberrechte obliegen dem Autor.

Ein MyCoRe Dokument im Dublin-Core-Format
========================================
.
.
.
```

Anhang B

Installation des Cross Archive Searching Service

Im Installationsverzeichnis des Services befinden sich 4 Unterverzeichnisse.

- in `database` befinden sich Skripte zum Erstellen der Datenbank, der Datenbanktabellen und -indizes sowie zum Erstellen und Updaten der Text-Indizes des Text Information Extenders
- in `workdir` befinden sich Dateien mit HTML-Fragmenten, die in die dynamisch erstellten HTML-Seiten eingebaut werden. Außerdem befinden sich hier Skripte für das Harvesting. `workdir` muss das Arbeitsverzeichnis des Web Application Servers für die Servlets sein. Hier wird auch die Log-Datei für das Harvesting abgelegt.
- in `src` befinden sich die Java-Quellcodes
- in `webapp` befindet sich die Web-Applikation für den Service

Folgende Schritte müssen ausgeführt werden, um den Service zu installieren.

1. **Konfiguration des Java-CLASSPATH** Gebraucht werden die Klassen für den DB2-JDBC-Zugriff (`db2java.zip`), die Klassen für den Xerces XML-Parser (`xerces.jar`) sowie die Klassen für die Servlets (`servlet.jar`).
2. **Anlegen der Datenbank, sowie der einzelnen Tabellen und Indizes** Dazu das Skript `create_db.cmd` aus dem `database`-Verzeichnis aufrufen.
3. **Anlegen der Text-Indizes** mit dem Skript `create_textindices.cmd` aus dem `database`-Verzeichnis.
4. **Kompilieren der Source-Codes** Dazu den Befehl

$$\texttt{javac} \ *.\texttt{java} \ -\texttt{d} \ ../\texttt{webapp}/\texttt{WEB} - \texttt{INF}/\texttt{classes}$$

in dem `src`-Verzeichniss absetzen. Die kompilierten Klassen werden dadurch in `webapp\WEB-INF\classes`, im Verzeichnis der Web-Applikation abgelegt.

5. **Deployment des Web Application Servers** Das Verzeichnis `webapp` beinhaltet bereits die komplette Web-Applikation. Bei Benutzung von Tomcat muss sie zum Beispiel nur in ein Unterverzeichnis unterhalb des Tomcat-Verzeichnisses `webapps` kopiert werden.

6. **Harvesting** In dem `workdir`-Verzeichnis befinden sich bereits zwei Skripte (`harvestall.cmd` und `harvestselected.cmd`) für das Harvesting bestimmter OAI-Repositories. Sie können für ein Harvesting von dort angestoßen werden. Sie sollten als Ausgangspunkt dienen, von welchen Repositories man regelmäßig Metadaten einsammeln möchte.

7. **Updaten der Text Indizes** Jedes mal, nachdem sich die Metadaten in der Datenbank geändert haben, muss der Textindex auf den neuesten Stand gebracht werden, damit sich die Änderungen auch auswirken. Dafür gibt es das Skript `update_textindices_with_log.cmd`, welches sich in dem Verzeichnis `database` befindet.

8. **Konfiguration des Schedulers** Die letzten beiden Schritte müssen natürlich immer wieder ausgeführt werden, damit die Daten immer auf den neuesten Stand gebracht werden. Dazu legt man *Scheduled Tasks* unter Windows bzw. *CRON-Jobs* in der Linux- bzw. UNIX-Welt fest. Man bindet einfach die in den letzten beiden Schritten benutzten Skripte ein. Man sollte darauf achten, dass das Indizieren erst beginnt, nach dem der Harvesting-Prozess komplett beendet ist.

Hier sei nun noch das SQL-Skript gegeben, welches die Tabellen und Indizes in der Datenbank anlegt:

```
CREATE TABLE ARCHIVE (
  ID          VARCHAR(30) NOT NULL,
  NAME        VARCHAR(255) NOT NULL,
  BASE_URL    VARCHAR(255) NOT NULL,
  WAS_COMPLETED SMALLINT NOT NULL,
  TIMESTAMP   TIMESTAMP   NOT NULL,
  CONSTRAINT ARCHIVE_PK PRIMARY KEY (ID)
);
```

```
CREATE TABLE RECORDS (
  ID          VARCHAR(40) NOT NULL,
  OAIID       VARCHAR(255) NOT NULL,
  DATESTAMP   VARCHAR(50) NOT NULL,
  ARCHIVEID   VARCHAR(40) NOT NULL,
  TITLE       LONG VARCHAR,
  CREATOR     LONG VARCHAR,
  SUBJECT     LONG VARCHAR,
  DESCRIPTION LONG VARCHAR,
  CONTRIBUTOR LONG VARCHAR,
  PUBLISHER   LONG VARCHAR,
  DATE        LONG VARCHAR,
  TYPE        LONG VARCHAR,
  FORMAT      LONG VARCHAR,
  IDENTIFIER  LONG VARCHAR,
  SOURCE      LONG VARCHAR,
  LANGUAGE    LONG VARCHAR,
  RELATION    LONG VARCHAR,
  COVERAGE    LONG VARCHAR,
  RIGHTS      LONG VARCHAR,
  CONSTRAINT RECORDS_PK PRIMARY KEY (ID),
  CONSTRAINT RECORDS_FK FOREIGN KEY (ARCHIVEID) REFERENCES ARCHIVE (ID)
);

CREATE INDEX ARCHIVE_NAME ON ARCHIVE (NAME ASC);
CREATE INDEX RECORDS_OAIID ON RECORDS (OAIID ASC);
CREATE INDEX RECORDS_ARCHIVEID ON RECORDS (ARCHIVEID ASC);
```

Anhang C

Statistiken zum Harvesting des Cross Archive Searching Service

Ein OAI-Data-Provider kann sich bei der Open Archives Initiative als solcher registrieren. Die Initiative pflegt eine Liste dieser bei ihr registrierten Data Provider. Unter der URL *http://www.openarchives.org/Register/ListFriends.pl* ist eine aktuelle Liste im XML-Format abrufbar. Sie kann durch eben diese XML-Formatierung leicht in Applikationen eingebunden werden. Der im Rahmen dieser Arbeit implementierte Cross Archive Searching Service nutzt diese Liste für eine Funktion des Metadata-Harvesting von allen registrierten Data Providern. Am 24.04.2002 waren es 74 registrierte Data Provider. Am 22.06.2002 waren es schon 96.

Am 24.04.2002 wurde ein Harvesting über die 74 registrierten Data Provider gestartet. Es wurden in 29 Stunden und 40 Minuten fast 1 Million (978037) Metadaten-Records gesammelt. Von den 74 für das Einsammeln zuständigen Prozessen endeten aber nur 62 auf korrekte Art und Weise, die restlichen 12 wurden aufgrund von unerwarteten Laufzeitfehlern abgebrochen. Diese Laufzeitfehler wurden verursacht, weil entweder die Repositories nicht erreichbar waren oder sich die Anbieter nicht korrekt an den Standard hielten. Die Fehler lassen sich in fatale und nicht fatale unterscheiden, je nachdem, ob sie ein Fortsetzen des Harvesting-Prozesses erlauben. Die Fehler waren:

- Inkonformität bezügiglich des geforderten XML-Schemas (nicht fatal);
- Syntaktische Fehler im XML-Datenstrom (fatal)
  - XML-Dokument war nicht wohlgeformt,
  - Fehlen des Delimiters `;` beim Nutzen von Entity-Referenzen,
  - Nutzen ungültiger Zeichen,
  - XML-Attribute waren nicht in `'` oder `"` eingebettet
- HTTP-Request Timeout (fatal);

- Abbruch der Socket-Verbindung (fatal);
- Nichtexistenz eines angegebenen ResumptionTokens (fatal).

Am 22.05.2002 wurden die Metadaten von 40 ausgewählten Data Providern gesammelt und anschließend indiziert. Das Sammeln dauerte 5 Stunden und 21 Minuten. Das Indizieren dauerte 1 Stunde und 42 Minuten. Es wurden insgesamt 64449 Records gesammelt.

Die Repositories, deren Metadaten eingesammelt wurden, sind den einzelnen Tabellen C.1, C.2 und C.3 aufgelistet. Sie unterstützen alle OAI. Mit Hilfe des im Rahmen dieser Arbeit implementierten Cross Archive Searching Service, der unter der URL *http://corfu.informatik.uni-rostock.de/OAISearch* erreichbar ist, sind alle aufgelisteten Archive durchsuchbar.

ID des Repository	Anbieter / Beschreibung	Anzahl der Records
164.76.128.26	Center of Environmental Information Technology and Applications, Eastern Michigan University, Ypsilanti, USA	19
anu	The Australian National University, Canberra, Australien	237
AIM25	Archives in London and the M25 area, mehrere beitragende Institutionen aus dem Großraum London, Großbritannien	4283
caltechcstr	Computer Science Technical Reports, California Institute of Technology, Pasadena, USA	274
caltechEERL	Earthquake Engineering Research Laboratory Technical Reports, California Institute of Technology, Pasadena, USA	226
caltechETD	Electronic Theses and Dissertations, California Institute of Technology, Pasadena, USA	35
cav2001	Fourth International Symposium on Cavitation, California Institute of Technology, Pasadena, USA	145
cbold	Comparative Bantu Online Dictionary, University of Califonia, Berkeley, USA	89

Tabelle C.1: OAI Data Provider - Teil 1

ID des Repository	Anbieter / Beschreibung	Anzahl der Records
celebration	“A Celebration of Women Writers”-Archive, University of Pennsylvania, Philadelphia, USA	188
CCSDthesis	Centre pour la Communication Scientifique Directe, mehrere beitragendene Universitäten aus ganz Frankreich	135
CSTC	Computer Science Teaching Center, unterstützt von Association for Computing Machinery, USA	75
DLCommons	Digital Library of the Commons, Indiana University, Bloomington, USA	512
DUETT	Dissertations and other Documents of the Gerhard-Mercator-University Duisburg	231
eldorado	Elektronisches Dokumenten-, Archivierungs- und Retrievalsystem der Universität Dortmund	534
epsilondiss	EPSILON Dissertation Test Archive, Swedish University of Agricultural Sciences, Uppsala, Schweden	33
EKUTuebingen	Eberhard-Karls-Universität Tübingen	528
formations	Formations Media Studies Archive, University of Ulster, Großbritannien	23
sceti	Schoenberg Center for Electronic Text and Image, University of Pennsylvania, Philadelphia, USA	54
SADA	Studien- und Diplomarbeiten, Lehrstuhl Datenbanken und Informationsysteme, Universität Rostock (Testbetrieb)	7
techreports.larc.nasa.gov	NASA Technical Reports, USA	2767
theses.mit.edu	M.I.T. Theses, Massachusetts Institute of Technolgogy, Cambridge, USA	6678
UBC	Electronic Thesis and Dissertation Archive, University of British Columbia, Vancouver, Kanada	2
University OfNottingham	ePrints, University of Nottingham, Großbritannien	43
UUdiva	Digital Archive, University of Uppsala, Schweden	1653

Tabelle C.2: OAI Data Provider - Teil 2

ID des Repository	Anbieter / Beschreibung	Anzahl der Records
glasgow	EPrint Archive, University of Glasgow, Großbritannien	29
GenericEPrints	Instructional Technology ePrint Server, Utah State University, Logan, USA	3
hofprints	EPrint Archive, Hofstra University, Hempstead, USA	35
HKUTO	Theses Online, Hong Kong University, Hong Kong	8747
HUBerlin	Document Server, Humboldt-Universität Berlin	815
ioffe	Archive of the IOFFE Institut, St. Petersburg, Russland	340
LSUETD	Electronic Thesis and Dissertation Archive, Louisiana State University, Baton Rouge, USA	208
MONARCH	Multimedia Online Archiv Chemnitz, Technische Universität Chemnitz	528
naca. larc.nasa.gov	NACA Technical Reports, USA	7549
ncstrlh	Historical Collection, Virginia Polytechnic Institute and State University, USA	20517
physdoc	Physnet Document Server, Universität Oldenburg	319
RIACS	Eprint Archive, Research Institute for Advanced Computer Science, Moffett Field, USA	51
sammelpunkt	“Sammelpunkt. Elektronisch archivierte Theorie”, Bureau für Philosophie, Wien, Österreich	164
VTETD	Electronic Thesis and Dissertation Collection, Virginia Polytechnic Institute and State University, USA	3321
www. open-video.org	Open Video Project, University of North Carolina, Chapel Hill, USA	1654
www. perseus.tufts.edu	Perseus Digital Library, Tufts University, Medford, USA	1398

Tabelle C.3: OAI Data Provider - Teil 3

Abbildungsverzeichnis

2.1 Fragmentierung eines logischen Datenbankobjektes in Verteilten Datenbanksystemen 12
2.2 Die Architektur des Content Manager Version 8 18
2.3 Das Content Manager Version 8 Meta-Modell 21

3.1 Dienste einer Digitalen Bibliothek nach [EF00] 25
3.2 Eine Digitale Bibliothek mit Integrationsfähigkeit 26
3.3 Aufbau einer Digitalen Bibliothek nach BlueRose 28
3.4 Integrationsgraphen von Verbunden Digitaler Bibliothekssysteme . 31

4.1 Mischung der Ergebnisse 40
4.2 Euler-Diagramm für das Szenario der Suche auf einem Repository 44
4.3 Euler-Diagramm für das Szenario der Suche auf zwei Repositories 44

5.1 Aufruf einer entfernten Methode mit CORBA 60
5.2 Client-Server-Kommunikation mit Z39.50 67
5.3 Verteilte Suche über einen Z39.50 Gateway 69
5.4 Verteilte Suche mit Federated Search 71

6.1 Transformation mittels XSLT 77
6.2 Die MyCoRe-Architektur 78
6.3 Teil des MyCoRe-Kerns für die Persistierung der multimedialen Objekte 79
6.4 Teil des MyCoRe-Kerns für die Anfragebearbeitung 80
6.5 verteilte Suche in MyCoRe 81
6.6 Architektur des implementierten Cross Archive Searching Services 86

Tabellenverzeichnis

2.1 Die 15 Dublin Core Elemente 9
2.2 Bedeutung der Dublin Core Elemente 10

4.1 Beispiel: ‘Naives’ globales Ranking 48
4.2 Beispiel: Verbessertes globales Ranking 49
4.3 Die Abbildung von lokalen auf globale Deskriptoren mittels ρ . . 51
4.4 Beispiel: Finden gemeinsamer Eckpunkte – Ausgangslage 54
4.5 Beispiel: Finden gemeinsamer Eckpunkte – Endergebnis 55

5.1 Vergleich von Cross Archive Searching Services 66

C.1 OAI Data Provider - Teil 1 104
C.2 OAI Data Provider - Teil 2 105
C.3 OAI Data Provider - Teil 3 106

Literaturverzeichnis

[BCF+02] Boag, S.; Chamberlin, D.; Fernandez, M. F.; Florescu D.; Robie, J.; Siméon, J.; Stefanescu, M.: *XQuery 1.0: An XML Query Language*, W3C Working Draft, 2002.
http://www.w3.org/TR/xquery

[BEK+00] Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H. F.; Thatte, S.; Winer, D.: *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, 2000,
http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[BM01] Biron, P. V.; Malhotra, A.: *XML Schema Part 2: Datatypes*, W3C Recommendation, 2001.
http://www.w3.org/TR/xmlschema-2

[BPSM00] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.: *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 2000.
http://www.w3.org/TR/REC-xml

[BTN00] Barton J. J.; Thatte, S.; Nielsen, H. F.: *SOAP Messages with Attachments*, W3C Note, 2000.
http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211.

[CD99] Clark, J.; DeRose, S.: *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 1999.
http://www.w3.org/TR/xpath

[Cla99] Clark, J.: *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, 1999.
http://www.w3.org/TR/xslt

[corba] *Common Object Request Broker Architecture and Specification*, Object Management Group, Inc. (OMG), Version 2.6, 2001.
http://www.omg.org.

[cmv8] *Training Presentations*, IBM Content Manager Version 8 Beta Program, PartnerWorld For Developers, IBM, 2001.

[Dat90] Date, C. J.: *An Introduction to Database Systems*, Band 1. Addison-Wesley, Reading, MA, 5. Auflage, 1990.

[DMO01] DeRose, S.; Maler E.; Orchard, D: *XML Linking Language (XLink) Version 1.0*, W3C Recommendation, 2001.
http://www.w3.org/TR/xlink

[Eli44] Eliot, T. S.: *Four quartets*, London : Faber and Faber, 1944.

[EF00] Endres, A.; Fellner, D. W.: *Digitale Bibliotheken: Informatik-Lösungen für globale Wissensmärkte*, dpunkt.Verlag, 1.Auflage, 2000.

[Fal01] Fallside, D. C.: *XML Schema Part 0: Primer*, W3C Recommendation, 2001.
http://www.w3.org/TR/xmlschema-0

[FB96] Freed, N.; Borenstein, F.: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, Request for Comments 2046, 1996.
http://www.faqs.org/rfcs/rfc2046.html

[FGM+97] Fielding, R.; Gettys, J.; Mogul, J. C.; Frystyk, H.; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*, Request for Comments 2616, 1997.
http://www.faqs.org/rfcs/rfc2616.html

[GG97] Gravano, L.; Garcia-Molina, H.: *Merging Ranks from Heterogeneous Internet Sources* In: *Proceedings of the 23rd VLDB Conference*, 1997.

[Gri98] Griffel, F.: *Componentware – Konzepte und Techniken eines Softwareparadigmas*, dpunkt.Verlag, 1998.

[Har92] Harman D.: *Ranking Algorithms* In: Frakes, W.B.; Baeza-Yates, R. (Herausgeber): *Information Retrieval – Data Structures And Algorithms*, Kapitel 14, Prentice Hall PTR, 1992.

[HM01] Heuer, A.; Meyer, H.: *Digitale Bibliotheken*, Vorlesungsskript, 2001.
http://wwwdb.informatik.uni-rostock.de/Lehre/Vorlesungen/skripte/digbib2001/

[HMPT00] Heuer, A.; Meyer, H.; Porst B.; Titzler P.: *BlueView: Virtual Document Servers for Digital Libraries*, In: *Proceedings of IEEE Advances in Digital Libraries 2000 (ADL 2000)*, Washington D.C., Seiten 207–217, 2000.

[KHK02] Krönert, U.; Hegner, M.; Kupferschmidt, J.: *MyCoRe – Metadatenverarbeitung*, MyCoRe-Dokumentation, Komponente Dokumenten- und Personen-Metadaten, 2002.
http://www.mycore.de/repository/cgi-bin/viewcvs.cgi/∼checkout∼/mycore/documentation/MyCoRe_Metadaten.rtf

[LC95] Lasher, L.; Cohen, D.: *A Format for Bibliographic Records*, Request for Comments 1807, 1995.
http://www.faqs.org/rfcs/rfc1807.html

[LM78] Lockemann, P. C.; Mayr, H. C.: *Rechnergestützte Informationssysteme*, Springer-Verlag, Berlin, Heidelberg, New York, 1978.

[LMZN01] Liu, X.; Maly, K.; Zubair, M.; Nelson, M. L.: *Arc - An OAI Service Provider for Digital Library Federation*, D-Lib Magazine, Volume 7, Number 4, 2001.
http://www.dlib.org/dlib/april01/liu/04liu.html

[Mil99] Miller, P.: *Z39.50 for All*, Ariadne Issue 21, 1999.
http://www.ariadne.ac.uk/issue21/z3950

[MK60] Maron, M. E.; Kuhns, J.L.: *On Relevance, Probabilistic Indexing and Information Retrieval*, In: Journal of the ACM, Volume 7, Issue 3, Seiten 216–244, Association for Computing Machinery Press, New York, 1960.

[SB98] Sander-Beuermann, W.: *Schatzsucher – Die Internet-Suchmaschinen der Zukunft*, In: *c't Computertechnik*, Heft 13/1998, Seite 176, 1998.

[SH99] Saake, G.; Heuer, A.: *Datenbanken – Implementierungstechniken*, MITP-Verlag, Bonn, 1999.

[sql00] *SQL Multimedia and Application Packages (SQL/MM), Part 2: Full-Text*,International ISO/IEC Standard, 2000.

[sql01] *SQL Multimedia and Application Packages (SQL/MM), Part 5: Still Image*,International ISO/IEC Standrad, Comittee Draft, 2001.

[TBMM01] Thompson, H. S.; Beech D.; Maloney, M.; Mendelsohn, N.: *XML Schema Part 1: Structures*, W3C Recommendation, 2001.
http://www.w3.org/TR/xmlschema-1

[Tit99] Titzler, P.: *Entwurf und Implementierung eines Virtuellen DokumentenServers*, Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1999.

[VL01] Van de Sompel, H.; Lagoze C.: *The Open Archives Initiative Protocol for Metadata Harvesting*, Protocol Version 1.1, Open Archives Initiative, 2001.
http://www.openarchives.org/OAI/1.1/openarchivesprotocol.htm

[W3C98] World Wide Web Consortium, DOM Working Group members: *Document Object Model (DOM) Level 1 Specification: Version 1.0*, W3C Recommendation, 1998.
http://www.w3.org/TR/REC-DOM-Level-1

[z3950] ANSI/NISO Z39.50-1995, *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*, Juli 1995. *http://lcweb.loc.gov/z3950/agency/1995doce.html*

[Zei01] Zeitz, A.: *Personalisierungskonzepte für Digitale Bibliotheken*, Diplomarbeit, Universität Rostock, Fachbereich Informatik, 2001.

[ZZZ+01] Zeitz, A.; Zimmermann, G.; Zarick, M.; Romberg, C.; Below B.: *Ein Virtueller Dokumentenserver für Digitale Bibliotheken* In: Höpfner, Hagen (Herausgeber): *Beitragsband zum Studierenden-Programm bei der 9. Fachtagung "Datenbanken in Büro, Technik und Wissenschaft"*, Seiten 29–31. GI Fachausschuss 2.5, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2001.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 30. Juni 2002 Mathias Zarick

Thesen

1. Der Content Manager Version 8 wird wesentliche Novationen bringen, die dem MyCoRe-Projekt zugute kommen werden.

2. Eine Digitale Bibliothek braucht Schnittstellen nach außen, um von einem Metasystem für eine verteilte Suche integriert zu werden. Diese sollten möglichst standardisiert sein, um Interoperabilität zu schaffen. Solche Standards sind beispielsweise OAI oder Z39.50.

3. Es gibt 2 wesentliche Ausprägungen für eine Realisierung einer verteilten Suche, die der echten verteilten Suche “On-the-Fly” und die der Replizierung von Metadaten und oder Dokumenten in einen lokalen Datenbestand.

4. Die Dublettenerkennung bei einer verteilten Suche ist nur äußerst schwierig umsetzbar. Gleichheit müsste eher geraten werden. Die Nutzung von globalen Identifizierungsmechanismen, wie zum Beispiel der des DOI, können Abhilfe bringen.

5. Um ein globales Ranking adäquat zu realisieren, reicht es nicht aus, einfach nur die auf den autonomen Instanzen gültigen Rankingwerte zu übermitteln. Es müssen ferner die für eine Rankingfunktion notwendigen statistischen Parameter für eine Berechnung mitübermittelt werden. Das *Reranking* und das *Finden gemeinsamer Eckpunkte* sind zwei Heuristiken, die ohne eine solche Übermittlung von statistischen Informationen für ein globales Ranking auskommen.

6. Eine verteilte Suche konnte für MyCoRe relativ einfach implementiert werden, die Implementierung erfüllt nahezu alle wesentlichen Anforderungen an verteilte Suchen, die im Rahmen dieser Arbeit formuliert wurden.

7. OAI bietet keine Suchfunktionalität an und bietet deshalb auch keinen Ansatzpunkt für eine echte verteilte Suche.

8. Viele OAI-Anbieter (Data Provider) halten sich nicht korrekt an den Standard und erschweren dadurch ein Metadaten-Retrieval für eine Replizierung.