Entwurf und Implementierung eines Virtuellen Dokumenten-Servers

Diplomarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von Titzler, Patrick
geboren am 05.03.1973 in Rostock

Betreuer: Prof. Dr. Andreas Heuer
Dr.-Ing. Holger Meyer
Dipl.-Inf. Joachim Kröger

Abgabedatum: 31.05.1999

Zusammenfassung

Die Recherche in Dokumenten und Metadaten sowie der Zugriff auf die ermittelten Dokumente und Metadaten, die in heterogenen, autonomen Systemen (LDS) abgelegt sind, soll durch den Entwurf eines Virtuellen Dokumenten-Servers (VDS) ermöglicht werden. Die Architektur des VDS wird anhand von Anforderungen entworfen, die sich aus der Funktionalität der lokalen Dienste des VDS sowie der für externe Dienste bereitzustellenden Funktionalität ableiten lassen.

Eine prototypische Implementierung demonstriert die Basisfunktionalität des Systems und bildet den Grundstein für die Erweiterung, deren endgültige Funktionalität im Zusammenwirken mit externen Systemen ein globales Informationssystem — eine Digitale Bibliothek — bildet.

Abstract

The goal of this master thesis is the development of a system called Virtual Document Server (VDS), which allows the user to search in documents and their associated metadata and to retrieve the results. A VDS provides a uniform way to access the documents and metadata, which are stored in autonomous and heterogenous systems called LDS. The development of the system architecture is based on requirements of the local and external services, that a VDS has to provide.

The prototypic implementation results in a system with basic functionality. In conjunction with other systems, an enhanced version of this prototype will form a global information system — a digital library.

CR-Klassifikation

D.1.5. Object-oriented Programming
D.2.1. Requirements/Specifications
H.2.3. Query languages
H.2.5. Heterogeneous Databases
H.2.8. Database applications
H.3.3. Information Search and Retrieval

Key Words

Digitale Bibliotheken, Information Retrieval, verteilte Suche, Anbindung heterogener Datenbanken

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Suche nach Informationen stellt für einen Anwender einen wesentlichen Bestandteil seiner Arbeit dar. Die ersten Ansätze der Informationsbereitstellung basierten auf zentral gehaltenen Daten. Der Wandel von einer rein zentralen zu einer dezentralen Informationsbereitstellung wurde eine Notwendigkeit, die in dem stetigen Anwachsen der zu archivierenden Informationen (seien sie in Form von Dokumenten, Bildern, Videos usw. abgelegt) begründet lag. Der dadurch stark zunehmende Ressourcenbedarf (Speicher, Rechenleistung usw.) zur Aufbereitung, Verwaltung und Bereitstellung der Informationen setzt der rein zentralen Haltung von Informationen Grenzen.

Durch die dezentralisierte Bereitstellung von Informationen ergeben sich neue Anforderungen an Informationssysteme (*Digital Libraries*), die eine Suche in diesen Informationsquellen vornehmen. Laut [Pae96] sind die Anforderungen an eine Digitale Bibliothek

- die Lokalisierung und Selektion von Quellen,
- das Retrieval der Informationen,
- die Interpretation der Informationen,
- die lokale Verwaltung der Informationen sowie
- die Bereitstellung der Informationen.

Vor einem Anwender bleibt dabei idealerweise verborgen, wo und wie die Informationen zu suchen sind, in welcher Form sie vorliegen und wie auf sie zugegriffen werden kann.

Untersuchungen zur Entwicklung von Digitalen Bibliotheken sind Bestandteil von derzeit national und international durchgeführten bzw. bereits abgeschlossenen Projekten wie [dig98] und [med98].

Die in diesem Kapitel vorgenommene Vorstellung des zur Zeit in Deutschland laufenden Projektes *Global-Info* dient zur Motivation der vorliegenden Diplomarbeit. In der sich daran anschließenden Gliederung wird ein Überblick über den weiteren Aufbau der Arbeit gegeben.

1.1 Das Förderkonzept Global-Info

Das Ziel des vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) geförderten Global-Info-Konzeptes[1] [gi98] ist es, Wissenschaftlern von ihren Arbeitsplätzen aus einen effizienten Zugang zu elektronischen und multimedialen Volltext-, Literaturhinweis-, Fakten- und Softwareinformationen zu ermöglichen.

Für die Entwicklung der dazu erforderlichen globalen Informationssysteme wurden in [Sch98] fünf Schwerpunkte formuliert, die hier nur stichwortartig wiedergegeben werden:

1. Ergänzung und Bearbeitung von Inhalten (durch Autorenwerkzeuge usw.)
2. Vernetzung von Lehr- und Lernmaterialien zum elektronisch unterstützten Lehren und Lernen (Lernsoftware, Interaktive Lehrgänge usw.)
3. Erschließung und Beschreibung von Inhalten (Dokumentidentifikation, -beschreibung und -typisierung usw.)
4. Nutzung von Inhalten (Informationsvermittlung, Evaluierung usw.)
5. Wirtschaftlichkeitsmodelle, Billing und Abrechnung, Statistik (Lizenzfragen, Inkasso usw.)

Zur Bearbeitung der Schwerpunkte gliedert sich das Förderkonzept Gobal-Info in diverse Hauptprojekte, die als Sonderfördermaßnahmen (SFM) bezeichnet werden. Durch die Mitwirkung am zur Zeit laufenden Vorprojekt der SFM 6 ist die Universität Rostock an Global-Info beteiligt.

1.2 Die Sonderfördermaßnahme 6: BlueRose

Die SFM BlueRose[2] zum Global-Info Projekt dient der Entwicklung von Techniken zum Aufbau eines globalen Informationssystems für elektronisch verfügbare Fachinformationen. In Bezug auf die Ziele des Projekts Global-Info läßt die SFM sich in die Schwerpunkte der Erschließung und Beschreibung von Inhalten sowie der Nutzung von Inhalten einordnen.

Die Nutzung der Inhalte erfolgt beispielsweise durch Wissenschaftler, Bibliothekare und Studenten, die unter Verwendung der nachfolgend vorgestellten Dienste in der elektronisch verfügbaren Literatur recherchieren können.
In Abbildung 1.1 wird die Architektur der BlueRose-Dienste dargestellt, wobei die einzelnen Dienste folgende Funktionalitäten bereitstellen:

- Die *Dokumenten-Server* stellen Dokumente und Metadaten bei Informationsanbietern und -vermittlern bereit.

[1] Globale Elektronische und Multimediale Informationssysteme für Naturwissenschaft und Technik

[2] *B*uilding *L*ibraries *U*nifying *E*nhanced *R*etrieval-*O*riented user *S*ervices

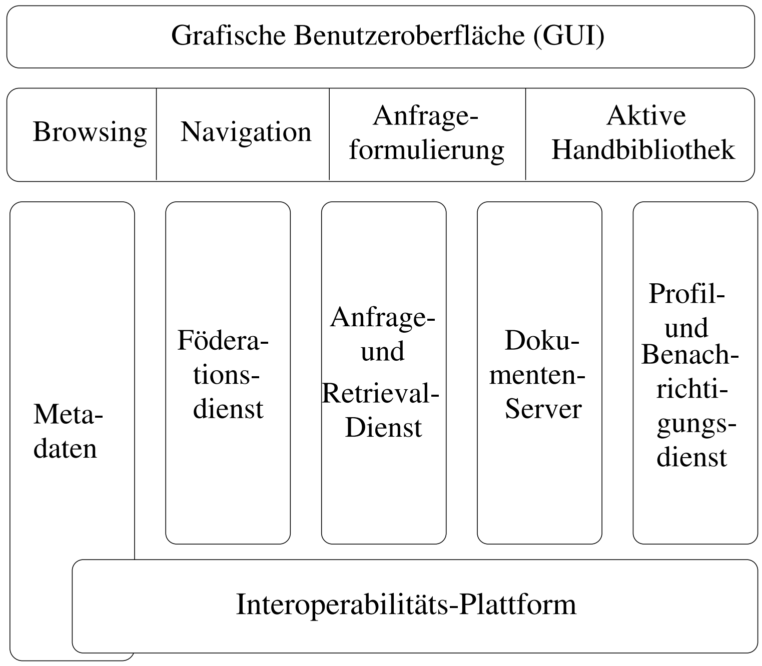

Abbildung 1.1: Architektur der BlueRose-Dienste

- Die *Anfrage- und Retrieval-Dienste* dienen zum Finden und Anfordern von Dokumenten und Metadaten.
- Der *Föderationsdienst* homogenisiert die heterogen vorliegenden Dokument- und Metadaten-Strukturen beim Zugriff und gestattet so die globale Interpretation von Daten.
- Der *Profil- und Benachrichtigungsdienst* informiert den Nutzer über neue bzw. aktualisierte Dokumente und Metadaten.
- Eine auf *Metadaten* gestützte Indexierung von Fachinformationen kann von Verlagen zum Anbieten von Informationen dienen.
- Andere Dienste dienen zum *Browsen* und zur *Navigation* in den lokalen bzw. durch Anfragen gefundenen Dokumenten und Metadaten sowie zur Unterstützung der Nutzer bei der *Formulierung von Nutzerprofilen und Anfragen.*
- Die *aktive Handbibliothek* verwaltet nutzerbezogene Dokumente und Metadaten sowie den Dokumenten zugeordnete Annotationen.
- Die einheitliche *Grafische Benutzungsoberfläche* (GUI) realisiert den Zugriff auf die bereitgestellten Dienste.

In [Heu99] ist eine Beschreibung der Anwendungsarchitektur der BlueRose-Dienste zu finden, die das Zusammenwirken der einzelnen Dienste verdeutlicht.

Der Fachbereich Informatik der Universität Rostock mit seinen beteiligten Partnern der Fachbereiche Mathematik, Chemie und Physik sowie der Universitätsbibliothek und dem Rechenzentrum zeichnet sich im zugeordneten Teilprojekt für

- die Verwaltung semistrukturierter Dokumente und strukturierter Metadaten in Datenbanken (Dokumenten-Servern),
- die Anbindung des Anfrage- und Retrieval-Dienstes an die Dokumenten-Server, existierende Literatur-Recherche-Dienste und Suchmaschinen sowie
- die Kopplung von verteilten Anfragen und verteiltem Retrieval in einer Sprache (IRQL) verantwortlich.

Im Rahmen dieses Teilprojektes werden am hiesigen Fachbereich zur Zeit drei Diplomarbeiten bearbeitet. Neben der vorliegenden Arbeit handelt es sich um [Por99] und [Jon99], auf deren Ziele zu einem späteren Zeitpunkt eingegangen wird.

1.3 Bezug zur vorliegenden Arbeit

Um den Zugriff und die Recherche von Nutzern bzw. Informationsvermittlern (z.B. Bibliotheken) auf Dokument-Metadaten und Dokumenten zu realisieren, ist der Aufbau von Virtuellen Dokumenten-Servern (VDS) erforderlich,

- die lokal Dokumente und deren Metadaten, die von Autoren, Verlagen oder anderen Informationsanbietern bereitgestellt werden, in Datenbanken verwalten und
- die Anfragen bzw. Zugriffe auf lokale Dokumente und Metadaten bearbeiten und extern zu bearbeitende Anfragen bzw. Zugriffe durch andere Informationsanbieter durchführen lassen. Die Weiterleitung der nicht lokal bearbeiteten Anfragen und Zugriffe geschieht für den Nutzer weitestgehend transparent, soweit keine Autorisierung bzw. Abrechnung für die Erbringung der Dienstleistung erfolgen muß.

Ziel der vorliegenden Arbeit ist es, die für die Erbringung der beschriebenen Funktionalität erforderlichen Dienste eines VDS zu analysieren, die Architektur des VDS zu entwerfen, die externen Schnittstellen der lokalen VDS-Dienste zu beschreiben und prototypisch zu implementieren.

1.4 Gliederung der Arbeit

Zur Realisierung der Ziele gliedert sich diese Arbeit wie folgt:

Kapitel 2. Ausgangspunkt dieser Arbeit bildet die Analyse der lokalen Dienste, aus denen sich ein VDS zusammensetzt. Ergänzend werden die Betrachtungen auf externe Dienste ausgedehnt, die Funktionalitäten für die VDS-Dienste bereitstellen bzw. deren Funktionalitäten genutzt werden. Durch die Analyse können die Schwerpunkte der weiteren Arbeit identifiziert und eine Ausgangsbasis für den Architekturentwurf geschaffen werden.

Kapitel 3. Aus den Anforderungen wird eine mögliche Architektur des Virtuellen Dokumenten-Servers hergeleitet und die Schnittstellen der VDS-Dienste beschrieben, die diese für in- und externe Dienste bereitstellen.

Kapitel 4. Um die an einen VDS adressierten Anfragen an Dokumente durch die LDS bearbeiten zu können, müssen diese Anfragen in Konstrukte der Anfragesprache der den LDS zugrunde liegenden Datenquellen überführt werden. Für drei konkrete Systeme werden dazu konkrete Untersuchungen vorgenommen, deren Ergebnis eine Einschätzung der Umsetzbarkeit der betrachteten IRQL-Klauseln in die Anfragesprache der Systeme bildet.

Kapitel 5. Die Beschreibung der Umsetzung des Architekturentwurfs in einen ersten Prototypen leitet den praktischen Teil dieser Arbeit ein. Dazu werden für die Implementierung des Basissystems, das erste prototypische Funktionalitäten bereitstellt, geeignete Ansätze vorgestellt und diskutiert.

Kapitel 6. In einer zusammenfassenden Betrachtung werden die wesentlichen Punkte dieser Diplomarbeit rekapituliert und ein Ausblick auf mögliche Folgearbeiten gegeben, die den Prototypen komplettieren.

Literaturverzeichnis. Das Literaturverzeichnis umfaßt Artikel, Bücher sowie online verfügbare Dokumente, die in der Diplomarbeit referenziert werden.

Anhang A. Die formale IDL-Spezifikation der externen Dienstschnittstellen bietet einen Überblick über die von den VDS-Diensten bereitgestellten Funktionalitäten.

Kapitel 2

Anforderungen an den VDS

In der Kurzvorstellung der BlueRose-Dienste wurde bereits dargelegt, welche Dienste die Funktionalität des zu entwickelnden Informationssystems bereitstellen. Nicht alle der aufgeführten Dienste müssen beim Entwurf der VDS-Architektur berücksichtigt werden, da einige keinen Einfluß auf dessen Funktionalität besitzen.

Ein VDS ist für das Finden und Anfordern von lokalen bzw. externen Dokumenten sowie deren Metadaten zuständig. Die Dokumente und ihre Metadaten werden von heterogenen, autonomen Lokalen Dokumenten-Servern (LDS) bereitgestellt, auf die durch einen vom VDS bereitgestellten Dienst homogen zugegriffen werden kann. Der VDS realisiert dabei eine Transparenz, die vor dem Nutzer verbirgt, in welchem LDS die angeforderten Daten gespeichert sind und wie auf sie zugegriffen wird. Wenn beim Zugriff oder der Recherche in Dokumenten eine Autorisierung erforderlich ist, wird ein gewisser Teil der Transparenz aufgehoben und eine zusätzliche Interaktion zwischen Nutzer und Lizenzgeber erforderlich, die durch den VDS vermittelt werden kann.

Da sich das Projekt in einer sehr frühen Phase befindet, liegen keine vertiefenden Informationen über die von den Projektpartnern zu entwickelnden Dienste vor. Infolgedessen können die Untersuchungen zu den externen Diensten nicht in dem Umfang vorgenommen werden, in dem es wünschenswert wäre. Deshalb ist es erforderlich, an einigen Stellen dieses Kapitels allgemeine Betrachtungen vorzunehmen und Annahmen zu treffen, um Anforderungen spezifizieren zu können.

Ziel dieses Kapitels ist es, die Anforderungen an die Dienste des VDS zu identifizieren um darauf aufbauend eine Gesamtarchitektur zu entwerfen, die die ermittelten Anforderungen adäquat erfüllt und Spielraum für Erweiterungen bzw. Änderungen läßt. Aufgrund des frühen Projektstadiums können sich im weiteren Verlauf neue Anforderungen an einen VDS ergeben, die integriert werden müssen.

In Abschnitt 2.1 werden die zu untersuchenden Dienste eingeführt und die Reihenfolge der sich daran anschließenden Abschnitte, in denen die Dienste analysiert werden, motiviert. Das Kapitel endet mit einer kurzen Zusammenfassung der wesentlichen Erkenntnisse.

2.1 Einführung der zu untersuchenden Dienste

Der Virtuelle Dokumenten-Server bildet ein offenes, verteiltes System das

- einen oder mehrere Lokale Dokumenten-Server,
- einen Anfrage- und Retrieval-Dienst sowie
- einen Replikations- und Caching-Dienst

umfaßt, die miteinander kommunizieren (Abbildung 2.1). Im Lokalen Dokumenten-Server werden Dokumente und Metadaten verwaltet, die dem Nutzer durch den Anfrage- und Retrieval-Dienst (Query-Server) bereitgestellt werden. Dokumente, die nicht in einem der zugeordneten LDS abgelegt sind, werden durch den Replikations- und Caching-Dienst von externen Datenquellen angefordert (3) und lokal zwischengespeichert. Die Funktionalität des VDS beschränkt sich nicht auf

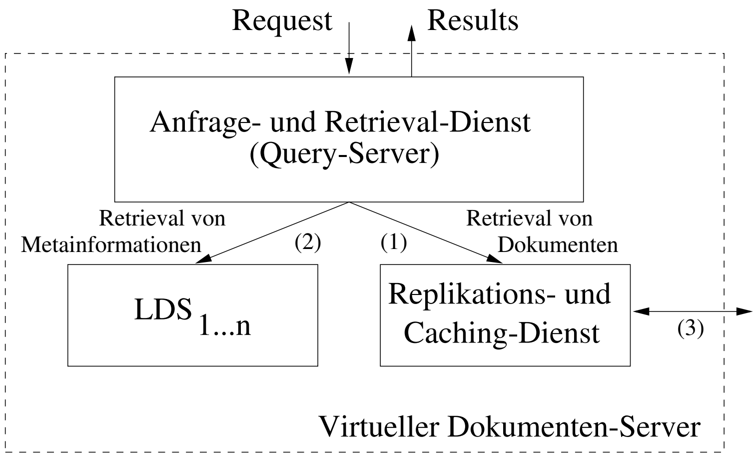

Abbildung 2.1: Zugriff auf lokale und externe Dokumente

das Retrieval von Dokumenten (1), sondern umfaßt z.B. auch die Bearbeitung von Anfragen, deren Ergebnisse Teildokumente bzw. Metainformationen sein können (2).

In dieser Arbeit werden die folgenden externen Dienste, die die Funktionalität des VDS verwenden, betrachtet:

- Föderationsdienste zur Integration verteilter Dokumentenbestände und
- Trader/Broker, die Vermittlungsdienste bereitstellen.

Weitere Dienste, wie der Profil- und Benachrichtigungsdienst, werden vom VDS genutzt bzw. nutzen dessen Funktionalität. Sie werden in dieser Arbeit jedoch nicht vertiefend analysiert.

Die Untersuchung zur Bestimmung der Anforderungen an den VDS beginnen mit dem Lokalen Dokumenten-Server in Abschnitt 2.2, da dieser die Datenquelle bildet, der die Dokumente bereitstellt. Die anschließende Betrachtung des Anfrage- und Retrieval-Dienstes in Abschnitt 2.3 ist eng mit den anderen Diensten verknüpft, da dieser für den Zugriff auf die Dienste des VDS sowie auf die externen Dienste verantwortlich ist. Wie der Zugriff auf externe Dokumente erfolgt, wird in der Betrachtung zum Replikations- und Caching-Dienstes in Abschnitt 2.4.1 näher beleuchtet. Den Abschluß der Untersuchungen bilden Betrachtungen zum Föderationsdienst in Abschnitt 2.5 und dem Komplex Trader/Broker in Abschnitt 2.6, die als externe Dienste die Funktionalität des VDS benutzen bzw. für den VDS Funktionalität bereitstellen. Neben den hier betrachteten Diensten gibt es weitere, die den VDS nutzen. Da sie aber keinen direkten Einfluß auf den zu entwickelnden Architekturvorschlag haben, werden sie in Abschnitt 2.7 nur der Vollständigkeit halber genannt. In Abschnitt 2.8 werden die Untersuchungen zusammengefaßt und Schwerpunkte der weiteren Arbeit identifiziert.

2.2 Lokaler Dokumenten-Server

Der in Abbildung 2.2 dargestellte Lokale Dokumenten-Server verwaltet die lokal gehaltenen Dokumente und dazu gehörende Metadaten. Bei den LDS handelt es sich um Datenbanksysteme, die Anfragemethoden auf strukturierten und semistrukturierten Dokumenten sowie Information-Retrieval-Methoden auf diesen Dokumenten bereitstellen. Ein Lokaler Dokumenten-Server bildet ein autonomes System, dessen Verwaltung nicht durch den VDS erfolgt. Aufgrund der Autonomie kann ein LDS auch nicht durch einen Dienst soweit gekapselt werden, daß der Zugriff von nicht-VDS-Diensten auf geschützte Dokumente unterbunden werden kann. Deshalb muß der Replikations- und Caching-Dienst für die Zwischenspeicherung von Dokumenten auf einen dedizierten LDS zurückgreifen, auf den eine volle Kontrolle ausgeübt werden kann.

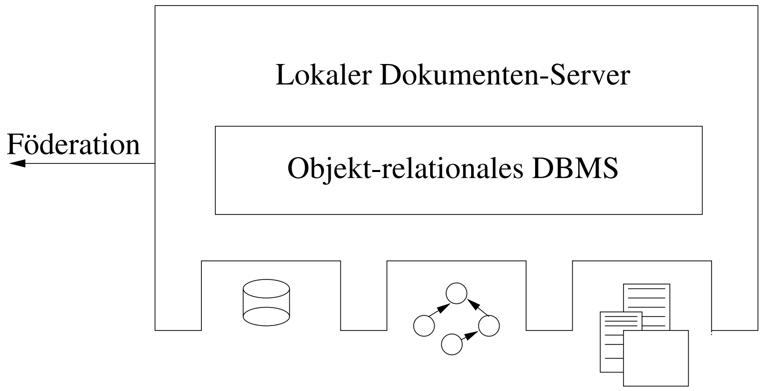

Abbildung 2.2: Lokaler Dokumenten-Server

Eine ganz spezielle Ausprägung des Lokalen Dokumenten-Servers stellen externe Literatur-Recherche-Dienste wie MathNet [mathnet] und PhysNet [physnet] dar.

Kennzeichnend für diese beiden Dienste ist, daß sie die indizierten Dokumente nicht selbst speichern, sondern lediglich Metadaten dieser Dokumente (Titel, Schlüsselwörter usw.) bereitstellen.[1]

Im Rahmen einer parallel laufenden Diplomarbeit werden zur Zeit Untersuchungen vorgenommen, die die Eignung der objektrelationalen Datenbanksysteme *DB2 Universal Database* und *INFORMIX Universal Server* zur Darstellung, Verwaltung und Suche von XML-Dokumenten bewerten [Por99].

Von besonderer Bedeutung für die vorliegende Arbeit sind die Ergebnisse der Untersuchungen zur Erweiterbarkeit der bereits vorhandenen Text-Datentyperweiterungen[2], da diese Erweiterungen die IR-Anfragebestandteile der IRQL (bspw. die contains-Funktion) realisieren sollen. In Abschnitt 4.2.3 werden dazu einige Betrachtungen vorgenommen.

Neben den objektrelationalen Datenbanksystemen wird in der prototypischen Implementierung das Volltextdatenbanksystem *Fulcrum SearchServer* [ssi98] als LDS verwendet, da es sich durch sehr gute IR-Anfragemöglichkeiten (und Ergebnisse [Mey96]) auszeichnet und mit Hilfe der SQL-artigen Anfragesprache auch strukturierte Anfragen an den Datenbestand gestellt werden können. In Abschnitt 4.2.1 werden Untersuchungen zur Anfragesprache des *Fulcrum SearchServers* vorgenommen.

2.2.1 Anforderungen

Die Hauptaufgabe des LDS besteht in der Verwaltung der Dokumente und der Bereitstellung einer Anfrageschnittstelle für die Dokumente. Die dabei auftretenden Anfragen können ein Dokument (Retrieval-Anfrage) bzw. Informationen über Dokumente anfordern. Die zu verwaltenden Dokumente können in frei verfügbare und lizenzrechtlich geschützte Dokumente unterteilt werden. Letztere erfordern eine Autorisierung durch den Nutzer. Betrachtungen zur Autorisierung werden bei der Vorstellung des Anfrage- und Retrieval-Dienstes in Abschnitt 2.3 vorgenommen.

Obwohl die Verwaltung der Dokumente in heterogenen LDS erfolgt, ist ein einheitlicher Zugriff auf die Dokumente und ihre Metainformationen zu gewährleisten, um den zugreifenden VDS-Diensten eine einheitliche Schnittstelle zur Verfügung zu stellen. Die Überwindung der Heterogenität erfordert deshalb die Einführung einer Komponente, die für die Abbildungen der Konstrukte der VDS-Anfragesprache auf die Anfragesprache des dem LDS zugrunde liegenden Datenbanksystems verantwortlich ist. Von besonderer Bedeutung ist außerdem die Vergabe einer eindeutigen Dokument-Identität (DOI, *Digital Object Identifier*), die

[1] Die beiden aufgeführten Dienste basieren beide auf dem Harvest-System, das die indizierten Dokumente nicht selbst verwaltet. Der Zugriff auf die Dokumente erfolgt in diesem Fall demzufolge auch unabhängig vom Anfrage-Dienst.

[2] In DB2 wird die Datentyperweiterung als *Text-Extender* [IBM97] und in Informix als *Text Datablade* [Inf97] bezeichnet.

das Laden der Dokumente überhaupt erst ermöglicht. Im Einzelnen können die folgenden Anforderungen an einen LDS aufgestellt werden:

Einheitlicher Zugriff auf die LDS. Lokale Dokumenten-Server können sich im Datenmodell, im Schema, in der Schnittstelle zum Zugriff auf die Funktionen (API) des zugrunde liegenden Datenbanksystems und den Anfragemöglichkeiten unterscheiden. Um zu verhindern, daß die Heterogenität dazu führt, daß für den Zugriff auf jeden LDS eine eigene Schnittstelle angeboten werden muß, wird eine Komponente eingeführt, die eine Homogenisierung vornimmt. Diese Komponente wird als Wrapper bezeichnet. Durch die Einführung dieser Komponente kann praktisch jeder LDS an einen VDS angebunden werden.

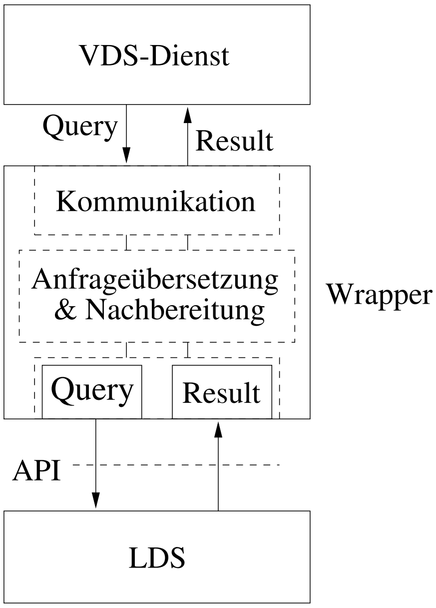

Abbildung 2.3: Aufbau eines Wrappers

Betrachtet man den Aufbau des Wrappers in Abbildung 2.3, so sind die Eingaben Anfragen in einer bestimmten Anfragesprache (IRQL) und die Ausgaben Dokumente bzw. Metainformationen. Die Aufgaben eines Wrappers umfassen

- die Kommunikation mit dem aufrufenden Dienst,
- die Kommunikation mit der Datenquelle (via API),
- die Umformung der IRQL-Anfrage in die Zielanfragesprache des LDS,
- die Nachbereitung der Ergebnisse (Erweiterung der Funktionalität einer Datenquelle durch die Ergebnisnachbearbeitung, Umformung in globales Datenmodell) und
- die Fehlerbehandlung und -weiterleitung.

Da sich die Anfragesprache des VDS (IRQL) und des LDS (z.B. SQL bzw. ein Subset von SQL) in der Syntax unterscheiden, muß eine Abbildung in die entsprechende Zielsprache erfolgen. Da auch die Mächtigkeit der Sprachen verschieden sein kann, besteht die Möglichkeit, bestimmte Konstrukte der Anfragesprache des VDS durch einen oder mehrere Konstrukte der Anfragesprache des LDS zu ersetzen bzw. diese Konstrukte durch den Wrapper simulieren zu lassen.[3] Je nach Komplexität des Wrappers können Funktionen ausgelagert werden bzw. weitere Funktionen[4] hinzukommen.

Verwaltung der Dokumente. Die wesentliche Aufgabe des LDS ist die Verwaltung der Dokumente und die Aufbereitung der extrahierten Daten für Anfragen. Zur Verwaltung der Dokumente zählen das Einfügen neuer und das Löschen alter Dokumente sowie das Ändern einzelner Dokumentbestandteile. Der LDS ist nur für die Speicherung der Dokumente zuständig. Sind zu einem Dokument mehrere Versionen zu speichern, werden der Zugriff und die Verwaltung der einzelnen Versionen durch den Replikations- und Caching-Dienst kontrolliert. Zur Aufbereitung der Daten gehören beispielsweise die Erzeugung eines Indexes für IR-Anfragen.

Vergabe der eindeutigen Dokument-Identität. Um ein Dokument eindeutig identifizieren zu können und den Zugriff auf das Dokument zu ermöglichen, ist es notwendig, jedem Dokument einen global eindeutigen Schlüssel zuzuordnen, der als DOI (*Digital Object Identifier*) bezeichnet wird.

Der Schlüssel könnte praktischerweise so gewählt werden, daß sich aus diesem semantische Informationen ableiten lassen, anhand derer der LDS ermittelt wird, in dem das Dokument gespeichert ist. Die Information, in welchem LDS das Dokument gespeichert ist, könnte vom für das Laden der Dokumente zuständigen lokalen VDS-Dienst genutzt werden, um den zu kontaktierenden LDS zu ermitteln. Dies würde den Zugriff auf das Dokument vereinfachen, da die Lokalisierung ohne großen Aufwand erfolgen kann. Die Integration von semantischen Informationen in den DOI kann jedoch auch Probleme hervorrufen, wenn beispielsweise die Implementierung der VDS-Dienste intern die semantische Information nutzt, dies jedoch nach außen nicht bekannt ist. Wenn die Struktur des DOI geändert wird, könnten so beispielsweise bei der Programmbearbeitung unerwartete Nebenwirkungen auftreten.

Deshalb ist es sinnvoller, für die interne Identifizierung und Lokalisierung eines Dokumentes einen physischen Schlüssel einzuführen, der für den Nut-

[3] Wenn z.B. eine Anfragesprache keine Projektion unterstützt, kann diese vom Wrapper nachträglich auf den Ergebnissen durchgeführt werden.

[4] Wie beispielsweise die Anfrageoptimierung bei der Bearbeitung von Anfragen, die in Teilanfragen zerlegt wurden, wenn die ursprüngliche Anfrage in der angegebenen Form nicht von der darunterliegenden Datenquelle bearbeitet werden konnte. (Diese Anfrageoptimierung ersetzt nicht die Anfrageoptimierung der Datenquelle.)

zer transparent ist. Die Trennung in logischen Schlüssel[5] und physischen Schlüssel[6] erweist sich als praktisch, da einem logischen Schlüssel mehrere physische Schlüssel zugeordnet werden können. Dies ist möglich, da zu einem Originaldokument mehrere Replikate existieren können, so daß diese zwar den gleichen DOI aber nicht den gleichen physischen Schlüssel besitzen (weil sie in verschiedenen LDS gespeichert sind).

Ein n-teiliger physischer Schlüssel könnte beispielsweise die folgende Struktur aufweisen

$$\underbrace{de.uni-rostock.informatik}_{eindeutigeDomäne}.\underbrace{ldsX}_{LDS-ID}.\underbrace{dokumentZ}_{Dokument-ID}$$

wobei jedem Teil des Schlüssels semantische Informationen zu entnehmen sind:

- die Domäne, die dem VDS zugeordnet ist
- die LDS-ID des LDS, in dem das Dokument abgelegt ist
- die für einen LDS eindeutige Dokumentidentifikation

Diese kurzen Betrachtungen zum physischen Schlüssel wurden nur der Vollständigkeit halber vorgenommen und sind nur für die Implementierung des Zugriffs auf die Dokumente von Bedeutung. Konkrete Entscheidungen zum Aufbau und der Nutzung des physischen Schlüssels sind in [Jon99] zu finden. Die weitere Diskussion bezieht sich nur auf den logischen Schlüssel, den DOI.

Aus Sicht des LDS muß sichergestellt werden, daß die Datenquelle den Zugriff auf die bereitgestellten Dokumente anhand des DOI durchführen kann. Ist dies nicht der Fall, muß eine eindeutige Abbildung des DOI auf den von der Datenquelle verwendeten Identifizierungsmechanismus für Dokumente vorgenommen werden. Beispielsweise könnte eine Datenquelle den Zugriff auf die von ihr verwalteten Dokumente durch Nutzung einer URL realisieren. Demzufolge müßte der Wrapper dieser Datenquelle bei der Anforderung eines Dokuments nicht dessen DOI sondern den URL an die Datenquelle weiterreichen.

Die Vergabe eines logischen Schlüssels für Dokumente wird von einer externen Institution erfolgen, die sicherstellt, daß eine globale Eindeutigkeit besteht. Mit der Erzeugung von DOIs beschäftigt sich die Sonderfördermaßnahme 12 zum Projekt Global-Info.[7]

[5] Wie lautet der DOI dieses Dokuments?

[6] Handelt es sich um ein Original oder ein Replikat und wo ist es gespeichert?

[7] Titel: "AAA-Services. Werkzeuge für die digitale Bibliothek: Authentifizierungs-, Autorisierungs- und Accounting-Dienste"

Export der Dokumente. Nutzer können alle in einem LDS gespeicherten Dokumente anfordern, sofern ihnen die Dokument-Identität und, falls erforderlich, die Autorisierungsinformationen bekannt sind. Anfragen, die ein komplettes Dokument (oder Teile davon) anfordern und Anfragen auf Metadaten der Dokumente müssen sich nicht zwangsläufig in der Syntax unterscheiden, sofern auf den Inhalt eines Dokumentes durch ein Attribut (z.B. Attribut `complete_content` eines geeigneten Typs) zugegriffen werden kann.

Export von Schemainformationen. Die Speicherung der Dokumente und ihrer Metadaten im LDS kann in verschiedenen Datenmodellen und Schemata erfolgen. Beim Zugriff auf externe LDS müssen deshalb zwangsläufig Transformationen durchgeführt werden, um Daten zwischen den verschiedenen VDS austauschen zu können. Diese Aufgabe wird vom Föderationsdienst übernommen, der dazu gewisse Informationen über die Modellierung des Datenbestandes benötigt. Aus diesem Grunde exportiert ein LDS auf Anfrage seine Schemainformationen (Dokumenttypen und deren Modellierung).

Schemainformationen sind statische Informationen. Deshalb können sie von der zentralen Informationsvermittlungsstelle des VDS, dem Anfrage- und Retrieval-Dienst persistent gehalten werden, um den wiederholten Zugriff auf den LDS zu vermeiden. Voraussetzung dafür ist ein Benachrichtigungssystem, das dem Anfrage- und Retrieval-Dienst mitteilt, sobald sich Schemainformationen geändert haben und ein erneutes Einlesen erforderlich ist.

Export von weiteren Metadaten. Die Selektion externer LDS, die zur Bearbeitung einer Anfrage hinzugezogen werden, erfolgt durch den Broker u.a. aufgrund von Metadaten, die den Datenbestand des LDS beschreiben (Anzahl der Dokumente und weitere statistische Daten wie Stichworthäufigkeiten und -verteilungen).

Diese Metadaten sind im Gegensatz zu den Schemainformationen dynamisch, da Änderungen im Datenbestand im Vergleich zu Schemaänderungen relativ häufig auftreten können. Die Verifikation der gespeicherten Metadaten kann in zyklischen Abständen oder auf explizite Aufforderung erfolgen.

2.2.2 Zusammenfassung

Ein Teil der beschriebenen Anforderungen richtet sich direkt an die Datenquelle, d.h. das Datenbanksystem, und hat keinen Einfluß auf den Architekturentwurf der VDS-Dienste. Zu diesen Anforderungen gehört die Integration des Dokumenten-Schlüssels in das Datenbankschema und die Verwaltung der Dokumente. Eine offene Anforderung ist der Entwurf eines geeigneten Schemas, das die Struktur bzw. die Metadaten der Dokumente angemessen abbildet. In Abhängigkeit von der (noch zu bestimmenden) Funktionalität der anderen Dienste können weitere Anforderungen an den LDS auftreten, die bisher nicht betrachtet wurden.

2.3 Anfrage- und Retrieval-Dienst

Der Anfrage- und Retrieval-Dienst, der als Query-Server bezeichnet wird, bildet im VDS den zentralen Anlaufpunkt für Nutzer- oder Dienst-Anfragen. Die Bearbeitung der Anfragen und ihre Verteilung auf die Lokalen Dokumenten-Server und den Replikations- und Caching-Dienst bilden einen Aufgabenschwerpunkt. Den zweiten Schwerpunkt stellt die Verwaltung der Metadaten der lokalen Dienste dar, die durch externe Dienste abgerufen werden können. Den dritten Schwerpunkt bildet die Verwaltung der lokalen VDS-Dienste selbst.

Der Query-Server kommuniziert mit einer Vielzahl von Diensten. Lokal besteht eine ständige Verbindung mit den LDS und dem Replikations- und Caching-Dienst. Die externen Dienste, wie der Föderationsdienst, werden lediglich bei Bedarf kontaktiert.

Anfragen an den Query-Server werden in einer eigens entwickelten Anfragesprache (IRQL[8]) spezifiziert, die die Vorteile von IR- und DB-Konzepten, wie z.B. vage Anfragen, Ranking, Relevance Feedback, Attributierung, Verknüpfung und Restrukturierung von Daten, in homogener Weise miteinander verknüpft. Der Query-Server ist für die Dekomposition der Anfrage in Teilanfragen zuständig, die durch die LDS bzw. deren Wrapper bearbeitet werden. Die Umformung der Teilanfragen in die Anfragesprache der Datenquelle wird durch die Wrapper erledigt, die diese anbinden. Die Ergebnisse werden vom Query-Server zusammengefaßt und gegebenenfalls nachbearbeitet.

Aufgrund der Heterogenität der den LDS zugrunde liegenden Schemata ist es erforderlich, ein globales Datenmodell zu schaffen, das die Ergebnisse einheitlich modellieren kann. Da der Query-Server keine Operationen auf den Dokument-Inhalten durchführen muß, kann diese Modellierung sehr einfach erfolgen. Für die Abbildung der Anfrageergebnisse auf dieses Datenmodell sind die Wrapper zuständig.

Zur Bearbeitung der Anfragen greift der Query-Server auf das Repository zurück, in dem Metadaten der zugeordneten LDS und Informationen zum Zugriff auf die Wrapper, die diese LDS anbinden, gespeichert sind. Das Repository gliedert sich in einen transienten und einen persistenten Bereich, zum einen um der Lebensdauer der Metadaten Rechnung zu tragen und zum anderen um die Performance positiv zu beeinflussen.

[8] *I*nformation *R*etrieval *Q*uery *L*anguage. Siehe Kapitel 4 für eine kurze Beschreibung.

2.3.1 Anforderungen

Die kurze Einführung in die Funktionalität des Query-Servers bietet einen guten Ansatzpunkt, um spezielle Anforderungen zu beschreiben, die aus den beschriebenen Aufgaben resultieren. Die Anfragebearbeitung wird einleitend betrachtet, um darauf aufbauend die notwendigen Metainformationen, die im Repository gespeichert sein müssen, spezifizieren zu können. Die Kommunikation zwischen dem Query-Server und den in- und externen Diensten rundet die Anforderungen ab und gibt eine Überleitung zum letzten noch nicht beschriebenen internen Dienst, dem Replikations- und Caching-Dienst.

Bearbeitung der Anfrage. Die LDS verwalten, wie bereits beschrieben, Dokumente und deren Dokument-Metadaten. Als Dokument-Metadaten sollen in dieser Arbeit die strukturiert vorliegenden Bestandteile der Dokumente, wie beispielsweise Autor, Titel, Abstrakt und Stichwörter, verstanden werden. Aber auch Daten, die nicht im Dokument gespeichert sind (wie die Größe und Prüfsumme) sondern aus dem Dokument abgeleitet werden bzw. dem Dokument zugeordnet sind, werden in dieser Arbeit als Dokument-Metadaten bezeichnet.

Der Zugriff auf die kompletten Dokumente bzw. ihre Bestandteile kann entweder frei sein oder aber eine Autorisierung bzw. eine kostenpflichtige Abrechnung erfordern. Für den Zugriff und die Suche in den Dokument-Metadaten der LDS gilt in der Regel, daß sie ohne Autorisierung oder Abrechnung durchgeführt werden können.[9]

Daraus folgt, daß bei der Bearbeitung von Anfragen berücksichtigt werden muß, ob für den Zugriff auf die Daten eine Autorisierung bzw. Abrechnung erforderlich ist oder nicht. Deshalb wird in der Anfragebearbeitung durch den Query-Server eine Unterscheidung in zwei Anfrageformen vorgenommen:

- *Metadaten-Retrieval-Anfragen* gestatten die Suche in den Dokument-Metadaten sowie in den einzelnen Bestandteilen der Dokumente. Als Ergebnis einer solchen Anfrage können nur die frei zugreifbaren Dokument-Metadaten sowie DOIs der ermittelten Dokumente zurückgeliefert werden. Diese DOIs können anschließend genutzt werden, um mit Hilfe von Dokument-Retrieval-Anfragen die Dokumente bzw. einzelne Bestandteile der Dokumente anfordern zu können.

[9] Die kommerziellen Anbieter von Dokumenten sind bestrebt, die in ihrem Besitz befindlichen Dokumente an Leser zu verkaufen. Der potentielle Kunde wird in der Regel vor dem Kauf daran interessiert sein, nähere Informationen (Abstrakt, eventuell ein Inhaltsverzeichnis) über die verfügbaren Dokumente zu erhalten. Diese Informationen stellen die Anbieter deshalb häufig kostenlos zur Verfügung, um ihm dafür eine Möglichkeit zu bieten. Es gibt natürlich auch Nachweissysteme, bei denen auch für die Recherche in diesen Informationen eine Autorisierung erforderlich ist. In dieser Arbeit wird davon ausgegangen, daß dies keinen der LDS betrifft.

Eine spezielle Ausprägung von Metadaten-Retrieval-Anfragen, die hier nur am Rande erwähnt wird, stellen Anfragen an die von den LDS aus dem Datenbestand ermittelbaren Metadaten (Anzahl der gespeicherten Dokumente, Anzahl der gespeicherten Dokumente eines Typs usw.) dar. Diese Anfragen liefern als Ergebnis die angeforderten Metadaten zurück.

- *Dokument-Retrieval-Anfragen* dienen zum Zugriff auf die Bestandteile der Dokumente (kompletter Inhalt, ein Kapitel usw.), wobei gegebenenfalls eine Autorisierung bzw. eine Abrechnung erfolgen muß. Als Ergebnis einer Dokument-Retrieval-Anfrage wird das Dokument bzw. der angeforderte Bestandteil des Dokuments zurückgeliefert. Dabei kann vom Anbieter für jeden LDS eine eigene Festlegung getroffen werden, auf welche Dokument-Metadaten bzw. Bestandteile von Dokumenten frei zugegriffen werden kann und welche Zugriffe autorisierungs- bzw. kostenpflichtig sind. Somit kann den individuellen Anforderungen der Anbieter Rechnung getragen werden.

Die Entscheidung, ob es sich bei einer Anfrage um eine Metadaten-Retrieval-Anfrage oder um eine Dokument-Retrieval-Anfrage handelt, kann anhand der Projektionsliste[10] der Anfrage getroffen werden. Treten in der Projektionsliste nur frei zugängliche Metadaten auf, handelt es sich um eine Metadaten-Retrieval-Anfrage. Für Metadaten-Retrieval-Anfragen gilt, daß autorisierungs- bzw. kostenpflichtige Dokument-Metadaten (oder Dokumentbestandteile) nicht im Anfrageergebnis enthalten sein dürfen. Die Suche, d.h. die Verwendung dieser Daten in Selektionsbedingungen, ist jedoch erlaubt. Ergebnis einer solcher Suche ist beispielsweise ein boolescher Wert (enthalten oder nicht) bzw., falls es sich um eine IR-Anfrage handelt, ein Ranking-Wert. Problematisch ist hierbei, daß verhindert werden muß, daß aus den Ergebnissen der Suche autorisierungs- bzw. kostenpflichtige Metadaten bzw. Dokumentbestandteile abgeleitet werden können.[11]

Eine kurze Beispiel-SQL-Anfrage verdeutlicht das bisher gesagte. Der Zugriff auf die Dokument-Metadaten *title* und *author* sei frei, der Zugriff auf den *Inhalt* des Dokumentes (`complete_content`) jedoch nicht. Die Anfrage

```
SELECT title, author
FROM documentbase
WHERE complete_content contains "Student"
```

[10] In einer Projektionsliste wird festgelegt, welche Attribute als Ergebnis einer Anfrage zurückgeliefert werden. In einer SQL-Anfrage entspricht dies den Attributen und Funktionen, die in der SELECT-Klausel spezifiziert werden.

[11] Wenn beispielsweise die Anforderung eines kompletten Buches autorisierungspflichtig ist, muß verhindert werden, daß in einer (oder mehreren Anfragen) alle Kapitel dieses Buches angefordert werden können.

wird als Metadaten-Retrieval-Anfrage bearbeitet, da nur die frei zugänglichen Dokument-Metadaten *title* und *author* der Dokumente, in denen Begriff *Student* vorkommt, das Ergebnis der Anfrage bilden.

Enthält die Projektionsliste hingegen Metadaten bzw. Dokumentbestandteile, für die der Zugriff eventuell autorisierungs- bzw. kostenpflichtig ist, muß es sich um eine Dokument-Retrieval-Anfrage handeln. Demnach ist die Anfrage

```
SELECT title, complete_content
FROM documentbase
WHERE title contains "Semesterferien"
```

eine Dokument-Retrieval-Anfrage, da für alle Dokumente, die den Begriff *Semesterferien* im Titel enthalten, der Inhalt (`complete_content`) zurückgeliefert werden soll.

Für Anbieter, die auch keine freie Suche in autorisierungs- bzw. kostenpflichtigen Metadaten und Dokumentbestandteilen zulassen wollen[12], muß die Unterscheidung zwischen Dokument-Retrieval-Anfragen und Metadaten-Retrieval-Anfragen anhand der Projektionsliste und den Selektionsbedingungen[13] der Anfrage erfolgen. In diesem Fall müßte auch die erste Beispielanfrage als Dokument-Retrieval-Anfrage behandelt werden, da die Suche in einem autorisierungs- bzw. kostenpflichtigen Bestandteil des Dokumentes erfolgen soll.

Metadaten-Retrieval. Die Bearbeitung von Anfragen, in denen von lokalen und externen LDS frei verfügbare Dokument-Metadaten angefordert werden, ist in Abbildung 2.4 dargestellt und erfolgt in den folgenden Schritten:

1. Empfang der Anfrage.
2. Analyse der Anfrage: Zerlegung der Anfrage in Teilanfragen, die durch die LDS bearbeitet werden sollen. Ermittlung der LDS aus der Anfrage, an die die Teilanfragen weitergeleitet werden müssen. Gegebenenfalls kann der Broker kontaktiert werden, der weitere externe LDS vorschlägt, die Dokumente enthalten, die die Anfrage erfüllen. Dazu verfügt der Broker über Informationen, die zur Selektion der erfolgversprechendsten Datenquellen erforderlich sind. Nachdem die zu kontaktierenden lokalen und externen LDS ermittelt sind, wird anhand der Informationen im Repository ermittelt, wie eine Verbindung zu den lokalen Wrappern aufgebaut werden kann.

[12] Ein Grund kann beispielsweise darin bestehen, daß anhand von Anfragen nicht der Inhalt (oder ein Teil) des Dokuments ableitbar sein soll.

[13] In den Selektionsbedingungen einer Anfrage werden die Dokumente ermittelt, die die Anfrage erfüllen.

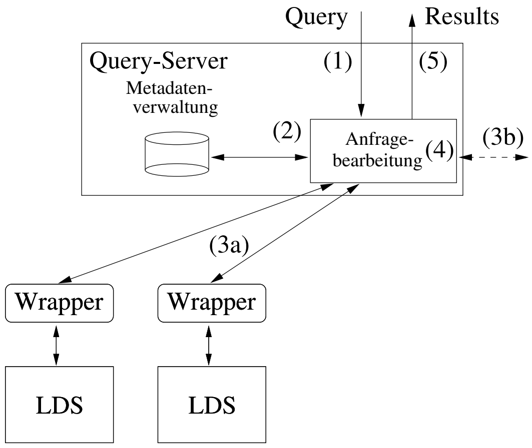

Abbildung 2.4: Metadaten-Retrieval-Anfragebearbeitung durch den Query-Server

3. Kontaktierung der LDS via Wrapper (a) oder Föderationsdienst (b): Die Wrapper nehmen die Umformung der Anfrage in die Anfragesprache der LDS vor, sammeln die Ergebnisse ein, bereiten sie auf und liefern sie an den Query-Server zurück. Externe LDS werden über den Föderationsdienst kontaktiert, der sicherstellt, daß die Anfrage und die Ergebnisse in ein vom Empfänger verständliches Format konvertiert werden.
4. Empfang und Nachbereitung der Ergebnisse: Wurde mehr als ein LDS zur Bearbeitung der Anfrage herangezogen, so ist eine Zusammenfassung und Aufbereitung der Ergebnisse erforderlich. Dazu zählen beispielsweise das globale Ranking und die Berücksichtigung der Nutzer-Präferenzen (z.B. maximale Anzahl der Ergebnistupel).
5. Rückgabe der Ergebnisse an den Aufrufer.

Eine Ergänzung ist zur Anfragebearbeitung nachzutragen. Zum einen können auch Anfragen an Replikate, die durch den Replikations-und Caching-Dienst in einem LDS gespeichert wurden, gestellt werden. Somit erstreckt sich die Suche auf alle Dokumente, die zu dem Zeitpunkt lokal gespeichert sind, auch wenn es sich nicht um Originale handelt.

Dokument-Retrieval. Die Vorgehensweise bei der Bearbeitung von Dokument-Retrieval-Anfragen ist in Abbildung 2.5 dargestellt. Bei der Anforderung eines Dokumentes kann die Notwendigkeit einer Autorisierung durch den Nutzer bestehen, wenn der Zugriff auf das Dokument durch Lizenzen eingeschränkt ist. Daraus ergibt sich die Forderung nach einem Dienst, der die Autorisierung vornimmt.

Wo dieser Dienst im VDS anzusiedeln ist, wurde bereits in der Projektbeschreibung festgelegt. Da der Replikations- und Caching-Dienst für die

Aktualisierung der lokal gehaltenen Replikate (Änderungen in der Lizenz, Ablauf der Lizenz, neue Version des Dokuments) zuständig ist und die Änderungen eng mit der Autorisierung verknüpft sind, soll die Anbindung direkt an diesem Dienst erfolgen.

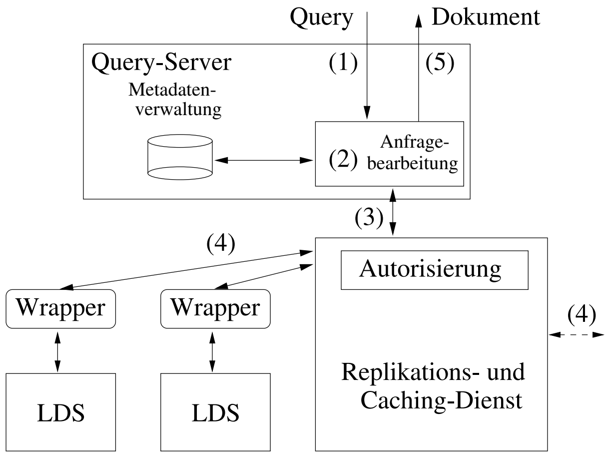

Abbildung 2.5: Retrieval eines Dokumentes durch den Query-Server

Das Retrieval des Dokumentes läuft in folgenden Schritten ab:

1. Empfang der Retrieval-Anfrage: Die Anfrage kann direkt durch einen lokalen Nutzer oder durch einen externen VDS ausgelöst worden sein.
2. Weiterleitung der Retrieval-Anfrage: Der Query-Server führt selbst keine Bearbeitung der Dokument-Retrieval-Anfragen durch, sondern leitet sie an den Replikations- und Caching-Dienst weiter.
3. Kontaktierung des Replikations- und Caching-Dienstes: Das Dokument wird beim Replikations- und Caching-Dienst angefordert. Die Retrieval-Anfrage enthält den DOI des angeforderten Dokumentes und gegebenenfalls erforderliche Autorisierungsinformationen. Der Replikations- und Caching-Dienst ermittelt, ob sich das Dokument in einem LDS befindet[14] oder von einem externen VDS angefordert werden muß. Dabei wird durch den Autorisierungsdienst überwacht, daß die Lizenzbedingungen beim Zugriff nicht verletzt werden.
4. Zugriff auf LDS bzw. entfernte VDS: Auslösen der Retrieval-Anfrage beim entsprechenden Dienst, sofern das Dokument nicht bereits vom Replikations-und Caching-Dienst lokal zwischengespeichert wurde und sofort verfügbar ist.
5. Rückgabe des Dokumentes: Unter der Voraussetzung, daß das Dokument gefunden wurde

[14] Siehe Diskussion über die Vergabe eindeutiger Dokument-Identitäten in Abschnitt 2.2.1.

- und frei verfügbar ist oder
- nicht frei verfügbar ist aber eine korrekte Autorisierung erfolgt,

kann das Dokument an den Nutzer ausgeliefert werden. Dokumente, die durch den Replikations- und Caching-Dienst von einem anderen VDS angefordert wurden, können unter Berücksichtigung der Lizenzbedingungen transient bzw. persistent zwischengespeichert werden, um den wiederholten Zugriff zu beschleunigen.

Lokalisierung von Dokumenten. Die Lokalisierung der angeforderten Dokumente erfolgt durch den Replikations- und Caching-Dienst. Sind die LDS, in denen die angeforderten Dokumente abgelegt sind, nicht aus dem DOI ableitbar, müssen alle lokalen LDS kontaktiert werden, wodurch bei sequentieller Vorgehensweise die Antwortzeit des Systems negativ beeinflußt wird und bei paralleler Vorgehensweise die Netzbelastung ansteigt. Eine Vermeidung dieses Problems ist möglich, wenn der Replikations-Dienst einen Index der von den LDS verwalteten Dokumente besitzt bzw. eine globale Instanz anhand einer Dokument-Identität den aktuellen Ort der Speicherung ermitteln kann.

Wenn in keinem der LDS das angeforderte Dokument abgelegt ist, müssen die externen VDS durch den Replikations- und Caching-Dienst kontaktiert werden, um eine Kopie des Dokuments zu erhalten.

Probleme bei der Dokument-Retrieval-Anfragebearbeitung. Die im vorherigen Abschnitt beschriebene Vorgehensweise bei der Dokument-Retrieval-Anfragebearbeitung bildet eine logische Folgerung aus den Projektvorgaben. In diesem Absatz werden einige Probleme aufgezeigt, die bei einer Verlagerung des Autorisierungsdienstes aus dem Replikations- und Caching-Dienst in einen eigenständigen Dienst vermieden werden könnten. Eine ausführliche Diskussion zu Fehlerfällen im VDS findet erst in Abschnitt 3.1.3 statt, nachdem die Architektur des VDS entworfen wurde.

- Der Replikations- und Caching-Dienst kann im Fehlerfall einen Engpaß im VDS bilden. Da alle lokalen und externen Retrieval-Anfragen durch einen Dienst bearbeitet werden, hat ein Ausfall dieses Dienstes zur Folge, daß ein VDS lokal keine Retrieval-Anfragen mehr bearbeiten kann.
  Ein möglicher Lösungsansatz besteht darin, den Query-Server bei Ausfall des lokalen Replikations- und Caching-Dienstes die Retrieval-Anfragen an einen anderen VDS weiterleiten zu lassen, dessen Replikations- und Caching-Dienst die Dokumente anfordert. Ein anderer Ansatz ergibt sich, wenn einem VDS mehrere Replikations- und Caching-Dienste zugeordnet werden. Bei Ausfall des primären Replikations- und Caching-Dienstes würde aus der Liste der verfügbaren Dienste der nächste selektiert werden.

Würde der Autorisierungsdienst aus dem Replikations- und Caching-Dienst ausgegliedert und der Zugriff auf Dokumente im LDS direkt über den Query-Server erfolgen, bestünde diese Problematik nicht, da bei Ausfall des Replikations- und Caching-Dienstes zumindest lokale Dokumente angefordert werden könnten, ohne auf externe VDS zugreifen zu müssen.

- Aus den Abbildungen 2.4 und 2.5, die den Ablauf der Anfragebearbeitung beschreiben, geht hervor, daß sowohl Query-Server als auch Replikations- und Caching-Dienst auf die LDS zugreifen müssen. Dazu ist es erforderlich, daß beide Informationen halten, die beschreiben, welche LDS verfügbar sind und wie diese angesprochen werden müssen. Da zur Laufzeit neue LDS in einem VDS angemeldet werden können, muß der Query-Server, der die Metadaten und Zugriffsinformationen der LDS verwaltet, den Replikations- und Caching-Dienst davon benachrichtigen.

  Diese zusätzliche Kommunikation zwischen Query-Server und Replikations- und Caching-Dienst wäre nicht erforderlich, wenn der Zugriff auf Dokumente direkt über den Query-Server erfolgen würde.

Eine alternative Dokument-Retrieval-Anfragebearbeitung. Die alternative Umsetzung der Dokument-Retrieval-Anfragebearbeitung, wie sie in Abbildung 2.6 dargestellt ist, führt zu einer Vermeidung dieser Probleme. Die

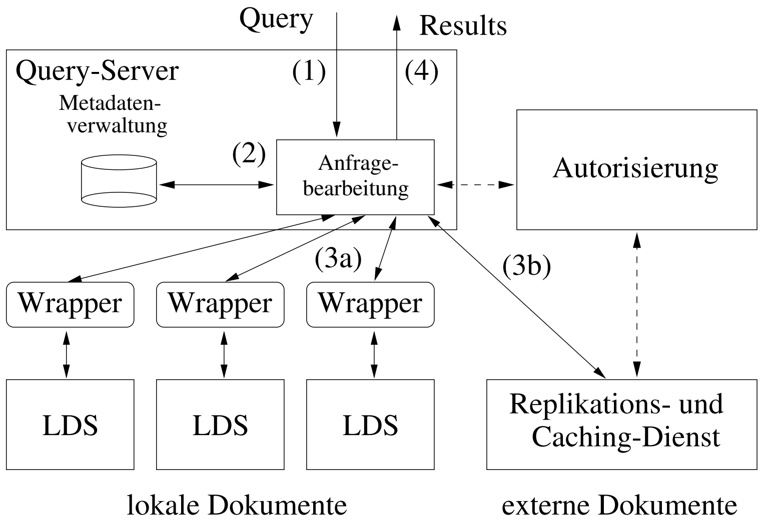

Abbildung 2.6: Veränderte Anfragebearbeitung bei Auslagerung des Autorisierungsdienstes

Auslagerung des Autorisierungsdienstes führt zu einer Entkopplung von LDS und Replikations- und Caching-Dienst. In diesem Ansatz werden lokale Dokumente vom Query-Server direkt von den LDS angefordert (3a), externe durch den Replikations- und Caching-Dienst[15] (3b). Beide Dienste greifen zur Autorisierung auf den eigenständigen Autorisierungsdienst zu.

Dieser Ansatz besitzt den Nachteil, daß das Caching der lokalen Dokumente durch den Wrapper oder durch den Query-Server erfolgen müßte. Da die dazu erforderliche Funktionalität bereits durch den Replikations- und Caching-Dienst bereitgestellt wird, sollte sie auch genutzt werden und nicht redundant vorliegen. Eine Auftrennung des Replikations- und Caching-Dienstes in zwei separate Dienste könnte dem entgegenwirken, erfordert jedoch auch eine Kommunikation zwischen Caching-Dienst und Autorisierungsdienst, da gecachte lizenzrechtlich geschützte Dokumente ohne eine erneute Prüfung der Gültigkeit einer Autorisierung nicht ausgeliefert werden dürfen.

Insgesamt betrachtet bietet dieser kurz vorgestellte Vorschlag zur Dokument-Retrieval-Anfragebearbeitung eine gute Alternative zur vorher ausführlich beschriebenen Vorgehensweise. Da aber die Projektvorgaben für die vorliegende Arbeit bindend sind, wird dieser Ansatz nicht weiter verfolgt.

Verwaltung der zugeordneten LDS. Bevor ein Virtueller Dokumenten-Server mit einem Lokalen Dokumenten-Server in Verbindung treten kann, muß sich der LDS beim VDS registrieren. Dazu werden die erforderlichen Informationen zum Zugriff auf den Wrapper, der die Datenquelle anbindet, und die Metainformationen der Datenquelle in einem Repository verwaltet.
Externe Dienste wie der Föderationsdienst können auf dieses Repository zugreifen, um die zur Erfüllung ihrer Aufgabe notwendigen Informationen abzufragen.

Zu klären bleibt, wie die Registrierung der LDS/Wrapper erfolgen soll und wie die externen Dienste über die An-/Abmeldung eines LDS informiert werden.

Metadaten-Verwaltung. Neben den Informationen zum Zugriff auf die Wrapper muß der Query-Server die Metadaten der LDS verwalten, um sie bei Bedarf den externen Diensten bereitzustellen.

Die zu verwaltenden Metadaten können in statische und dynamische Metadaten unterteilt werden. Diese Unterteilung wird eingeführt, um den unterschiedlichen Änderungsintervallen der Daten Rechnung zu tragen.

Statische Metadaten, wie z.B. Schemainformationen der LDS, werden durch den Query-Server persistent gespeichert, um Anfragen direkt beantworten zu können, ohne wiederholt mit dem LDS in Verbindung treten zu müssen. Diese Metadaten werden in der Regel selten (bis gar nicht) geändert und deshalb nur nach expliziter Aufforderung neu von den LDS angefordert.

[15]Dieser verfügt über einen eigenen LDS, in dem die Replikate von extern gespeicherten Dokumente verwaltet werden.

Dynamische Metadaten, zu denen beispielsweise IR-spezifische Statistiken (Worthäufigkeiten und -verteilungen u.ä.) zählen, werden anhand des Datenbestandes des LDS ermittelt. Sie ändern sich in Abhängigkeit von der Häufigkeit von Verwaltungsoperationen (Einfügen, Ersetzen, Löschen von Dokumenten) auf diesem Datenbestand. Auch diese Metadaten werden persistent von der Metadaten-Verwaltung gespeichert und entweder nach expliziter Aufforderung oder in periodischen Abständen verifiziert.

Durch die persistente Speicherung der Metadaten im Query-Server wird erreicht, daß nach dem Auftreten von kritischen Fehlern, die zum Verlust der nur transient vorliegenden Daten führen können, diese Metadaten nicht erneut von den LDS angefordert werden müssen.

Einen alternativen Ansatz zur aktiven Überprüfung des Metadatenbestandes durch den Query-Server bildet die Verwendung einer Alerting-Funktion, bei der die LDS den Query-Server selbständig über Änderungen informieren.

Kommunikation mit den LDS. Da die LDS durch Wrapper angebunden werden, steht dem Query-Server eine einheitliche Schnittstelle für die Kommunikation bereit. Die Kommunikation mit den Wrappern sollte sprachunabhängig stattfinden, da der Zugriff auf die LDS durch verschiedene Dienste (Query-Server, Replikations-Dienst) erfolgt, die den Anforderungen entsprechend in der jeweils geeignetesten Programmiersprache entwickelt werden.

Kommunikation mit dem Replikations- und Caching-Dienst. Die Kommunikation zwischen den beiden Diensten umfaßt Nachrichten zur Anforderung von Dokumenten und Nachrichten zur Autorisierung beim Zugriff aus diese Dokumente. Die Autorisierung des Nutzers wird durch den Replikations- und Caching-Dienst durchgeführt, da die Dokumente hier angefordert werden. Für die Realisierung der Kommunikation empfiehlt sich die Verwendung von protokoll- und sprachunabhängigen Ansätzen. Für die Protokollunabhängigkeit spricht eine im Vergleich zum protokollabhängigen Ansatz einfache Erweiterbarkeit und für die Sprachunabhängigkeit die Möglichkeit, daß auch in einer anderen Programmiersprache implementierte externe Dienste die Funktionalität des Replikations- und Caching-Dienstes nutzen können.[16]

Kommunikation mit den externen Diensten. Da zu diesem Zeitpunkt noch keine Informationen über die Kommunikation zwischen externen Diensten und dem VDS vorliegen, sollte eine sprachunabhängige Form gewählt werden um auf weitere Entwicklungen vorbereitet zu sein.

[16] Ein Grund für den Zugriff externer VDS wurde in der Diskussion zu “Problemen bei der Retrieval-Anfragebearbeitung“ auf Seite 21 gegeben.

2.3.2 Zusammenfassung

Die bisher erfolgte Beschreibung der Aufgaben des Query-Servers umfaßt die wesentlichen Punkte der Anfragebearbeitung und Metadaten-Verwaltung. Im Entwurf des Query-Servers sind jedoch weitere Betrachtungen zu Problemen wie An- und Abmeldung von LDS und die Propagierung von Änderungen im Repository erforderlich.

Vernachlässigt wurden vorläufig auch Anforderungen an den Query-Server, die sich aus der Zusammenarbeit mit weiteren externen Diensten ergeben. Beispielhaft sei der Benachrichtigungs-Dienst genannt, der in periodischen Abständen über Änderungen in Dokumenten bzw. am Datenbestand der LDS informiert werden muß.

2.4 Replikations- und Caching-Dienst

Der Zugriff auf Dokumente, die nicht im LDS, sondern in einem externen VDS gespeichert sind, erfordert, daß ein Dienst bereitgestellt wird, der diese Dokumente bei der externen Datenquelle anfordert und ausliefert. Diese Aufgaben werden vom Replikations-Dienst übernommen.

Die Replikation der externen Dokumente wird vorgenommen, um die Verfügbarkeit zu verbessern, die Zugriffszeit zu vermindern und eine Lastverteilung durchzuführen. Erreicht werden diese Ziele durch die lokale Zwischenspeicherung der Dokumente, die einen erneuten entfernten Zugriff verhindert. Handelt es sich um ein lizenzrechtlich geschütztes Dokument, ist eine zeitliche Begrenzung der Speicherung einzuführen, da das Replikat nach Ablauf der Lizenz wieder gelöscht werden muß.

Die Verwendung eines Replikates erfordert, daß Änderungen des Original-Dokumentes durch den Replikations-Dienst propagiert werden. Damit ist sichergestellt, daß dem Nutzer beim Zugriff auf das Replikat die jeweils aktuelle Version bereitgestellt wird. Auswirkungen auf die anderen Komponenten des VDS hat die Versionierung insofern, daß die Speicherung der Versionen im LDS erfolgen muß und dem Nutzer bei Bedarf mehrere Versionen eines Dokumentes präsentiert werden können. Dabei ist jedoch zu berücksichtigen, daß eine Lizenz an eine bestimmte Version gebunden sein kann.

Eine weitere Möglichkeit, den Zugriff auf externe Dokumente zu beschleunigen, bietet die Einführung eines Caching-Dienstes, der diese Dokumente für einen begrenzten Zeitraum lokal zwischenspeichert und in Abhängigkeit von der gewählten Ersetzungsstrategie wieder aus dem Cache entfernt. Bei der Anforderung eines Dokumentes ist deshalb ein Zwischenschritt erforderlich, bei dem ermittelt wird, ob sich das Dokument bereits im Cache befindet oder nicht. Sollte sich das Dokument nicht im Cache befinden, ist eine Anforderung des Dokumentes notwendig. Für Dokumente im Cache gelten natürlich die gleichen lizenzrechtlichen Bedingungen wie für Replikate, die durch den Replikations-Dienst verwaltet werden.

Die Verwaltung von Dokumenten, die Lizenzbedingungen unterworfen sind, erfordert, daß ein Autorisierungsdienst integriert wird, der die Einhaltung der Lizenzbedingungen bei der Speicherung, der Änderung und dem Zugriff überwacht.

2.4.1 Anforderungen

Da die Funktionalität des Replikations- und Caching-Dienstes Bestandteil von Untersuchungen sind, die in einer anderen Diplomarbeit vorgenommen werden, sei für vertiefende Informationen auf [Jon99] verwiesen. An dieser Stelle ist lediglich festzulegen, welche Schnittstellen vom Replikations- und Caching-Dienst für den Query-Server bereitgestellt werden müssen und welche Schnittstellen des LDS genutzt werden.

Auslieferung von externen Dokumenten. Der Query-Server fordert ein Dokument an, indem er die Dokument-Identität und, wenn erforderlich, die dazugehörige Autorisierungsinformation bereitstellt. Es ist Aufgabe des Replikations- und Caching-Dienstes, zu ermitteln, wo das Dokument gespeichert ist, und das Dokument beim LDS bzw. beim externen VDS anzufordern. Da die Dienste des VDS in einer heterogenen Umgebung laufen können, sollte die zugrunde liegende Kommunikation sprachunabhängig erfolgen.

Autorisierung lizenzpflichtiger und Abrechnung kostenpflichtiger Dokumente. Beim Zugriff auf lizenzpflichtige bzw. kostenpflichtige Dokumente ist die Autorisierung und Abrechnung durchzuführen sowie die Einhaltung der Lizenzbedingungen (z.B. Gruppenlizenz, Einzellizenz) zu überprüfen. Der Query-Server leitet die Autorisierungsinformationen vom Nutzer zum Replikations- und Caching-Dienst, greift aber nicht aktiv in die Autorisierung ein.

Zugriff auf Dokumente der LDS. Der Zugriff auf die Dokumente erfolgt via Wrapper, wobei dem Wrapper des kontaktierten LDS lediglich der DOI übergeben werden muß, da die Autorisierung vom Autorisierungsdienst des Replikations- und Caching-Dienstes durchgeführt wird.

2.4.2 Zusammenfassung

Der Replikations- und Caching-Dienst stellt dem Anfrage- und Retrieval-Dienst bei Dokument-Retrieval-Anfragen alle angeforderten Dokumente bereit, die lokal oder auf entfernten VDS gespeichert sind. Beim Zugriff auf Dokumente bzw. Dokument-Metadaten, die in externen VDS abgelegt sind, ist jedoch nicht gewährleistet, daß diese ihre Daten im gleichen Format zurückliefern, wie es von den Diensten des lokalen VDS erwartet wird. Deshalb wird der Föderationsdienst eingeführt, der für die Transformation der Daten zuständig ist.

2.5 Föderationsdienst

Die LDS der VDS bilden eine Föderation von autonomen Datenbanksystemen, die in Bezug auf das Datenmodell, das Schema, den DB-Inhalt, die Anfragesprache uvm. heterogen sein können. Um die Heterogenität überwinden zu können, werden Transformationen durchgeführt, deren Ergebnis ein einheitliches Datenbankmodell und Schema bildet [Rah94, Kapitel 12]. Damit ist der Zugriff auf die Daten anderer VDS möglich.

Doch nicht nur die Überwindung der Heterogenität der Datenbanksysteme wird durch den Föderationsdienst gewährleistet. Da Informationen über ein Dokument in unterschiedlichen Datenbanken gespeichert sein können, ist eine Verknüpfung dieser Informationen möglich. Durch die Erkennung und Zusammenfassung von Dokumenten, deren Inhalt in einer Beziehung steht[17] kann der Föderationsdienst zu einem Dokument mehr Informationen liefern als eine einzelne Datenbank, sofern die in den Datenbanken gespeicherten Informationen nicht nur redundant sondern auch ergänzend sind.[18]

Um die erforderlichen Transformationen durchführen zu können, verwaltet der Föderationsdienst die statischen und dynamischen Metadaten der Föderation in einem Datenverwaltungssystem. Diese Metadaten werden vom Query-Server des VDS bereitgestellt, der eine Schnittstelle für den Föderationsdienst anbietet.

Der Föderationsdienst wird durch den Projektpartner in Magdeburg konzipiert und zur Verfügung gestellt.

2.5.1 Anforderungen

Die BlueRose-Dienste nutzen den Föderationsdienst, um Metainformationen und Dokumente von VDS anzufordern, ohne sich um die Heterogenität der Systeme kümmern zu müssen.

Homogenisierung von Daten und Anfragen. Die Homogenisierung erfolgt anhand von Metainformationen über die Informationsquellen. Diese Informationen umfassen z.B. LDS-Schemabeschreibungen und Beschreibungen der VDS-Anfragesprachen. Für den homogenen Zugriff auf die LDS innerhalb eines VDS durch den Query-Server oder den Replikations- und Caching-Dienst sind die Wrapper zuständig, die die notwendigen Transformationen durchführen.

[17] Beispielsweise könnten Dokumente Informationen über die Autoren und andere Dokumente wiederum Informationen über Verlage und deren unter Vertrag stehende Autoren enthalten.

[18] In dem vorherigen Beispiel könnten demnach neben den Informationen zum Autor auch Informationen über den Verlag mitgeliefert werden.

2.5.2 Zusammenfassung

Der Query-Server greift auf den Föderationsdienst zu, um Anfragen an andere VDS bzw. einen Broker stellen zu können und wird vom Föderationsdienst angesprochen, wenn andere VDS seine Dienste in Anspruch nehmen. Der Replikations- und Caching-Dienst nutzt den Föderationsdienst, um externe Dokumente und ihre Metadaten von anderen VDS anzufordern.

2.6 Anbindung von Tradern/Brokern

In einer digitalen Bibliothek gibt es eine Vielzahl von Virtuellen Dokumenten-Servern und anderen Diensten. Deshalb ist es erforderlich, diese zu registrieren und bei Bedarf an einen Dienstleistungsnehmer zu vermitteln. Diese Aufgabe wird durch eine Trader/Broker-Komponente erledigt. Die Projektpartner in Hamburg entwickeln dafür ein allgemeines Trader/Broker-Framework, aus dem fachspezifische Trader und Broker konstruiert werden können.

Da zum Zeitpunkt der Bearbeitung dieser Diplomarbeit keine vertiefenden Informationen über diese Komponente vorliegen, beschränken sich die Untersuchungen zu den Anforderungen auf allgemeine Trader und Broker. Zum besseren Verständnis werden eingangs deren Aufgaben beschrieben, bevor die daraus resultierende Spezifikation der Anforderungen erfolgt.

2.6.1 Trader

In einem verteilten System kann es eine Vielzahl von Diensten geben, die von beliebigen Teilkomponenten des Systems in Anspruch genommen werden. Die Vermittlung dieser Dienste an einen potentiellen Dienstnehmer erfolgt, wie in Abbildung 2.7 dargestellt, durch den Trader anhand der vom Client geforderten Dienst-Eigenschaften.

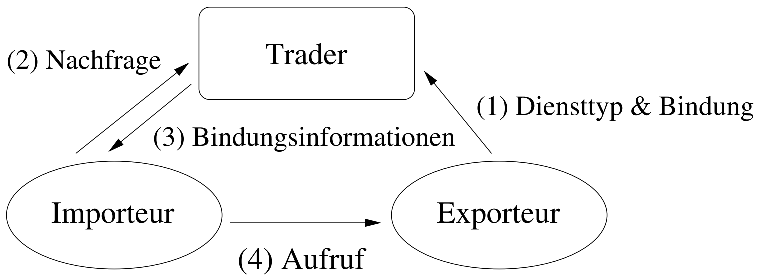

Abbildung 2.7: Ablauf des Tradings [Gri98]

In einem ersten Schritt erfolgt die Registrierung des Dienstes beim Trader (1), wobei der bereitgestellte Diensttyp (z.B. Druckdienst) und der Zugriff darauf (Bindungsinformation) spezifiziert werden muß. Ein Diensttyp wird durch einen unveränderlichen Namen, eine Menge von Schnittstellen und (statischen und dynamischen) Attributen dargestellt, wobei Attribute die Eigenschaften des Dienstes beschreiben. Anhand der statischen Attribute können Fähigkeiten und anhand der dynamischen Attribute der Zustand des Dienstes ermittelt werden. Das folgende CORBA-IDL-Listing beschreibt einen Signatur-Dienst:

```
typedef sequence <octet> document;
interface Signature
 {
   readonly attribute string algorithm;
   boolean sign(in document doc, out string signature);
   boolean verify(in document doc, in string signature);
 };
```

Der beschriebene Dienst *Signature* ermittelt zu einem Dokument doc die Signatur (*sign*) bzw. verifiziert die Signatur des Dokumentes (*verify*). Das Attribut *algorithm* beschreibt den dabei verwendeten Algorithmus.

Will ein Client einen Dienst in Anspruch nehmen, stellt er an den Trader eine Anfrage (2), in der der erforderliche Diensttyp, die Schnittstellen und relevante Attribute beschrieben sind. Im obigen Beispiel könnte der Client beispielsweise fordern, daß der angeforderte Dienst die Signatur anhand des MD5-Algorithmus bestimmen soll. Der Trader ermittelt unter Zuhilfenahme der Anforderungen die in Frage kommenden registrierten Dienste und liefert die dazugehörigen Bindungen an den Client zurück (3). Der Client kann dabei z.B. Einfluß auf die Anzahl und die Reihenfolge der Ergebnisse nehmen, indem er die vom Trader verwendete Strategie (Policy) verändert.

Im letzten Schritt bindet sich der Client an den Dienstanbieter und kann die gewünschte Methode aufrufen (4). Die dazu notwendige Kommunikation findet direkt zwischen dem Client und dem Dienstanbieter statt, ohne die Dienste des Traders weiter in Anspruch zu nehmen.

Der Vorteil des Tradings ist, daß zur Laufzeit Komponenten ausgewechselt bzw. rekonfiguriert werden können, da die Bindungsinformationen des Dienstes bei jeder Vermittlung neu an den Client übertragen werden. Somit ist eine dynamische Anpassung der Komponenten an die Anforderungen des Systems möglich.

Der beschriebene Trading-Ansatz ist von einfacher Natur und weist einige Nachteile auf (u.a. keine Skalierbarkeit). [Vsu98] listet diese Nachteile auf und zeigt aktuelle Trading-Ansätze.

2.6.2 Broker

In einer digitalen Bibliothek wird eine riesige Anzahl von Dokumenten gespeichert. Da diese Dokumente in einer Vielzahl von Datenquellen gehalten werden, kann es aus Performance-Gründen sinnvoll sein, die Suche auf einige vielversprechende Datenquellen zu beschränken. Diese Aufgabe wird vom Broker übernommen, der, auf Informationen über den Datenbestand aufbauend, die Selektion der Datenquellen durchführt, was in Abbildung 2.8 durch die gestrichelte Linie dargestellt wird. In der Architektur von BlueRose wird der Broker durch den

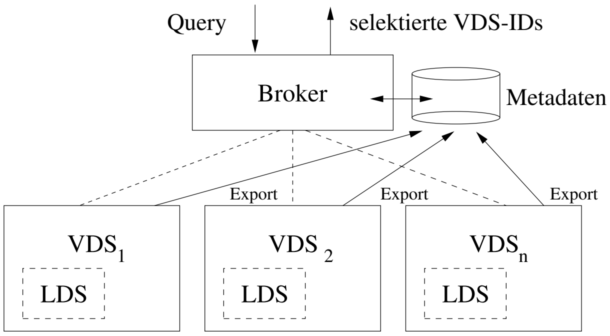

Abbildung 2.8: Selektion der Datenquelle durch den Broker

Föderationsdienst unterstützt, die die Heterogenität der Datenquellen verbirgt.

Für den Export der Metadaten der Lokalen Dokumenten-Server ist der Föderationsdienst zuständig, der entweder via Query-Server oder direkt auf den LDS zugreift.[19] Zu den vom VDS bereitzustellenden Metadaten gehören die Schemadefinition und statistische Daten über den Datenbestand. Welche statistischen Daten dies im Einzelnen sind, wird durch Dienste festgelegt, die sich mit dem verteilten Information Retrieval beschäftigen.

2.6.3 Anforderungen

Aus den vorherigen Abschnitten lassen sich folgende allgemeine Anforderungen an den VDS stellen:

Export der bereitgestellten Dienste (Trader). Damit ein Trader die vom VDS bereitgestellten Dienste vermitteln kann, muß diesen Diensten ein eindeutiger Typ zugeordnet und dieser an den Trader exportiert werden. Da die Traderkomponente zur Zeit noch nicht vorliegt, kann die Umsetzung des Exportes nur sehr eingeschränkt erfolgen.

Bereitstellung der Bindungsinformation(Trader). Die Anbindung der bereitgestellten Dienste an den Dienstnehmer, der z.B. der Broker sein kann, wird durch die Bindungsinformation beschrieben. Da zu diesem Zeitpunkt auch keine Aussage über die im Verlauf des Projektes gefällten Entscheidungen bezüglich der Kommunikation der externen Dienste untereinander getroffen werden kann, sollte bei der Wahl der Anbindung darauf geachtet werden, daß ein Zugriff durch die externen Dienste erfolgen kann.

[19]In Abschnitt 2.2 wurde bereits dargelegt, daß der Zugriff auf ein autonomes System, das ein LDS darstellt, nicht durch den VDS verhindert werden kann.

Export von Metadaten (Broker). Die von den externen Diensten zur Erfüllung ihrer Aufgaben erforderlichen Metadaten sind an den Föderationsdienst zu exportieren. Da der LDS nicht gekapselt ist, kann dieser Export via Query-Server oder direkt durch den Föderationsdienst initiiert werden.

2.6.4 Zusammenfassung

Die hier grob beschriebenen Anforderungen lassen sich zu diesem Zeitpunkt nicht näher spezifizieren, da die zu den beiden Komponenten verfügbaren Informationen sich auf Projektanträge beschränken.

2.7 Weitere externe Dienste

Neben den in dieser Arbeit vorgestellten Diensten gibt es weitere externe Dienste wie die GUI[20] und den Profil- und Benachrichtigungsdienst, die die Funktionalität des VDS verwenden, aber keine speziellen Anforderungen stellen. Beispielhaft sei der Profil- und Benachrichtigungsdienst vorgestellt.

Der Profil- und Benachrichtigungsdienst informiert den Nutzer über neu verfügbare Informationen (Dokumente) bzw. Datenquellen, die seinem Interessenprofil entsprechen. Dabei werden Informationsanbieter in kooperative und nichtkooperative Anbieter unterteilt. Erstere stellen von sich aus die zur Erkennung und Benachrichtigung erforderliche Funktionalität bereit, während letztere lediglich eine Schnittstelle zum Zugriff auf die Dokumente und ihre Metadaten anbieten.

Unter der Voraussetzung, daß ein VDS als ein Informationsanbieter angesehen wird, der selbst keine Informationen verwaltet, sondern lediglich den Zugriff auf die Informationen bereitstellt, kann dieser als kooperativer oder nichtkooperativer Anbieter auftreten. Wenn ein VDS als kooperativer Anbieter agieren soll, ist in einen der VDS-Dienste die entsprechende Funktionalität zu integrieren. Diese Aufgabe könnte vom Query-Server oder vom Replikations- und Caching-Dienst übernommen werden. Es bietet sich an, dafür den letzteren zu wählen, da er über Änderungen des replizierten Datenbestandes informiert ist und da er durch die lokalen LDS (bspw. unter Nutzung eines lokalen Alerting-Services) über Änderungen am Datenbestand informiert werden kann. Als nicht-kooperativer Anbieter sind an den VDS keine weiteren Anforderungen an die Funktionalität zu stellen, da er bereits ein Query-Interface bereitstellt und den Zugriff auf die Dokumente realisiert.

[20] Grafische Benutzungsoberfläche

2.8 Zusammenfassung

Bei den Untersuchungen zu den einzelnen Diensten und den resultierenden Anforderungen an die Architektur des Virtuellen Dokumenten-Servers haben sich drei Schwerpunkte hervorgetan. Zwei dieser Schwerpunkte werden im folgenden Kapitel bearbeitet, während der dritte in den Aufgabenbereich einer anderen Arbeit fällt.

1. Die Anfragebearbeitung durch den Query-Server bildet Schwerpunkt eins. Dazu gehören die Analyse und Verteilung der Anfragen auf die LDS, den Replikations- und Caching-Dienst und externe VDS, die Zusammenfassung und Nachbereitung der Ergebnisse (globales Ranking), die Verwaltung der Metadaten und Zugriffsinformationen im Query-Server aber auch die Bereitstellung der Metadaten der LDS, die von externen Diensten angefordert werden können. Im nächsten Kapitel wird dieser Schwerpunkt behandelt und die daraus resultierenden Schnittstellen abgeleitet.
2. Schwerpunkt zwei bilden LDS und Wrapper, die den Zugriff auf die lokalen Dokumente gestatten. Zu den zu realisierenden Anforderungen gehört der Zugriff auf die LDS und die damit verbundenen Aufgaben wie die Abbildung der Anfrage in die Anfragesprache der LDS und die Ergebnisaufbereitung. Auch dieser Schwerpunkt wird durch die folgenden Kapitel abgedeckt. Einige Grundlagen für die Verwendung eines objektrelationales Datenbanksystems als LDS werden durch die Erweiterung der Anfragesprache um IR-Konstrukte, die strukturierte Anfragen auf semi-strukturierten Dokumenten realisieren, in [Por99] geschaffen.
3. Der Zugriff auf die Dokumente, die damit verbundene Autorisierung und die Speicherung durch den Replikations- und Caching-Dienst bildet Schwerpunkt drei. Dieser Schwerpunkt wird im Rahmen einer anderen Diplomarbeit untersucht [Jon99]. Für die vorliegende Arbeit ist von Bedeutung, welche Schnittstellen der Replikations- und Caching-Dienst dem Query-Server bereitstellt, um Dokumente anfordern zu können.

Der Aufbau des folgenden Kapitels, in dem die Architektur und Dienstschnittstellen beschrieben sind, gliedert sich entsprechend der bereits beschriebenen Schwerpunkte. Da ein Aspekt aus Schwerpunkt zwei, die Abbildung der IRQL-Anfrage auf die Anfragesprache des LDS, von der Anfragesprache der verwendeten Datenquelle (Datenbanksystem, Suchdienst) abhängt, werden die dazu notwendigen Betrachtungen in einem separaten Kapitel (Kapitel 4) vorgenommen.

Kapitel 3

Architekturbeschreibung des VDS

Die Architektur des Virtuellen Dokumenten-Servers setzt sich aus den bereits beschriebenen Diensten Query-Server, LDS sowie Replikations- und Caching-Dienst zusammen. Aus der Beschreibung der Anforderungen haben sich bereits Kriterien für den Architekturaufbau des VDS ergeben, die in diesem Kapitel aufgegriffen werden und zu einer Beschreibung der Gesamtarchitektur führen.

Die Bearbeitung von Anfragen, die durch externe Dienste bzw. andere VDS gestellt werden, erfolgt durch die Zusammenarbeit der lokalen Dienste des VDS. Daraus erhebt sich die Forderung nach Schnittstellen, die diese Dienste bereitstellen müssen, damit andere Dienste deren Funktionalität nutzen können.

Die Architekturbeschreibung wird im Abschnitt 3.1 hergeleitet, wobei dargestellt wird, welche Dienste zur Lösung einer Aufgabe miteinander kommunizieren. Eine kurze Betrachtung zum Verhalten des VDS in Fehlerfällen in Abschnitt 3.1.3 zeigt Probleme auf und bietet einen ersten Ausblick auf Erweiterungsmöglichkeiten des Systems. Aufbauend auf der Architekturbeschreibung werden in Abschnitt 3.2 die Dienstschnittstellen für die einzelnen Dienste festgelegt und beschrieben. Dabei orientiert sich der weitere Aufbau dieses Kapitels an den im vorherigen Kapitel identifizierten Schwerpunkten: Query-Server, Wrapper/LDS sowie Replikations- und Caching-Dienst. Begonnen wird in Abschnitt 3.2.1 mit der Beschreibung der Schnittstellen des Query-Servers, die zur Anfragebearbeitung (Analyse, Vorbearbeitung, Nachbearbeitung), zum Zugriff auf die lokalen und externen Dienste und zur Verwaltung der Metainformationen erforderlich sind. Im Anschluß daran werden in Abschnitt 3.2.2 die Dienstschnittstellen des LDS und in Abschnitt 3.2.3 die Schnittstellen des Replikations- und Caching-Dienstes spezifiziert.

Die formale Definition der Dienstschnittstellen wurde aus Gründen der Übersichtlichkeit in den Anhang A ausgelagert.

3.1 Architekturentwurf

Der Virtuelle Dokumenten-Server bildet ein verteiltes System aus Client- und Server-Anwendungen, die in ihrem Zusammenwirken die Funktionalität des VDS bereitstellen. Ein VDS setzt sich aus einem Query-Server, einem Replikations- und Caching-Dienst sowie mehreren LDS zusammen. Die Anzahl der Lokalen Dokumenten-Server eines VDS kann im laufenden Betrieb variieren, wenn eine neue Datenquelle angeschlossen wird bzw. wenn eine Datenquelle kurzfristig nicht verfügbar ist.

Dieser Abschnitt gliedert sich in die Herleitung der Architektur (3.1.1), Betrachtungen zur Kommunikation der VDS-Dienste mit externen Diensten (3.1.2) und Betrachtungen zum Verhalten des VDS in Fehlerfällen (3.1.3).

3.1.1 Herleitung der Architektur

Die Architektur des VDS setzt sich aus den lokalen Diensten zusammen und beschreibt die zur Bearbeitung notwendige Kommunikation der Dienste untereinander sowie die Kommunikation mit externen Diensten. Da die wesentliche Aufgabe des VDS die Anfragebearbeitung darstellt, bietet sich hier ein guter Ansatzpunkt für die Herleitung der Architektur.

In den Anforderungen an den Query-Server wurden zwei Formen von Anfragen identifiziert: das Metadaten-Retrieval und das Dokument-Retrieval. In

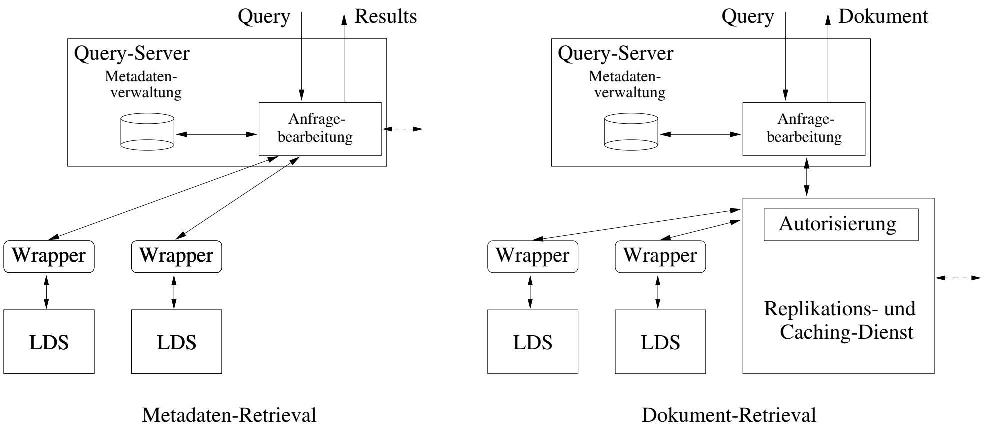

Abbildung 3.1: Anfragebearbeitung durch den Query-Server

Abbildung 3.1 ist die zur Bearbeitung der Anfragen notwendige Kommunikation zwischen den Diensten nochmals dargestellt. Metadaten-Retrieval-Anfragen (Abbildung links) werden durch direkte Kontaktierung der LDS bzw. die Kontaktierung externer Dienste[1] und Dokument-Retrieval-Anfragen durch Weiterleitung

[1] Ermittlung vielversprechender externer VDS durch den Broker und anschließende Kontaktierung dieser VDS.

der Anfrage an den Replikations- und Caching-Dienst (Abbildung rechts) bearbeitet.[2] Aus den Abbildungen läßt sich bereits das Zusammenwirken der einzelnen Dienste zur Erfüllung der jeweils erforderlichen Aufgaben entnehmen.

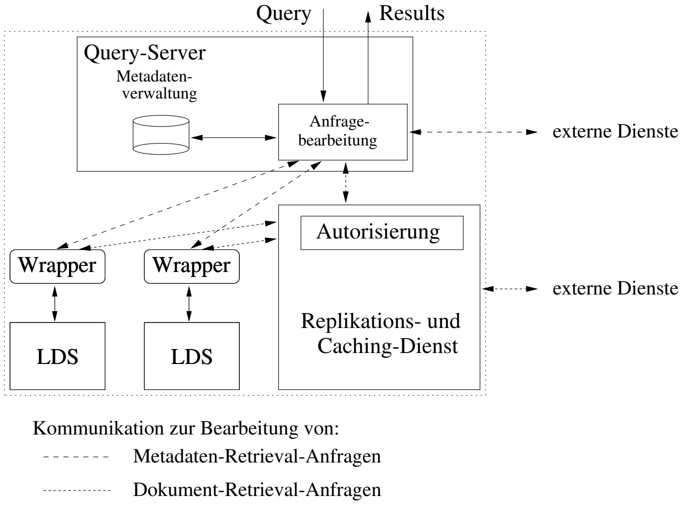

Abbildung 3.2: Gesamtarchitektur des VDS

Eine Zusammenführung beider Bearbeitungsweisen führt zu der in Abbildung 3.2 dargestellten Architektur, wobei wiederum die Kommunikation der einzelnen Dienste untereinander durch Pfeile verdeutlicht wird. Damit sind bereits die wesentlichen Punkte der Architektur des Virtuellen Dokumenten-Servers beschrieben. Der Query-Server und der LDS werden in Abschnitt 3.2 und der Replikations- und Caching-Dienst in [Jon99] genauer beschrieben.

Aus Abbildung 3.2 ist nicht ersichtlich, welche externen Dienste auf den Query-Server und auf den Replikations- und Caching-Dienst zugreifen können bzw. welche externen Dienste durch diese beiden Dienste genutzt werden. Dazu werden im Folgenden einige Betrachtungen angestellt.

3.1.2 Kommunikation mit externen Diensten

In den allgemeinen Anforderungen aus Kapitel 2 sind für den Query-Server bereits einige Aussagen getroffen worden. Im Wesentlichen handelt es sich bei den zu kontaktierenden Systemen um den Föderationsdienst und den Broker, da beide Dienste Funktionalität bereitstellen, die vom VDS benötigt wird. Demzufolge sind Schnittstellen für den Zugriff auf diese Dienste zu schaffen. Die anderen externen Dienste, wie der Profil- und Benachrichtigungsdienst[3], können via An-

[2] Eine ausführliche Beschreibung der allgemeinen Vorgehensweise bei der Bearbeitung von Anfragen wurde bereits in Abschnitt 2.3.1 gegeben.

[3] In der Annahme, daß der VDS kein kooperativer Anbieter ist, erfolgt der Zugriff über die Anfrageschnittstelle. Andernfalls erfolgt eine direkte Kommunikation zwischen Replikations- und Caching-Dienst sowie Profil- und Benachrichtigungsdienst.

frageschnittstelle auf den Query-Server zugreifen, so daß diese vorläufig nicht zu berücksichtigen sind.

Es ist in der vorliegenden Arbeit nur sehr eingeschränkt möglich, Aussagen bezüglich der Kommunikation zwischen Replikations- und Caching-Dienst sowie externen Diensten zu treffen, da die Aufgaben des Dienstes nicht auf die Anforderung von Dokumenten beschränkt sind, sondern zusätzliche Verwaltungsaufgaben (Lizensierung usw.) anfallen, die eine Kommunikation mit weiteren externen Diensten, die in dieser Arbeit nicht betrachtet wurden, erfordern. Aus diesem Grund reduziert sich die Diskussion auf den für diese Arbeit wichtigen Aspekt, die Dokument-Retrieval-Anfragen. Für weitergehende Informationen zum Replikations- und Caching-Dienst und dessen Schnittstellen sei erneut auf [Jon99] verwiesen.

Die lokalen und externen Dokumente werden durch den Replikations- und Caching-Dienst bereitgestellt. Somit ist es seine Aufgabe zu ermitteln, in welchem VDS (im lokalen oder in einem externen?) das Dokument gespeichert ist[4] und den entsprechenden VDS zu kontaktieren. Der Zugriff auf ein Dokument, das in einem entfernten VDS gespeichert ist, kann wie in Abbildung 3.3 dargestellt auf zwei Wegen erfolgen. Zum einen kann die Retrieval-Anfrage an den Query-Server (1), zum anderen an den Replikations- und Caching-Dienst (2) des entsprechenden VDS gesendet werden. Diese zwei Ansätze sind eine logische Folgerung aus der

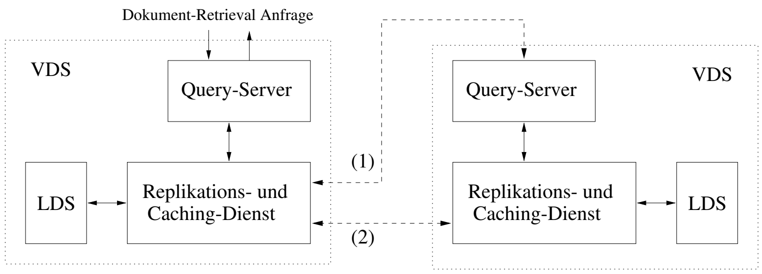

Abbildung 3.3: Zugriff auf ein externes Dokument

Forderung, daß alle Zugriffe auf lokale Dokumente über den Replikations- und Caching-Dienst erfolgen.

Der erste Ansatz hat den Vorteil, daß der Dokument-Retrieval-Zugriff auf den Replikations- und Caching-Dienst nur durch lokale Dienste erfolgt und somit Änderungen der Dienstschnittstellen vor externen Diensten verborgen bleiben. Als Nachteil ist aufzuführen, daß der Query-Server bei starker Belastung zum Flaschenhals werden kann.[5] Dieser Ansatz bildet im Vergleich zum zweiten Ansatz

[4] Wie dies geschieht, wird in dieser Arbeit nicht betrachtet, da diese Aufgabe in den Bereich einer anderen Diplomarbeit [Jon99] fällt.

[5] An dieser Stelle kann keine pauschale Entscheidung über die Wahrscheinlichkeit, daß ein Query-Server zum Hemmnis für das System wird, getroffen werden. Deshalb sind weitere Untersuchungen zum Lastverhalten von Query-Servern anhand eines Prototypen vorzunehmen.

ein geschlosseneres System, da Anfragen an den VDS nur über den Query-Server entgegengenommen werden können.

Wird, wie in Ansatz zwei beschrieben, ein VDS als ein offenes System unabhängiger Dienste betrachtet, bei dem durch externe VDS auch auf den Replikations- und Caching-Dienst zugegriffen werden kann, so wird der Nachteil aus Ansatz eins vermieden. Wenn davon ausgegangen wird, daß alle Replikations- und Caching-Dienste homogene Schnittstellen anbieten, wirken sich Änderungen an der Schnittstelle auf alle Replikations- und Caching-Dienste des VDS-Verbundes aus. Somit wird der Vorteil von Ansatz eins bedeutungslos. Da beim direkten Zugriff auf den Replikations- und Caching-Dienst der Umweg über den Query-Server eingespart wird, muß das Antwortzeitverhalten besser als im beschriebenen ersten Ansatz sein. Somit erscheint der direkte Zugriff die bessere Wahl zu sein.

Neben dem Dokument-Retrieval-Zugriff gibt es eine weitere Form von Zugriffen auf die Dokumente, die sich aus der Verwaltung der Replikate durch den Replikations- und Caching-Dienst ergibt und in einer Kommunikation mit einem externen Dienst (einem anderen Replikations- und Caching-Dienst) resultiert. Die Replikations-Dienste kommunizieren direkt miteinander, um sich über Änderungen der Dokumente usw. zu informieren, und sind nicht von der Entscheidung für einen der beiden vorgestellten Ansätze betroffen.

3.1.3 Betrachtungen zu Fehlerfällen

In den bisherigen Beschreibungen der Architektur wurden keine Aussagen zum Verhalten des Systems in Fehlerfällen getroffen. Für die vorliegende Arbeit sind folgende Fehlerfälle interessant:

1. der Query-Server ist nicht verfügbar
2. der Replikations- und Caching-Dienst ist nicht verfügbar
3. ein LDS ist nicht verfügbar
4. ein externer Dienst ist nicht verfügbar

Jeder der Fehlerfälle hat mehr oder minder schwere Auswirkungen auf die Verfügbarkeit des VDS bzw. auf die Bearbeitung von Anfragen. In den folgenden Betrachtungen wird nach Metadaten-Retrieval-Anfragen und nach Dokument-Retrieval-Anfragen unterschieden.

Fehlerfall 1: Ausfall des Query-Servers. Beim Ausfall des lokalen Query-Servers können keine Metadaten-Retrieval-Anfragen an den Dokumentenbestand des VDS gestellt werden. Ein Lösungsansatz besteht darin, die LDS einem anderen VDS zuzuordnen. Der Aufwand bei dieser Vorgehensweise ist jedoch beträchtlich, da die LDS beim neuen VDS angemeldet werden müssen und Informationen über die Neuzuordnung den davon betroffenen externen Diensten zu übermitteln sind. Aus diesem Grund scheint es in diesem Fehlerfall sinnvoller zu sein, keine Metadaten-Retrieval-Anfragen auf diesen Datenbestand zuzulassen, solange der Query-Server nicht verfügbar ist.

Dokument-Retrieval-Anfragen sind vom Ausfall des Query-Servers nicht betroffen, da der Zugriff auf lokal vorhandene Dokumente über einen anderen VDS erfolgen kann. Die Vorgehensweise entspricht dabei dem Retrieval eines externen Dokumentes.

Fehlerfall 2: Ausfall des Replikations- und Caching-Dienstes. Der Ausfall dieses Dienstes hat keinen Einfluß auf die Bearbeitung von Metadaten-Retrieval-Anfragen, da diese durch den Query-Server und die LDS erfolgt.

Ein Zugriff auf die Dokumente kann nicht erfolgen, ohne daß ein Ersatzdienst die Aufgabe des ausgefallenen Dienstes übernimmt.[6] Dieser Lösungsansatz kann beispielsweise realisiert werden, indem dem Query-Server eine Liste von Replikations- und Caching-Diensten zugeordnet wird, die im Fehlerfall aktiviert werden können, bzw. indem ein neuer Dienst durch den Trader vermittelt wird. Inwieweit eine solche Vorgehensweise praktikabel ist, hängt auch davon ab, wie komplex der Replikations- und Caching-Dienst ist und wie hoch seine Anforderungen an Ressourcen sind.[7]

Fehlerfall 3: Ausfall eines LDS. Bei Ausfall eines LDS können keine Metadaten-Retrieval-Anfragen an den Datenbestand dieses LDS gestellt werden. Eine praktikable Lösung des Problems, die ohne Replikation des Datenbestandes auskommt, gibt es nicht. Eine Ausnahme gilt für lokale Dokumente, die auf externen LDS repliziert vorliegen. Da für diese Dokumente die Metadaten auch im externen LDS gespeichert sind, könnten die Anfragen auf die Metadaten der Kopien angewendet werden. Dies ist jedoch ein sehr aufwendiger Prozeß (Ermittlung ob und wo Replikate vorliegen, Weiterleitung der Anfrage an die entsprechenden VDS und Einschränkung der Anfrage auf diese replizierten Dokumente) und erscheint nur sinnvoll, wenn eine hohe Verfügbarkeit erforderlich ist.

Dokument-Retrieval-Anfragen können nur bearbeitet werden, wenn das Dokument bereits vorher aufgrund einer internen oder externen Anfrage vom Replikations- und Caching-Dienst angefordert wurde und daher im Cache vorliegt oder wenn ein Replikat in einem anderen VDS existiert.

Fehlerfall 4: Ausfall eines externen Dienstes. Da die Auswirkungen beim Ausfall eines externen Dienstes derart vielfältig sein können, kann an dieser Stelle keine vollständige Betrachtung durchgeführt werden. Sämtliche Aus-

[6] Ein direkter Zugriff auf die LDS zur Anforderung eines Dokumentes darf nicht erfolgen, da die gegebenenfalls erforderliche Autorisierung nicht durchgeführt werden kann.

[7] Verwendet der Dienst z.B. Datenbanken zur Speicherung von Dokumenten?

sagen gelten für den Fall, daß für den ausgefallenen Dienst kein Ersatzdienst bereitsteht, der von einem Trader vermittelt werden kann.

Beim Ausfall des Föderationsdienstes kann auf keine Daten externer VDS zugegriffen werden, da keine Homogenisierung der Anfragen und Ergebnisse stattfindet.

Der Ausfall des Brokers führt dazu, daß bei Metadaten-Retrieval-Anfragen keine Selektion von vielversprechenden VDS erfolgen kann, an die Anfragen weitergeleitet werden können. Eine Weiterleitung von Anfragen an andere VDS ist möglich, jedoch kann eine qualitative Bewertung der VDS in Bezug auf Anfragen nicht erfolgen.

Umfangreiche Auswirkungen kann der Ausfall eines Lizensierungsdienstes haben, da eine Verifikation der Autorisierung nicht erfolgen kann. Es hängt von der Strategie der Autorisierungsdienste im Replikations- und Caching-Dienst ab, ob und wie der Zugriff auf ein Dokument erfolgt. Betrachtungen zu dieser Problematik werden in [Jon99] vorgenommen.

Für die hier beschriebenen Fehlerfälle kann es Lösungen geben, die im günstigsten Fall vom System automatisch, wesentlich häufiger aber manuell angewandt werden müssen. In den anschließenden Definitionen der Dienstschnittstellen wurden Fehlerfälle insofern berücksichtigt, daß das System in einem stabilen Zustand verbleibt; es werden jedoch keine umfangreichen automatischen Problembehebungen durchgeführt.

Dies bedeutet nicht, daß in Fehlerfällen, die aus der fehlenden Verfügbarkeit von Datenquellen resultieren, die Anfragebearbeitung nicht durchgeführt wird. Stattdessen könnte in Abhängigkeit von der Semantik der Anfrage eine eingeschränkte Bearbeitung stattfinden, sofern das dabei ermittelte Ergebnis für den Anwender trotzdem von Nutzen sein kann. Ein kurzes Beispiel illustriert dies. Bei einer einfachen Selektion von Dokumenten aus mehreren LDS - es findet also eine Vereinigung der Ergebnismengen statt - wird festgestellt, daß ein LDS nicht ansprechbar ist. Die Ergebnismenge wird folglich nur anhand der ansprechbaren LDS ermittelt und eine entsprechende Warnung erzeugt. Das ermittelte Ergebnis ist zwar nicht vollständig, aber für den Nutzer unter Umständen doch akzeptabel. Vorteil dieser Vorgehensweise ist die größere Fehlertoleranz des Systems. Die Erkennung, ob die Sematik einer Anfrage bei eingeschränkter Verfügbarkeit der LDS hinreichend erfüllt werden kann, ist jedoch eine sehr komplexe Aufgabe und stellt dementsprechend hohe Anforderungen an die Umsetzung.

3.2 Definition der Dienstschnittstellen

Die Architekturbeschreibung in Abbildung 3.2 auf Seite 35 gab bereits einen Überblick darüber, zwischen welchen Diensten des VDS Schnittstellen erforderlich sind. Doch insbesondere die Komplexität des Query-Servers erfordert, daß der Aufbau dieses Dienstes näher beschrieben wird, da hier der Schwerpunkt der Anfragevorbearbeitung und der Anfragenachbearbeitung liegt. Gemäß der in Kapitel 2 identifizierten Schwerpunkte gliedert sich die Beschreibung der Dienstschnittstellen folgendermaßen:

- Query-Server in Abschnitt 3.2.1
- Lokaler Dokumenten-Server in Abschnitt 3.2.2
- Replikations- und Caching-Dienst in Abschnitt 3.2.3

Die Dienstschnittstellen werden anhand eines Ausschnittes aus der IDL-Beschreibung erläutert. An den gegebenen Stellen wird auf die komplette formale Spezifikation in Anhang A verwiesen. Die Verwendung der IDL[8] ermöglicht die sprachunabhängige Spezifikation von Schnittstellen, die auf verschiedene Programmiersprachen abgebildet werden kann.

3.2.1 Anfrage- und Retrieval-Dienst

Die Aufgaben des Query-Servers umfassen die Kommunikation mit externen und lokalen VDS-Diensten, die Bearbeitung von Anfragen und die Verwaltung von Metadaten. Dementsprechend gliedert sich die in Abbildung 3.4 dargestellte Query-Serverarchitektur.

Das in Abschnitt 3.2.1.1 vorgestellte Query-Interface ist für den Empfang der Anfragen zuständig und löst die Anfragebearbeitung aus. Die Anfragebearbeitung umfaßt die Analyse der Anfrage, die Ermittlung der zu kontaktierenden Komponenten, die Kontaktierung dieser Komponenten und die Nachbearbeitung der Ergebnisse. Die Kontaktierung der zur Anfragebearbeitung erforderlichen lokalen und externen VDS-Dienste erfolgt Verwendung der Schnittstellen zum Wrapper (Abschnitt 3.2.1.3), zum Replikations- und Caching-Dienst (Abschnitt 3.2.1.4) sowie zum Föderationsdienst (Abschnitt 3.2.1.5). Die Verwaltung der dem VDS zugeordneten LDS erfolgt in einer separaten Komponente, der Ressourcen-Verwaltung (Abschnitt 3.2.1.2), die ein dynamisches Einbinden neuer LDS zur Laufzeit unterstützt.

Die in Abbildung 3.4 dargestellten Ressourcen- sowie Replikations- und Caching-Dienst-Interfaces bilden keine eigenständigen Komponenten, sondern sind als eine Art Abstraktionshilfe anzusehen.[9] Diese Interfaces werden durch die Anfragebearbeitung und die Ressourcen-Verwaltung benutzt, um auf die lokalen VDS-Dienste zuzugreifen.[10] Bei dem Query- und dem Föderationsdienst-Interface handelt es sich um eigenständige Komponenten innerhalb des Query-Servers, die

[8] *I*nterface *D*efinition *L*anguage: Eine formale Spezifikation ist in [cor98] verfügbar.
[9] Aus diesem Grund sind sie in der Abbildung 3.4 durch eine gestrichelte Linie markiert.
[10] Dies wird in der Abbildung durch die gestrichelten Pfeile ausgedrückt.

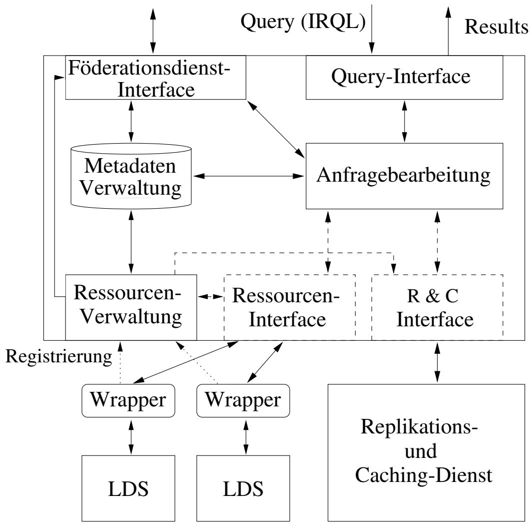

Abbildung 3.4: Aufbau des Query-Servers

für den Empfang von Anfragen zuständig sind. Ihre interne Schnittstelle zur Anfragebearbeitungskomponente entspricht weitestgehend ihrer externen Schnittstelle zum Empfang der Anfragen. Die Metadaten-Verwaltung speichert die statischen und dynamischen Metadaten des VDS und der zugeordneten LDS und stellt diese Metadaten dem Föderationsdienst-Interface, der Anfragebearbeitung und der Ressourcen-Verwaltung bereit.

Die Komponenten des Query-Servers laufen auf einem Host und sind so konzipiert, daß jede eingehende Anfrage eine eigene Anfragebearbeitung auslöst, in deren Verlauf die internen und externen Dienste des VDS parallel kontaktiert werden. Eine genaue Beschreibung der Bestandteile des Query-Servers wird in den folgenden Abschnitten gegeben. Die vollständige, formale Spezifikation der im Anschluß spezifizierten Schnittstellen des Query-Servers ist in Anhang A.1 nachzulesen.

3.2.1.1 Query-Interface

Anfragen an den Query-Server werden durch das Query-Interface entgegengenommen und anschließend die Anfragebearbeitung ausgelöst.[11] Eine Anfrage an den Query-Server kann durch einen Nutzer, einen externen Replikations-und Caching-Dienst oder durch den Föderationsdienst[12] gestellt werden.

[11] Pro eingehenden Request eine eigene Anfragebearbeitung.

[12] Metadaten-Retrieval-Anfragen können von einem externen Query-Server an einen Broker weitergeleitet werden, der vielversprechende VDS ermittelt. An diese wird die Anfrage via Föderationsdienst weitergeleitet. Der Föderationsdienst an sich ist nur an Anfragen interessiert, die den Datenbestand der LDS beschreiben. Diese Anfragen werden aber direkt an das Föderationsdienst-Interface gestellt.

Um eine Überlastung des Query-Servers zu verhindern, kann die maximale Anzahl gleichzeitig bearbeiteter Anfrage-Requests durch Einführung eines Maximums `maxOpenRequests` beschränkt werden. Anhand des ersten Prototypen sollte ermittelt werden, welcher Wert für dieses Maximum praktikabel ist. Ist zu einem Zeitpunkt die Summe der eingehenden und bearbeiteten Requests größer als das erlaubte Maximum, werden die über das Maximum hinausgehenden Requests abgewiesen. Als Alternative könnte in diesem Fall auch eine FIFO-Warteschlange eingerichtet werden.

Die von einem Query-Server bearbeiteten Anfragen wurden in der Beschreibung des Anfragebearbeitungsprozesses in Dokument-Retrieval-Anfragen und Metadaten-Retrieval-Anfragen unterteilt. Jeder dieser beiden Anfrageformen kann neben der eigentlichen Anfrage, die in der IRQL spezifiziert ist, weitere Informationen enthalten, die zur Bearbeitung der Anfrage hinzugezogen werden. Da sich diese Informationen bei den beiden Anfrageformen unterscheiden, ist eine differenzierte Betrachtung beim Entwurf des Query-Interfaces erforderlich.

Dokument-Retrieval-Anfragen. Neben dem DOI können Dokument-Retrieval-Anfragen auch Autorisierungsinformationen enthalten. Stellt sich bei der Bearbeitung der Anfrage heraus, daß eine Autorisierung erforderlich ist, aber die dazu notwendige Autorisierungsinformation nicht der Anfrage beigefügt ist, wird die Anfragebearbeitung beendet und an den Aufrufer die entsprechende Fehlermeldung zurückgeliefert. Für das Retrieval des Dokumentes muß demzufolge eine neue Anfrage gestellt werden, die die Autorisierungsinformation enthält. Durch diesen Ansatz wird die Funktionalität des Query-Servers vereinfacht, da im Hinblick auf die Kommunikation mit dem Aufrufer eine zustandslose Bearbeitung erfolgt.

Zum Retrieval von Dokumenten stellt das Query-Interface die Methode

```
void retrieveDocument(in string documentIdentifier,
                      in data options,
                      in data authorizationInformation,
                      out data document)
     raises (documentNotFoundException,
             miscRetrieveException,
             authorizationRequiredException,
             authorizationFailedException);
```

bereit, die als Parameter die Dokument-Identität (`documentIdentifier`) des zu ladenden Dokumentes, Optionen (`options`) und Autorisierungsinformationen (`authorizationInformation`) enthält. Die Methode liefert im Erfolgsfall das angeforderte Dokument und dazugehörige Metadaten (wie Größe, letztes Änderungsdatum uvm.) zurück. In Abschnitt 3.2.1.6 wird skizziert, welche konkreten Metadaten zurückgeliefert werden und wie das von der Methode zurückgelieferte Ergebnis modelliert werden kann. Wenn während der Anfragebearbeitung ein Fehler auftritt, wird dies durch das Auslösen einer Ausnahme signalisiert.

Durch die Optionen könnte beispielsweise der Anteil des Dokumentes[13] festgelegt werden, der durch die Methode zurückgeliefert wird. Dazu ist es erforderlich, daß dem Nutzer die Struktur des Dokumentes bekannt ist, was vor dem Retrieval eines Dokumentes eigentlich nicht der Fall sein sollte, sofern sich aus der Klassifizierung eines Dokumentes nicht ein Teil der Struktur ableiten läßt.[14] Ziel der Einführung dieses Parameters ist die Erweiterbarkeit von Dokument-Retrieval-Anfragen, um neben kompletten Dokumenten auch einzelne nicht frei zugreifbare Bestandteile eines Dokuments, die bei einer Suche als relevant ermittelt wurden, anfordern zu können. Die Festlegung auf eine bestimmte Struktur des Parameters `options` erfolgt an dieser Stelle nicht, da dazu Betrachtungen zur Modellierung der zurückgelieferten Dokumente erforderlich sind, die erst zu einem späteren Zeitpunkt vorgenommen werden.

Der zweite übergebene Parameter `authorizationInformation` kann die zum Zugriff auf das Dokument erforderlichen Autorisierungsinformationen enthalten. Soll einer Anfrage keine Autorisierungsinformation beigefügt werden, wird dieser Parameter mit einem bestimmten Wert belegt (entspricht bspw. dem NIL-Zeiger (oder auch NULL) in einigen Programmiersprachen). Anhand dieses Wertes kann der Replikations- und Caching-Dienst erkennen, daß keine Autorisierungsinformation für diese Anfrage spezifiziert wurde. Wenn die Autorisierungsinformation bei der Anforderung eines lizenzrechtlich geschützten Dokumentes fehlt, darf das Dokument nicht ausgeliefert werden (`authorizationRequiredException`). Die `authorizationFailedException` signalisiert, daß die übergebene Autorisierungsinformation für das angeforderte Dokument ungültig ist und der Zugriff auf dieses Dokument verweigert wurde.

Wenn kein Dokument mit dem angegebenen DOI gefunden werden konnte, wird eine `documentNotFoundException` ausgelöst. Andere allgemeine Fehler bei der Bearbeitung der Anfrage führen zu einer `miscRetrieveException`.[15]

In der Spezifikation für die der Methode übergebenen Parameter `options` und `authorizationInformation` und des zurückgelieferten Dokument-Objektes wird kein konkreter Typ angegeben, da dazu weitere Betrachtungen zur Autorisierung[16] und zur Verwaltung von Dokumenten[17] erforderlich sind. Einige Betrachtungen zur Modellierung der Struktur des zurückgelieferten Dokument-Objektes

[13] Ein Kapitel eines Buches, zwei Absätze eines Reports usw.

[14] Als Beispiel sei an dieser Stelle ein Buch gewählt, das in der Regel Kapitel enthält.

[15] Anmerkung: Aus Gründen der Übersichtlichkeit werden im Folgenden bestimmte Ausnahmen, die bei der Methodenausführung auftreten, zu einer allgemeinen Ausnahme (misc*) zusammengefaßt. Die allgemeinen Ausnahmen werden in der Methodenspezifikation mit * gekennzeichnet und enthalten eine verbale Information, die den konkreten Fehler näher beschreibt.

[16] Welche Informationen müssen übertragen werden und wie werden sie beschrieben?

[17] Gibt es für ein Dokument verschiedene Versionen, die vom Nutzer angefordert werden können? Wie wird diese Versionierung im Dokument abgelegt?

werden in Abschnitt 3.2.1.6 vorgenommen.

Metadaten-Retrieval-Anfragen. Eine Metadaten-Retrieval-Anfrage umfaßt neben der IRQL-Anfrage die Präferenzen des Nutzers. Diese Präferenzen charakterisieren die Einstellungen, die der Nutzer im Regelfall auf die Anfrage anwenden will. Zu diesen Einstellungen kann die maximale Anzahl der zurückzuliefernden Ergebnisse, die Fehlertoleranz und vieles mehr gehören. Die Präferenzen werden zusammen mit der eigentlichen Anfrage an die Anfragebearbeitung weitergereicht, die entsprechend der Vorgaben die Bearbeitung durchführt.

Das Query-Interface stellt für Metadaten-Retrieval-Anfragen die Methode

```
void findRelevantResults(in string query,
                         in data preferences,
                         out data resultList,
                         out string statusInfo)
     raises (miscFindException);
```

bereit, deren Parameter die in der IRQL spezifizierte Anfrage `query` und die bei der Bearbeitung zu berücksichtigenden Präferenzen `preferences` bilden. Die Methode liefert die von der Anfragebearbeitung ermittelte Ergebnisliste in Form eines Handles (`resultList`) und eine Statusinformation (`statusInfo`) zurück. Das Handle kann dazu genutzt werden, um auf die Ergebnistupel zuzugreifen. Die Statusinformationen geben Auskunft über Ereignisse, die die Ergebnisermittlung beeinflußt haben. Aus Gründen der Übersichtlichkeit werden die Präferenzen und die Modellierung der Ergebnisliste erst in Abschnitt 3.2.1.6 näher beschrieben.

Da die Bearbeitung von Metadaten-Retrieval-Anfragen eine komplexe Aufgabe darstellt, die in Zusammenarbeit von mehreren lokalen und externen VDS-Diensten gelöst wird, kann eine Vielzahl von Fehlern auftreten, die bei der Vorbearbeitung durch die Anfragebearbeitungskomponente beginnend über die Kontaktierung der anderen VDS-Dienste, die Umformung der Anfragen, die Bearbeitung der Anfragen durch die LDS bis hin zur abschließenden Nachbearbeitung der Ergebnisse durch die Anfragebearbeitungskomponente auftreten können. Da nicht alle Fehler für den Nutzer des Query-Servers relevant sind, sondern meistens nur ein Interesse am Ergebnis besteht[18], werden entsprechend der Präferenzen des Nutzers nur die jeweils allgemeinsten Fehler durch die `miscFindException`* weitergereicht.

Fehler, die nicht zum Abbruch der Anfragebearbeitung führen aber für den Nutzer von Interesse sein können, da sie das Ergebnis beeinflußt haben, werden

[18] Als Nutzer eines Suchdienstes ist man nur bis zu einem bestimmten Grad an Informationen über den Anfrageprozeß interessiert. Während beispielsweise das Wissen von Nutzen sein kann, daß eine Ressource nicht ansprechbar ist, kann der Grund für dieses Verhalten irrelevant sein, solange kein Einfluß auf die Fehlerbehebung genommen werden kann.

neben dem Ergebnis als Statusinformation zurückgeliefert und nicht durch das Auslösen einer Ausnahme signalisiert.[19]

Wenn die Anfrage keine Ergebnisse liefert, wird dies durch eine leere Liste und nicht wie bei Dokument-Retrieval-Anfragen durch eine Ausnahme signalisiert.

Die komplette formale Spezifikation der hier vorgestellten Query-Schnittstelle ist in Anhang A.1.1 nachzulesen. Eine Schnittstelle zur Bearbeitung einer spezielleren Form von Anfragen, die vom Föderationsdienst an den Query-Server gestellt werden ohne das Query-Interface zu verwenden, wird in Abschnitt 3.2.1.5 vorgestellt.

3.2.1.2 LDS/Ressourcen-Verwaltung

Einem VDS können laut allgemeiner Architekturbeschreibung beliebig viele LDS zugeordnet werden. Damit die Dienste des VDS diese LDS ansprechen können, ist eine Registrierung in der Ressourcen-Verwaltung erforderlich. Die Aufgaben der Ressourcen-Verwaltung umfassen die Anmeldung (`addResource`), die dauerhafte Abmeldung (`removeResource`), die kurzzeitige Abmeldung während der Wartung des Systems o.ä. (`deactivateResource`) und die Wiederanmeldung (`reactivateResource`) einer Ressource und die Informierung der lokalen und externen Dienste über diese Änderungen. Zu den zu informierenden Diensten gehören der lokale Replikations- und Caching-Dienst, der auf die Dokumente des LDS zugreift sowie der Föderationsdienst und der Broker, denen die statischen und dynamischen Metadaten des LDS mitgeteilt werden müssen, damit diese die neuen Informationen in ihren Datenbestand aufnehmen können.

Anmeldung eines neuen LDS. Für die Anmeldung eines neuen LDS wird von der Ressourcen-Verwaltung die Methode

```
void addResource(in data resourceDescription,
                 out string resourceIdentifier)
      raises (resourceNotFoundException,
              resourceManagementException);
```

bereitgestellt, deren Argument die Beschreibung der Ressource bildet, die angemeldet werden soll. Die Methode liefert bei erfolgreicher Anmeldung eine LDS-Identifikation (`resourceIdentifier`) zurück, die für den gesamten VDS eindeutig ist. Bei der Anmeldung einer neuen Ressource wird durch die Verwaltung geprüft, ob die Ressource angesprochen werden kann.[20] Ist dies nicht der Fall wird eine `resourceNotFoundException` ausgelöst und die Anmeldung als gescheitert betrachtet. Tritt ein anderweitiger Fehler bei der Anmeldung auf, wird eine `resourceManagementException`* ausgelöst.

[19] Siehe dazu die Betrachtungen zum Verhalten des Systems in Fehlerfällen in Abschnitt 3.1.3.
[20] Die dazu verwendete Methode ist in Abschnitt 3.2.1.3 beschrieben.

Die Struktur einer Ressourcenbeschreibung soll an dieser Stelle nicht festgelegt werden, da neben allgemeinen Informationen auch implementierungsspezifische Informationen[21] beschrieben werden müssen. Die allgemeinen Informationen können sowohl informale Beschreibungen als auch formale Beschreibungen[22] enthalten.

Im Anschluß an die erfolgreiche Registrierung werden die statischen und dynamischen Metadaten des LDS angefordert, durch die Metadaten-Verwaltung im Repository abgelegt und die in- und externen Dienste über die Neuanmeldung informiert. Diese Dienste müssen ihrerseits sicherstellen, daß die Ressource von ihnen bei Bedarf auch genutzt werden kann.[23]

Abmeldung eines LDS. Wenn ein LDS dauerhaft aus dem VDS ausgegliedert werden soll, kann dies mit Hilfe der Methode

```
void removeResource(in string resourceIdentifier)
      raises (resourceIdNotFoundException,
              resourceManagementException);
```

geschehen. Das Argument dieser Methode ist der bei der Anmeldung vergebene Identifikator des LDS. Ist der Ressourcen-Verwaltung kein LDS mit diesem Identifikator bekannt, wird eine `resourceIdNotFoundException` ausgelöst. Tritt bei der Bearbeitung ein allgemeiner Fehler auf, so wird dies durch das Auslösen einer `resourceManagementException`* gekennzeichnet.

Bei der Abmeldung eines LDS sind die lokalen und externen Dienste über die Abmeldung zu informieren, so daß durch diese keine weiteren Anfragen an den LDS gerichtet werden. Die statischen und dynamischen Metadaten können dann durch die Metadaten-Verwaltung aus dem Repository entfernt werden.

Zeitlich begrenzte Deaktivierung eines LDS. Neben der dauerhaften Entfernung eines LDS aus dem VDS kann es Anwendungsszenarien geben, in denen ein LDS nur für einen begrenzten Zeitraum abgemeldet werden soll.[24] Solange eine Ressource als abgemeldet markiert ist, kann durch keinen Dienst des VDS auf diese Ressource zugegriffen werden. Dafür stellt die Ressourcen-Verwaltung die Methode

```
void deactivateResource(in string resourceIdentifier)
      raises (resourceIdNotFoundException,
              resourceManagementException);
```

[21] Wie wird mit der Ressource kommuniziert?

[22] Klassifizierung des Inhaltes des LDS nach bestimmten Vorgaben.

[23] Der Verantwortliche für den Föderationsdienst muß beispielsweise die für die Homogenisierung notwendigen Transformationen bestimmen und der Broker die dynamischen Metadaten bei der Ermittlung von zu kontaktierenden VDS berücksichtigen usw.

[24] Ein Beispiel für einen solchen Fall kann sein, daß der LDS wegen Wartungsarbeiten für einen begrenzten Zeitraum abgeschaltet wird.

bereit. Die Methode erhält als Argument den bei der Anmeldung vergebenen eindeutigen Identifikator des LDS. Ist der Ressourcen-Verwaltung kein LDS mit diesem Identifikator bekannt, wird eine `resourceIdNotFoundException` ausgelöst. Treten andere Fehler bei der Bearbeitung auf, wird das durch das Auslösen der `resourceManagementException`* signalisiert.

Im Gegensatz zur endgültigen Abmeldung dürfen bei der kurzzeitigen Deaktivierung die Metainformationen des LDS nicht aus dem Repository der Ressourcen-Verwaltung entfernt werden. Zum jetzigen Zeitpunkt läßt sich nicht abschätzen, ob es erforderlich ist, die externen Dienste über die Deaktivierung zu informieren, da sie nicht direkt auf die LDS zugreifen und indirekte Zugriffe (durch Anfragen) durch die lokalen VDS-Dienste erfolgen, die über die Deaktivierung informiert sind. Eine Abstimmung mit den externen Projektpartnern ist in diesem Fall notwendig.

Reaktivierung eines LDS. Nachdem ein LDS für einen begrenzten Zeitraum abgemeldet wurde, kann er durch die Methode

```
void reactivateResource(in string resourceIdentifier)
      raises (resourceIdNotFoundException,
              resourceNotFoundException,
              resourceManagementException);
```

wieder für die Anfragebearbeitung freigegeben werden. Da der Grund für eine zeitweilige Abschaltung des LDS bspw. Änderungen an der Modellierung des Datenbestandes sein können, wird bei der Reaktivierung des LDS der statische und dynamische Metadatenbestand des LDS vom Query-Server neu eingelesen und im Repository gespeichert. Durch Einführung dieses Deaktivierungs/Reaktivierungs-Konzeptes wird eine Möglichkeit geschaffen, den Query-Server über Änderungen des dynamischen und statischen Metadatenbestandes eines LDS zu informieren, so daß dieser nicht unbedingt von sich aus in zyklischen Abständen die Metadaten anfordern muß, um Änderungen zu erkennen.

Auf eine Erläuterung der Schnittstelle soll an dieser Stelle verzichtet werden, da die Parameter, die Rückgabewerte und die ausgelösten Fehlerfälle in den vorherigen drei Methodenbeschreibungen ausführlich vorgestellt wurden. Bei der Reaktivierung eines LDS wird den Diensten des VDS eine Nachricht (siehe Abschnitt 3.2.1.4) zugesandt, die darüber informiert, daß der LDS wieder angesprochen werden kann.

Neben den in diesem Abschnitt definierten Methoden bietet die Ressourcen-Verwaltung weitere allgemeine Methoden an, die über die aktuell verwalteten Ressourcen Auskunft geben. Da diese Methoden insbesondere durch eine Administratorschnittstelle[25] und nicht durch die Dienste des VDS genutzt werden, sei

[25] Eine einfache GUI, die die komfortable Administration gestattet.

an dieser Stelle auf eine Festlegung verzichtet. Die Schnittstellenspezifikation der hier beschriebenen Ressourcen-Verwaltung ist in Anhang A.1.2 enthalten.

3.2.1.3 Schnittstelle zum Wrapper/LDS[26]

Der Query-Server stellt eine Schnittstelle für den einheitlichen Zugriff auf die Wrapper der LDS bereit. Die Schnittstelle setzt sich aus einer Methode zum Aufruf der Metadaten-Retrieval-Anfragebearbeitung der Wrapper (`processQuery`), Methoden zum Import von statischen LDS-Metadaten (`getStaticMetadata`) und dynamischen Metadaten (`getDynamicMetaData`) sowie einer Methode `ping` zur Überprüfung, ob eine Verbindung zum Wrapper aufgebaut werden kann, zusammen. Eine Schnittstelle für die Bearbeitung von Dokument-Retrieval-Anfragen wird durch diese Schnittstelle nicht bereitgestellt, da der Zugriff auf die Dokumente durch den Replikations- und Caching-Dienst erfolgt.

Die Schnittstellenspezifikation der nun vorgestellten Methoden des Ressourcen-Interfaces ist in Anhang A.1.3 nachzulesen.

Auslösen der Anfragebearbeitung im Wrapper. Die Anfragebearbeitungskomponente des Query-Servers ermittelt anhand der Anfrage, welche LDS zu kontaktieren sind und löst die Anfragebearbeitung durch die Wrapper aus. Dafür wird die Methode

```
void processQuery(in string query,
                  in data preferences,
                  out data resultList)
     raises (connectFailedException,
             miscWrapperException);
```

bereitgestellt, deren Parameter die IRQL-Anfrage `query` sowie dazugehörige Präferenzen `preferences` sind, sofern diese für die Anfragebearbeitung durch den Wrapper von Bedeutung sind. Beispielsweise ist die maximale Anzahl der Ergebnistupel eine solche Präferenz, die an den Wrapper weitergeleitet wird. Aufgrund dieser Information verkürzt der Wrapper die Ergebnisliste des LDS auf die gewünschte Größe.

Das durch den Wrapper erzeugte Ergebnis-Handle `resultList` kann der Query-Server nutzen, um auf die Ergebnisse zuzugreifen. Trat bei der Kontaktierung des Wrappers ein Fehler auf, so wird eine `connectFailedException` ausgelöst. Alle anderen Fehlerfälle werden durch die `miscWrapperException`* repräsentiert.

Die Anfragebearbeitungskomponente des Query-Servers wartet auf die Beendigung der Methode und führt auf allen Ergebnissen, die durch die LDS bzw. indirekt durch den Broker ermittelt wurden, eine abschließende Nachbearbeitung aus (Mischen und globales Ranking).

[26] In Abbildung 3.4 wurde diese Komponente als Ressourcen-Interface und nicht als LDS-Interface bezeichnet, um eine Verwechslung mit der Schnittstelle des LDS zu vermeiden.

Import von Metadaten der LDS. Die im Repository des Query-Servers gespeicherten Metadaten der LDS werden von der Anfragebearbeitungskomponente durch die Methoden `getStaticMetadata` und `getDynamicMetaData` importiert. Der Import von statischen Metadaten erfolgt durch die Methode

```
void getStaticMetadata(out data metaData)
     raises (connectFailedException,
             miscMetadataImportException);
```

die keine Parameter enthält und als Ergebnis alle statischen Metadaten eines LDS in einer hier nicht näher spezifizierten Form zurückliefert. Scheitert der Verbindungsaufbau zum Wrapper/LDS, wird eine `connectFailedException` ausgelöst. Fehler, die bei der Bearbeitung durch den LDS auftreten, werden durch das Auslösen der `miscMetadataImportException`* signalisiert.

Der Import von dynamischen Metadaten erfolgt durch Nutzung der Methode

```
void getDynamicMetadata(out data metaData)
     raises (connectFailedException,
             miscMetadataImportException);
```

die keine Parameter übergeben bekommt, eine Menge von dynamischen Metadaten eines LDS (`metaData`) zurückliefert und die im Fehlerfall die gleichen Ausnahmen wie die bereits beschriebene Methode zum Import von statischen Metadaten auslösen kann.

In den beiden zuletzt angegebenen Schnittstellenbeschreibungen wurde keine Aussage zur Struktur der Metadatenbeschreibungen gegeben. Eine Festlegung auf eine konkrete Form ist insofern kompliziert, da hierzu Entscheidungen und Wünsche von externen Projektpartnern, die diese Metainformationen verwenden, zu berücksichtigen sind. Die Struktur dieser Daten ist lediglich für die Metadaten-Verwaltung von Bedeutung, da diese die Daten speichert. Eine Interpretation der Daten findet jedoch in den Diensten des VDS nicht statt. Aus diesem Grund wurde an dieser Stelle von der konkreten Struktur abstrahiert.

Verbindungstest. Bei der Neuanmeldung eines LDS bzw. bei anderen Verwaltungsoperationen der Ressourcen-Verwaltungskomponente kann es erforderlich sein, zu überprüfen, ob eine Verbindung zum Wrapper des LDS aufgebaut werden kann. Dafür wird die Methode

```
void ping(out boolean alive);
```

bereitgestellt, die keine Parameter enthält und deren Rückgabe ein boolescher Wert ist, der Auskunft darüber gibt, ob der Wrapper ansprechbar ist (*true*) oder nicht (*false*).

Eine Ausweitung der Überprüfung auf die Verfügbarkeit des LDS ist auf einfache Weise möglich, indem der Wrapper die dazu notwendige Funktionalität bereitstellt und das Ergebnis dieser Überprüfung in den Rückgabewert der Methode einfließen läßt.

Wird bei der Überprüfung festgestellt, daß ein Wrapper/LDS nicht ansprechbar ist, kann diese Ressource in der Ressourcen-Verwaltung als temporär nicht erreichbar markiert werden. Solange eine Ressource als nicht erreichbar markiert ist, werden an diese keine Anfragen weitergeleitet.

3.2.1.4 Schnittstelle zum Replikations- und Caching-Dienst

Die Schnittstelle zum Replikations- und Caching-Dienst gestattet es der Anfragebearbeitung und der Ressourcen-Verwaltung, Dokument-Retrieval-Anfragen durch Nutzung der Methode `retrieveDocument` und Nachrichten über neu angemeldete bzw. abgemeldete Ressourcen (siehe Abschnitt 3.2.1.2) an selbigen zu schicken.

Dokument-Retrieval-Anfragen. Dokument-Retrieval-Anfragen werden bekanntlich nicht vom Query-Server bearbeitet sondern an den Replikations- und Caching-Dienst weitergereicht. Infolgedessen besitzt die entsprechende Methode des Query-Interfaces und die hier spezifizierte Methode die gleiche Signatur[27]:

```
void retrieveDocument(in string documentIdentifier,
                      in data authorizationInformation,
                      in data options,
                      out data document)
     raises (documentNotFoundException,
             miscRetrieveException,
             authorizationRequiredException,
             authorizationFailedException);
```

Der Methode werden als Parameter der DOI (`documentIdentifier`), optionale Autorisierungsinformationen (`authorizationInformation`) und Informationen (`options`) zur Beschreibung des zu ladenden Dokumentbestandteiles übergeben.[28] Diese Methode löst beim Replikations- und Caching-Dienst das Retrieval des Dokumentes und, falls erforderlich, die Verifizierung der Autorisierungsinformation durch den Autorisierungsdienst aus. Falls der Replikations- und Caching-Dienst das Dokument laden konnte, wird es in einer hier nicht konkretisierten Form an den Query-Server zurückgeliefert, wobei neben dem eigentlichen Dokument weitere Metadaten, die für den Nutzer bzw. für die weitere Bearbeitung vom System von Interesse sind, enthalten sein können. Im Fehlerfall werden die bereits vorher beschriebenen Ausnahmen ausgelöst.

[27] Eine ausführliche Beschreibung der Methoden wurde bereits auf Seite 42 gegeben.

[28] Siehe Diskussion in Abschnitt 3.2.1.1.

Anmeldung/Abmeldung eines LDS. Da der Query-Server für die Verwaltung der zugeordneten LDS zuständig ist, liegt es in seiner Verantwortung, den Replikations- und Caching-Dienst über Änderungen der verwalteten LDS zu informieren, damit kein Zugriff auf einen nicht erreichbaren LDS erfolgt. Dazu stellt die Schnittstelle zum Replikations- und Caching-Dienst eine Methode zur Anmeldung (`registerResource`), eine Methode zur Abmeldung (`deleteResource`), eine Methode zur zeitlich begrenzten Abmeldung (`deactivateResource`) und eine Methode zur Wiederanmeldung (`reactivateResource`) eines LDS bereit. Die Registrierung eines neuen LDS beim Replikations- und Caching-Dienst erfolgt durch die Methode

```
void registerResource(in string resourceIdentifier,
                      in data resourceAccessDescription)
     raises (miscManagementException);
```

deren Parameter die eindeutige LDS-Identität `resourceIdentifier` und eine Beschreibung `resourceAccessDescription`[29] bilden. Nach erfolgreicher Registrierung kann der Replikations- und Caching-Dienst via Wrapper auf den LDS zugreifen. Wenn die Registrierung scheitert, wird eine `miscManagementException`* ausgelöst, die aussagt, daß kein Zugriff auf den LDS erfolgen kann.

Ein LDS kann solange durch den Replikations- und Caching-Dienst angesprochen werden, bis seine Registrierung durch Aufruf der Methode

```
void deleteResource(in string resourceIdentifier)
     raises (resourceIdNotFoundException,
             miscManagementException);
```

gelöscht wird. Die Methode erhält als Parameter den Identifikator des LDS, unter dem die zu entfernende Ressource registriert wurde. Ist diese Ressource beim Replikations- und Caching-Dienst nicht registriert, wird eine Ausnahme (`resourceIdNotFoundException`) ausgelöst. Andere kritische Fehlerfälle werden durch das Auslösen der `miscManagementException`* signalisiert.

Die zeitlich begrenzte Abmeldung einer Ressource wurde eingeführt, um in bestimmten Situationen, beispielsweise während Wartungsarbeiten, den Zugriff auf die Ressource zu verhindern. Erfolgen kann die Abmeldung durch Aufruf der Methode

```
void deactivateResource(in string resourceIdentifier)
     raises (resourceIdNotFoundException,
             miscManagementException);
```

deren Parameter wiederum die Identität ist, unter dem die Ressource registriert wurde. Die im Fehlerfall ausgelösten Ausnahmen entsprechen denen der Methode `deleteResource`.

[29] Diese gibt an, wie auf den Wrapper des LDS zugegriffen wird. Dazu sind implementierungsabhängige Informationen erforderlich.

Soll eine bereits registrierte Ressource, die zwischenzeitlich deaktiviert worden ist, dem Replikations- und Caching-Dienst wieder zur Verfügung gestellt werden, erfolgt dies durch den Aufruf der Methode

```
void reactivateResource(in string resourceIdentifier)
     raises (resourceIdNotFoundException,
             miscManagementException);
```

Somit besitzt der Replikations- und Caching-Dienst eine eigene Ressourcen-Verwaltung, die durch den Query-Server gesteuert wird.

Die Spezifikation der Replikations- und Caching-Dienst-Schnittstelle des Query-Servers ist in Anhang A.1.4 nachzulesen.

3.2.1.5 Schnittstelle zum Föderationsdienst

Der Query-Server nutzt den Föderationsdienst um Metadaten-Retrieval-Anfragen an entfernte VDS zu senden und stellt den externen Diensten via Föderationsdienst seine Datenbestände zur Verfügung. In Anlehnung an diese Aufgaben besitzt die Föderationsdienstschnittstelle Methoden zum Empfang von Anfragen über den statischen Metadatenbestand (`fetchStaticMetadata`) und den dynamischen Metadatenbestand (`fetchDynamicMetadata`) des VDS und der LDS und Methoden zum Senden von Metadaten-Retrieval-Anfragen an entfernte VDS (`forwardQuery`), die durch den Broker ermittelt werden. Metadaten-Retrieval-Anfragen, die durch den Föderationsdienst an einen entfernten VDS gesendet werden, werden vom Empfänger-VDS nicht durch die Föderationsdienstschnittstelle sondern durch das in Abschnitt 3.2.1.1 beschriebene Query-Interface entgegengenommen.

Die in diesem Abschnitt erläuterten Methoden sind in Anhang A.1.5 formal spezifiziert.

Anfragen zum dynamischen und statischen Metadatenbestand. Externe Dienste wie der Föderationsdienst und der Broker arbeiten auf den Metadaten der VDS und der LDS. Der dynamische Metadatenbestand wird hauptsächlich vom Broker verwendet, der aufgrund dieser Daten die Selektion der zu kontaktierenden VDS vornimmt. Diese Metadaten können vom Broker automatisch ausgewertet werden. Zum Zugriff auf die dynamischen Metadaten wird die Methode

```
void fetchDynamicMetadata(out data metaData)
     raises (miscFetchException);
```

bereitgestellt, die keine Parameter besitzt und einen Metadatenstrom zurückliefert.[30] Im Fehlerfall löst die Methode eine `miscFetchException`* aus, die das

[30] Aus Gründen der Übersichtlichkeit wird in der hier vorliegenden Schnittstellenbeschreibung von der Struktur des Metadatenstroms abstrahiert.

aufgetretene Problem[31] näher beschreibt.

Statische Metadaten können in der Regel nicht automatisch durch externe Dienste interpretiert und verwendet werden. Als Beispiel seien die Schemainformationen der LDS genannt, die der Föderationsdienst zur Homogenisierung verwendet. Da viele der auftretenden Probleme nicht durch automatisierte Vorgänge gelöst werden können, ist eine Einwirkung einer dafür verantwortlichen Person notwendig. Aus diesem Grund kann die Exportschnittstelle für statische Metadaten als unterstützende Hilfe zum Zugriff auf Informationen, die ansonsten nicht frei verfügbar wären, angesehen werden. Die Methode

```
void fetchStaticMetadata(out data metaData)
     raises (miscFetchException);
```

besitzt keine Parameter und liefert einen Metadatenstrom zurück. Die Struktur eines solchen Datenstroms wird, wie auch bereits bei den dynamischen Metadaten zum Ausdruck gebracht wurde, an dieser Stelle vernachlässigt, da eine Interpretation der Daten durch den Query-Server nicht erfolgt. Im Fehlerfall wird wie bereits bei der dynamischen Variante der Methode die Exception `miscFetchException`* ausgelöst.

Änderungen am Datenbestand der LDS führen eine Änderung des dynamischen Datenbestandes nach sich. Da die LDS von sich aus nicht den Query-Server über diese Änderungen informieren müssen, kann es erforderlich sein, daß dieser die Metadaten in zyklischen Abständen oder nach expliziter Aufforderung anfordert.[32] Für den expliziten Ansatz wird die Methode

```
void verifyMetadata()
     raises (miscVerifyException);
```

bereitgestellt, die eine Überprüfung des dynamischen Metadatenbestandes des Repositories im Query-Server auslöst. Dazu kontaktiert der Query-Server alle ihm zugeordneten LDS und fordert die aktuellen Metadaten an. Die Methode besitzt keine Parameter und liefert keine Ergebnisse zurück.[33] Bei erfolgreicher Bearbeitung der Methode wird sowohl der transient als auch der persistent gehaltene Metadatenbestand auf den neuesten Stand gebracht. Durch den Aufruf der `fetchDynamicMetadata`- bzw. der `fetchStaticMetadata`- Methode kann der Föderationsdienst den aktualisierten Metadatenbestand des VDS abfragen.

[31] Ein solches Problem kann beispielsweise sein, daß der Datenbestand gerade aktualisiert wird.

[32] Würde gefordert, daß die LDS den Query-Server selbständig über Änderungen informieren, könnte diese Methode entfallen, da die Metadaten im Repository bei Bedarf sofort nach der Benachrichtigung neu geladen werden würden.

[33] Würde die Methode als Ergebnis zurückliefern, ob der Metadatenbestand geändert wurde, wäre es erforderlich, daß ein Vergleich der Metadaten erfolgen muß. Um diesen Vergleich durchzuführen ist es aber erforderlich, die Struktur der Metadaten zu kennen. Dieses soll im gewählten Ansatz aber vermieden werden, da der Query-Server eine reine Vermittlerrolle spielt.

Scheitert die Verifikation, wird von der Methode eine `miscVerifyException`* ausgelöst.

Greift eine Methode während des Updates auf den Metadatenbestand zu, wird bei ersterer eine Ausnahme ausgelöst. Änderungen am statischen Metadatenbestand finden sehr selten statt und beschränken sich im Wesentlichen auf den Eintrag neuer LDS bzw. deren Schemata. Diese Neueintragungen können automatisch erkannt werden, da das Repository in seinem dynamischen Metadatenbestand eine Liste aller angemeldeten LDS verwaltet.

In den vorangegangenen Methoden wurde davon ausgegangen, daß der komplette dynamische bzw. der komplette statische Metadatenbestand des Repositories durch den Föderationsdienst angefordert werden muß. In Abhängigkeit vom Umfang der Daten kann es jedoch sinnvoller sein, für jeden LDS diese Daten getrennt anzufordern. Die bisherigen Schnittstellenbeschreibungen müßten in diesem Fall dahingehend geändert werden, daß ein weiteres Argument, das den `resourceIdentifier` enthält, eingeführt wird.

Anmeldung/Abmeldung eines LDS. Wenn ein LDS in einem VDS an- oder abgemeldet wird (Abschnitt 3.2.1.2), erfolgt eine Benachrichtigung der externen Dienste durch den Query-Server. Dazu stellt die Föderationsdienstschnittstelle die Methode `notify` zur Verfügung, die kein Argument enthält und keinen Rückgabewert liefert. Im Fehlerfall löst die Methode eine `miscNotifyException`* aus:

```
void notify()
     raises (miscNotifyException);
```

Durch den Aufruf der Methode wird der Föderationsdienst veranlaßt, den Metadatenbestand des VDS neu einzulesen, um die erfolgten Änderungen zu erfassen.

Weiterleitung von Anfragen an entfernte VDS. Im Rahmen der Anfragebearbeitung kann der Query-Server entscheiden, ob eine Metadaten-Retrieval-Anfrage an externe VDS weitergeleitet werden soll. Dazu kontaktiert der Query-Server via Föderationsdienst den Broker, der einige vielversprechende VDS ermittelt und die Anfrage an diese VDS weiterleitet. Die Kontaktierung des Föderationsdienstes erfolgt durch Aufruf der Methode

```
void forwardQuery(in string query,
                  in data preferences,
                  out data resultList)
     raises (miscForwardException);
```

deren Parameter die Anfrage `query` und die Präferenzen[34] `preferences` bilden. Das Ergebnis ist wiederum ein Handle, durch das auf die Anfrageergebnisse zugegriffen werden kann. Im Fehlerfall wird durch diese Methode eine allgemeine Ausnahme (`miscForwardException`*) ausgelöst.

[34] Siehe Metadaten-Retrieval-Anfragen in Abschnitt 3.2.1.1

Die Ergebnisliste einer Anfrage, die durch einen externen Dienst bearbeitet wurde, ist um Informationen angereichert, welche die Herkunft der Ergebnisse näher beschreiben.[35] Die Methode liefert eine leere Ergebnisliste zurück, wenn bei der Recherche durch den Broker kein VDS bestimmt werden konnte oder wenn in den VDS keine Ergebnisse gefunden wurden.

3.2.1.6 Weitergehende Betrachtungen

In den vorherigen Abschnitten wurden die externen Dienstschnittstellen des Query-Servers vorgestellt und ihre Aufgaben beschrieben. An einigen Punkten wurden jedoch keine konkreten Festlegungen zu Datentypen bzw. Strukturen der Parameter getroffen, da Entscheidungen zur Modellierung der entsprechenden Objekte (Dokumente, Ergebnislisten, Präferenzen, Metadaten) noch ausstanden. Dies wird für einige wesentliche Objekte als Abschluß der Beschreibung des Aufbaus und der Funktionalität des Query-Servers in diesem Abschnitt nachgeholt.

Modellierung eines Dokumentes. Der Query-Server kann durch externe Dienste beauftragt werden, ein Dokument auszuliefern. Dazu ist es erforderlich, innerhalb des VDS ein Dokument-Objekt zu modellieren, das den Inhalt dieses Dokumentes enthält. Dabei kann es sinnvoll sein, daß ein solches Dokument-Objekt weitere Metadaten umfaßt, die das Dokument genauer beschreiben. Würde die Modellierung eines Dokument-Objektes keine Metadaten beinhalten, wäre es für einige Dienste (wie den Profil- und Benachrichtigungsdienst) erforderlich, diese Metadaten in einem getrennten Schritt anzufordern.

Da nicht alle für das Dokument verfügbaren Metadaten in einem Dokument-Objekt vertreten sein sollen, ist eine Selektion wesentlicher Metadaten erforderlich. Die hier beschriebene Modellierung enthält einige Metadaten, die dem *Dublin Core Element Set*[36] entnommen sind. Dieses wurde durch die *Dublin Core Metadata Initiative* [dc99] eingeführt, um elektronische Ressourcen (Web-Dokumente) beschreiben zu können. Der Dublin Core zeichnet sich durch Verständlichkeit (nutzbar sowohl durch Fachleute als auch durch Laien), Erweiterbarkeit und im Vergleich zu speziellen Beschreibungen, die im Bibliothekswesen entstanden sind (bspw. MARC), durch Allgemeingültigkeit aus. Die Metadaten der Dokument-Objekte können von Suchmaschinen dazu genutzt werden, die Suche auf spezielle Informationen einzugrenzen und somit die Ergebnisse zu verbessern.

Die Modellierung eines Dokument-Objektes könnte unter Berücksichtigung der im Anschluß vorgeschlagenen Verwendung eines Handles beispielsweise die folgende Struktur aufweisen:

---

[35] Eine solche Information kann beispielsweise sein, in welchem externen VDS gesucht wurde. Diese Information kann etwa an den Nutzer weitergeleitet werden, falls dieser aufgrund von positiven Erfahrungen in nachfolgenden Anfragen diese Ressource explizit durchsuchen möchte.

[36] `http://purl.oclc.org/dc/about/element_set.htm`

```
struct documentType
 {
  string          documentIdentifier;
  string          dateCreated;
  string          dateLastModified;
  string          md5;
  long            size;
  documentHandle  complete_content;
  string          mimeType;
  string          xmlDTD;
 };
```

Die hier aufgeführten Metadaten dienen nicht der Beschreibung des Inhaltes des Dokumentes, sondern beschreiben das Dokument selbst. Diese Informationen sind nicht unbedingt für den Nutzer von Bedeutung; sie können vielmehr von den an der Anfragebearbeitung beteiligten VDS-Diensten verwendet werden. In dem vorliegenden ersten Modellierungsvorschlag setzt sich ein Dokument-Objekt aus

- dem DOI (der Dokument-Identität),
- dem Datum der Erzeugung[37],
- dem Datum der letzten Änderung[37],
- einer Prüfsumme (RFC1321) [md5],
- der Größe,
- einem Handle zum Zugriff auf den Inhalt des Dokumentes,
- dem Format, in dem das Dokument vorliegt (Postscript, Text), sowie
- der *Document Type Definition* (DTD), die die Struktur des XML-Dokuments beschreibt (falls es sich um ein XML-Dokument handelt)

zusammen. Anhand des DOI kann das Dokument-Objekt eindeutig identifiziert werden. Das Datum der Erzeugung und das Datum der letzten Änderung können für den Nutzer von Interesse sein. Anhand der Prüfsumme können Veränderungen am Dokument erkannt werden. Die Größe des Dokumentes kann dem Nutzer als statistische Information präsentiert werden, wenn das zu übertragende Dokument sehr umfangreich ist und die Übertragung eine längere Zeit in Anspruch nimmt. Eine Beschreibung, in welchem Format ein Dokument vorliegt, ist eigentlich nicht erforderlich, da die Lokalen Dokumenten-Server Dokumente verwalten, die in XML/RDF vorliegen. Aus Gründen der Erweiterbarkeit ist dieses Attribut jedoch sinnvoll, da nicht sichergestellt werden kann, daß externe Literatur-Recherche-Dienste, die ebenfalls in einen VDS eingebunden werden können, nur

[37] Darstellung beispielsweise entsprechend ISO 8601 kompakte Form: 19990305 für den 5. März 1999. In Abhängigkeit von den Anforderungen könnte jedoch auch eine kleinere Granularität (YYYYMMDDHHMM) erforderlich sein.

XML-Dokumente enthalten. Anhand dieses Attributes ist es dem Nutzersystem möglich, das diesem Format zugeordnete Tool zu starten, um das Dokument korrekt darzustellen. Dazu bietet sich eine Kodierung im MIME-Format [mime] an, da dieses bereits eine standardisierte Klassifizierung von Daten (Text, Image, Audio, Video, Application usw.) einführt. Handelt es sich bei dem Dokument um ein XML-Dokument, für das die DTD bekannt ist, wird diese im entsprechenden Attribut abgelegt.

An dieser Stelle wird davon ausgegangen, daß die LDS über die beschriebenen Metainformationen verfügen und sie auch an die Dienste des VDS weiterleiten.

Es bietet sich an, diese Modellierung im gesamten VDS beizubehalten, so daß zwischen den Diensten des VDS keine Metadaten-Konvertierung durchgeführt werden muß. Es ist hier eine Abstimmung mit dem Replikations- und Caching-Dienst notwendig, da dieser eventuell weitere Informationen benötigt.

In der Strukturbeschreibung des Dokument-Objektes wurde bewußt offen gelassen, wie der Zugriff auf den Inhalt des Dokumentes erfolgen kann. Dies wird nun nachgeholt.

Zugriff auf den Inhalt eines Dokumentes. Für den Zugriff auf den Inhalt eines Dokumentes werden in Abbildung 3.5 zwei Ansätze dargestellt, die skizzieren, wie auf eine Dokument-Retrieval-Anfrage ein Dokument-Objekt zurückgeliefert werden kann. In Ansatz eins ist der Inhalt des Dokumentes Teil der Rück-

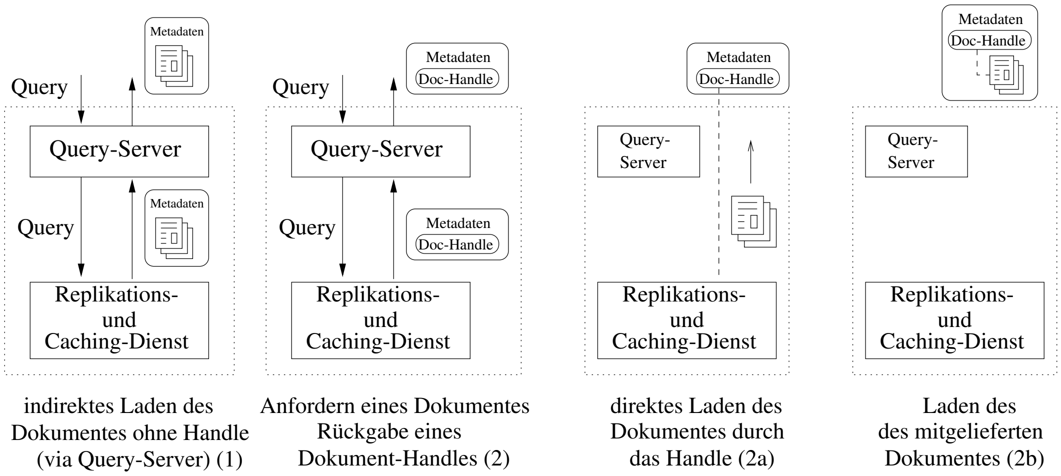

Abbildung 3.5: Laden eines Dokument-Objektes

gabe (1). Dieser Ansatz besitzt den Nachteil, daß sehr umfangreiche Dokumente beim Transport Probleme hervorrufen können, da von jedem der beteiligten Dienste (je nach Implementierung) entsprechender Speicherplatz bereitgestellt werden muß.[38] Weiterhin werden bei dieser Vorgehensweise Dienste (in diesem Fall der

[38]Wenn davon ausgegangen wird, daß Dokumente nicht als ein kontinuierlicher Datenstrom übertragen werden.

Query-Server) beansprucht, die selbst nur eine Weiterleitungsfunktion besitzen.

Im zweiten Ansatz wird ein Handle eingeführt, bei dem auf eine Dokument-Retrieval-Anfrage nicht das Dokument selbst, sondern ein Handle zurückgeliefert wird (2), dessen Methoden zum Laden des Dokumentes genutzt werden. Wie diese Methoden das Dokument laden, kann in der Implementierung anhand bestimmter Kriterien (wie beispielsweise der Größe des Dokumentes) entschieden werden. Handelt es sich um ein großes Dokument, so ist das direkte Laden (2a) praktischer, bei dem mit Hilfe des Handles der Inhalt des Dokumentes beim Replikations- und Caching-Dienst direkt angefordert wird, ohne weitere VDS-Dienste in Anspruch zu nehmen. Die Inhalte relativ kleiner Dokumente können auch als Bestandteil des Handles zurückgeliefert werden, so daß die Methoden auf die lokal vorhandenen Daten zugreifen können (2b). Vor dem Nutzer bleibt die tatsächlich verwendete Vorgehensweise weitestgehend verborgen, da er die Implementierung der Handle-Funktionen nicht kennt.

Ein einfaches Handle stellt nur eine Methode bereit, die den Inhalt des Dokumentes anfordert:

```
interface DocumentHandle
 {
  // die Bytes, aus denen sich das Dokument zusammensetzt
   typedef sequence<octed> documentContentData;

  // Laden des Dokumentes
   void getContent(out documentContentData documentContent)
    raises (miscHandleException);
 };
```

Das Handle kann auch weitere Methoden anbieten, die beispielsweise nur die Teile des Dokumentes laden, die der Nutzer spezifiziert hat. In einem konkreten Beispiel könnte dies bedeuten, daß ein Nutzer nicht ein ganzes Buch übertragen bekommt, sondern nur ein Kapitel, das für ihn von Interesse ist.

Insgesamt betrachtet bietet der Handle-Mechanismus eine sehr große Flexibilität, die Spielraum für Erweiterungen läßt aber auch den Anforderungen gerecht wird, die aus dem Umfang eines Dokumentes resultieren. In diesem Abschnitt wurde bewußt nur eine vereinfachte Darstellung des sich anbietenden Handle-Konzeptes gegeben. Weitere Betrachtungen sind in Bezug auf den Zugriff auf die Daten (Welche Schnittstelle des Replikations- und Caching-Dienstes nutzt das Handle um die Daten zu laden?), die Sicherheit (Wie wird sichergestellt, daß nur derjenige das Handle benutzen kann, der das Dokument angefordert hat? Dies ist insbesondere dann von Bedeutung, wenn eine Autorisierung für den Zugriff auf das Dokument durchgeführt wurde.) und auf die Lebensdauer eines Handles (Wie lange ist ein Handle gültig?) erforderlich.

Beschreibung der Präferenzen einer Anfrage. Die Metadaten-Retrieval-Anfrage eines Nutzers setzt sich, wie bereits in der Beschreibung des Query-Interfaces erwähnt, aus der eigentlichen Anfrage und Präferenzen zusammen. In der Anfrage spezifiziert der Nutzer, in welchen Ressourcen nach welchen Kriterien gesucht werden soll. Präferenzen erweitern die Anfrage dahingehend, daß standardmäßige Festlegungen zu Struktur und Umfang der Ergebnisliste, zur Selektion von zu kontaktierenden Ressourcen u.ä. getroffen werden. Diese Festlegungen gelten immer dann, wenn in der Anfrage keine gegenteiligen Angaben gemacht wurden. Die für die Anfragebearbeitung wichtigen Präferenzen dienen

- der Beschränkung der Ergebnismenge und
- der Beschreibung der Struktur der Ergebnisse.

Die Ergebnismenge kann durch die Festlegung einer maximalen Ergebnistupelanzahl und die Vorgabe einer Extension eingeschränkt werden. Letzteres bedeutet, daß die Suche nur auf ausgewählten Ressourcen durchgeführt wird. Eine daraus resultierende Konsequenz ist, daß das Nutzersystem beim Query-Server Informationen über die verfügbaren Ressourcen anfordern muß, um sie dem Nutzer zur Selektion präsentieren zu können. Sinnvoll kann deshalb beispielsweise die Festlegung sein, daß standardmäßig alle lokalen LDS in die Suche einbezogen werden.

Präferenzen, die die Struktur der Ergebnisse beschreiben, können entweder durch das Nutzersystem (für den Nutzer nicht sichtbar) direkt in die IRQL-Anfrage integriert werden, die an den Query-Server übermittelt wird, oder getrennt von der IRQL-Anfrage übermittelt werden. Im letzteren Fall ist es Aufgabe des Query-Servers, die endgültige IRQL-Anfrage zu ermitteln.
Ein kurzes Beispiel verdeutlicht den letzteren Fall:

```
// eine Liste von Zeichenketten
 typedef sequence<string> listOfString;

struct preferencetype
 {
  // maximale Anzahl von Ergebnistupeln
   short maxResults;
  // Attribute, aus denen sich ein Ergebnistupel zusammensetzt
   listOfString defaultAttributeList;
  // LDS, die bei einer Anfrage kontaktiert werden sollen
   listOfString defaultResourceList;
 };
```

Angewendet auf einen konkreten Anwendungsfall könnten die Präferenzen

- maxResults=20,

- defaultAttributeList={Relevance, Title} und
- defaultResourceList={local}

dem vom Nutzer spezifizierten Anfrageterm *Student* zugeordnet werden. Aus dem Anfrageterm und den Präferenzen kann die folgende IRQL-Anfrage zusammengesetzt werden:

```
SELECT Relevance, Title, DocId
FROM   local
WHERE  complete_content CONTAINS "Student"
RANK BY hitscount
LIMIT TO 20
```

Beim Vergleich der Vorgaben für die zurückzuliefernde Attributliste und der daraus generierten SELECT-Klausel der IRQL-Anfrage fällt auf, daß ein zusätzliches Attribut `DocId` eingefügt wurde. Dieses Attribut wird standardmäßig in jedem Anfrageergebnis zurückgeliefert, um dem Nutzer die Möglichkeit zu geben, die in der Ergebnisliste referenzierten Dokumente anzufordern. Präferenzen, die nicht mit Hilfe der IRQL ausgedrückt werden können, werden als zusätzlicher Parameter an den Wrapper übergeben (siehe Abschnitt 3.2.1.3). Im folgenden Abschnitt wird beschrieben, wie die Ergebnisliste strukturiert ist.

Modellierung der Ergebnisliste. Die LDS eines VDS ermitteln anhand der ihnen übermittelten Anfrage und Präferenzen eine geordnete Liste von Ergebnistupeln, die diese erfüllen. Die hier vorgenommenen Betrachtungen umfassen sowohl die Struktur der einzelnen Ergebnistupel als auch die Modellierung der Liste.

Die Wrapper der LDS generieren eine in der Struktur einheitliche Ergebnisliste, deren Tupel sich aus den folgenden Bestandteilen zusammensetzen:

- den Attributen (bzw. Funktionen und arithmetischen Operationen) der Projektionsliste
- einem Schlüssel zum Zugriff auf das Dokument
- Informationen zum lokalen Ranking

Die Attribute des Selektionsprädikates sind die Dokument-Metadaten (Titel, Abstrakt usw.), die der Nutzer in seiner Anfrage an den VDS spezifiziert hat. Der Schlüssel zum Zugriff auf das Dokument, der DOI, gestattet dem Nutzer das Auslösen einer Dokument-Retrieval-Anfrage zum Laden eines Dokumentes der Ergebnisliste. Der dritte, für die Nachbearbeitung der Anfrage wesentlichste Teil der Ergebnisliste, setzt sich aus dem lokalen Ranking-Wert des Dokumentes sowie weiteren statistischen Daten[39] zusammen, die vom Query-Server zur Bestimmung eines globalen Rankings verwendet werden können.

Eine Festlegung, welche Informationen neben dem Ranking-Wert durch die LDS bzw. die kontaktierten externen VDS zu jedem Ergebnistupel zur Bestimmung eines globalen Rankings bereitgestellt werden müssen, soll in dieser Arbeit nicht getroffen werden, da die Problematik der Bestimmung eines globalen Rankings Bestandteil anderweitiger aktueller Forschungen ist. An dieser Stelle sei auf [GCGMP97] verwiesen, die Probleme bei der Ermittlung eines globalen Rankings aufzeigen und gleichzeitig einen Lösungsansatz vorstellen.

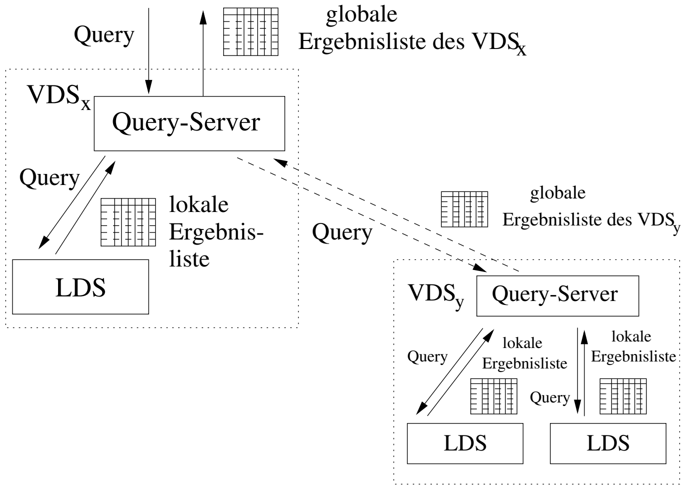

Abbildung 3.6: Mischen der Ergebnislisten

Wenn mehr als ein lokaler oder externer LDS zur Anfragebearbeitung hinzugezogen wurde, ist eine Nachbearbeitung der Ergebnislisten durch den Query-Server erforderlich, da die Ergebnisse zusammengemischt und eventuell verknüpft[40] werden müssen (Abbildung 3.6). Jeder der an der Anfragebearbeitung beteiligten VDS ermittelt für die von ihm kontaktierten LDS eine globale Ergebnisliste, die an den aufrufenden VDS zurückgeliefert wird.

Zugriff auf die Ergebnisliste. Der Umfang der Liste, die von den LDS an den jeweils lokalen Query-Server zurückgeliefert werden muß, kann sich von keinem Ergebnistupel bis hin zu vielen tausend Ergebnistupeln erstrecken. Das Volumen der dabei zu übertragenden Daten kann im schlechtesten Fall die Ressourcen des Query-Servers weit überfordern, so daß eine Übertragung aller Ergebnistupel zu einem Zeitpunkt keinen geeigneten Ansatz darstellt. Stattdessen könnte

[39] Dabei kann es sich um Worthäufigkeiten, Wortverteilungen und andere statistische Informationen zum Dokument und zum Datenbestand handeln, die von der Datenquelle bereitgestellt werden.

[40] Die Verknüpfung von Ergebnissen durch den Query-Server ist erforderlich, wenn die IRQL-Anfrage einen JOIN-Operator enthält.

ein Handle-Mechanismus verwendet werden, bei dem nicht die Ergebnisse selbst, sondern eine Möglichkeit zum Zugriff auf die Ergebnisse zurückgeliefert wird. Dadurch kann ein Query-Server nach Bedarf jedes einzelne Ergebnistupel (bzw. Teile des Tupels) anfordern und die darauf erforderlichen Operationen der Nachbearbeitung durchführen. Der Vorteil dieser Vorgehensweise ist, daß durch den Query-Server selbst auf umfangreiche Ergebnislisten bzw. Ergebnistupel problemlos zugegriffen werden kann.

In Abbildung 3.7 wird beispielhaft die Erzeugung der Ergebnisliste und die Verwendung eines Ergebnis-Handles dargestellt.

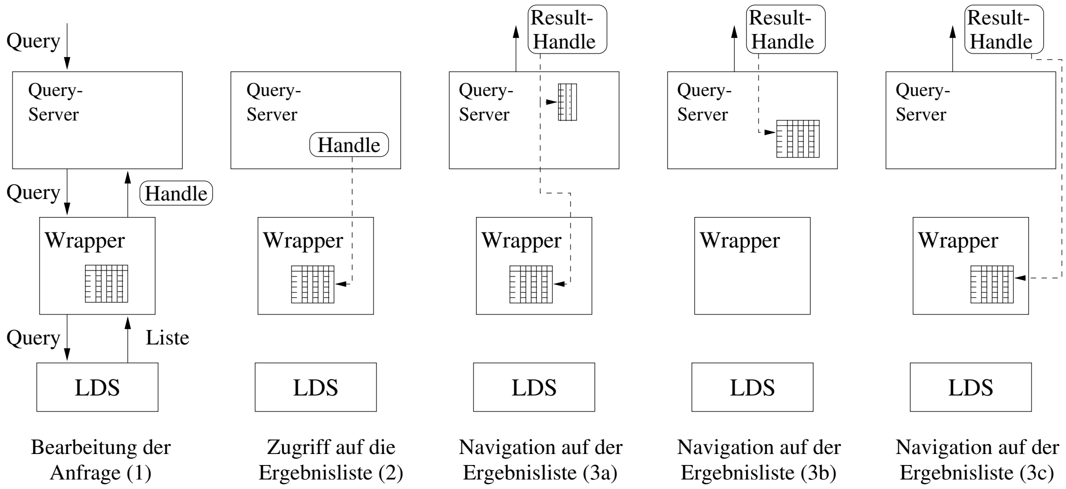

Abbildung 3.7: Zugriff auf die Ergebnisliste mittels Handle

Der Query-Server sendet eine Metadaten-Retrieval-Anfrage an den Wrapper des LDS, der die Anfrage umformt und durch den LDS bearbeiten läßt. Dieser liefert eine Ergebnisliste zurück. Dieser Ergebnisliste wird durch den Wrapper ein Handle zugeordnet, das dem Query-Server zurückgeliefert wird (1). Dieses Handle kann der Query-Server nutzen, um auf die Tupel der Ergebnisliste zuzugreifen (2), um beispielsweise die Nachbearbeitung der Ergebnisse durchführen zu können. Der Query-Server erzeugt im Verlauf der Nachbearbeitung ein globales Handle, das an den Aufrufer zurückgeliefert wird. Dieses Handle kann der Aufrufer verwenden, um auf die globalen Ergebnisse des VDS zuzugreifen (3a-c). Vor dem Aufrufer bleibt dabei verborgen, wie auf die Ergebnisdaten zugegriffen wird. In Abhängigkeit vom Umfang der zurückgelieferten Ergebnisse und gewählten Implementierungsentscheidungen kann der Zugriff auf verschiedenen Wegen erfolgen. Drei davon sind in Abbildung 3.7 kurz skizziert:

- In (3a) wird ein Teil der Attribute der Ergebnistupel vom Query-Server für den Zugriff zwischengespeichert. Attribute, deren Inhalt sehr groß ist, können im Speicher des Wrappers (bzw. alternativ im LDS) verbleiben. Auf die im Query-Server und im LDS zwischengespeicherten Attribute greift der

Aufrufer unter Zuhilfenahme des Handles zu. Diese Vorgehensweise bietet den Vorteil, daß der Query-Server nur Attribute vom LDS anfordern muß, die er für die Nachbearbeitung der Anfrage benötigt (bspw. bei der Verknüpfung von Teilergebnissen mittels JOIN-Operator). Alle anderen Attribute verbleiben im Wrapper bzw. LDS und werden vom Aufrufer durch den Zugriff auf das Handle angefordert. Dadurch werden die vom Query-Server benötigten Ressourcen (Hauptspeicher) reduziert.

- Ist der zur Zwischenspeicherung der Ergebnisliste notwendige Ressourcenbedarf relativ gering, könnte diese auch komplett im Query-Server in einer Ergebnismengenverwaltung zwischengespeichert werden (3b). Damit werden wiederholte Zugriffe auf die Wrapper vermieden.
- Die Ergebnistupel können auch komplett auf dem Wrapper verbleiben, so daß der Aufrufer der Anfragebearbeitung diese direkt anfordern kann (3c). Durch diese Vorgehensweise muß der Query-Server keine Ergebnismengenverwaltung durchführen, da diese durch den Wrapper erfolgt. Dieser Ansatz besitzt den Vorteil, daß sich der Query-Server nach der Bearbeitung der Anfrage nicht mehr um den Zugriff auf die Ergebnisse kümmern muß.

Jeder der skizzierten Ansätze kann in bestimmten Anwendungsszenarien seine Stärken ausspielen, so daß eine allgemeingültige Entscheidung für nur einen der Ansätze nicht getroffen werden kann. Deshalb sollte durch den Query-Server anhand von Daten über das Ergebnisvolumen (bspw. Anzahl der Ergebnisse, Größe der Tupel) die für den Anwendungsfall jeweils günstigste Variante ausgewählt werden. Weitere Variationen zur Realisierung der Handles können entstehen, wenn nicht der Wrapper direkt die Ergebnislisten zwischenspeichert, sondern ein dafür dedizierter Dienst.

Die Einführung eines Handles zum schrittweisen Zugriff bietet Vorteile bei der Verwaltung von Ergebnismengen, die in Bezug auf den Umfang (Summe der Größe aller Daten, die in der Liste zusammengefaßt sind) problematisch sind. Neben diesem positiven Effekt steigt allerdings auch die Komplexität des Query-Servers, da je nach verwendetem Ansatz eine Ergebnismengenverwaltung implementiert und Schnittstellen zum Zugriff vorgesehen werden müssen. Ein dabei bestehendes Problem stellt die Lebensdauer der Ergebnisse dar, da diese nicht zeitlich unbeschränkt gespeichert werden können. Wie die Ergebnismengenverwaltung und die Handles zum Zugriff auf die Ergebnisse in der Praxis umgesetzt werden können, soll in diesem Kapitel nicht weiter betrachtet werden. In Anhang A auf Seite 140 ist eine einfache Schnittstellenbeschreibung eines Ergebnislisten-Handles spezifiziert.

Strukturierung der Metadaten. In der Schnittstellenbeschreibung des Föderationsdienstes wurden keine Vorschläge zu Umfang und Struktur der zurückgelieferten Metadaten gegeben. Deshalb werden an dieser Stelle einige kurze Denkansätze vorgestellt.

Die statischen Metadaten einer Datenquelle umfassen Informationen zur Modellierung der komplex strukturierten Dokumente. Diese Informationen, die sich aus dem/den Schema(ta) der DBMS entnehmen lassen, setzen sich aus Attributnamen und Attributtypen zusammen. Dabei kann ein Attributtyp nicht nur einfache sondern auch komplexe Datentypen beschreiben, da es sich bei den Datenquellen um Objektrelationale DBMS handeln kann. Neben den Attributnamen sind gegebenenfalls weitere Informationen zur semantischen Bedeutung des Attributes erforderlich, da eine eindeutige Ableitung der Semantik aus dem Namen nicht möglich ist. Die semantischen Informationen lassen sich nicht aus dem Datenbankschema entnehmen, sondern müssen auf andere Art und Weise durch den Datenbankadministrator bereitgestellt werden. Die statischen Metadaten lassen sich demnach beispielsweise in der folgenden einfachen Form ausdrücken, wobei geeignete Darstellungsformen zu finden sind:

$$\{\text{Attributname, Attributtyp, informale semantische Informationen}\}$$

Dynamische Metadaten beschreiben den Datenbestand der Datenquelle. Dazu gehören die Anzahl der gespeicherten Dokumente und IR-spezifische Daten, sofern die Datenquelle IR-Anfragen unterstützt. Sie können teilweise durch Anfragen an Systemtabellen bzw. die Tabellen, in denen die Dokumente gespeichert sind, ermittelt werden. Es kann allerdings auch erforderlich sein, aufwendigere Analysen durchzuführen, wenn diese Daten nicht frei verfügbar sind bzw. keine Schnittstelle zur Abfrage bereitgestellt wird. Für die dynamischen Metadaten muß eine geeignete Auswahl derjenigen Daten getroffen werden, die durch die Datenquellen bereitzustellen sind. Diesen Daten sind dann eindeutige Bezeichner zuzuordnen, so daß eine automatische Verarbeitung der durch den Wrapper aufbereiteten Metadaten erfolgen kann.

Die vorhergehenden Feststellungen sind nur dann von Bedeutung, wenn die Datenquellen kooperativ sind und ihre Metadaten nach außen bereitstellen. Externe Literatur-Recherche-Dienste, die durch den Query-Server wie ein LDS angebunden werden, sind in der Regel als unkooperativ anzusehen, da sie ihre Metadaten nicht nach außen bereitstellen. Dementsprechend schwer (bzw. unmöglich) ist die qualitative Einordnung von solchen Systemen, die bei der auf den Metadaten basierenden Selektion der zu kontaktierenden LDS durch den Broker erfolgt. Es liegt in der Verantwortung der Implementierer des Wrappers diese Metadaten zu ermitteln (bspw. anhand von Tests) und sie durch den Wrapper bereitstellen zu lassen.

Abschließende Bemerkungen

In den vergangenen Abschnitten wurden die externen Dienstschnittstellen des Query-Servers beschrieben, die sich aus den Anforderungen in Kapitel 2 ergaben. Offen geblieben sind die Anforderungen, die sich durch die Kommunikation mit dem Trader ergeben. Dazu gehören die Anmeldung des VDS und die Bereitstellung der Bindungsinformation. Eine Spezifikation der dazu notwendigen Schnittstellen wird in dieser Arbeit nicht erfolgen, da zu wenig Informationen zu diesem Dienst vorliegen.

Nicht spezifiziert und erläutert wurden interne Vorgänge bei der Analyse und der Nachbearbeitung der Anfrage, da diese für die Beschreibung der Dienstschnittstelle des Query-Servers nicht von Bedeutung sind. Auf jeden Fall sind weitere Untersuchungen zu dieser Thematik erforderlich, da sie den Kern des Query-Servers bildet. Aus diesem Grund sei an dieser Stelle auf weiterführende Arbeiten verwiesen, die sich aus den Erkenntnissen der vorliegenden Arbeit ergeben.

Damit ist Schwerpunkt eins der in der Zusammenfassung des vorhergehenden Kapitels (Abschnitt 2.8) ermittelten drei Schwerpunkte bearbeitet. Der zweite Schwerpunkt, der sich aus den LDS und deren Wrappern zusammensetzt, wird teilweise im folgenden Abschnitt behandelt.

3.2.2 Lokaler Dokumenten-Server

Die lokalen VDS-Dienste greifen via Wrapper auf die Lokalen Dokumenten-Server zu. Für diese Dienste sind somit lediglich die Schnittstellen der Wrapper von Bedeutung, da diese den Zugriff auf die LDS realisieren. Die Schnittstellen, die die Wrapper nutzen um die Datenquelle anzusprechen, sind von den darunterliegenden Datenquellen (DBMS, Literatur-Recherche-Dienst) abhängig. Aus diesem Grund kann erst im Kapitel 5, in dem die konkreten Systeme vorgestellt werden, eine Beschreibung der dazu verwendeten Schnittstellen erfolgen.

Entsprechend der in Abschnitt 2.2 ermittelten Anforderungen besitzt jeder Wrapper Methoden

- zur Dokument-Retrieval-Anfragebearbeitung,
- zur Metadaten-Retrieval-Anfragebearbeitung und
- zum Zugriff auf die statischen und dynamischen Metadaten der jeweiligen Datenquelle.

Neben diesen Methoden besitzt jeder Wrapper eine weitere Methode, deren Notwendigkeit sich aus dem Architekturentwurf des Query-Servers ergibt. Diese Methode realisiert eine Servicefunktion, die testet, ob ein LDS ansprechbar ist oder nicht. In Anhang A.2 sind die Spezifikationen der nachfolgend beschriebenen Methoden zusammengefaßt.

3.2.2.1 Dokument-Retrieval-Anfragen

Um ein Dokument seines Datenbestandes aufgrund einer Dokument-Retrieval-Anfrage ausliefern zu können, benötigt der LDS einen eindeutigen Schlüssel, anhand dessen er das Dokument identifizieren kann. Die Methode

```
void getDocument(in string documentIdentifier,
                 in data options,
                 out data document,
                 out data requiredAuthorizationInformation)
     raises (documentNotFoundException,
             wrapperException,
             sourceException);
```

liefert das Dokument, dem der eindeutige `documentIdentifier` zugeordnet ist. Dabei können durch den Parameter `options` weitere Festlegungen zum Retrieval getroffen werden.[41] Der Wrapper formt die Dokument-Retrieval-Anfrage in die Anfragesprache der Datenquelle um, sendet die Anfrage an die Datenquelle und fordert weitere Metainformationen zum Dokument an. Diese Metainformationen bilden zusammen mit dem eigentlichen Dokument das zurückgelieferte Dokument-Objekt `document`. An dieser Stelle soll offen bleiben, wie die Modellierung des Dokument-Objektes erfolgen soll. In den Betrachtungen zur "Modellierung eines Dokumentes im Query-Server" in Abschnitt 3.2.1.6 wurde bereits ein Vorschlag angegeben, der an dieser Stelle nicht ausreichend erscheint, da der Replikations- und Caching-Dienst eventuell weitere Informationen zum Dokument benötigt, die jedoch nicht an den Query-Server weitergegeben werden. Aus diesem Grund bildet die hier skizzierte Schnittstelle lediglich einen ersten Ansatz, der durch die Anforderungen des Replikations- und Caching-Dienstes revidiert werden kann. Neben dem eigentlichen Dokument müssen weitere Informationen (`requiredAuthorizationInformation`) zum Dokument zurückgeliefert werden, die gegebenenfalls die Autorisierungspflicht beim Zugriff auf das Dokument signalisieren. Wenn ein Dokument eine Autorisierung erfordert, wird diese durch den Replikations- und Caching-Dienst vor der Auslieferung durchgeführt.

Wenn in der Datenquelle kein Dokument mit dem angegebenen DOI gespeichert ist, löst die Methode eine `documentNotFoundException` aus. Ist bei der Anfragebearbeitung durch den Wrapper bzw. die Datenquelle ein Fehler aufgetreten, wird dies durch das Auslösen der `wrapperException` bzw. der `sourceException` kenntlich gemacht.

3.2.2.2 Metadaten-Retrieval-Anfragen

Im Verlauf der Metadaten-Retrieval-Anfragebearbeitung kontaktiert der Query-Server mit Hilfe der bereitgestellten Wrapper die aus der Anfrage ermittelten lokalen Datenquellen. Dazu nutzt der Query-Server das in Abschnitt 3.2.1.3 beschriebene Ressourcen-Interface, das wiederum die Methode

```
void documentMetaDataQuery(in string query,
```

[41] Siehe dazu die Beschreibung der Schnittstelle zum Replikations- und Caching-Dienst des Query-Servers in Abschnitt 3.2.1.1.

```
                              in data preferences,
                              out data resultList)
     raises (wrapperException,
             sourceException);
```

des Wrappers aufruft, die die Anfragebearbeitung im Wrapper auslöst.

Die Parameter der Methode setzen sich aus der IRQL-Anfrage `query` und den dazugehörigen Präferenzen `preferences` zusammen.[42] Wenn die Anfrageübersetzung, die Anfrageausführung durch die zugrunde liegende Datenquelle und die Anfragenachbearbeitung durch den Wrapper erfolgreich abgeschlossen werden konnten, liefert die Methode eine Liste von Ergebnistupeln in der vom Query-Server geforderten Form zurück.[43] Im Fehlerfall wird in Abhängigkeit vom Ort des Auftretens (Wrapper oder Datenquelle) eine Ausnahme (`wrapperException`*, `sourceException`*) ausgelöst. Die Ausnahmen werden vom Ressourcen-Interface des Query-Servers ausgewertet und lösen in diesem gegebenenfalls eine allgemeine `miscWrapperException`* aus.

3.2.2.3 Export der statischen und dynamischen Metadaten

Lokale Dokumenten-Server exportieren auf Anfrage ihren statischen und dynamischen Metadatenbestand an den Query-Server. Dazu werden bei der Implementierung jedes Wrappers eine Menge von Operationen (Anfragen o.ä.) festgelegt, die bei Aufruf der Methode ausgeführt werden. Die Operationen ermitteln auf geeignete Weise die gewünschten Metadaten. Diese werden durch den Wrapper in eine (für den VDS) einheitliche Form umgeformt und zurückgeliefert. Verfügt die Datenquelle über keine Schnittstelle zum Abfragen der Metadaten, ist eine geeignete Alternative durch den Wrapper-Implementierer zu finden. Die statischen Metadaten der Datenquelle werden vom Wrapper durch Aufruf der Methode

```
void getResourceStaticMetaData(out data metaData)
     raises (wrapperException,
             sourceException);
```

und die dynamischen Metadaten der Datenquelle durch Aufruf der Methode

```
void getResourceDynamicMetaData(out data metaData)
     raises (wrapperException,
             sourceException);
```

angefordert. Beide Methoden besitzen keine Parameter und liefern bei erfolgreicher Anfragebearbeitung alle statischen bzw. alle dynamischen Metadaten der

[42] Siehe dazu “Beschreibung der Präferenzen einer Anfrage“ in Abschnitt 3.2.1.6.
[43] Siehe dazu “Modellierung der Ergebnisliste“ in Abschnitt 3.2.1.6.

Datenquelle in einer hier nicht näher spezifizierten Form zurück.[44] Diese Metadaten werden vom Query-Server zwischengespeichert und auf Anfrage an die externen Dienste (Föderationsdienst, Broker) weitergeleitet.

Im Fehlerfall kann in Abhängigkeit vom Ort des Auftretens (Wrapper oder Datenquelle) eine `wrapperException`* bzw. `sourceException`* ausgelöst werden.

In den Betrachtungen zum Query-Server wurde davon ausgegangen, daß dieser das Laden der Metadaten vornimmt, um zu überprüfen, ob sich die Metadaten der LDS geändert haben. Dieser aktive Ansatz könnte durch einen passiven Ansatz ersetzt werden, wenn diese Aufgabe an die Wrapper des LDS delegiert wird. Diese würden bei Bedarf den Query-Server informieren, der daraufhin die geänderten Metadaten der betroffenen LDS anfordern kann. Der Vollständigkeit[45] halber sei für diesen Ansatz die Methode

```
void notify()
     raises (miscNotifyException);
```

eingeführt, die keine Parameter besitzt, keine Ergebnisse zurückliefert und im Fehlerfall eine `miscNotifyException`* auslöst. Durch den Aufruf dieser Methode wird der Query-Server veranlaßt, die Metadaten des LDS neu einzulesen.

Die Verwendung dieser Methode wirft ein Problem auf. Die Kommunikation zwischen Query-Server und LDS wurde in den bisherigen Schnittstellenbeschreibungen immer durch den Query-Server initiiert, so daß der LDS praktisch nicht wissen mußte, welchem Query-Server (und damit VDS) er zugeordnet wurde und wie er ihn kontaktieren kann. Um die hier beschriebene Methode nutzen zu können, ist eine beidseitige Kontaktierung der Dienste untereinander erforderlich. Aus diesem Grund wird dieser Ansatz in der vorliegenden Arbeit nicht weiter verfolgt, sondern kann als Erweiterung des ersten Entwurfes angesehen werden.

3.2.2.4 Verbindungstest

Bei der Neuanmeldung einer Ressource bzw. bei der Wartung der Ressourcen-Liste durch die Ressourcen-Verwaltung des Query-Servers wird die Methode `ping` des Ressourcen-Interfaces (siehe Abschnitt 3.2.1.3) genutzt, um zu testen, ob eine Verbindung zum Wrapper aufgebaut werden kann. Diese Methode ruft die Methode

```
void pong(out boolean alive);
```

des Wrappers auf, die keine Parameter übergeben bekommt und als Ergebnis einen booleschen Wert zurückliefert, der Auskunft über die Erreichbarkeit der

[44] Siehe Betrachtungen zur “Strukturierung der Metadaten“ in Abschnitt 3.2.1.6.

[45] Im Anhang wird diese Methode nicht formal spezifiziert, da zur Zeit eine Umsetzung nicht erfolgen soll.

darunterliegenden Datenquelle gibt. Die Erreichbarkeit des Wrappers wird bereits durch den erfolgreichen Aufruf der Methode festgestellt, denn bei einem gescheiterten Verbindungsaufbau kann die Methode nicht aufgerufen werden.

Abschließende Bemerkungen

In den vorherigen Abschnitten wurde ein Teil der dem zweiten Schwerpunkt aus Abschnitt 2.8 zugeordneten Schnittstellen beschrieben. Diese umfassen im Wesentlichen die vom Query-Server benötigten Methoden zum Zugriff auf die Wrapper der LDS. Weitere Schnittstellen können sich aus der Zusammenarbeit mit dem Replikations- und Caching-Dienst ergeben, da dieser einen dedizierten LDS als persistenten Speicher für replizierte Dokumente verwenden kann. Aus diesem Grund sei für weitere Wrapper-Schnittstellenbeschreibungen auf die laufenden Untersuchungen zum Replikations- und Caching-Dienst in [Jon99] verwiesen.

3.2.3 Replikations- und Caching-Dienst

Der dritte und letzte Schwerpunkt aus Abschnitt 2.8 umfaßt die Schnittstellen des Replikations- und Caching-Dienstes. Dieser Schwerpunkt wird, wie bereits mehrfach dargelegt, im Rahmen einer anderen Diplomarbeit [Jon99] bearbeitet.

In den Ausführungen zur Schnittstelle des Query-Servers zum Zugriff auf den Replikations- und Caching-Dienst in Abschnitt 3.2.1.1 wurde bereits erläutert, daß eine Schnittstelle erforderlich ist, die das externe Dokument-Retrieval auslöst und eine Schnittstelle für Verwaltungsaufgaben der LDS benötigt wird.

In den Schnittstellenbeschreibungen des LDS im vorherigen Abschnitt wurde eine weitere Schnittstelle skizziert, die den Zugriff des Replikations- und Caching-Dienstes auf den Wrapper realisiert.[46]

Bisher wurde nicht berücksichtigt, daß ein Replikations-Dienst einen eigenen LDS nutzen kann, der Replikate von extern gespeicherten Dokumenten verwaltet. Der Wrapper dieses LDS muß dementsprechend Schnittstellen zum schreibenden Zugriff bereitstellen.

Weitere Dienstschnittstellen ergeben sich aus der Zusammenarbeit der Replikations-Dienste, die sich gegenseitig über Änderungen informieren müssen. Der im Replikations- und Caching-Dienst verankerte Autorisierungsdienst benötigt ebenfalls eine externe Schnittstelle, mit der eine Kommunikation mit anderen Autorisierungsdiensten erfolgen kann.

3.3 Zusammenfassung

In diesem Kapitel wurde die Architektur des VDS aus den in Kapitel 2 spezifizierten Anforderungen abgeleitet und die Dienstschnittstellen der VDS-Dienste

[46] Die dort beschriebene Schnittstelle muß gegebenenfalls verfeinert werden.

beschrieben. Dabei war es nicht immer möglich, endgültige Festlegungen zu treffen, da weitere Untersuchungen zu Teilproblemen (Autorisierung, Metadaten) erforderlich sind. Diese Untersuchungen konnten nicht im Rahmen dieser Arbeit erfolgen, da die dafür benötigten Grundlagen erst zu einem späteren Zeitpunkt durch externe Projektpartner gelegt werden.
Mit dem Abschluß dieses Kapitels bleiben noch viele Fragen bezüglich der internen Funktionalität der VDS-Dienste offen:

- Query-Server:
  - Wie erfolgt die konkrete Anfragevorbearbeitung und die Anfragenachbearbeitung durch den Query-Server?
  - Wie erfolgt die Verwaltung der Metadaten?
- LDS/Wrapper:
  - Wie erfolgt die Übersetzung der IRQL-Anfrage in die Anfragesprache der Datenquelle?
- Replikations- und Caching-Dienst:
  - Wie ist er aufgebaut und wie wird das Retrieval der Dokumente realisiert? Welche weiteren Funktionen sind dazu zu erfüllen?

Alle diese Fragen können und sollen nicht in der vorliegenden Arbeit beantwortet werden, da sie lediglich die Basis für weitere Untersuchungen bildet.

Im folgenden Kapitel werden einige Grundlagen für die Beantwortung der Frage zur Übersetzung der IRQL-Anfrage geschaffen. Dazu werden die Anfragesprachen der anzubindenden Systeme vorgestellt und mit der IRQL verglichen. Aus dem Vergleich lassen sich bereits einige Aussagen darüber ableiten, ob für die Konstrukte der IRQL eine Abbildung auf die Anfragesprache der Datenquelle angegeben werden kann bzw. welche Konstrukte (wenn möglich) durch den Wrapper zu simulieren sind.

Kapitel 4

Die Anfragesprache IRQL

Um Anfragen an die in einer Digitalen Bibliothek gespeicherten Dokumente bzw. deren Metadaten stellen zu können, wird eine Anfragesprache benötigt, die Anfragen auf den strukturiert und unstrukturiert bzw. semistrukturiert vorliegenden Teilen des Dokumentes durchführen kann. Dies ist deshalb erforderlich, da nicht für alle Dokumente eine einheitliche Struktur angegeben werden kann und somit zwangsläufig Teile eines Dokumentes unstrukturiert in der Datenbank abgespeichert werden müssen.

Aus diesem Grund ist eine Verknüpfung der Konzepte von Datenbankanfragen (Attributierung, Verknüpfung, Restrukturierung, typspezifische Operatoren) und Information-Retrieval-Anfragen (inhaltsbasierte Suche, Ranking, Relevance Feedback) erforderlich. Dies wird bei der Entwicklung der Anfragesprache IRQL[1] [HP99] angestrebt. Durch die Verknüpfung der Vorteile der DB- und IR-Anfrage-Konzepte entsteht eine mächtige Sprache, die für die Problematik der Anfragen an Digitale Bibliotheken sehr gut geeignet ist.

Da sich die Entwicklung der IRQL in einem sehr frühen Stadium befindet, können in Abschnitt 4.1 lediglich einige allgemeine Betrachtungen zur IRQL vorgenommen werden. Im Anschluß werden in Abschnitt 4.2 die Anfragesprachen der Systeme vorgestellt, die in der Implementierung des ersten Prototypen als LDS verwendet werden, und mit der IRQL verglichen. Daraus können Erkenntnisse gewonnen werden, auf denen die Anfrageübersetzung des Wrappers aus der IRQL in die konkrete Anfragesprache des jeweiligen Systems aufbaut. Die Zusammenfassung in Abschnitt 4.3 schließt dieses Kapitel ab.

Bei den betrachteten Systemen handelt es sich um das Information-Retrieval System *Fulcrum SearchServer* (Abschnitt 4.2.1), das objektrelationale Datenbanksystem *DB2* (Abschnitt 4.2.3) und einem auf dem *Harvest*-Broker basierenden System (Abschnitt 4.2.2).

[1] *I*nformation *R*etrieval *Q*uery *L*anguage

4.1 Sprachumfang der IRQL

An dieser Stelle soll keine komplette Beschreibung der IRQL angegeben werden. Vielmehr beschränken sich die folgenden Betrachtungen auf einige wenige (für die Übersetzung der Anfrage durch die Wrapper wichtige) Bestandteile der Anfrage. Es liegt in der Verantwortung der Anfragebearbeitung des Query-Servers, die für die jeweilige Datenquelle relevanten Bestandteile zu ermitteln und an die Wrapper weiterzuleiten. Diese übersetzen die (Teil)-Anfragen in die Anfragesprachen der Datenquellen und liefern die Ergebnisse zurück. Die Anfragebearbeitung des Query-Servers verknüpft die Ergebnisse entsprechend der Anfrage miteinander bzw. führt weitere Operationen durch, die das endgültige Anfrageergebnis bestimmen.

Durch Nutzung der IRQL können Anfragen an strukturiert, semistrukturiert und unstrukturiert vorliegende Dokumente sowie deren Metadaten gestellt werden. Eine IRQL-Anfrage setzt sich aus je einer SELECT-, FROM- und WHERE-Klausel[2] und weiteren optionalen Klauseln (Gruppierung, Sortierung usw.) zusammen. Da beim Entwurf der IRQL Kompatibilität mit dem SQL3-Standard [sql3] angestrebt wird, können in einer Anfrage in jeder SELECT-, FROM- und WHERE-Klausel wieder Anfragen auftauchen; die Anfragesprache ist orthogonal. Jede dieser Unteranfragen kann eine eigene Datenquelle betreffen, durch die die Anfragebearbeitung durchgeführt werden muß.

Durch die WHERE-Klausel werden u.a. die Dokumente selektiert, deren Metadaten bzw. Inhalte als Ergebnis der Anfrage zurückgeliefert werden. Von besonderem Interesse für die hier vorgenommenen Betrachtungen sind dabei die Prädikate, anhand derer die Selektion durchgeführt wird.

In der WHERE-Klausel einer IRQL-Anfrage können die vier nachfolgend klassifizierten Formen von Anfrageprädikaten durch boolesche Operatoren miteinander verknüpft werden. In Klammern wird jeweils darauf hingewiesen, aus welchem Bereich (Datenbanken=DB, Information-Retrieval=IR) das Prädikat ursprünglich entstammt.

Exakte Suche (DB). Unter der exakten Suche sollen die Anfrageprädikate und Operatoren verstanden werden, die mit Hilfe von Punktanfragen (Konstanten-Selektion, Attribut-Selektion), Bereichsanfragen (Bereichsselektion) und Ungewißheitsselektionen gebildet werden. Jede SQL-konforme Anfragesprache muß diese Form von Anfrageprädikaten unterstützen.

Einfache Suche (IR). Die einfachste Form von IR-Anfrageprädikaten bildet das Prädikat, das erfüllt ist, wenn der Suchterm im Text vorkommt.

Proximity-Suche (IR). Bei der Suche kann es manchmal sinnvoll sein, zu fordern, daß Suchterme relativ nahe beieinander im Text liegen, um einen

[2]Die WHERE-Klausel stellt bei Anfragen in der einfachen Form auch einen optionalen Bestandteil dar.

semantischen Zusammenhang zu verdeutlichen. Dabei kann festgelegt werden, ob die Reihenfolge des Auftretens relevant ist oder nicht.

Phonetische Suche (IR). Ziel der phonetischen Suche ist das Auffinden von Begriffen, die einen ähnlichen Klang wie der angegebene Suchterm besitzen. Diese Form der Suche ist dann sinnvoll, wenn die genaue Schreibweise des Suchterms nicht bekannt ist.

Eine wesentliche Eigenschaft von IR-Systemen bildet die qualitative Bewertung der Anfrageergebnisse. Die Qualität der Bewertung hängt vom verwendeten Retrieval-Modell (Boolesches-Retrieval-Modell[3], Vektorraummodell[3], probabilistisches Retrieval-Modell [Fuh97, Kapitel 6]) ab. In der IRQL kann die zu verwendende Ranking-Funktion festgelegt werden, so daß auch bei exakter Suche aufgrund dieser Funktion ein Ranking-Wert ermittelt werden kann (was in DBMS ansonsten nicht der Fall ist).

Weitere Beschreibung der Prädikate

Die IR-Prädikate der WHERE-Klausel einer IRQL-Anfrage setzen sich aus einem obligatorischen und einem optionalen Anteil zusammen. Der optionale Anteil umfaßt die folgenden Klauseln:

Fehlertoleranz. In einigen IRQL-Anfrageprädikaten kann eine Fehlertoleranz angegeben werden, die festlegt, wieviele Vertauschungen bzw. Verdrehungen von Zeichen ein Wort maximal aufweisen darf, um trotzdem das Prädikat zu erfüllen. Wie der Fehlerwert bestimmt wird, hängt vom zugrunde liegenden Algorithmus ab.

Wichtung. Den Suchtermen einer Anfrage kann ein Gewicht zugeordnet werden, das den Einfluß der Suchterme auf den berechneten Ranking-Wert eines Dokumentes mitbestimmt.

Minimum/Maximum. Durch die Festlegung eines Minimums und Maximums kann festgelegt werden, wie oft ein Suchterm mindestens auftreten muß bzw. höchstens auftreten darf, um ein Prädikat zu erfüllen. Dies kann beispielsweise sinnvoll sein, um die Qualität der Anfrageergebnisse zu verbessern.[4]

Case-Sensitivität. Festlegung, ob Groß- und Kleinschreibung des Suchterms und der Wörter im Dokument berücksichtigt werden sollen oder nicht.

[3]Dieses Modell wurde bereits in der vorangehenden Studienarbeit [Tit98] vorgestellt.

[4]Die Beschränkung auf ein Maximum beugt beispielsweise dem Mißbrauch vor, der seit einiger Zeit mit einigen Suchmaschinen betrieben wird. Da diese den Ranking-Wert eines Dokumentes u.a. aufgrund der Häufigkeit des Vorkommens des Suchterms bestimmen, führt die häufige Aufzählung des Begriffes im Dokument (vorzugsweise im Kopfteil) zu einem verzerrten Ranking-Wert.

Teilstring-Suche. Festlegung, ob der Suchterm Teil eines anderen Wortes im Dokument sein darf oder nicht.

Für alle Klauseln können Standardvorgaben existieren, die durch explizite Angabe der Klauseln im Prädikat überschrieben werden. Eine häufig in Suchmaschinen verwendete Vorgabe bildet beispielsweise die Kombination {keine Schreibfehler zugelassen, Gewichtung eins (normal), Minimum eins (kein Maximum), Case-insensitiv und Teilstring-Suche}. Klauseln eines Prädikats können die Ergebnismenge sowohl verkleinern als auch vergrößern. Es besteht natürlich in ungünstigen Fällen auch die Möglichkeit, daß ein Prädikat die Ergebnismenge nicht beeinflußt.

Einige kurze Beispiele sollen das bisher gesagte illustrieren[5], wobei für alle Beispiele die folgenden Standardvorgaben gelten:

- keine Schreibfehler
- Gewicht eins
- Suchterm muß mindestens einmal auftreten, kein Maximum
- Case-insensitive Suche
- keine Teilstringsuche

Ein Beispiel für die einfache Suche stellt die Anfrage

```
SELECT title, docId
FROM local
WHERE abstract CONTAINS "Semester"
       ATLEAST 3
```

dar. Sie liefert Titel und DOI derjenigen Dokumente der Datenquelle *local* zurück, in deren Abstrakt der Begriff *Semester* (in beliebiger Groß- bzw. Kleinschreibung) vorkommt. Die Klausel ATLEAST schränkt die Ergebnismenge dahingehend ein, daß nur Dokumente berücksichtigt werden, in deren Abstrakt der Begriff *Semester* mindestens dreimal auftaucht. Die folgende Proximity-Suchanfrage

```
SELECT title, abstract, docId
FROM local
WHERE abstract CONTAINS "Freizeit"
       BEFORE "Arbeit"
       CASE SENSITIVE
       SUBSTRING
```

[5] Die verwendete Syntax entspricht nicht unbedingt der in der IRQL tatsächlich verwendeten Form.

liefert Titel, Abstrakt und DOI derjenigen Dokumente, in deren Abstrakt der Begriff *Freizeit* vor dem Begriff *Arbeit* auftaucht, wobei die Groß/Kleinschreibung eine Rolle spielt. Durch die Klausel SUBSTRING wird die Ergebnismenge gegenüber einer gleichen Anfrage ohne diese Klausel vergrößert, da beide Begriffe zusätzlich Bestandteil eines anderen Begriffes sein dürfen. Ein Beispiel für die phonetische Suche und die Verknüpfung mehrerer Prädikate stellt die Anfrage

```
SELECT title, author, docId
FROM local
WHERE author SOUNDEX "Meier" AND
      abstract CONTAINS "wissenschaftliche Arbeit"
       CASE SENSITIVE
       ATMOST 2
```

dar. Diese Anfrage liefert Titel, Autor und DOI der Dokumente, deren Autor einen Namen besitzt, der ähnlich wie *Meier* klingt, und in deren Abstrakt die Phrase *wissenschaftliche Arbeit* nicht mehr als zweimal auftritt.

4.2 Vergleich der IRQL ...

In den folgenden Abschnitten werden die Anfragesprachen der Systeme, an die der Query-Server die IRQL-Anfrage weiterleitet, daraufhin untersucht, ob und inwieweit sie die Funktionalität der im vorherigen Abschnitt vorgestellten Prädikate und zugehörigen Klauseln bereitstellen. Davon ausgehend kann ermittelt werden, für welche der Systeme der zugeordnete Wrapper mit vertretbarem Aufwand die fehlende Funktionalität des Prädikates simulieren kann. Das Ergebnis dieser Untersuchungen bildet eine Einordnung nach folgendem Maßstab:

Bewertung $\oplus$: Die Anfragesprache des Systems unterstützt Selektionen anhand dieses Prädikats. Der Wrapper muß nur einen geringen Teil bzw. gar keine der fehlenden Klauseln simulieren.

Bewertung $\otimes$: Die Anfragesprache des Systems unterstützt Selektionen anhand dieses Prädikats nur eingeschränkt. Durch den Wrapper muß dieser Teil der fehlenden Klauseln simuliert werden.

Bewertung $\ominus$: Die Anfragesprache des Systems unterstützt keine Selektionen anhand dieses Prädikats. Das Prädikat und eventuell auftretende Klauseln müssen durch den Wrapper simuliert werden.

In dieser Einordnung wird nicht berücksichtigt, wie aufwendig die Simulation eines Prädikats oder einer Klausel ist, sondern lediglich aufgezeigt, wie viele der Klauseln nicht unterstützt werden und simuliert werden müssen. Deshalb wird zusätzlich eine ergänzende Einordnung vorgenommen, aus der hervorgeht, wie gut die Simulation der Prädikate auf den vom System bereitgestellten Daten und Funktionen aufbauen kann.

Bewertung $\nearrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System gute bis sehr gute Voraussetzungen, wobei keine wesentlichen Einschränkungen zu berücksichtigen sind.

Bewertung $\rightarrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System mit einigen wesentlichen Einschränkungen gute Voraussetzungen.

Bewertung $\searrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System keine guten Voraussetzungen.

Die Einordnung ist nicht als Gesamtbewertung der Anfragesprachen der Systeme zu sehen, da Prädikate und Klauseln, die neben den geforderten unterstützt werden, nicht positiv in die Bewertung einfließen. Es geht vielmehr darum, die Umsetzbarkeit der IRQL-Prädikate in die entsprechenden Anfragesprachen zu prüfen.

Nach dieser Festlegung können in den folgenden Abschnitten die Anfragesprachen der Systeme einerseits vorgestellt und andererseits daraufhin eingeordnet werden, ob sie die in der Klassifikation vorgestellten Prädikate realisieren können. Eine kurze tabellarische Zusammenfassung schließt diese Betrachtungen ab.

An dieser Stelle sei bereits darauf hingewiesen, daß keine der Anfragesprachen der Datenquellen alle der in SQL3 eingeführten Konzepte umsetzt. Aus diesem Grund werden lediglich die zum SQL89-Kern [sql89] gehörenden Prädikate betrachtet.

4.2.1 ... mit der Anfragesprache von Fulcrum

Der *Fulcrum SearchServer* [ssi98] ist ein IR-System, das einige DBMS-Funktionalität besitzt. Die vom System indexierten Dokumente werden nicht intern gespeichert, sondern es werden Referenzen auf die Dokumente gehalten. Dokumente beliebigen Typs können vom SearchServer indexiert werden, sofern für diesen Typ ein Programm existiert, das die Metadaten aus dem Dokument extrahieren kann. Die extrahierten Metadaten können neben weiteren Daten in einer Tabelle gespeichert werden, auf die mit Hilfe der Anfragesprache zugegriffen wird. Der Fulcrum SearchServer zeichnet sich durch qualitativ sehr gute Retrievaleigenschaften aus, besitzt aber im Bereich von Datenbankanfragen einige wesentliche Schwächen, die im folgenden Unterabschnitt aufgezeigt werden.

Der SearchServer stellt eine SQL-artige Anfragesprache namens *SearchSQL* [sss98] bereit, die auf einer Teilmenge des SQL92-Standards [sql92] basiert und IR-spezifische Spracherweiterungen aufweist. Die Anfragesprache bietet von den bekannten Mengenoperationen lediglich die Vereinigung kompatibler Tabellen an.

Eine Verknüpfung mittels JOIN ist nicht erlaubt. Eine wichtige negative Eigenschaft ist, daß Anfragen nicht geschachtelt werden können. Auf die Prädikate der WHERE-Klausel können die booleschen Operatoren Konjunktion, Disjunktion und Negation angewendet werden.

Im Folgenden werden die für die Untersuchung wesentlichen Anfrageprädikate vorgestellt und ermittelt, inwieweit sie das entsprechende Prädikat der IRQL umsetzen. Die Untersuchungen beziehen sich auf die Version 3.7 des *Fulcrum SearchServers*.

4.2.1.1 Exakte Suche (DB)

Bei der exakten Suche zeigen sich deutliche Schwächen der Anfragesprache im Vergleich zu SQL-konformen Anfragesprachen (kein JOIN, keine Schachtelung).

Punktanfragen werden nicht vollständig unterstützt. Während die Konstantenselektion (gleich, ungleich, größer, größer gleich, kleiner, kleiner gleich) auf numerischen Werten und Datumswerten gestattet ist[6], wird die Attributselektion nicht unterstützt.

Bereichsanfragen können auf numerischen Werten und Datums-Werten durchgeführt werden.

Ungewißheitsselektionen werden unterstützt, wobei Wildcards (%[7] und _[8]) verwendet werden dürfen.

Weitere Prädikate zur exakten Suche. Da keine geschachtelten Anfragen in der Anfragesprache vorgesehen sind, gibt es keine Quantoren und Mengenvergleiche. Lediglich das *IN*-Prädikat steht in eingeschränkter Form zur Verfügung, wobei der Vergleich auf eine Konstantenselektion beschränkt ist.

Für die vom System bereitgestellten Prädikate der exakten Suche gelten Einschränkungen, die durch den Wrapper aufgehoben werden könnten. Die Attributselektion kann problemlos in der Nachbearbeitung der Ergebnisse durch den Wrapper simuliert werden. Prädikate, deren Parameter wiederum Anfragen darstellen, müßten ebenfalls durch den Wrapper simuliert werden. Dieser übergibt in einem ersten Schritt die einzelnen Teilanfragen an die Datenquelle, die diese bearbeitet, und simuliert auf den Ergebnissen die Funktionalität der Prädikate. Diese Ergebnisse könnten gegebenenfalls als Konstanten wieder in eine weitere Teilanfrage einfließen, die der Datenquelle zur Bearbeitung übergeben wird. Dabei ist zu berücksichtigen, daß eine gewisse Form von Optimierung, die ansonsten durch die Datenquelle vorgenommen werden würde, durch den Wrapper zu erfolgen hat,

[6] Der Vergleich von Zeichenketten auf Gleichheit ist ebenfalls gestattet.
[7] beliebig viele Zeichen
[8] ein beliebiges Zeichen

um die Antwortzeit nicht extrem negativ zu beeinflussen. Obwohl die Voraussetzungen für die Simulierung der fehlenden Operatoren und Prädikate relativ gut sind, führen das Fehlen des JOIN-Operators und die fehlende Möglichkeit zur Schachtelung von Anfragen zu einer Abwertung.

Bewertung: $\otimes \rightarrow$

4.2.1.2 Einfache Suche (IR)

SearchSQL stellt das `CONTAINS`-Prädikat bereit, um in unstrukturiertem Text zu suchen. Dabei werden sowohl die Suche nach

- einem Term oder mehreren Termen,
- einem Term oder mehreren Termen, die Wildcards enthalten als auch
- einer Phrase

unterstützt. Während die Suche nach einem oder mehreren Termen bzw. die Phrasensuche von vielen IR-Systemen bzw. DB-Systemen unterstützt wird, bietet der *Fulcrum SearchServer* weitergehend eine linguistische Vorbearbeitung des Suchterms an. Dadurch erfüllen nicht nur genaue Vorkommen des Suchterms im Text das Prädikat, sondern auch solche Begriffe, die aus dem Suchterm gebildet werden können (Mehrzahl, andere Zeitform bei Verben, Ersetzung von Umlauten durch andere Schreibweisen (ä=ae, ...)). Neben diesen Variationen des Suchterms können auch weitere Begriffe das Prädikat erfüllen, wenn sie ein Synonym des Suchterms darstellen bzw. wenn sie in einem Thesaurus diesem Begriff zugeordnet worden sind.[9]

Folgende Anmerkungen sind in Bezug auf die Klauseln, die im Prädikat vorkommen können, zu treffen:

- Eine Festlegung der Fehlertoleranz kann nicht erfolgen. Um einen alternativen Lösungsansatz zu finden, muß bei der Bestimmung des Fehlerwertes das Fehlen von Zeichen, das Einfügen von Zeichen und das Vertauschen von Zeichen zwischen den zu vergleichenden Termen erkannt werden. Während das Einfügen von zusätzlichen Zeichen durch Wildcards erkannt werden kann, ist das Fehlen von Zeichen bzw. die Vertauschung nicht mit Wildcards darstellbar. Dazu wäre die Abbildung in einen mächtigeren regulären Ausdruck notwendig, der aber in einer Anfrage nicht spezifiziert werden kann. Einen wesentlich praktikableren (wenn auch aufwendigen) Lösungsansatz bietet dagegen die Vorgehensweise, bei der der Index der verwalteten Suchterme komplett (oder teilweise) vom Wrapper persistent vorgehalten wird und bei einer Anfrage anhand eines Algorithmus die zu diesem Suchterm ähnlichen Terme des Index ermittelt und in die Anfrage integriert werden.

[9] Dies kann beispielsweise sinnvoll sein, wenn nach Abkürzungen gesucht wird. So könnte die Suche nach *OFG* automatisch um die Phrase *Opportunities For Growth* erweitert werden.

- Für jeden Suchterm (bzw. Phrase) kann ein Gewicht festgelegt werden.
- Die Forderung nach einer minimalen bzw. nach einer maximalen Anzahl des Auftretens des Suchterms kann ebenfalls nicht durch die Anfragesprache *SearchSQL* ausgedrückt werden. Eine Simulation durch den Wrapper ist relativ problemlos möglich, da abgefragt werden kann, wie oft ein Suchterm in einem konkreten Attribut aufgetreten ist. In der Nachbearbeitung der Anfrage ist anhand dieser Daten eine Selektion der Ergebnistupel durch den Wrapper möglich.
- Die Verwendung der Case-Sensitivitäts-Klausel aus der IRQL wird nur unterstützt, wenn beim Anlegen der Tabelle die entsprechende Option aktiviert wurde.[10] In Abhängigkeit von dieser Festlegung ist entweder nur eine Case-sensitive oder nur eine Case-insensitive Suche möglich. Darüber hinausgehende Funktionalität (beide Formen der Suche gemischt) muß durch den Wrapper simuliert werden.
- Die Teilstring-Suche kann durch die Verwendung von Wildcards simuliert werden.

Im Bereich der einfachen Suche erfüllt die Anfragesprache des *SearchServers* mit einigen Einschränkungen alle Anforderungen. Lediglich die Bestimmung des Fehlerwertes stellt ein Problem dar, das beim vorgestellten Lösungsansatz eine umfangreiche Vorbearbeitung durch den Wrapper erforderlich macht. Abgesehen von diesem Handicap kann jedoch eine positive Einordnung im Hinblick auf die Umsetzung getroffen werden.

Bewertung: $\otimes \nearrow$

4.2.1.3 Proximity-Suche (IR)

Auch die Proximity-Suche wird in *SearchSQL* durch das CONTAINS-Prädikat abgedeckt. Das Prädikat ist erfüllt, wenn die Suchterme innerhalb des vorgegebenen Abstandes (in Zeichen) voneinander im Text vorkommen. Die Reihenfolge, in der die Terme im Text vorkommen müssen, kann dabei festgelegt werden. Eine andere Granularität des Abstandes (Satz, Paragraph usw.) kann jedoch nicht gewählt werden, obwohl diese für Texte weitaus besser geeignet wären.

Das Prädikat kann durch die gleichen Klauseln beeinflußt werden, die bereits im Abschnitt über die einfache Suche untersucht wurden. Aus diesem Grund sei an dieser Stelle lediglich eine kurze Zusammenfassung der wesentlichen Punkte gegeben.

[10] Für alle leistungsfähigen IR-Systeme gilt, daß bereits bestimmte Festlegungen (Art des Index usw.) getroffen und Verwaltungsoperationen (Indizieren der Dokumente) durchgeführt werden müssen, bevor eine Anfrage an das System gestellt werden kann. Die dabei getroffenen Festlegungen bezüglich des Index können während der Bearbeitung einer Anfrage nicht rückgängig gemacht werden, da ansonsten der Index neu erstellt werden müßte.

- Die Berechnung eines Fehlerwertes muß durch den Wrapper erfolgen.
- Eine Gewichtung aller Suchterme der Proximity-Suche kann erfolgen.
- Die Prüfung, ob die Terme in der vorgegebenen (bzw. irrelevanten) Reihenfolge mindestens bzw. maximal sooft wie im Minimum bzw. Maximum gefordert auftreten, muß ebenfalls durch den Wrapper erfolgen.
- Ist eine Unterscheidung der Suchterme nach Groß- und Kleinschreibung gefordert, muß dies bereits beim Anlegen der Tabelle spezifiziert worden sein. Eine Case-insensitive Suche auf Daten, die Case-sensitiv indexiert sind, ist daher nicht möglich.
- Die Proximity-Suche kann auch auf Teilstrings angewendet werden, indem Wildcards verwendet werden.

Daraus ergibt sich ebenfalls eine positive Bewertung, da bis auf die Fehlertoleranz alle Klauseln ohne Probleme umsetzbar sind.

Bewertung: $\otimes \nearrow$

4.2.1.4 Phonetische Suche (IR)

Die phonetische Suche wird vom *SearchServer* nicht unterstützt und muß dementsprechend durch den Wrapper simuliert werden. Dieser verwendet einen Algorithmus, der die zum Suchterm ähnlich klingenden Terme bestimmt, nach denen anschließend in einer Anfrage an den *SearchServer* gesucht wird. In Abhängigkeit von der gewählten Implementierung des Algorithmus können einige bzw. alle Klauseln umgesetzt werden, die die phonetische Suche beeinflussen. Für die Simulation des Prädikats bietet der *SearchServer* somit gute Voraussetzungen.

Bewertung: $\ominus \nearrow$

4.2.1.5 Gesamteinschätzung

Die Anfragesprache des *SearchServers* bietet mit Ausnahme der phonetischen Suche gute Voraussetzungen für die Übersetzung der IRQL-Anfragekonstrukte. Die gesamte im Bereich der unscharfen Suche aufgezeigte, fehlende Funktionalität kann durch den Wrapper simuliert werden. Als negativ sind die sehr eingeschränkte Unterstützung bei der exakten Suche und die fehlende Funktionalität zur phonetischen Suche aufgefallen. Dementsprechend umfangreich sind die vom Wrapper zu simulierenden Operatoren und Prädikate. Insgesamt betrachtet bietet das System gute Voraussetzungen für die Simulation der nicht oder nur teilweise unterstützten Prädikate.

Der *Fulcrum SearchServer* bietet im Bereich der IR-Operationen viele Ansätze, die zu einer Verbesserung der Anfrageergebnisse führen. Dazu zählen unter anderem die Stoppwortliste, die flexible Auswahl von Retrieval-Modell (Strict Boolean, Fuzzy Boolean, Vector Space) und Relevanzberechnungsalgorithmus (*hits* *count*, *terms count*, *terms ordered* und *critical terms ordered*).[11] Darüber hinaus wird eine intuitive Suche ermöglicht, bei der in der Anfrage ein Teil eines Textes oder ein Dokument selbst angegeben wird, aus dem durch linguistische Bearbeitung die relevanten Terme ermittelt werden. Auf diesen Termen basiert die daran anschließende Suche durch den *SearchServer*.

Für weitere Informationen zur Anfragesprache *SearchSQL* und den im *Fulcrum SearchServer* verwendeten Konzepten sei abschließend auf die Dokumentation [sss98] und [ssd98] verwiesen.

4.2.2 ... mit der Anfragesprache von Harvest

Das *Harvest*-System [harvest] setzt sich aus mehreren modularen Komponenten zusammen [HSW96], die für die Indexierung der Dokumente (*Gatherer*), die Suche auf diesem Index (*Broker*) und weiteren Aufgaben zuständig sind. Eine Vorstellung des Systems wurde bereits in der dieser Diplomarbeit vorausgehenden Studienarbeit [Tit98, Kapitel 3] gegeben. Für die hier vorzunehmende Betrachtung zur Anfragesprache ist es erforderlich, die Anfragesprache des Query-Managers des Brokers zu untersuchen. Diese wird auf die Anfragesprache der darunterliegenden Suchmaschine abgebildet. Jedes Textretrieval-System, das attributbasierte Anfragen unterstützt, kann als Suchmaschine verwendet werden. Die Anfragesprache der Suchmaschine sollte dabei zumindest alle Konstrukte der Anfragesprache des Brokers realisieren können.

Da in der vorliegenden Arbeit nicht alle in Frage kommenden Anfragesprachen von Suchmaschinen bezüglich der Kriterien eingeordnet werden können, wurde die Suchmaschine ausgewählt, die zusammen mit dem *Harvest*-System ausgeliefert wird und deshalb eine recht große Verbreitung in laufenden *Harvest*-Diensten besitzt. Hierbei handelt es sich um *Glimpse* [MW94].[12] Neben dieser Suchmaschine unterstützt der *Harvest*-Broker explizit *WAIS*, dessen frei verfügbaren Ableger *freeWAIS* und *Nebula*.

Die Anfragesprache ist nicht SQL-konform und beschränkt sich auf Operatoren zur exakten Suche und zur einfachen IR-Suche. In einer Anfrage kann nur spezifiziert werden, aus welchen Attributen sich ein Ergebnistupel zusammensetzt und welche Attribute mit welcher Konstante zur Ermittlung der Ergebnismenge verglichen werden sollen. Die Anfragesprache unterstützt die Verknüpfung der bereitgestellten Operatoren durch die Konjunktion und Disjunktion sowie die Negation. Eine wichtige Charakteristik des Harvest-Systems (und damit der Anfragesprache) zeigt sich darin, daß die zu indexierenden Dokumente anhand eines vorgegebenen Kriteriums (Domain, geographische Lage, Thematik usw.) in einem

[11] In [sss98, Kapitel 2] werden die verwendeten Modelle und Relevanzalgorithmen genauer erklärt.

[12] Der Harvest-Broker des in der prototypischen Implementierung anzubindenden Literatur-Recherche Dienstes *Math-Net* [mathnet] setzt auf dieser Suchmaschine auf.

Index zusammengefaßt werden.[13] Die Festlegung, auf welchen Index die Anfrage angewendet wird, wird nicht in der Anfrage selbst, sondern durch die Kontaktierung des entsprechenden Brokers getroffen, der diesen Index verwaltet.

4.2.2.1 Exakte Suche (DB)

Wie sich in der Einführung gezeigt hat, reduziert sich eine Anfrage auf die Projektion und die Selektion der Tupel. Im Bereich der exakten Suche fehlen viele der in SQL festgelegten Prädikate und Operatoren.

Punktanfragen werden nur sehr eingeschränkt unterstützt. Für die Konstantenselektion wird lediglich der Operator zum Test auf Gleichheit angeboten. Alle anderen Operatoren der Konstantenselektion werden ebenso wie die Attributselektion nicht unterstützt.

Bereichsanfragen können nicht gestellt werden.

Ungewißheitsselektionen können bearbeitet werden, wobei die bereitgestelltem regulären Ausdrücke mächtiger sind, als die beiden Wildcards, die das entsprechende *LIKE*-Prädikat in SQL unterstützt.

Weitere Prädikate zur exakten Suche werden nicht angeboten.

Die betrachtete Anfragesprache ist nicht ausreichend, um alle Operatoren und Prädikate zur exakten Suche umzusetzen. Diese müßten komplett durch den Wrapper simuliert werden. Das System bietet keine erwähnenswerten Ansätze, auf denen eine Simulation der fehlenden Prädikate und Operatoren aufbauen könnte. Da eine Aufzählung der negativen Aspekte die Aufzählung positiver Aspekte bei weitem übertreffen würde, kann demzufolge nur eine negative Bewertung in Bezug auf die Umsetzbarkeit der Prädikate und Operatoren vergeben werden.

Bewertung: $\ominus$ $\searrow$

4.2.2.2 Einfache Suche (IR)

Im Bereich der einfachen Suche bietet die Anfragesprache die Möglichkeit, nach

- einem oder mehreren Termen,
- einem Term, der Wildcards enthält und
- Phrasen

zu suchen. Die Anfragesprache unterstützt die folgenden Klauseln des entsprechenden IRQL-Anfrageprädikats:

[13] Beispielsweise können alle Dokumente in einem Index zusammengefaßt sein, die auf einem Web-Server der Domain `uni-rostock.de` gespeichert sind; Dokumente eines Web-Servers der Domain `informatik.uni-rostock.de` in einem anderen Index usw.

- Die Fehlertoleranz kann festgelegt werden. Die dabei spezifizierte Ziffer bezieht sich auf die Anzahl der maximal erlaubten Löschungen, Einfügungen oder Ersetzungen eines Zeichens innerhalb des Suchterms.
- Eine Wichtung der Suchterme kann nicht erfolgen, da durch die Suchmaschine kein Ranking erfolgt.
- Es kann keine minimale und maximale Anzahl des Auftretens eines Suchterms gefordert werden. Um diese Klausel zu simulieren, muß der Wrapper die Ergebnisse des Brokers auswerten, in denen die Zeilen aufgelistet werden, die den Suchterm enthalten. Anhand dieser Daten kann die Selektion der zutreffenden Ergebnisse erfolgen.
- Die Case-sensitive und die Case-insensitive Suche werden unterstützt.
- Die Suche nach Termen bzw. Phrasen kann sowohl auf Wortgrenzen als auch auf Teilstrings erfolgen.

Mit Ausnahme der Wichtung der Suchterme können alle Klauseln des Prädikats zur einfachen Suche umgesetzt werden. Da die Suchmaschine, auf der der Broker aufsetzt, keine Relevanzberechnung durchführt, ist diese durch den Wrapper zu simulieren. Die dafür notwendige Implementierung von Relevanzberechnungsalgorithmen kann für einfache Vertreter[14] ohne großen Aufwand erfolgen. Die qualitativ wesentlich besseren Algorithmen, die z.B. Wortverteilungen im gesamten Dokumentenbestand berücksichtigen, lassen sich praktisch nicht umsetzen, da keine statistischen Informationen vom Suchsystem (*Glimpse*) ermittelt und demzufolge auch nicht von externen Anwendungen abgefragt werden können.

Bewertung: $\otimes \rightarrow$

4.2.2.3 Proximity-Suche (IR)

Die Proximity-Suche wird durch die Anfragesprache nicht abgedeckt. Die Simulation dieses Prädikates durch den Wrapper kann in zwei Schritten erfolgen. Im ersten Schritt wird eine Anfrage gestellt, in der geprüft wird, ob die Terme, auf denen die Proximity-Suche stattfindet, im entsprechenden Attribut auftreten. Im zweiten Schritt wird der Abstand dieser Terme voneinander bestimmt und das Ergebnis des Prädikates ermittelt.

Für die Umsetzung der Klauseln gelten die gleichen Aussagen, die für die einfache Suche getroffen wurden. Die fehlende Relevanzberechnung und der Mangel, keine Berechnung von qualitativ höherwertigen Relevanzwerten durchführen zu können, führt zu einer Abwertung in Bezug auf die Umsetzbarkeit.

Bewertung: $\ominus \rightarrow$

[14] Beispielsweise sei der vom *Fulcrum SearchServer* angebotene *hits count*-Algorithmus erwähnt, bei dem aus der Häufigkeit des Auftretens des Suchterms im Dokument der Relevanzwert ermittelt wird.

4.2.2.4 Phonetische Suche (IR)

Ein Prädikat zur phonetischen Suche wird nicht bereitgestellt. Dieses muß dementsprechend mit allen dazugehörenden Klauseln komplett durch den Wrapper simuliert werden. In Abhängigkeit vom implementierten Algorithmus könnten einige oder alle der geforderten Klauseln umgesetzt werden. Eine mögliche Vorgehensweise bei der Implementierung im Wrapper läuft darauf hinaus, eine Menge von Wörtern zu ermitteln, die ähnlich wie der Suchterm klingen, und diese in einer Anfrage disjunktiv zu verknüpfen.

Die Umsetzung des Prädikates an sich kann relativ problemlos erfolgen. Die bereits mehrfach erwähnten Probleme bei der Relevanzberechnung führen jedoch wiederum zu einer Abwertung.

Bewertung: $\ominus \rightarrow$

4.2.2.5 Gesamteinschätzung

Die Anfragesprache des *Harvest*-Brokers stellt nur einen geringen Teil der durch die IRQL-Prädikate realisierten Funktionalität bereit. Da sowohl die Proximity-Suche als auch die phonetische Suche komplett und die exakte Suche fast komplett durch den Wrapper simuliert werden müssen, ist für die Umformung der IRQL-Anfrageprädikate im Wrapper ein erheblicher Implementierungsaufwand erforderlich. Neben den hier betrachteten Prädikaten bietet die Anfragesprache des *Harvest*-Brokers keine weiteren IR-Konzepte wie Relevance Feedback und Ranking an. Aufgrund der fehlenden Basis in Bezug auf IR- und DB-Konzepte kann das *Harvest*-System nicht für die Verwendung als LDS empfohlen werden.

Insgesamt betrachtet bietet die Anfragesprache dieses Systems nur den positiven Aspekt, daß viele der Klauseln durch die Anfragesprache unterstützt werden, so daß zumindest eine gewisse Basis vorhanden ist, auf der aufgebaut werden kann. Das Fehlen einer Rankingfunktion zur Berechnung eines Relevanzwertes führt jedoch dazu, daß bei der nachträglichen Ermittlung dieses Wertes ein hoher Aufwand betrieben werden muß, da keine Metadaten zum Datenbestand des Indexes durch den Broker exportiert werden. Die Güte der Anfrageergebnisse ist im Vergleich zu IR-Systemen (wie Fulcrum) als kritisch zu bewerten. Aus diesem Grund erfolgte eine durchgängig negative Bewertung für die Abschätzung des Aufwandes bei der Simulierung der Prädikate durch den Wrapper. Reichen sehr einfache Berechnungsalgorithmen für die Bestimmung eines Relevanzwertes aus, relativiert sich die Bewertung und kann mit Ausnahme der exakten Suche als durchaus positiv bezeichnet werden.

4.2.3 ... mit der Anfragesprache von DB2

Beim *DB2 Universal Database*-System (UDB) [Cha98] von IBM handelt es sich um ein objektrelationales Datenbanksystem, für das Erweiterungen (*Extender*) existieren, die Suchanfragen über Grafiken, Audio- und Videodaten sowie Texten realisieren. Für die vorliegende Arbeit ist der *TextExtender* [IBM97] von Bedeutung, der Funktionen zur Volltextsuche bereitstellt. Die Untersuchungen basieren auf der Version 5.2 des *TextExtenders*.

Im Vergleich mit dem bereits betrachteten Volltextdatenbanksystem *Fulcrum SearchServer* bietet das Datenbanksystem *UDB* eine größere Funktionalität im Hinblick auf exakte Anfragen an, ist jedoch im Bereich der IR-Funktionen auf Texten wesentlich schwächer als der eigens dafür entwickelte *Fulcrum SearchServer*. Dies betrifft unter anderem die Beschränkung auf das boolesche Retrieval-Modell und die starre Ranking-Funktion.

Eine ausführliche Vorstellung der durch den *TextExtender* bereitgestellten Funktionalität wird beispielsweise in [Por99] gegeben.

4.2.3.1 Exakte Suche (DB)

Da es sich bei der *UDB* um ein Datenbanksystem handelt, dessen Anfragesprache den SQL92-Standard erfüllt, werden alle Prädikate der exakten Suche in IRQL-Anfragen unterstützt. Aus diesem Grund werden die Prädikate und Operatoren für Punktanfragen, Bereichsanfragen und Ungewißheitsselektionen an dieser Stelle nicht explizit vorgestellt. Neben diesen Prädikaten werden das *IS NULL*-Prädikat[15], das *IN*-Prädikat[16], das *EXISTS*-Prädikat[17] und die Quantoren *all*, *any* und *some* unterstützt.

Die Prädikate zur exakten Suche der IRQL können somit vollständig in die Prädikate der Anfragesprache der *UDB* umgeformt werden, ohne sie durch den Wrapper simulieren zu müssen. Deshalb kann eine Bewertung in Bezug auf den Umfang des zur Umsetzung erforderlichen Aufwandes entfallen.

Bewertung: $\oplus$

4.2.3.2 Einfache Suche (IR)

Um eine unscharfe Suche in Texten vornehmen zu können, ist eine Indexierung der Dokumente erforderlich. Der *TextExtender* bietet vier verschiedene Indexierungsformen an, wobei jede nur einen Teil der zur Verfügung stehenden Operationen unterstützt.[18]

[15] Test, ob ein Attribut mit dem NULL-Wert belegt ist.

[16] Ein Wert ist in einer Menge enthalten, die statisch oder dynamisch durch eine Unteranfrage festgelegt wird.

[17] Von der Unteranfrage wurde mindestens ein Ergebnistupel zurückgeliefert.

[18] Die unterstützten Indexformen sind der *linguistische Index* (linguistische Bearbeitung der Begriffe eines Dokumentes), der *präzise Index* (exakte Form der Begriffe im Index), der *duale Index* (Kombination von linguistischem und präzisem Index) und der *Ngram Index* (Ähnlichkeit von Begriffen). In [Por99, Kapitel 4, Abschnitt 4.2] wird ein tabellarischer Überblick über die Indizes und die von ihnen unterstützte Suchfunktionalität gegeben.

Da für jedes Dokument (bzw. Attribut) nur eine Indexierungsform zulässig ist, bilden die in diesem und den folgenden Abschnitten vorgenommenen Betrachtungen einen zur Zeit in der Praxis nicht erreichbaren Idealfall, da nicht alle der hier berücksichtigten Prädikate bzw. deren Klauseln anhand eines der Indizes ermittelt werden können. Ein kurzes Beispiel illustriert dies. Bei Verwendung eines linguistischen Index, bei dem eine Normalisierung der Terme eines Dokumentes während der Indexierung vorgenommen wird, kann keine exakte Suche (in Bezug auf den zu findenden Term z.B. *laufend*) stattfinden, da nur die Wortstämme der Begriffe des Dokumentes (im Beispiel *lauf*) im Index gespeichert sind. Da die Suche auf dem Index und nicht auf dem Dokument selbst stattfindet, kann nicht eindeutig entschieden werden, ob der Suchterm in einem Dokument vorkommt oder nicht. Um auf die Anfrage trotzdem ein Ergebnis zu erhalten, ist durch den Wrapper eine aufwendige Suche im Dokument selbst notwendig.

Der *TextExtender* stellt diverse Funktionen (*User Defined Functions*) für die Volltextsuche bereit. Die für die hier vorgenommenen Betrachtungen wichtigste Funktion stellt die *CONTAINS*-Funktion dar, da durch sie nach

- einem oder mehreren Termen,
- einem oder mehreren Termen, die Wildcards enthalten und
- Phrasen

im Text gesucht werden kann. Wie auch beim *Fulcrum SearchServer* kann die Suche um linguistische Variationen des Terms, um Synonyme bzw. andere durch den Thesaurus ermittelte Begriffe erweitert werden, wobei diese Formen der Anfrageerweiterung nicht auf allen durch das System angelegten Indizes angewandt werden können.

Die im IRQL-Prädikat zur einfachen Suche vorkommenden Klauseln können teilweise unter Zuhilfenahme der vom *TextExtender* bereitgestellten Funktionen umgesetzt werden.

- Die Vorgabe einer maximalen Fehlertoleranz ist nur bei Verwendung des *Ngram Index* möglich. Alle anderen Formen von Indizes unterstützen keine fehlertolerante Suche.
- Eine Wichtung der Suchterme einer Anfrage ist nicht möglich. Diese kann jedoch in einem Nachbearbeitungsschritt im Wrapper nachträglich auf den Rückgabewert der Funktion *NO_OF_MATCHES* angewendet in die Relevanzberechnung einfließen.
- Der *TextExtender* stellt eine Funktion *NO_OF_MATCHES* bereit, die auf dieses Prädikat angewendet, die Anzahl der Treffer zurückliefert. Anhand dieses Wertes kann vom System ermittelt werden, ob das Ergebnis innerhalb des spezifizierten Minimum/Maximum-Intervalls liegt.

- In Abhängigkeit von der gewählten Indexform kann die Case-Sensitivität bei der Suche berücksichtigt werden oder nicht. Für die vom *TextExtender* erzeugten Indizes bedeutet dies im Einzelnen: eine Case-sensitive Suche wird vom *linguistischen Index* nicht unterstützt, da eine Normalisierung der Begriffe durchgeführt wird. In einem *präzisen Index* kann nur Case-sensitiv gesucht werden, da die Begriffe in exakt der Form indiziert werden, in der sie im Text auftreten. Beim *dualen Index* treffen je nach verwendetem Index die Feststellungen der vorher genannten Indizes zu. Im *Ngram Index* sind sowohl die Case-sensitive als auch die Case-insensitive Suche möglich, wobei im ersteren Fall bereits beim Anlegen des Index' explizit festgelegt werden muß, daß die Indexierung Case-sensitiv zu erfolgen hat.
- Die Suche nach einem Term innerhalb einer Zeichenkette kann durch die Integration von Wildcards in den Suchterm auf allen Indizes erfolgen. Da beim *linguistischen Index* eine Normalisierung der Begriffe durchgeführt wird, können bei der Suche nach Teilstrings einige Vorkommen jedoch nicht erkannt werden.[19] Dies ist bei der Festlegung des Suchterms zu berücksichtigen.

Bei der Kombination der betrachteten Klauseln können je nach verwendetem Index weitere Einschränkungen gelten, auf die an dieser Stelle nicht eingegangen wird.[20] Daraus können eventuell weitere Vor- oder Nachbearbeitungen durch den Wrapper erforderlich werden, die aus dieser kurzen Betrachtung nicht hervorgehen.

Aus den vorgenommenen Betrachtungen wird deutlich, daß im Bereich der einfachen Suche zwar alle Klauseln umgesetzt werden können, jedoch je nach verwendetem Index Einschränkungen auftreten. Da dies große Auswirkungen auf die praktische Verwendbarkeit besitzt und aufgrund der Nebenwirkungen der Aufwand zur Simulierung der nicht miteinander kombinierbaren Klauseln steigt, kann im Vergleich zum *Fulcrum SearchServer* nur eine negative Bewertung des erforderlichen Aufwandes erfolgen.

Bewertung: $\otimes \rightarrow$

4.2.3.3 Proximity-Suche (IR)

Der *TextExtender* unterstützt die Proximity-Suche, wobei bei der Abstandsbestimmung zwischen den Suchtermen die minimale Granularität ein Satz und die maximale Granularität ein Abschnitt darstellen kann. Eine andere Granularität (Wort, Zeichen) kann, wie auch die Reihenfolge, in der die Terme auftreten sollen, nicht festgelegt werden.

---

[19] Beispielsweise wird das Verb *fuhren* zu *fahren* normalisiert. Die Suche nach *fuh%* findet dementsprechend kein Vorkommen des Teilstrings im Index.

[20] Bei der Teilstringsuche auf einem *dualen Index* ist beispielsweise nur eine Case-sensitive Suche möglich.

Die in dem Proximity-IRQL-Prädikat vorkommenden Klauseln können nur teilweise durch die vom *TextExtender* bereitgestellten Funktionen umgesetzt werden.

- Die Vorgabe einer maximalen Fehlertoleranz ist nur bei Verwendung des *Ngram Index* möglich. Alle anderen Formen von Indizes unterstützen keine fehlertolerante Suche.
- Eine Wichtung der Suchterme der Anfrage ist nicht möglich.
- Die Funktion *NO_OF_MATCHES* liefert bei Anwendung auf das Proximity-Prädikat die Gesamtsumme der Vorkommen der Suchterme innerhalb des Textes zurück. Anhand dieses Wertes kann jedoch nicht geprüft werden, ob die Suchterme mindestens bzw. maximal sooft wie gefordert innerhalb eines Satzes bzw. eines Abschnittes auftreten. Deshalb muß dieser Test durch den Wrapper durchgeführt werden.
- In Abhängigkeit von der gewählten Indexform kann die Case-Sensitivität bei der Suche berücksichtigt werden oder nicht. Eine vertiefende Erklärung dafür wurde bereits bei der Betrachtung zur einfachen Suche gegeben.
- In einem Proximity-Prädikat kann eine Teilstringsuche erfolgen. Es gilt die gleiche Einschränkung für den *linguistischen Index*, die bereits bei den Betrachtungen zur einfachen Suche erläutert wurde.

Die Umsetzung des IRQL-Proximity-Prädikates durch die vom *TextExtender* bereitgestellte Funktion kann nur recht eingeschränkt erfolgen, da einerseits die Reihenfolge, in der die Suchterme auftreten sollen, nicht festgelegt werden kann, und andererseits diverse Klauseln nicht unterstützt werden. Eine Simulation dieser Klauseln ist demzufolge notwendig. Da auch für dieses Prädikat die Einschränkungen bezüglich der Kombinierbarkeit der Indizes und damit der Klauseln gelten, ist im Vergleich zum Aufwand, der bei der Umsetzung des Prädikates im *Fulcrum SearchServer* erforderlich ist, der hier erforderliche Aufwand wesentlich größer.

Bewertung: $\otimes \rightarrow$

4.2.3.4 Phonetische Suche (IR)

Der *TextExtender* stellt ein Prädikat für die phonetische Suche bereit, das auf den *linguistischen*, den *präzisen*[21] und den *dualen* Index angewendet werden kann.

- Die Vorgabe einer maximalen Fehlertoleranz ist, wie bereits mehrfach erwähnt, nur bei Verwendung des *Ngram Index* möglich. Da diese Indexform

[21] Laut Dokumentation kann die phonetische Suche auf diesem Index durchgeführt werden. In einem praktischen Test konnte dies jedoch nicht nachvollzogen werden, da die Anfragebearbeitung jedesmal fehlschlug. Dies deckt sich mit den Erfahrungen, die in [Por99, Abschnitt 4.3] gemacht wurden.

jedoch keine phonetische Suche unterstützt, ist eine Umsetzung dieser Klausel durch den *TextExtender* nicht möglich. Demzufolge muß diese Funktionalität durch den Wrapper simuliert werden.

- Eine Wichtung des Suchterms der Anfrage ist nicht möglich.
- Die Funktion *NO_OF_MATCHES* liefert bei Anwendung auf das Prädikat der phonetischen Suche die Anzahl der Vorkommen der ähnlich klingenden Begriffe im Text zurück. Anhand dieses Wertes kann geprüft werden, ob dieser im in der Klausel spezifizierten Minimum/Maximum-Intervall liegt.
- Das Prädikat der phonetischen Suche wird sowohl durch komplette Begriffe als auch durch Teilbegriffe, die ähnlich klingen, erfüllt. Eine Festlegung, daß nur erstere (oder letztere) bei der Suche berücksichtigt werden, kann jedoch nicht getroffen werden.

Das IRQL-Prädikat für die phonetische Suche kann nur sehr eingeschränkt durch die vom *TextExtender* bereitgestellte Funktion umgesetzt werden. Der Großteil der Klauseln muß durch den Wrapper simuliert werden, was insbesondere dadurch erschwert wird, daß nicht immer erkennbar ist, welcher Begriff oder Teilbegriff im Text das Prädikat erfüllt hat. Eine Highlight-Funktion, wie sie beispielsweise der *Fulcrum SearchServer* bereitstellt, wäre in diesem Fall eine nützliche Hilfe. Durch das Fehlen dieser Funktion wird die Simulation der fehlenden Klauseln sehr erschwert (wenn nicht gar unmöglich gemacht), da nicht sicher ist, auf welche der Begriffe die Klauseln anzuwenden sind. Deshalb wird der Aufwand bei der Simulation als hoch eingestuft.

Bewertung: $\otimes \searrow$

4.2.3.5 Gesamteinschätzung

Mit Ausnahme der exakten Suche erfordert die Umsetzung der betrachteten IRQL-Prädikate häufig die Simulation durch den Wrapper, da die bereitgestellten *TextExtender*-Funktionen nicht ausreichend sind. Negativ wirkt sich insgesamt betrachtet aus, daß einige Funktionen nur auf bestimmten Indizes angewendet werden können. Da für die Umsetzung der IRQL-Prädikate und ihrer zugeordneten Klauseln jedoch die Kombination mehrerer *TextExtender*-Funktionen notwendig wäre, was jedoch nicht auf allen Indizes möglich ist, muß zwangsläufig eine Entscheidung gegen die Verwendung eines Teil der vom *TextExtender* bereitgestellten Funktionen getroffen werden. Diese müssen demzufolge durch den Wrapper simuliert werden.

Neben den hier betrachteten Funktionen bietet der *TextExtender* weitere Möglichkeiten zur Anfrageverbesserung, wie die Expansion der Anfrageterme durch die Nutzung von Synonymen und Thesauri an. Bei der *Freitext-Suche* kann ein (Teil-)Satz angegeben werden, aus dem relevante Suchbegriffe ermittelt und darauf aufbauend Dokumente selektiert werden.[22]

Eine aus der Untersuchung der *TextExtender*-Funktionen abgeleitete Forderung für die Verbesserung des *TextExtenders* ist die gleichzeitige Zuordnung mehrerer Indizes zu einem Attribut. Daraus resultieren natürlich einige negative Effekte wie höherer Speicherplatzbedarf und eine verschlechterte Performance, die aber zumindest durch die höhere Funktionalität gerechtfertigt werden können. Weitere Verbesserungen in Bezug auf die Ergebnisqualität könnten durch die Bereitstellung verschiedener Relevanzberechnungsalgorithmen (siehe auch *Fulcrum SearchServer* in Abschnitt 4.2.1.5) erzielt werden.

Für vertiefende Informationen zum *TextExtender* und den bereitgestellten Funktionen sei auf die Dokumentation in [IBM97] und auf eine etwas ausführlichere (und unabhängig von der vorliegenden Arbeit vorgenommene) Untersuchung der gesamten *TextExtender*-Funktionalität in [Por99] verwiesen.

4.3 Zusammenfassung

In den vorherigen Abschnitten wurden die Anfragesprachen des IR-Systems *Fulcrum SearchServer*, des *Harvest*-Brokers und des objektrelationalen Datenbanksystems *DB2* daraufhin untersucht, inwieweit sich mit ihnen die betrachteten Prädikate der WHERE-Klausel einer IRQL-Anfrage umsetzen lassen. Die zusätzlich durchgeführte grobe Aufwandsabschätzung gestattet eine differenzierte Betrachtung der Ergebnisse. Ein System, das weniger Klauseln oder Prädikate der IRQL umsetzt, kann eine bessere Wahl bilden als ein anderes System, bei dem zwar weniger Prädikate oder Klauseln simuliert werden müssen, aber der dafür erforderliche Aufwand wesentlich größer ist. Das Ergebnis dieser Untersuchung wird in Tabelle 4.1 zusammengefaßt.
Bei der Betrachtung der Tabelle fällt auf, daß keine der Anfragesprachen die betrachteten IRQL-Prädikate inklusive ihrer Klauseln vollständig umsetzen kann. Jedes der Systeme besitzt seine Schwächen, die sich im Aufwand der Simulation niederschlagen und die Gesamtbewertung negativ oder positiv beeinflussen.

In den vorherigen Abschnitten wurde für die davon betroffenen Klauseln ein erster Lösungsvorschlag gegeben, sofern das System eine Funktionalität bereitstellt, auf der dieser Lösungsvorschlag aufbauen kann. Die wesentlichen Punkte zu den einzelnen Systemen sind abschließend kurz zusammengefaßt.

Fulcrum SearchServer. Bei der Untersuchung der Anfragesprache des *Fulcrum SearchServers* sind folgende Schwachstellen aufgefallen, die eine Bearbeitung der Anfragen durch den Wrapper erforderlich machen.

Einige wesentliche, im Anfrageteil von SQL festgelegte Operatoren (Verbund) und Prädikate (die auf geschachtelten Anfragen arbeiten) werden nur in einge-

[22]Dies entspricht in etwa der intuitiven Suche des *Fulcrum SearchServers*.

Kriterium	SearchServer	Harvest	DB2
exakte Suche	$\otimes \rightarrow$	$\ominus \searrow$	$\oplus$
einfache Suche	$\otimes \nearrow$	$\otimes \rightarrow$	$\otimes \rightarrow$
Proximity-Suche	$\otimes \nearrow$	$\ominus \rightarrow$	$\otimes \rightarrow$
phonetische Suche	$\ominus \nearrow$	$\ominus \rightarrow$	$\otimes \searrow$

Erläuterung der verwendeten Symbole:

Bewertung $\oplus$: Die Anfragesprache des Systems unterstützt Selektionen anhand dieses Prädikats. Der Wrapper muß nur einen geringen Teil bzw. gar keine der fehlenden Klauseln simulieren.

Bewertung $\otimes$: Die Anfragesprache des Systems unterstützt Selektionen anhand dieses Prädikats nur eingeschränkt. Durch den Wrapper müsen die fehlenden Klauseln simuliert werden.

Bewertung $\ominus$: Die Anfragesprache des Systems unterstützt keine Selektionen anhand dieses Prädikats. Das Prädikat muß durch den Wrapper simuliert werden.

Bewertung $\nearrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System gute bis sehr gute Voraussetzungen, wobei keine wesentlichen Einschränkungen zu berücksichtigen sind.

Bewertung $\rightarrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System mit einigen wesentlichen Einschränkungen gute Voraussetzungen.

Bewertung $\searrow$: Für die Simulation der von der Anfragesprache des Systems nicht oder nur teilweise unterstützten Prädikate bietet das System keine guten Voraussetzungen.

Tabelle 4.1: Eignung der Anfragesprache für die Umsetzung der betrachteten IRQL-Anfrageprädikate

schränkter Form (*IN*) bzw. gar nicht (*EXISTS*) umgesetzt. Diese müssen somit vom Wrapper durch Zerlegung der Anfrage in Teilanfragen, die vom System bearbeitet werden können, und die Implementierung der Funktionalität der entsprechenden Operatoren und Prädikate, die auf Teilanfragen dieser Teilanfragen arbeiten, simuliert werden.

Im Bereich der vagen Anfragen unterstützt die Anfragesprache mit Ausnahme der phonetischen Suche alle betrachteten Prädikate. Die Umsetzung der darin auftretenden Klauseln kann im Wesentlichen relativ einfach erfolgen. Das Fehlen einer Funktion zur Fehlertoleranzbeeinflussung verhindert jedoch eine sehr gute Bewertung für die Umsetzung der entsprechenden IRQL-Prädikate.

Harvest. Die betrachtete Anfragesprache des *Harvest*-Systems bietet nur sehr wenige positive Ansätze für die Umsetzung der Prädikate. Die vom System bereitgestellte Funktionalität ist derart gering, daß eine Abwägung erforderlich ist, ob eine Anbindung (und die damit verbundene Simulation aller nicht unterstützen Prädikate und Operatoren) tatsächlich sinnvoll ist. Dagegen spricht, daß eigentlich fast die gesamte Funktionalität durch den Wrapper bereitgestellt werden muß und das zugrunde liegende System nur einen sehr geringen Teil der Aufgaben erledigt.

DB2 inklusive TextExtender. Die Anfragesprache von *DB2* unterstützt im Bereich der exakten Anfragen alle betrachteten Prädikate und Operatoren und bildet in diesem Vergleich damit die beste Wahl.

Die Funktionalität des *TextExtenders*, durch die die IR-spezifischen Prädikate realisiert werden, bietet mit Einschränkungen gute Ansätze für die Umsetzung, da alle Prädikate unterstützt werden. Als sehr nachteilig hat sich herausgestellt, daß mehrere Indexformen verwendet werden, auf denen jeweils nur ein Teil der Prädikate und Klauseln angewendet werden kann. Damit kann zwangsläufig ein Teil der vom System bereitgestellten Funktionalität nicht genutzt werden. Einen weiteren Kritikpunkt bildet die unzureichende Unterstützung verschiedener Relevanzberechnungsalgorithmen und die fehlende Zuweisung von Gewichten.

Schlußfolgerungen

In diesem Kapitel wurden Schwächen der Systeme bei der Umsetzung aufgedeckt und, wenn möglich, geeignete Alternativen aufgezeigt, die sich aus der vom System bereitgestellten Funktionalität ergeben. Aus den dabei angestellten Betrachtungen zur Umsetzung der Prädikate können folgende Erkenntnisse gewonnen werden:

- Keines der Systeme kann mit seiner Anfragesprache die Prädikate (und ihre Klauseln) einer IRQL-Anfrage vollständig umsetzen.
- Der *Fulcrum SearchServer* bietet zwar eine gute Basis als IR-System und damit für die Umsetzung der entsprechenden IRQL-Prädikate, sollte aber wesentlich mehr Datenbankfunktionalität besitzen und die fehlenden Prädikate und Operatoren des SQL-Anfrageteils implementieren.
- Das *Harvest*-System erscheint für die Umsetzung von IRQL-Anfragen nicht geeignet.
- Der *DB2 TextExtender* bietet Funktionen, die alle betrachteten IR-Prädikate umsetzen. Diese funktionieren jedoch teilweise nur auf einer Indexform und weisen Einschränkungen auf. Eine kombinierte Anwendung dieser Funktionen ist demnach nicht immer möglich, wodurch die nutzbare Funktionalität der Anfragesprache reduziert wird.
- Eine Kombination der DB-Funktionalität von *DB2* mit der IR-Funktionalität des *Fulcrum SearchServers* (als externes Modul zur Suche in Texten)

würde die beste Voraussetzung für die Umsetzung der IRQL-Anfragesprache bieten.

Aus den Untersuchungen ergeben sich einige offene Fragen zur Umsetzung der IRQL-Anfrage-Prädikate, da die von den Anfragesprachen nicht unterstützten Prädikate entweder simuliert oder aber auf bereitgestellte Prädikate oder Operatoren abgebildet werden müssen. Deren Ergebnismenge weicht jedoch häufig von der korrekten Ergebnismenge ab, wodurch eine Nachbearbeitung der Ergebnisse erforderlich wird.[23]

Ein weiteres Problem ergibt sich, wenn die Anfragesprache des Zielsystems zwar ein IRQL-Prädikat umsetzen kann, aber der dazu verwendete Algorithmus in den Systemen auf einer unterschiedlichen Granularität arbeitet. Beispielhaft sei die Proximity-Suche genannt, bei der im *Fulcrum SearchServer* die Abstandsbestimmung in Zeichen erfolgt, während sie der *DB2 TextExtender* auf Basis von Sätzen bzw. Paragraphen durchführt.

Die Lösung dieser Probleme bildet eine wesentliche Aufgabe, die bei der Realisierung des Virtuellen Dokumenten-Servers zu lösen ist. In der vorliegenden Arbeit wird diese Aufgabe jedoch nicht übernommen, da der dazu erforderliche Aufwand den zeitlichen Rahmen dieser Diplomarbeit bei weitem übersteigen würde. In der anschließend beschriebenen prototypischen Implementierung wird deshalb auch nur das einfachste der betrachteten IR-Prädikate (die einfache Suche) umgesetzt, das von allen Systemen unterstützt wird.

In den bisherigen Kapiteln wurden die Anforderungen an einen VDS klassifiziert, darauf aufbauend die Architektur entworfen und erste Betrachtungen zur Umsetzung der IRQL-Prädikate in die Prädikate der Anfragesprache der anzubindenden Systeme vorgenommen. Auf den Ergebnissen dieser Kapitel baut die im folgenden Kapitel beschriebene prototypische Implementierung des VDS auf.

[23]Beispielsweise könnte das Ergebnis der Proximity-Suche (*attribute CONTAINS searchterm BEFORE searchterm2*) durch das Ergebnis der konjunktiven Verknüpfung der Suchterme (*(attribute CONTAINS searchterm) AND (attribute CONTAINS searchterm2)*) näherungsweise ermittelt werden, wobei natürlich die Ergebnismenge der Proximity-Suche eine Untermenge des Ergebnismenge der ersatzweise kombinierten Anfrageprädikate darstellt. Eine Nachbearbeitung der Ergebnisse ist somit erforderlich. Im Rahmen des *Stanford Digital Library Projects* [dig98] wurden dazu bereits Untersuchungen zum Mapping von Booleschen Anfragen auf heterogene Informationsquellen durchgeführt [CGMP96].

Kapitel 5

Prototypische Implementierung

Ausgehend vom bereits vorgestellten Architekturentwurf und der Schnittstellenspezifikation wird in diesem Kapitel die prototypische Implementierung des Query-Servers und der Wrapper der LDS beschrieben.

In der Einleitung in Abschnitt 5.1 wird erläutert, welche Programmiersprache zur Implementierung verwendet wurde und weshalb die Entscheidung für diese Sprache getroffen wurde. Eine Vorstellung der Konzepte, die im Rahmen der Implementierung verwendet wurden, erfolgt in Abschnitt 5.2. Im Anschluß daran werden in Abschnitt 5.3 die Implementierung des Query-Servers und in Abschnitt 5.4 die Implementierung der Wrapper für die drei anzubindenden Systeme dargelegt und auf aufgetretene Probleme und zu berücksichtigende Einschränkungen in der Implementierung hingewiesen. Eine grundsätzliche Diskussion zu Erweiterungsmöglichkeiten findet in diesem Kapitel nicht statt, da das Ziel der Implementierung ein Basissystem darstellen soll. Dieses wird erst durch weiterführende Arbeiten in einen voll nutzbaren Virtuellen Dokumenten-Server überführt und weist somit viele Erweiterungsmöglichkeiten auf.

Die Beschreibung der Klassen, die für die Bereitstellung der Funktionalität der einzelnen Dienste zuständig sind, wurde aus Gründen der Übersichtlichkeit aus dieser Arbeit ausgelagert und ist in einem gesonderten Technischen Report [Tit99] verfügbar. In diesem sind auch eventuell vorgenommene Änderungen dokumentiert, die sich während der Implementierung als sinnvoll erwiesen haben, aber nicht mehr in die vorliegende schriftliche Arbeit einfließen konnten.

5.1 Wahl der Implementierungssprache

Für die Implementierung des Prototypen wurde die Programmiersprache Java[1] (JDK 1.2) ausgewählt. Dafür sprechen

- das schnelle Prototyping,

[1] `http://java.sun.com/`

- die Plattformunabhängigkeit,
- die Sicherheit,
- die einfache Anbindung von Datenbanken (JDBC, SQLJ),
- die Objektorientierung,
- die gute Basis für verteilte Anwendungen (z.B. RMI, CORBA),
- die gut unterstützte parallele Verarbeitung (Threading) sowie
- die Verfügbarkeit vieler wichtiger Basis-Bibliotheken (Netzwerk, Datenbankanbindung) auf allen für Java verfügbaren Plattformen.

Als alternative Implementierungssprache stand die Programmiersprache C zur Verfügung, gegen die aber

- die kompliziertere Programmierung (Speicherplatzverwaltung etc.),
- die mangelnde Plattformunabhängigkeit,
- die Anbindung von Datenbanken, die mittels ODBC erfolgt (schwerer als JDBC erlernbar und komplizierter) sowie
- die fehlenden objektorientierten Konzepte sprechen.

Insbesondere durch die Tatsache, daß eine Einarbeitung in jede der zur Auswahl stehenden Sprachen erforderlich gewesen wäre, fiel die Entscheidung für Java, was sicherlich auch in Hinblick auf die Zukunftsträchtigkeit der Sprache geschehen ist.

5.2 Allgemeine Konzepte

Neben der Auswahl einer geeigneten Implementierungssprache gibt es weitere Entscheidungen, die für oder gegen ein bestimmtes Konzept bei der Implementierung zu treffen sind. In den folgenden Abschnitten werden einleitend einige Konzepte zur Entwicklung verteilter Anwendungen mit Java und zum Zugriff auf Datenbanken vorgestellt und ihre Eignung für die Verwendung in der prototypischen Implementierung eingeschätzt.

5.2.1 Entwicklung verteilter Anwendungen mit Java

Der Virtuelle Dokumenten-Server stellt eine verteilte Anwendung dar, deren bereitgestellte Dienste aus den verschiedensten Gründen auf mehreren Rechnern verteilt residieren können. So kann es beispielsweise erforderlich sein, daß ein Wrapper auf einem Rechner laufen muß, der Client einer bestimmten Datenbank ist, um auf diese zugreifen zu können. Aus Gründen der Lastverteilung kann die Verteilung von Query- und Replikations- und Caching-Dienst-Server auf verschiedene Rechner sehr sinnvoll sein. Deshalb ist es erforderlich, zwischen den einzelnen Diensten eine Kommunikation via Netzwerk durchzuführen. Dafür stellt Java verschiedene Konzepte bereit, die sich durch den Umfang der Programmierung, die Geschwindigkeit, die Anforderungen an das System und die Flexibilität unterscheiden.

In der vorliegenden Arbeit werden stellvertretend

- die Low-Level Netzwerkprogrammierung,
- die Remote Procedure Calls (RPC),
- das Java RMI sowie
- CORBA

kurz vorgestellt und ihre Eignung für die prototypische Implementierung eingeschätzt. Die zu betrachtenden Konzepte umfassen nicht alle zur Zeit verfügbaren Ansätze zur Realisierung von verteilten Anwendungen, geben jedoch für diese Arbeit einen ausreichenden Einblick in verschiedenste Vorgehensweisen. Für vertiefende Informationen wird an geeigneten Stellen auf allgemeine bzw. weiterführende Literatur verwiesen.

5.2.1.1 Low-Level Netzwerkprogrammierung

Java bietet mit den im `java.net` Package zusammengefaßten Klassen eine relativ einfache Unterstützung zur netzwerkweiten Kommunikation zwischen Anwendungen, die auf Sockets basiert. Zur Herausforderung wird jedoch die Entwicklung eines komplexen Kommunikationsprotokolls und die Aufbereitung der Daten, die über das Netzwerk verschickt werden sollen.

Für die Verwendung der Low-Level Netzwerkprogrammierung spricht die sehr gute Performance und die sprachunabhängige Verknüpfung mehrerer Anwendungen. Einschränkungen sind insofern hinzunehmen, daß ein Austausch von Objekten zwischen Anwendungen, die nicht in derselben Sprache geschrieben sind, nicht möglich ist. Für den Austausch von reinen Datenströmen hingegen ist die Low-Level Netzwerkprogrammierung vollkommen ausreichend.

Für die schnelle prototypische Implementierung der Kommunikation der VDS-Dienste untereinander scheint die Low-Level Netzwerkprogrammierung relativ gut geeignet zu sein, solange die Kommunikation der Dienste untereinander einfach gestaltet wird. Im Vergleich zu anderen Ansätzen wird allerdings nur ein sehr geringes Abstraktionsniveau bereitgestellt, so daß der vom Implementierer vorzunehmende Arbeitsumfang zur Realisierung der kompletten Funktionalität, die die Überwindung der Heterogenität der beteiligten Systeme (beispielsweise in Bezug auf die Repräsentation von Daten) beinhaltet, relativ groß erscheint.

Die Netzwerkprogrammierung mit Java wird in vielen Büchern, wie beispielsweise in [HHSW97] und [Har97], anhand von Beispielen beschrieben, was besonders den Einstieg in die Problematik der Socket-Programmierung erleichtert.

5.2.1.2 Remote Procedure Calls (RPC)

Ein höheres Abstraktionsniveau als die Low-Level Netzwerkprogrammierung bieten, indem in Anwendungen scheinbar lokale Prozeduren aufgerufen werden, die in Wirklichkeit auf entfernten Servern abgearbeitet werden. Somit wird die Entwicklung einer verteilten Anwendung vereinfacht, da die internen Vorgänge bei der Kommunikation der Anwendungen untereinander bereits komplett durch das RPC-System abgedeckt wird. Somit entfällt bei der Implementierung die Eigenentwicklung eines Protokolls und beispielsweise die Berücksichtigung der heterogenen Repräsentation von Daten in den verschiedenen Systemen.

Die Abbildung 5.1 stellt eine allgemeine Vorgehensweise beim Aufruf eines RPCs dar. RPC-Systeme fassen beim Aufruf der Prozedur die Parameter zusammen, konvertieren sie in ein ein allgemeingültiges Darstellungsformat, senden sie an den Server, auf dem die aufzurufende Applikation (Prozedur) läuft, sammeln die Ergebnisse ein und senden sie an den Aufrufer zurück.

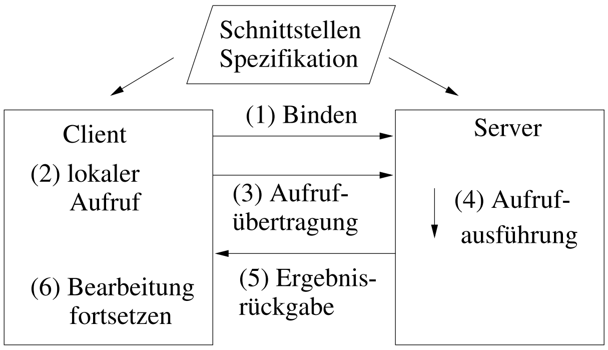

Abbildung 5.1: Grundlegender RPC-Mechanismus [Gri98]

Um einen RPC ausführen zu können, muß die Spezifikation der Schnittstelle in einer sprachunabhängigen Form unter Verwendung einer *Interface Definition Language* (IDL) erfolgen. Diese Spezifikation wird durch eine Übersetzung in die Zielsprache(n) abgebildet und ist nach der anschließenden Implementierung der Schnittstellenfunktionalität plattformübergreifend nutzbar. Vom Konzept her entspricht RPC der prozeduralen Programmierung, was nicht unbedingt für die Verwendung in einer objektorientierten Sprache spricht.

Viele weitergehende Ansätze bauen auf dem grundlegenden RPC-Konzept auf und führen Erweiterungen bzw. Abwandlungen ein, die beispielsweise wie bei Java RMI zu einer sprachgebundenen Spezifikation und einer guten Integration in die Sprache führen.

5.2.1.3 Java RMI

Eine reine Java-Implementierung stellt die *Remote Method Invocation* (RMI) [rmia] dar, die die einfache Verteilung von Objekten auf einen oder verschiedene Hosts gestattet, ohne sich um die zugrunde liegende Kommunikation beim Aufruf der Methoden dieser Objekte kümmern zu müssen. Da die zur Kommunikation erforderlichen Objektschnittstellen durch einen Compiler direkt aus der Klassendefinition erzeugt werden, muß keine separate Schnittstellenspezifikation (beispielsweise in einer IDL) erfolgen.

In Abbildung 5.2 ist die grobe Architektur eines Java-RMI-Systems dargestellt.[2] Anhand einer allgemeinen Beschreibung des Vorgangs beim Aufruf eines Remote-Objektes, also einem Objekt, daß sich nicht auf Seiten den Clients sondern des Servers befindet, wird die Funktionalität der einzelnen Schichten deutlich.

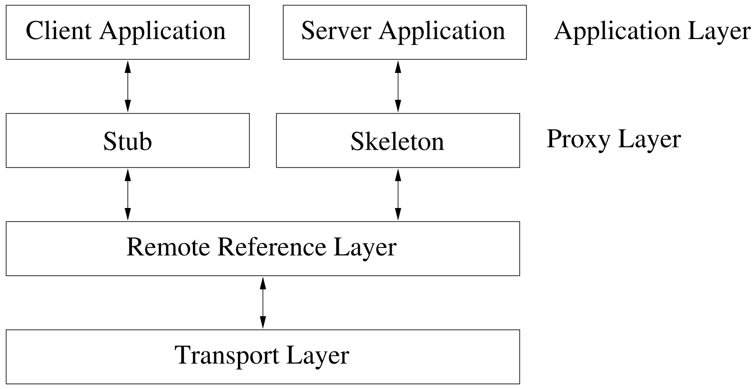

Abbildung 5.2: RMI: 4-Schichten Modell [Dow98]

Ein Client-Programm ruft die Methode eines Remote-Objektes auf, das auf einem Server vorliegt. Der Client greift dabei nicht direkt auf das Remote-Objekt zu, sondern nutzt dazu ein Proxy-Objekt (Stellvertreter), das auf Client-Seite als *Stub* bezeichnet wird. Dieses ist für den Datenaustausch und den damit verbundenen Aufgaben wie die Umwandlung (*marshaling*) der Methodenargumente in ein transportfähiges Format zuständig.

Server-seitig wird der Aufruf einer Methode des Remote-Objektes durch ein Proxy-Objekt entgegengenommen, das als *Skeleton* bezeichnet wird. Dieses dekodiert beim Aufruf die Methodenargumente, ruft die Implementierung der abzuarbeitenden Methode auf und wandelt die von der Methode zurückgelieferten Ergebnisse (Daten und ausgelöste Ausnahmen) in ein transportfähiges Format um und sendet diese via *Remote Reference Layer* an den Client zurück.

In der kurzen Beschreibung des Ablaufs eines Methoden-Aufrufes wurden nur die für das Verständnis wichtigen Aufgaben der einzelnen Teilkomponenten erwähnt. Für vertiefende Beschreibungen wie ein Remote-Objekt registriert, aufgefunden und angefordert werden kann, sei an dieser Stelle auf die bereits angegebene Literatur verwiesen.

[2] Eine detailierte Erklärung des Java RMI ist beispielsweise in [Dow98] und [rmib] nachzulesen.

Es spricht einiges für die Verwendung von Java RMI zur Kommunikation der VDS-Dienste untereinander. Im Vergleich zur Low-Level Netzwerkprogrammierung wird ein sehr hohes Abstraktionsniveau eingeführt, das die Anwendungsentwicklung sehr vereinfacht und die dazu erforderlichen notwendigen Aufgaben auf das Wesentliche — die Implementierung der durch die Objekte bereitzustellende Funktionalität und nicht der zur Kommunikation erforderliche Overhead — beschränkt. Die schnelle Anwendungsentwicklung ist besonders bei der Erstellung eines ersten Prototypen von Vorteil. Besonders interessant sind die im JDK 1.2 eingeführten persistenten (Remote-)Objekte, die bei Aufruf einer Methode durch den Client wieder aktiviert werden können [rmib, Kapitel 7].

Aufgrund der sprachgebundenen Beschreibung der Interfaces (im Gegensatz zur IDL), können nur Java-Anwendungen mittels Java RMI kommunizieren. Da jedoch ein Java-IDL Mapping [LBS98] existiert und eine IIOP-Bridge[3] zwischen IDL und RMI in Entwicklung ist, gibt es mehrere Möglichkeiten, Methoden von Anwendungen aufzurufen, die nicht in Java implementiert sind.

5.2.1.4 CORBA

Die von der *Object Management Group* (OMG[4]) eingeführte *Object Management Architecture* (OMA) ist ein Referenzmodell für verteilte, objektorientierte, heterogene Systeme.

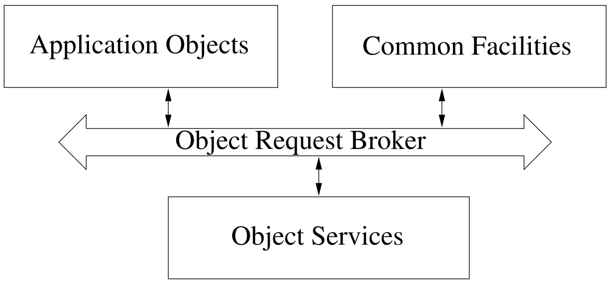

Abbildung 5.3: Object Management Architecture

Die in Abbildung 5.3 dargestellten Schlüsselkomponenten der Architektur bilden die *Object Services*[5] (Standard-Dienste wie z.B. Naming, Security, Trading usw.), die *Common Facilities* (optionale Dienste, die anwendungsspezifisch sind), die *Application Objects* (Anwendungen, die für bestimmte Aufgaben entwickelt wurden) und der *Object Request Broker (ORB)* (das Medium, das die Kommunikation zwischen den Komponenten realisiert). Die Common Object Request Broker Architecture (CORBA) [cor] ist eine konkrete Definition eines ORBs.

[3] IIOP (Internet Inter-ORB Protocol) [cor]: Protokoll zur Kommunikation zwischen unterschiedlichen ORBs (siehe auch nächster Abschnitt)

[4] `http://www.omg.org/`

[5] Ein Service (Dienst) umfaßt mehrere zusammengehörige Funktionen, die eine Dienstleistung erbringen.

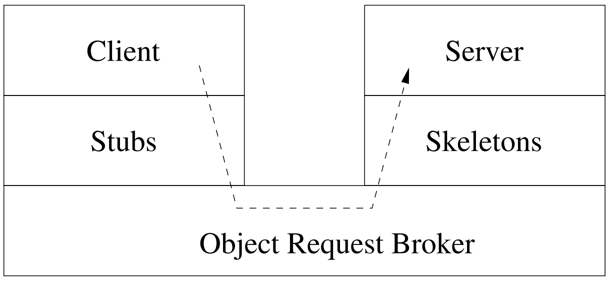

Abbildung 5.4: CORBA: Methodenaufruf

In Abbildung 5.4 wird schematisch der Prozeß des Methodenaufrufes dargestellt.

Die von Servern bereitgestellten Methoden-Interfaces müssen in einer sprachunabhängigen *Interface Definition Language* (IDL)[6] definiert werden. Ein Compiler generiert daraus in der gewünschten Sprache[7] Proxy-Objekte, die als Client-Stub und Server-Skeleton bezeichnet werden. Nach Aufruf einer Methode nutzt der Stub, der das Interface für diese Methode bereitstellt, den ORB um den Server (auf dem das Objekt vorliegt) zu ermitteln, fordert via ORB eine Verbindung an und sendet an das Skeleton die Methoden-Parameter. Das Skeleton wandelt den Aufruf und die Parameter in das von der Implementierung geforderte Format um und ruft das Objekt auf. Die Ergebnisse und eventuell ausgelösten Ausnahmen werden via ORB an den Client-Stub zurückgeliefert.

Ein Client kann die Methoden eines Objektes aufrufen, ohne wissen zu müssen, von welchem Server das Objekt bereitgestellt und wie es implementiert ist, da spezielle Dienste das Auffinden ermöglichen. Weiterhin besteht seit der in CORBA 2.0 vorgenommenen Einführung des IIOP (*Internet InterORB Protocol*) die Möglichkeit, verschiedenste herstellerabhängige Implementierungen eines ORBs miteinander zu verbinden. Dadurch kann ein Client auch auf Methoden eines Objektes zugreifen, das nicht durch den “lokalen“ ORB, sondern durch einen anderen ORB, ansprechbar ist. Für den Client ist dabei transparent, welcher ORB zum Aufruf einer Methode kontaktiert werden muß.

Der bisher gegebene kurze Einblick reicht bei weitem nicht aus, um als Einführung in die Verwendung von CORBA zu dienen. Deshalb sei an dieser Stelle auf die einführende und weiterleitende Literatur in [Sie96], [VD98], [OH98] und [LBS98] verwiesen.

Die Verwendung von CORBA als Kommunikationsbasis für den VDS bietet sehr gute Voraussetzungen, da zum einen die Erweiterbarkeit des Systems relativ problemlos vorgenommen werden kann und durch die sprachunabhängige Anbindung der Dienste die Integration in das Gesamtsystem des Projektes unterstützt

[6]In einer Interface-Definition werden öffentliche Attribute, Methoden (und ihre Parameter) und (durch Methoden ausgelöste) Ausnahmen spezifiziert.

[7]Es existieren Mappings für C, C++, Smalltalk, COBOL, Java und Ada.

und vereinfacht wird. Die Sprachunabhängigkeit führt dazu, daß jede Komponente des verteilten Systems in der Sprache implementiert werden kann, die für die Realisierung der Aufgabe am besten geeignet ist. Im VDS kann also durchaus der Query-Server in Java und der Replikations-Server in C++ implementiert werden. Die Integration der VDS-Dienste in das mit Abschluß des Projektes vorliegenden Gesamtsystem wird vereinfacht, da der Zugriff auf die einzelnen Dienste ORB-übergreifend erfolgen kann und somit keine Beschränkung auf einen bestimmten ORB erfolgen muß.

5.2.1.5 Fazit

Von den betrachteten Ansätzen scheint die Verwendung von CORBA als Basis für das verteilte System die besten Voraussetzungen zu bieten. Die Low-Level-Netzwerkprogrammierung weist nur ein geringes Abstraktionsniveaus auf und erfordert so einen erhöhten Aufwand bei der Implementierung. Das Java RMI verhindert durch die Sprachbindung die Kommunikation mit Diensten, die nicht in Java implementiert sind. Andere sprachunabhängige Ansätze, wie beispielsweise das *Inter Language-Unification System* (ILU) [ilu], die die Entwicklung verteilter Anwendungen unterstützen und somit alternativ zur CORBA eingesetzt werden könnten, wurden in der vorliegenden Arbeit nicht betrachtet. Da alle lokalen VDS-Dienste externe Dienstschnittstellen aufweisen, die die verwendete Kommunikationsbasis für die Anwendung transparent gestalten, ist ein nachträglicher Umstieg auf eine andere Kommunikationsbasis möglich.

5.2.2 Datenbankanbindung

Bei der Entwicklung von Anwendungen, bei denen zum Zugriff auf Datenbanken SQL-Kommandos eingebettet werden (*embedded SQL*), gibt es zwei Ansätze. Beide Ansätze haben gemeinsam, daß sie unabhängig vom verwendeten Datenbanksystem sind und somit Anwendungen, die auf einem bestimmten System aufsetzen, ohne Änderungen auch auf einem anderen System laufen, das den verwendeten Ansatz unterstützt. Da einige Systeme jedoch datenbankspezifische Funktionen bereitstellen, die andere Systeme nicht bieten bzw. deren Treiber nicht umsetzen kann, existieren häufig gewisse Einschränkungen bezüglich der Datenbankunabhängigkeit wenn diese Funktionen tatsächlich genutzt werden sollen. Somit muß die Unabhängigkeit von einem konkreten System häufig durch eine geringere nutzbare Funktionalität erkauft werden.

Beim ersten Ansatz, der als *dynamic SQL* bezeichnet wird, erfolgt die Einbettung des SQL-Kommandos in eine Zeichenkette, die erst bei Bearbeitung der Anfrage durch das Datenbanksystem interpretiert wird:

```
EXEC SQL("SELECT attributX FROM tableZ")
```

Diese Vorgehensweise ist zwar flexibel (da erst zur Laufzeit das auszuführende SQL-Kommando bekannt sein muß), hat aber den Nachteil, daß zur Compilezeit keine Syntax-Prüfung stattfinden kann.
Im Gegensatz dazu kann beim zweiten Ansatz (*static SQL*) das SQL-Kommando, das durch einen Pre-Compiler in die Hostsprache übersetzt wird, direkt in den Quellcode integriert werden:

```
#sql{ SELECT attributX INTO :variableY FROM tableZ };
```

Somit kann bereits zur Übersetzungszeit eine Syntaxanalyse stattfinden und eine größere Sicherheit als beim ersten Ansatz erreicht werden.

Für die Programmiersprache Java gibt es für beide Ansätze Umsetzungen. Der dynamische Ansatz wird schon seit einiger Zeit durch JDBC verwirklicht, während für den statischen Ansatz, der durch SQLJ realisiert wird, erst wenige Referenzimplementierungen vorliegen.[8]
Zum besseren Verständnis werden beide Ansätze kurz vorgestellt.

5.2.2.1 Anbindung von Datenbanken mittels JDBC

Das von JavaSoft entwickelte JDBC-API[9] [jdbc] bietet dem Java-Anwendungsentwickler eine standardisierte Schnittstelle für den Zugriff auf relationale Datenbankmanagementsysteme. Durch die Definition von Java-Klassen, die Datenbank-Verbindungen, SQL-Anfragen, Ergebnismengen und Metadaten repräsentieren, können Anwendungen unabhängig von der darunterliegenden Datenbank entwickelt werden. (Eine gute deutsche Einführung in die Nutzung von JDBC bietet beispielsweise [Dic97].)
Die zur Zeit vorhandenen JDBC-Treiber lassen sich im Allgemeinen in eine von vier Kategorien einordnen [HCF97]:

JDBC-ODBC bridge plus ODBC driver (Typ-1-Treiber). Der Zugriff auf die Datenbank erfolgt via ODBC-Treiber, d.h. die JDBC-Anfragen werden in ODBC-Anfragen übersetzt. Diese Treiber sind durch folgende Eigenschaften gekennzeichnet:

- Client-seitige Installation nativer Datenbankbibliotheken erforderlich
- Client-seitige Ausführung aller Prozesse
- Client-seitige Installation des ODBC-Treibers notwendig

[8] Die von ORACLE durchgeführte Referenzimplementierung ([sqlja]), verwendet zur Zeit zur Kommunikation mit dem Datenbanksystem den JDBC-Treiber. Durch diese Vorgehensweise kann die SQLJ-Referenzimplementierung bereits in einem frühen Entwicklungsstadium (es sind schließlich keine neuen Treiber erforderlich) und ohne großen Aufwand (die Kommunikation mit dem Datenbanksystem wird bereits durch bewährte Treiber realisiert) genutzt werden.

[9] Bei der Entwicklung des JDBC-APIs orientierte man sich an der ODBC-API (*O*pen *D*atabase *C*onnectivity) von Microsoft [odbc].

Die Verwendung dieses Treiber-Typs ist vorteilhaft, wenn auf den Client-Systemen bereits ein ODBC-Treiber installiert ist und für den Java-basierten Zugriff zusätzlich eine JDBC-Schnittstelle bereitgestellt werden soll. Eine Plattformunabhängigkeit der Clients ist nicht gewährleistet, da für das zugrunde liegende System die entsprechenden nativen Datenbankbibliotheken verfügbar sein müssen.

Native-API partly-Java driver (Typ-2-Treiber). Der Zugriff auf die Datenbank erfolgt direkt durch die Verwendung einer nativen Schnittstelle (DB-API). Gekennzeichnet sind die Typ-2-Treiber durch

- Client-seitige Installation nativer Datenbankbibliotheken
- Client-seitige Ausführung aller Prozesse
- keine Installation von ODBC-Treibern notwendig

Für diesen Treiber-Typ besteht die gleiche Plattformbeschränkung wie für die Typ-1-Treiber, da auch hier native Datenbankbibliotheken auf dem Client vorhanden sein müssen.

JDBC-Net pure Java driver (Typ-3-Treiber). Die Architektur ist dadurch gekennzeichnet, daß eine Middleware zwischen Client und Datenbank-Server eingeführt wird, die für die Umsetzung der Kommunikation mit dem Server zuständig ist. Diese Kommunikation kann via ODBC bzw. nativer Schnittstellen erfolgen. Die wesentlichen Eigenschaften dieses Treibertyps sind:

- keine Client-seitige Installation nativer Datenbankbibliotheken bzw. nativem Code notwendig
- Client- oder Server-seitige Ausführung der Prozesse (Aufteilung beliebig)
- keine Client-seitige Installation von ODBC-Treibern erforderlich

Den wohl flexibelsten Ansatz für die JDBC-Anbindung von Datenbanken bietet dieser Treiber-Typ. Eine Plattformbeschränkung wie bei den Typ-1- und Typ-2-Treibern gibt es nicht. Damit kann praktisch von jedem Client-System aus eine Datenbankanbindung erfolgen, ohne daß spezielle Voraussetzungen geschaffen werden müssen. Die Verteilung der Prozesse auf Client oder Server ermöglicht eine Anpassung an die vorhandenen Systemressourcen.

Native-protocol pure Java driver (Typ-4-Treiber). Dieser Treiber erfordert keinen Client-seitigen nativen Code, da der Zugriff auf die Datenbank direkt erfolgt. Dementsprechend positiv sind die kennzeichnenden Eigenschaften:

- keine Client-seitige Installation nativer Datenbankbibliotheken bzw. nativem Code erforderlich
- Client-seitige Ausführung der Prozesse
- kein ODBC-Treiber notwendig

Für diesen Treiber-Typ gelten im Wesentlichen die Aussagen, die zum Typ-3-Treiber getroffen wurden, wobei eine Verteilung der Prozesse auf Client und Server nicht vorgenommen werden kann.

Zusammenfassend betrachtet, bieten die Typ-3- und Typ-4-Treiber die besten Voraussetzungen für die JDBC-Anbindung von Datenbanken an den VDS, da die Plattformunabhängigkeit gewährleistet ist und das dynamische Laden der erforderlichen JDBC-Treiber durch den Client erfolgen kann. Leider stellen viele Hersteller zur Zeit nur Typ-1- und Typ-2-Treiber bereit, so daß bis zum Erscheinen der Typ-3- und Typ-4-Treiber jeder Client bestimmte Voraussetzungen (bspw. native Bibliotheken) erfüllen muß.

5.2.2.2 Anbindung von Datenbanken mittels SQLJ

SQLJ (Embedded SQL for Java) [sqlj] ist eine Umsetzung der statischen Einbettung von SQL in die Programmiersprache Java. Die Einbettung der SQL-Kommandos erfolgt durch Nutzung einiger zusätzlicher Java-Klauseln (`#sql`), die von einem Pre-Compiler in Javacode umgewandelt werden. Die Verwendung von SQLJ (als Alternative zu JDBC) weist folgende Vorteile auf:

- Syntaxprüfung bereits zur Compilezeit
- Typprüfung bereits zur Compilezeit[10]
- Schemaprüfung bereits zur Compilezeit[11]
- bessere Ausführungszeit durch vorübersetzten Code
- weniger Quell-Code erforderlich als bei vergleichbaren JDBC-Programmen
- Anfrage-Optimierungsmöglichkeiten bereits zur Compilezeit (und nicht erst bei Abarbeitung des Kommandos durch das Datenbanksystem)

Viele der hier genannten Vorteile haben positive Auswirkungen auf den Prozeß der Anwendungsentwicklung und sprechen somit für die Verwendung von SQLJ zum Zugriff auf Datenbanken, zumal in einem SQLJ-Programm auch problemlos JDBC-Statements verwendet werden können. Damit können die Vorteile beider Ansätze in einer Anwendung genutzt werden und die jeweils bessere Alternative für die konkrete Umsetzung verwendet werden.

[10] Typkompatibilität zwischen Java- und SQL-Variablen
[11] z.B. existiert das Attribut in dem Schema?

Durch SQLJ ist jedoch nicht nur eine statische Einbettung von SQL-Code in Java möglich. In der SQLJ-Planung sind beispielsweise die Verwendung von Java zur Implementierung von *Stored Procedures* (interne Datenbankfunktionen) (SQLJ Teil 1 [sql98b]) und zum Anlegen sowie den Zugriff auf nutzerdefinierte komplexe Datentypen (SQLJ Teil 2 [sql98a]) vorgesehen.

Da der Standardisierungsprozeß der Teile 1 und 2 der SQLJ-Spezifikation noch nicht abgeschlossen und demzufolge keine Referenzimplementierung verfügbar ist, sollten die weiteren Entwicklungen im Auge behalten werden. Die Nutzung nutzerdefinierter Datentypen zur Modellierung von komplex strukturierten Dokumenten und zur Definition von Dokument-Typ spezifischen Methoden, die nicht an ein konkretes Datenbanksystem gebunden sind, schafft wesentlich bessere Voraussetzungen als sie zur Zeit vorhandene Objektrelationale Datenbanksysteme bieten können.

5.2.2.3 Fazit

Die zur Zeit verfügbaren Implementierungen von JDBC und SQLJ weisen im Hinblick auf die Anfragemöglichkeiten einen gleichwertigen Umfang auf, so daß prinzipiell beide Ansätze genutzt werden können. Um eine endgültige Entscheidung für einen Ansatz treffen zu können, müßte u.a. analysiert werden, welche IRQL-Anfragen an einen Wrapper gestellt und welche daraus erzeugten Teilanfragen an das Datenbanksystem weitergereicht werden. Dadurch könnte ermittelt werden, ob die generierten Teilanfragen durch eine Menge statischer Anfragen abgedeckt werden oder ob eine dynamische Anfrageerzeugung unbedingt erforderlich ist.

In dieser Arbeit wird eine vorläufige Entscheidung zugunsten von JDBC zur Anbindung des *Fulcrum SearchServers* und *DB2* getroffen. Dies geschieht auch im Hinblick darauf, daß für den *Fulcrum SearchServer* zur Zeit keine SQLJ-Implementierung vorliegt und somit eine Zwischenlösung gewählt werden müßte, die beispielsweise auf der Oracle Referenzimplementierung [sqlja] aufbaut.

Mit den in den vorherigen Abschnitten getroffenen Entscheidungen sind für die Anbindung der lokalen VDS-Dienste und der Datenquellen einige grundlegenden Entscheidungen getroffen worden. Zum einen wird als Basis für die Kommunikation zwischen den lokalen VDS-Diensten CORBA eingesetzt und zum anderen erfolgt die Anbindung der LDS, die eine JDBC-Schnittstelle bereitstellen, unter Nutzung von JDBC. Da die anzubindenden Literaturrecherche-Dienste eine eigens entwickelte Schnittstelle zum Zugriff auf die Daten bereitstellen, muß diese genutzt werden.

Im folgenden Abschnitt wird allgemein die prototypischen Implementierung des Query-Servers beschrieben.

5.3 Implementierung des Query-Servers

Der Query-Server stellt einen wesentlichen Bestandteil der im Rahmen dieser Arbeit vorgenommenen Implementierung dar. Entsprechend der in Abschnitt 3.2.1 festgelegten Architektur und den bereits spezifizierten Dienstschnittstellen werden in den folgenden Abschnitten die im Rahmen der Implementierung getroffenen Entscheidungen zur Umsetzung

- des Query-Interfaces,
- der Anfragebearbeitung,
- des Ressourcen-Interfaces,
- der Ressourcen-Verwaltung und
- des Replikations- und Caching-Dienst-Interfaces

dokumentiert.

Bei der Umsetzung des Query-Servers spielt die parallele Bearbeitung der eingehenden IRQL-Anfragen eine wesentliche Rolle, da an das System durch Nutzer und externe Dienste gleichzeitig eine Vielzahl von Anfragen gestellt werden können. Eine serielle Bearbeitung der eingehenden Anfragen ist aufgrund der relativ hohen Antwortzeit der LDS und der daraus resultierenden Gesamtantwortzeit des Query-Servers nicht sinnvoll.

Die Umsetzung der Parallelität erfolgt mit Hilfe von Threads. Als Thread wird die parallele Ausführung von Programmcode innerhalb eines Prozesses bezeichnet [KSS96]. Da Java eine Multithread-fähige Sprache darstellt, die die Verwendung von Threads durch die Bereitstellung einer eigenen Klasse (`java.lang.Thread`) vereinfacht, wird in der Implementierung auf diese zurückgegriffen.

Durch die parallele Ausführung von Programmcode steigt jedoch der Aufwand, der zur Wahrung der Konsistenz von Daten, auf die mehrere Threads lesend und schreibend zugreifen, erforderlich ist. So ist beispielsweise zu berücksichtigen, daß der Replikations- und Caching-Dienst Updates auf den repliziert vorliegenden Dokumenten durchführen kann. Eine kurze Diskussion über diese Problematik wird in Abschnitt 5.4 vorgenommen.

Für Informationen über die Thread-Programmierung sei auf die entsprechende Literatur (z.B. allgemeine Thread-Programmierung in C [KSS96], Threads-Tutorial für Java [threads]) verwiesen.

5.3.1 Query-Interface

Bei der Implementierung des Query-Interfaces, das die gleichzeitige Bearbeitung von Anfragen mehrerer Nutzer oder externer Dienste unterstützt, wurde der in Abbildung 5.5 dargestellte Ansatz gewählt. Für jede eingehende Anfrage wird von einem Master-Thread ein eigener Thread gestartet, in dem die Anfragebearbeitung (siehe nächster Abschnitt) durchgeführt wird (in der Abbildung links). Nach erfolgreicher oder -loser Bearbeitung der Anfrage und Rückgabe der Ergebnisse wird dieser Thread vom Master-Thread wieder beendet.

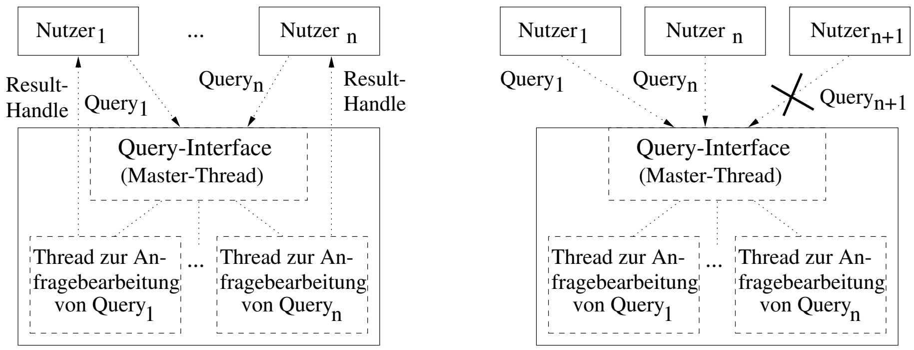

Abbildung 5.5: Multi-Threaded Query-Server – das Query-Interface

Die Aufgabe des Master-Threads besteht im Empfang der Anfragen, der Erzeugung, Verwaltung und Terminierung der Anfragebearbeitungs-Threads. Dabei gilt, daß gleichzeitig nur eine fest vorgegebene maximale Anzahl von Query-Requests durch den Master-Thread bearbeitet werden darf, damit das System nicht überlastet wird. Die Ermittlung eines optimalen Wertes hängt von den verfügbaren Systemressourcen und der Auslastung des Hosts ab. Steigt die Anzahl gleichzeitig zu bearbeitender Queries über das vorgegebene Maximum an, so wird eine Bearbeitung der über den Maximalwert hinausgehenden Anfragen abgelehnt (in der Abbildung rechts). Diese Anfragen müssen dementsprechend zu einem späteren Zeitpunkt erneut gestellt werden.

Eine Verbesserung dieses Problem-Verhaltens stellt die Einführung einer Warteschlange dar, die die eingehenden Anfragen sammelt und je nach Verfügbarkeit des Systems zur Bearbeitung freigibt. Im ersten Prototypen ist die Implementierung einer Warteschlange jedoch nicht vorgesehen. Durch Tests, die zum jetzigen Zeitpunkt aufgrund des frühen Entwicklungsstadiums noch nicht vorgenommen werden konnten, sollte ermittelt werden, wie häufig die Abweisung einer Anfrage in der Praxis vorkommen würde. In Abhängigkeit vom ermittelten Wert kann eine Entscheidung für oder gegen die Einführung der Warteschlange getroffen werden.

5.3.2 Anfragebearbeitung

Die Anfragebearbeitung des Query-Servers wird für jeden der vom Master-Thread des Query-Interfaces erzeugten Threads getrennt durchgeführt. Durch die parallele Bearbeitung der Anfragen wird eine Verbesserung der Gesamtantwortzeit des Query-Servers im Vergleich zur sequentiellen Bearbeitung erreicht. Die prototypische Umsetzung der Anfragebearbeitung des Query-Servers wird in den folgenden Abschnitten vorgestellt.

Metadaten-Retrieval-Anfragebearbeitung

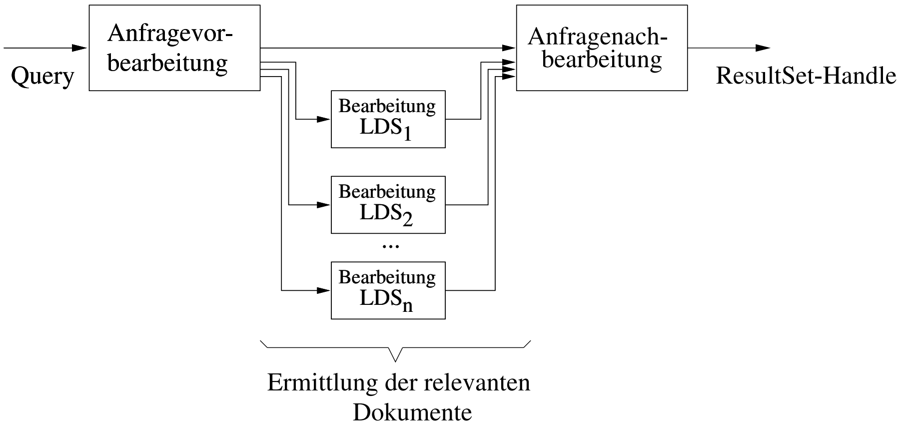

Abbildung 5.6: Bearbeitung einer Metadaten-Retrieval-Anfrage im Query-Server

Die in Abbildung 5.6 dargestellte Anfragebearbeitung einer Metadaten-Retrieval-Anfrage setzt sich aus der Anfragevorbearbeitung, der Anfrageausführung durch die LDS und der Anfragenachbearbeitung zusammen. Während der Anfragevorbearbeitung wird die in der IRQL spezifizierte Anfrage analysiert, eventuell Optimierungen vorgenommen und die Query-Dekomposition durchgeführt. Die ermittelten Teilanfragen werden parallel durch die kontaktierten LDS bearbeitet und die homogenisierten Ergebnisse einer Nachbearbeitung unterzogen. Das Ergebnis einer Metadaten-Retrieval-Anfrage bildet ein Handle zum Zugriff auf die ermittelten relevanten Daten.

Die vorliegende Implementierung der Anfragevorbearbeitung von Metadaten-Retrieval-Anfragen beschränkt sich auf die Dekomposition einfacher Anfragen der Form

```
SELECT attributListe
FROM ressourceName1 UNION ... UNION ressourceNameN
WHERE condition
```

wobei nur einige ausgewählte Selektionsprädikate durch die Wrapper umgesetzt werden. Die implementierte Nachbearbeitung der Anfragen umfaßt nur das Mischen der Ergebnisse[12], wobei kein globaler Ranking-Wert ermittelt wird (und somit keine globale Rangfolge, die durch diesen Wert festgelegt ist).

Das an den Nutzer zurückgelieferte Ergebnislisten-Handle wurde prototypisch umgesetzt, so daß zur Zeit die kompletten Ergebnisse mit dem Handle zum Query-Server übertragen werden.

Da zur Zeit keine externen VDS-Dienste prototypisch vorliegen, beschränkt sich die Anfragebearbeitung vorerst auf lokale VDS-Dienste. Eine Optimierung der Anfragen wird im vorliegenden Prototypen noch nicht vorgenommen.

[12]Dazu wird das Round-Robin-Verfahren genutzt.

Auswirkungen der Heterogenität der LDS auf die globale Anfragebearbeitung. Daß es sich bei den LDS um heterogene Systeme handeln kann, deren Anfragefunktionalität nicht für die Umsetzung von allen IRQL-Anfragen ausreichend ist, die Operationen auf bestimmte Attribute beschränken oder anderweitige Einschränkungen aufweisen, muß bei der Bearbeitung von Anfragen durch den Query-Server berücksichtigt werden.

Um zu ermitteln, ob ein System eine gegebene Anfrage bearbeiten kann, muß bekannt sein, welche Anfragefunktionalität durch dieses bereitgestellt wird; eine Beschreibung der unterstützten Anfragefunktionalität ist demzufolge notwendig. Diverse Untersuchungen und Projekte haben sich mit dieser Problematik beschäftigt. Zwei Ansätze, die sich vielleicht in ähnlicher Form auf die Anfragebearbeitung im VDS übertragen lassen, seien an dieser Stelle kurz erwähnt. Während in dem einen Ansatz eine einfache Spezifikation der durch die LDS bereitgestellten Anfragefunktionalität erfolgt, verzichtet der zweite Ansatz auf eine solche Spezifikation.

In [GMY98] werden jedem Attribut der Datenquelle Eigenschaften zugeordnet, die beschreiben, wie es in einer Anfrage verwendet werden darf. Beispielsweise könnte eine Eigenschaft festlegen, daß ein Attribut in einer Anfrage auftreten muß. Die Anfragefunktionalität einer Datenquelle wird letztendlich durch eine Menge von Templates spezifiziert, wobei ein Template für eine Anfrage festlegt, welche Attribute in dieser auftreten können und welche Einschränkungen für ihre Verwendung gelten. Durch Templates kann also festgelegt werden welche Attribute in der Selektionsbedingung einer Anfrage oder in der Projektionsliste der Anfrage auftreten dürfen. Anhand dieser Templates kann von der zentralen Anfragebearbeitungskomponente demnach entschieden werden, ob eine Anfrage von einer bestimmten Datenquelle bearbeitet werden kann oder nicht. Ist anhand der Templates ermittelt worden, daß die Bearbeitung der ursprünglichen Anfrage in der spezifizierten Form nicht möglich ist, kann sie beispielsweise durch Dekomposition in Teilanfragen zerlegt werden, deren Bearbeitung durch die Datenquelle unterstützt wird.

Ein anderer Ansatz bei der Anfragebearbeitung wurde im *Garlic*-System [garlic] gewählt. In diesem System besitzt die Anfragebearbeitungskomponente keine Informationen über die von den Datenquellen unterstützten Anfragefunktionalitäten [RS97]. Um eine Anfrage bearbeiten zu können, werden in Zusammenarbeit mit den Wrappern für eine Anfrage mehrere Anfragepläne generiert, wobei die Wrapper entscheiden, welchen Teil der Anfrage sie bearbeiten können. Somit wird durch die Wrapper gekapselt, welche Anfragefunktionalität sie bereitstellen, was die Erweiterbarkeit und die Einbindung beliebiger neuer Datenquellen vereinfacht.

Eine Einschätzung, welcher der Ansätze geeigneter ist, soll in dieser Arbeit nicht getroffen werden, da dazu eine genauere Untersuchung erforderlich ist. In der prototypischen Implementierung wurde davon ausgegangen, daß der Query-Server die Anfrage durch alle aus der Anfrage ermittelbaren LDS bearbeiten läßt.

Die Wrapper liefern

- die Ergebnisse der durch sie vollständig umgesetzten und bearbeiteten Metadaten-Retrieval-Anfrage oder
- keine Ergebnisse, da die Anfrage nicht bearbeitet werden kann.

Damit treffen zur Zeit letztendlich die Wrapper die Entscheidung, welche Anfrage sie bearbeiten können.

Die Homogenität in der Struktur der zurückgelieferten Ergebnisse wird durch die Wrapper sichergestellt, die eine Transformation der von der Datenquelle zurückgelieferten Ergebnisse in das innerhalb eines VDS verwendete globale Modell zur Repräsentation der Ergebnisse vornehmen.

Dokument-Retrieval-Anfragebearbeitung

Die in Abbildung 5.7 skizzierte Bearbeitung von Dokument-Retrieval-Anfragen erfordert vom Query-Server die Weiterleitung der Dokument-Anforderungen an den Replikations- und Caching-Dienst, so daß lediglich ein Aufruf der entsprechenden Schnittstelle und eine Auswertung der Rückgabe erfolgen muß. Das vom Replikations- und Caching-Dienst zurückgelieferte Handle zum Zugriff auf das Dokument wird vom Query-Server an den Auslöser der Anfrage weiterreicht, damit dieser auf das Dokument oder Teile davon zugreifen kann.

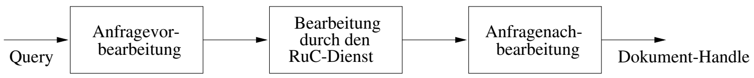

Abbildung 5.7: Bearbeitung einer Dokument-Retrieval-Anfrage im Query-Server

Da sich die Implementierung des Replikations- und Caching-Dienstes mit Abschluß dieser Arbeit noch im Anfangsstadium befindet, konnte die Umsetzung der Dokument-Retrieval-Anfragebearbeitung und des Dokument-Handles nur partiell erfolgen.

Hybride Anfragebearbeitung

In der Analyse der Anfragebearbeitung des VDS in Abschnitt 2.3.1 wurde eine Trennung der an einen Query-Server gestellten Anfragen in Dokument-Retrieval-Anfragen und Metadaten-Retrieval-Anfragen vorgenommen. Durch diese Trennung kann die in der Praxis verwendete Vorgehensweise bei der Suche nach Dokumenten realisiert werden. Dabei will der Nutzer in der Regel nicht sofort die ermittelten Dokumente selbst betrachten, sondern ist an einem Überblick über die Ergebnisse und eine Kurzzusammenfassung der Dokumente (Abstrakt usw.)

interessiert.[13] Erst in einem zweiten Schritt werden vom Nutzer die kompletten Dokumente oder die in der Anfrage ermittelten relevanten Teile der Dokumente angefordert.

Da in der IRQL beide Anfrageformen in einer Anfrage kombiniert auftreten können, ist zur Bearbeitung einer solchen Anfrage eine Zerlegung in mehrere Teilanfragen erforderlich, wobei im ersten Schritt Metadaten-Retrieval-Anfragen zur Ermittlung der relevanten Dokumente und im zweiten Schritt eine Menge von Dokument-Retrieval-Anfragen zum Zugriff auf die ermittelten Dokumente bzw. Teildokumente durchgeführt werden (Abbildung 5.8).

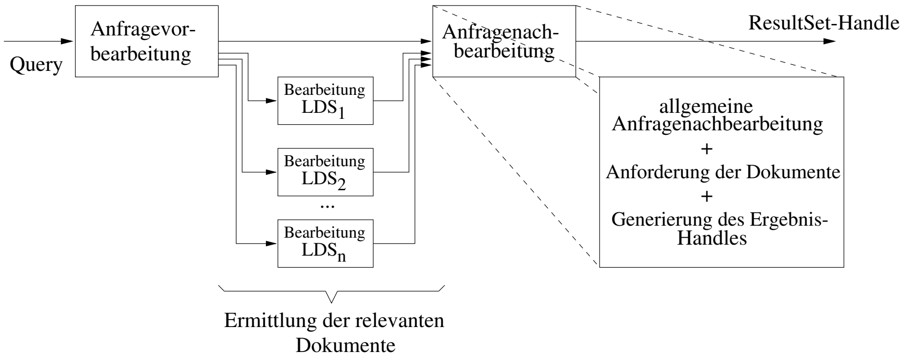

Abbildung 5.8: Bearbeitung einer hybriden Anfrage

Ein kurze IRQL-Beispielanfrage, in der alle Dokumente und ihre Titel von allen lokalen LDS angefordert werden, in deren Titel die Phrase *camel case* auftritt, verdeutlicht die Vorgehensweise bei der Bearbeitung einer solchen Anfrage:

```
SELECT title, complete_content
FROM local
WHERE title CONTAINS 'camel case'
```

Als Ergebnis der Analyse dieser Anfrage würden im Rahmen der Anfragebearbeitung Anfragen an alle ermittelten LDS gesendet werden, in denen die Ermittlung der Titel und DOIs der relevanten Dokumente erfolgt. Für einen LDS könnte diese Anfrage beispielsweise so formuliert werden:

```
SELECT title, docId
FROM sourceId
WHERE title CONTAINS 'camel case'
```

[13]Es hängt von der Güte der verwendeten Retrieval-Algorithmen ab, wie groß der Anteil der auf eine Anfrage zurückgelieferten nicht-relevanten bzw. nicht zurückgelieferten relevanten Dokumente ist. Aufgrund der qualitativ schlechten Ergebnisse (sei es aufgrund mangelhafter Anfrageformulierung, Mächtigkeit der Sprache oder verwendeter Algorithmen) muß der Nutzer deshalb zusätzlich anhand der zurückgelieferten Dokument-Metadaten eine Selektion der aus seiner Sicht relevanten Dokumente aus der vom System ermittelten Menge bestimmen.

Der Query-Server erhält als Ergebnis der Anfrage ein Handle zum Zugriff auf die Elemente der Ergebnisliste. Für jedes Element der Liste (also die Dokumente, die das Selektionsprädikat erfüllen) wird daraufhin vom Replikations- und Caching-Dienst ein Handle zum Zugriff auf das entsprechende Dokument angefordert.

Das vom Query-Server erzeugte und zurückgelieferte Ergebnislisten-Handle (Abbildung 5.9) besitzt also Methoden zur Navigation innerhalb der Ergebnisliste (a) und für jedes Element der Liste Methoden zum Zugriff auf die angeforderten Attribute (b) und Dokument-Inhalte (c).

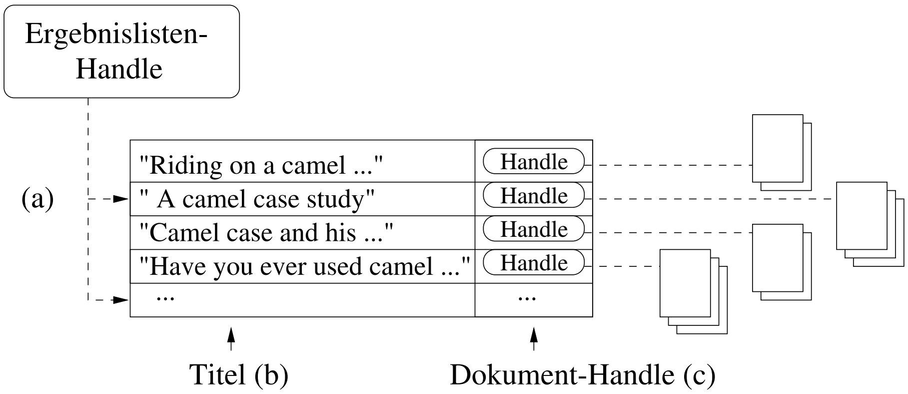

Abbildung 5.9: erzeugtes Ergebnislisten-Handle einer hybriden Anfrage

Um erkennen zu können, ob es sich bei einer Anfrage um eine Metadaten-Retrieval-Anfrage, eine Dokument-Retrieval-Anfrage oder eine hybride Form handelt, müssen unter anderem die Attribute der Projektionsliste analysiert werden.[14] Ist das Zurückliefern aller Attribute der Projektionsliste ohne eine Autorisierung oder Abrechnung gestattet, handelt es sich um eine Metadaten-Retrieval-Anfrage. Treten in der Projektionsliste nur Attribute auf, deren Auslieferung eine Autorisierung oder Abrechnung erfordert, handelt es sich um eine Dokument-Retrieval-Anfrage. Wenn keine der beiden Voraussetzungen zutrifft, stellt die Anfrage eine hybride Anfrageform dar.

In der vorliegenden Implementierung wurde eine Festlegung der in einer Anfrage spezifizierbaren Attributnamen vorgenommen, so daß die Einordnung, ob der Zugriff auf ein Attribut eine Dokument-Retrieval-Anfrage erfordert, auf einfache Art und Weise erfolgen kann. Der Einfachheit halber wurde nur ein Attribut festgelegt, für das dies der Fall ist: `complete_content` — das komplette Dokument.

Damit sind einige der für diese Arbeit wesentlichen Aspekte der Implementierung der Anfragebearbeitung im Query-Server beschrieben worden. Im folgenden

[14] Eine Betrachtung der Attribute der Selektionsbedingung ist nicht erforderlich, da für die Verwendung der Attribute in den Selektionsbedingungen laut der für diese Arbeit getroffenen Annahmen keine Autorisierungspflicht besteht. Siehe dazu die Erläuterungen im Abschnitt “Bearbeitung der Anfrage“ auf Seite 16.

Abschnitt wird kurz angerissen, wie die Anbindung der LDS und des Replikations- und Caching-Dienstes an den Query-Server erfolgt.

5.3.3 Ressourcen-Interface / Replikations- und Caching-Dienst-Interface

Die Implementierung der Schnittstellen zum Zugriff auf die Wrapper der LDS und das Replikations- und Caching-Dienst-Interface beschränkt sich auf den Aufruf der durch die Wrapper bereitgestellten Methoden und die Auswertung der zurückgelieferten Ergebnisse bzw. Ausnahmen. Da CORBA als Grundlage für die Kommunikation der VDS-Dienste untereinander gewählt wurde, erfolgt der Zugriff auf die Methoden der VDS-Dienste durch den Aufruf der Methode des entsprechenden Proxy-Objektes im Query-Server. Eine Beschreibung der Implementierung, wie dieser Zugriff geschieht, soll nicht gegeben werden, da dieser durch die Middleware-Schicht (also CORBA) geregelt wird.

5.3.4 Ressourcen-Verwaltung

Die Entscheidung CORBA als Basis für die Entwicklung des VDS zu wählen, führt dazu, daß ein Teil der in der Spezifikation festgelegten Funktionalität der Ressourcen-Verwaltung bereits durch einen CORBA-Dienst übernommen werden kann.

Dabei handelt es sich um den *Naming Service*. Bei diesem registrieren sich Dienste (Server), die ihre Dienstleistungen (im Falle der Wrapper bspw. die Bearbeitung von Anfragen durch konkrete Datenquellen) damit öffentlich zur Verfügung stellen. Ein Client kann anhand der vom *Naming Service* bereitgestellten Informationen die Dienste eines Servers in Anspruch nehmen.

Bei der Neuanmeldung eines LDS registriert sich der Wrapper beim *Naming Service* und kann somit von jedem der VDS-Dienste (Query-Server und Replikations- und Caching-Dienst) angesprochen werden. Die dauerhafte Abmeldung eines Dienstes kann ebenfalls mit Hilfe der durch einen Naming-Service bereitgestellten Funktionalität realisiert werden.

Um die komplette in Abschnitt 3.2.1.2 beschriebene Funktionalität der Ressourcen-Verwaltung nutzen zu können und nicht zu sehr von CORBA abhängig zu sein, wird jedoch ein eigener Dienst (der den Naming-Service nutzt) bereitgestellt.

5.4 Implementierung der Wrapper

Die Bereitstellung der Wrapper-Basisfunktionalität stellt den zweiten und letzten Kernpunkt der im Rahmen der vorliegenden Arbeit vorgenommenen Implementierung dar. Ein Wrapper stellt einen Server-Prozeß dar, der für Clients — die anderen lokalen VDS-Dienste — die Bearbeitung von Requests durchführt. Ein

Request symbolisiert dabei den Aufruf einer vom Wrapper bereitgestellten Methode zur Bearbeitung von

- Dokument-Anforderungen durch den Replikations- und Caching-Dienst,
- Metadaten-Retrieval-Anfragen,
- Exportaufforderungen der statischen/dynamischen-Metadaten des LDS sowie
- weiteren hier nicht beschriebenen Verwaltungsoperationen am Datenbestand, die durch den Replikations- und Caching-Dienst ausgelöst werden.

Für die Bearbeitung von Requests gelten im Wesentlichen die in Abschnitt 5.3.1 getroffenen Aussagen bezüglich der parallelisierten Ausführung: für jeden eingehenden Request wird, wie in Abbildung 5.10 dargestellt, ein eigener Thread gestartet.

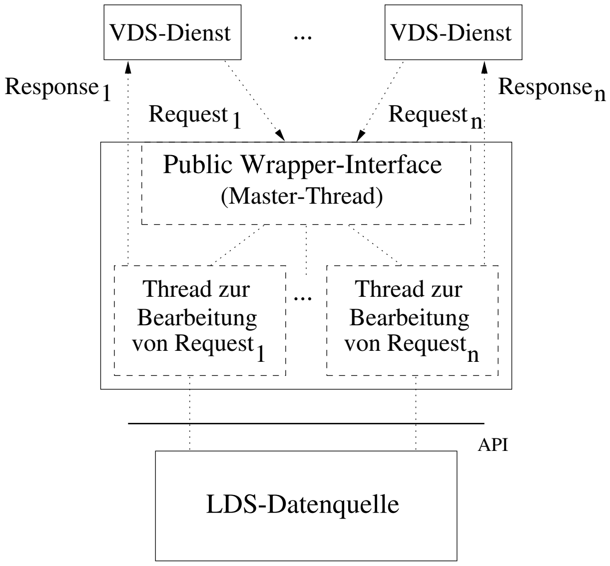

Abbildung 5.10: Multi-Threaded Wrapper

Die durch einen Wrapper zu bearbeiteten Requests lösen Methoden aus, die lesend und/oder schreibend auf dem Datenbestand der zugrunde liegenden Datenquelle operieren. Requests, die schreibende Operationen in der Datenquelle nach sich ziehen, werden nur durch den Replikations- und Caching-Dienst ausgelöst. Diese Art von Requests kann nur an den (oder die) LDS gerichtet werden, auf den (oder denen) der Replikations- und Caching-Dienst eine volle Zugriffskontrolle besitzt. Es handelt sich dabei in der Regel um dedizierte LDS, die nur Dokument-Replikate verwalten.

Da Requests parallel bearbeitet werden und eine vom Replikations- und Caching-Dienst ausgelöste Änderungsoperation auf dem Dokumentenbestand sich aus mehreren Teiloperation zusammensetzen kann, muß ein Transaktionskonzept bereitgestellt werden, das die Konsistenz der Daten gewährleistet.[15] Da diese Problematik sehr eng mit der Funktionalität des Replikations- und Caching-Dienstes verknüpft ist, wird in der vorliegenden Arbeit auf eine Betrachtung verzichtet und auf weiterführende Arbeiten verwiesen.

Eine genaue Beschreibung, wie die Implementierung der Wrapper erfolgt ist, wird in dieser Arbeit nicht gegeben. Stattdessen werden einige grundlegende Voraussetzungen dargelegt, die für die Anbindung und Verwendung

- des Volltextdatenbanksystems Fulcrum,
- der Harvest-basierten Literatur-Recherche-Dienste (MathNet, PhysNet) sowie
- von DB2

als LDS zu schaffen sind.

Jeder der implementierten Wrapper bietet eine Basisfunktionalität, die den Empfang, die Analyse, die Weiterleitung an die Datenquelle und die Nachbearbeitung von Anfragen umfaßt. Aus Zeitgründen wurde lediglich die Umsetzung von einfachen Anfragen der Form

```
SELECT attributListe
FROM ressourceName
WHERE condition
```

realisiert. Da drei der vier in Kapitel 4 untersuchten Selektionsprädikate der IRQL nicht von allen Systemen ohne Simulation der Funktionalität umgesetzt werden können, wurde vorerst nur das Prädikat zur einfachen Suche (CONTAINS) auf die entsprechenden Konstrukte der Anfragesprache der Systeme abgebildet.

5.4.1 Anbindung des Volltextdatenbanksystems Fulcrum

Zur Anbindung des Volltextdatenbanksystems Fulcrum wurde das *SearchBuilder for Java* Entwicklungswerkzeug [sbj98] zur JDBC-Anbindung verwendet.[16] In Abbildung 5.11 ist schematisch die Anbindung eines Wrappers an einen *SearchServer* dargestellt.

Durch Nutzung der vom Entwicklungswerkzeug bereitgestellten Java-Klassen erfolgt der Aufbau der Verbindung, das Senden von SQL-Kommandos an den *SearchServer*, die Ergebnismengenverwaltung sowie das Retrieval der Dokumente. Der JDBC-Treiber realisiert dabei via API die Kommunikation des Wrappers mit dem Datenbanksystem.

[15] Eventuell kann dabei auf den von CORBA bereitgestellten Transaktionsdienst [ots98] zurückgegriffen werden.

[16] Eine ODBC-Anbindung ist mit Hilfe von C [sbc98], Visual C++ und Visual Basic [sbb98] möglich.

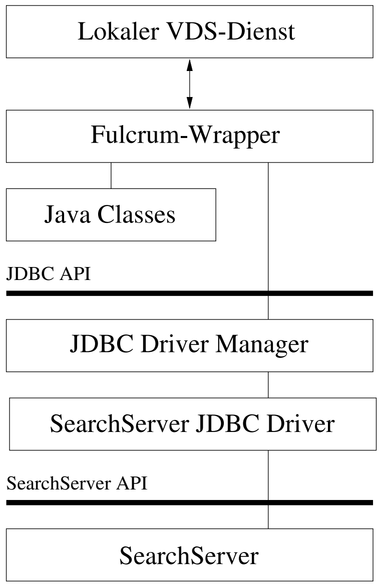

Abbildung 5.11: JDBC-Anbindung an einen Fulcrum SearchServer

Bemerkungen

Um die Anbindung des *Fulcrum SearchServers* zu ermöglichen, sind die folgenden Voraussetzungen zu schaffen:

Client-seitiger nativer Code erforderlich. Beim für Fulcrum bereitgestellten JDBC-Treiber handelt es sich um einen *Native-API partly Java driver* (Typ-2-Treiber), der erfordert, daß auf dem Client-Rechner binärer Code (hier in Form von shared libraries) vorliegt.

Da zur Zeit für den Fulcrum SearchServer keine Typ-3- oder Typ-4-Treiber angeboten wird, müssen diese Bibliotheken auf dem Client verfügbar sein, auf dem der Fulcrum-Wrapper läuft.

Umgebungsvariablen. Der vorhandene Treiber erfordert, daß für die Bearbeitung von Anfragen einige Fulcrum-Umgebungsvariablen gesetzt sind. Da in Java das Konzept der Umgebungsvariablen durch Properties[17] ersetzt wurde, um die Systemunabhängigkeit zu gewährleisten, können diese Variablen nicht durch den Wrapper gesetzt werden.

Da ein Wrapper einen Server-Prozeß darstellt, der bis zum gewollten bzw. ungewollten Shutdown aktiv ist, reicht das einmalige Setzen der Umgebungsvariablen in einer Shell-Umgebung vor dem Start des Wrappers.

Zum Zeitpunkt der Bearbeitung dieser Arbeit gelten für die Verwendung des *Fulcrum SearchServers* folgende Einschränkungen, die bei der Verwaltung der Dokumente zu berücksichtigen sind:

[17] Siehe dazu [Fla98, Kapitel 14:Systemeigenschaften].

Kein Schlüsselkonzept. Bei der Verwaltung der Dokumente zu berücksichtigen, daß dem *SearchServer* kein Schlüsselkonzept bekannt ist[18], mit dem sichergestellt werden kann, daß ein vergebener DOI innerhalb einer Relation eindeutig ist. Somit hat der Administrator sicherzustellen, daß eine eindeutige Identifikation eines Dokumentes möglich ist.

XML-Unterstützung. Da für den Fulcrum SearchServer zur Zeit kein XML-Textreader[19] verfügbar ist, wurden die Testdaten semiautomatisch generiert. Die Entwicklung eines XML-Textreaders [sbt98] (der beispielsweise auf dem existierenden SGML-Textreader aufsetzt) bildet eine Aufgabe, die in dieser Arbeit nicht durchgeführt werden kann, da dieser erst kostenpflichtig erworben werden muß. Eine Einschätzung des Umfangs der notwendigen Implementierungen kann deshalb auch nicht vorgenommen werden.

Die Anbindung des *Fulcrum SearchServers* wirft keine Probleme auf. Für die Verwaltung der Dokumente und die Extraktion von Metadaten aus XML-Dokumenten durch den *SearchServer* ist hingegen noch nicht die erforderliche Funktionalität verfügbar, so daß diese in weiterführenden Arbeiten umgesetzt werden muß.

5.4.2 Anbindung Harvest-basierter Systeme

Die Anbindung der Harvest-basierten Systeme wurde unter Nutzung der von Java im Package `java.net` bereitgestellten Klassen vorgenommen. Durch die Verwendung der Low-Level-Netzwerkprogrammierung erfolgt der direkte Zugriff auf die (fast undokumentierte) Schnittstelle des Brokers.

Bei der Bearbeitung einer Anfrage wird, wie aus Abbildung 5.12 ersichtlich, die transformierte Anfrage (2) an den Harvest-Broker weitergereicht (3a), der anhand der bei der Indexierung gewonnenen Metadaten (1) die Anfragebearbeitung durchführt. Wenn es sich bei der Anfrage um die Anforderung eines Dokumentes handelt, dann kann diese nicht durch den Broker bearbeitet werden, da dieser keinen Zugriff auf das Dokument besitzt.[20] Deshalb kontaktiert der Wrapper zum Retrieval eines Dokumentes den WWW-Server (3b), auf dem dieses Dokument gespeichert ist, und nicht den Broker. Dazu wurde bei der Implementierung wiederum auf die von Java bereitgestellten Klassen zur Netzwerkprogrammierung zurückgegriffen.

[18] In Abschnitt 4.2.1 wurde bereits erwähnt, daß die bereitgestellte Datenbankfunktionalität sehr eingeschränkt ist.

[19] Der Fulcrum SearchServer verwendet Textreader zur Indexierung und zum Retrieval der extern gespeicherten Dokumente [ssd98, Kapitel 5: Using External Text]. Durch dieses Konzept können praktisch aus jedem Dokument-Format, für das ein Textreader existiert, Metadaten extrahiert werden.

[20] Siehe dazu auch die Vorstellung des Harvest-Systems in der dieser Diplomarbeit vorangegangenen Studienarbeit [Tit98].

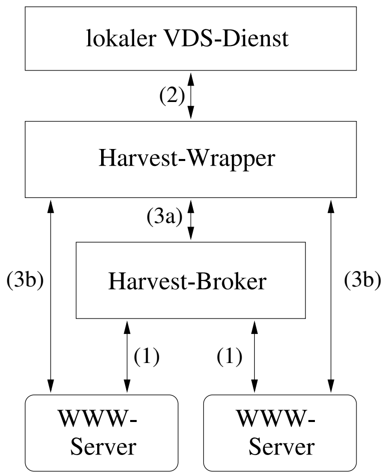

Abbildung 5.12: Anbindung von Harvest-basierten Systemen

Bemerkungen

Für die Anbindung eines Wrappers an ein Harvest-System sind keine Voraussetzungen erforderlich. Für die Nutzung eines Harvest-Systems als Lokalen Dokumenten-Server sind hingegen die folgenden Voraussetzungen zu schaffen:

Integration eines DOI. In der Identifikation der Anforderungen an einen LDS in Kapitel 2 wurde für die Identifikation eines Dokumentes ein logischer Schlüssel (DOI) und für die Lokalisierung eines Dokumentes ein physischer Schlüssel eingeführt.

Der Harvest-Wrapper greift unter Verwendung der einem Dokument zugeordneten URL auf den Inhalt zu. Demnach könnte der physische Schlüssel, der dem Wrapper bei der Anforderung eines Dokumentes übergeben wird, aus dem entsprechenden URL gebildet werden. Eine gewisse Abstraktion wird hingegen eingeführt, wenn dem Schlüssel anstatt der URL ein synthetisch erzeugter Wert zugeordnet wird, anhand dessen der Harvest-Wrapper auf geeignete Weise den URL ermitteln und daraufhin das Dokument anfordern kann. Somit wird eine Transparenz eingeführt, die in diesem Fall verbirgt, wo genau das Dokument gespeichert ist (Servername+Verzeichnis) und wie darauf zugegriffen werden kann (zu verwendendes Protokoll). Der positive Aspekt dieses Ansatzes ist, daß damit für alle LDS eines VDS diese Transparenz besteht, egal auf welchem System sie aufsetzen.[21]

Ein eindeutiger physischer Schlüssel, der dieser Transparenz genügt, kann

[21] Auf Dokumente, die beispielsweise durch einen *Fulcrum SearchServer* indexiert wurden, kann von externen Anwendungen in der Regel nur durch Verwendung der von Fulcrum bereitgestellten Funktionen zugegriffen werden, selbst wenn bekannt ist, wo das Dokument gespeichert ist.

beispielsweise während der Indexierung durch den Gatherer[22] für jedes Dokument generiert werden. Diese Vorgehensweise ist jedoch nicht trivial und erfordert die Kooperation des Harvest-Administrators. Eine andere Vorgehensweise bildet die Vergabe, persistente Speicherung und Verwaltung der Schlüssel durch den Harvest-Wrapper. Dabei wird davon ausgegangen, daß vor der Anforderung eines Dokumentes eine Metadaten-Retrieval-Anfrage gestellt worden ist. Für die dabei zurückgelieferten Dokumentreferenzen werden vom Wrapper die eindeutigen synthetischen Schlüssel zum Zugriff vergeben und die Zuordnung Schlüssel $\longrightarrow$ URL vom Wrapper gespeichert. Das Laden eines Dokumentes, für das ein synthetischer Schlüssel vergeben wurde, ist nun problemlos möglich.

XML-Unterstützung. Zur Extraktion der Dokument-Metadaten eines Dokumentes verwendet das Harvest-System eine Komponente namens *Gatherer*. Diese Komponente kann anhand von vorgegebenen Regeln den Typ eines Dokumentes bestimmen und durch Anwendung eines für diesen Typ entwickelten *Summarizers* die Dokument-Metadaten extrahieren. Durch Verwendung eines SGML-Summarizers und einer Spezifikation der zu verwendenden DTD kann dies für XML-Dokumente geschehen [HSW96, Abschnitt 4.4: Extracting data for indexing]. Nachdem die XML-Dokument-Metadaten indexiert sind, kann die Suche auf diesen Daten durchgeführt werden.

Die Anbindung von Harvest-basierten Systemen erweist sich als relativ problemlos; die erforderliche Implementierung jedoch aufgrund der geringen Unterstützung durch das System als umfangreich. Die Bereitstellung von XML-Dokumenten durch die Literatur-Recherche-Dienste kann zur Zeit nicht realisiert werden, da die Integration von XML-Dokumenten erst möglich wird, wenn eine Erweiterung der Gatherer-Funktionalität zur Extraktion der Dokument-Metadaten vorgenommen wird.

5.4.3 Anbindung von DB2

Das Datenbanksystem DB2 bietet in Java die Möglichkeit, sowohl durch die statische [db2sqlj] als auch dynamische [Cha98, Kapitel 8] Einbettung von SQL-Kommandos in eine Anwendung auf die gespeicherten Daten zuzugreifen.[23] In Abbildung 5.13 wird der Zugriff eines Wrappers mittels JDBC auf das Datenbanksystem schematisch dargestellt. Dieser erfolgt, wie auch bereits beim *Fulcrum SearchServer*, durch Verwendung eines JDBC-Treibers und speziellen bereitgestellten Java-Klassen.

[22]Um konkret zu sein während des Post-Summarizing-Prozesses [HSW96].

[23]Bei der Implementierung des DB2-Wrappers wurde die JDBC-Anbindung verwendet.

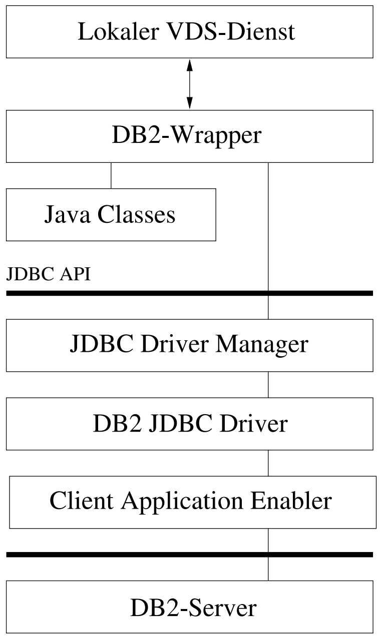

Abbildung 5.13: JDBC-Anbindung an einen DB2-Server

Bemerkungen

Die Anbindung eines Wrappers an DB2 erfordert, daß die folgenden Voraussetzungen erfüllt sind:

DB2-Client. Damit der vom Wrapper verwendete JDBC-Treiber mit dem DB2 Server kommunizieren kann, muß der Host, auf dem der Wrapper läuft, ein DB2 Client sein. Dazu ist Client-seitig ein *Client Application Enabler* zu installieren, der die Kommunikation zwischen Client-Rechner und DB2-Server realisiert.

Handelt es sich bei der Client-Anwendung um ein Applet (was hier nicht der Fall ist), muß der *Client Application Enabler* nicht installiert werden. Stattdessen muß auf dem WWW-Server ein *Applet-Server* laufen, der wiederum mit dem DB2-Server kommuniziert.

Client-seitiger nativer Code erforderlich. Der für DB2 verfügbare JDBC-Treiber (*Native-API partly Java driver* (Typ-2-Treiber)) erfordert, daß auf dem Client-Rechner binärer Code (hier in Form von shared libraries) vorliegt.

Umgebungsvariablen. Wie auch bereits in der Fulcrum-Anbindung dargelegt wurde, sind Client-seitig einige Umgebungsvariablen zu setzen. Da in Java das Konzept der Umgebungsvariablen durch Properties[24] ersetzt wurde, um

[24] Siehe [Fla98, Kapitel 14:Systemeigenschaften].

die Systemunabhängigkeit zu gewährleisten, können diese Variablen nicht durch den Wrapper gesetzt werden. Dementsprechend sind diese einmalig vor dem Start des Wrappers zu setzen.

XML-Unterstützung. DB2 bietet eine gewisse Unterstützung zur Integration der Bestandteile von XML-Dokumenten in Tabellen, die jedoch nicht näher untersucht wurde, da die Entwicklung dieses XML-Extenders bei IBM noch andauert. Ergänzende Informationen zur XML-Unterstützung von DB2 sind beispielsweise in [Por99] zu finden, die eine XML-Suchfunktionalität auf dem *DB2 TextExtender* aufsetzt.

Die Anbindung eines Wrappers an DB2 mittels JDBC erweist sich somit als einfach und wirft keine Probleme bei der Implementierung auf.

5.5 Zusammenfassung

Die in diesem Kapitel vorgenommene Vorstellung einiger Implementierungsansätze ermöglicht eine erste Einschätzung des für die Realisierung des Query-Servers und der Wrapper erforderlichen Aufwands. Der vorliegende Prototyp stellt ein Basissystem dar, dessen wesentliche Funktionalitäten in Bezug auf die IRQL-Anfragebearbeitung erst durch ergänzende Arbeiten implementiert werden.

Bei der Implementierung der Wrapper wurde der Fokus auf die Anfragebearbeitung gelegt. In die erweiterte Implementierung muß jedoch noch einfließen, daß der Replikations- und Caching-Dienst auf einige LDS auch schreibend zugreifen kann und somit die Konsistenz der Daten sichergestellt werden muß.

Bei der Anbindung der Wrapper an die drei betrachteten Systeme traten keine wesentlichen Probleme auf. Bei der Integration weiterer Systeme als LDS, für die ein JDBC-Treiber verfügbar ist, kann auf die in dieser Arbeit vorgenommene Implementierung der Wrapper-Basisarchitektur zurückgegriffen werden, da sowohl die Kommunikation mit den VDS-Diensten als auch die Kommunikation mit den darunterliegenden Datenquellen bereits umgesetzt wurde.

Da in diesem Kapitel lediglich die verwendeten Ansätze beschrieben wurden und nur teilweise die konkrete Umsetzung, wird die vorliegende Implementierungsbeschreibung durch einen Technischen Report ergänzt, der separat unter [Tit99] verfügbar ist. In diesem werden konkrete Aussagen zur Realisierung der Funktionalität der im Architekturentwurf spezifizierten Schnittstellen getroffen und die dazu bereitgestellten Klassen dokumentiert. Auf dieser Beschreibung aufbauend können weitere Arbeiten das bestehende Basissystem in einen voll nutzbaren Virtuellen Dokumenten-Server überführen.

Für Informationen über die Implementierung des Replikations- und Caching-Dienstes sei abschließend auf die parallel laufende Diplomarbeit von [Jon99] verwiesen.

Kapitel 6

Abschließende Bemerkungen

In der vorliegenden Diplomarbeit wurden die Anforderungen an die VDS-Dienste identifiziert, die sich aus der Analyse der allgemeinen Funktionalitäten der VDS-Dienste und externer Dienste ergaben. Auf diesen ersten Anforderungen aufbauend wurde die Grobarchitektur des VDS hergeleitet und die bereitzustellenden Dienstschnittstellen vorgestellt, erläutert und formal spezifiziert. Im Verlauf dieses Prozesses wurden weitere Anforderungen erkannt und in den Entwurf integriert. Da der Umfang einer Diplomarbeit nicht ausreicht, um alle aufgeworfenen Probleme erschöpfend zu diskutieren und Lösungen aufzuzeigen, wird dieses letzte Kapitel dazu genutzt, einen kurzen Überblick über den erreichten Stand der Dinge zu geben.

In Abschnitt 6.1 werden die in dieser Arbeit ermittelten wesentlichen Punkte kurz rekapituliert und in Abschnitt 6.2 einige Erweiterungsmöglichkeiten aufgezeigt.

6.1 Zusammenfassung

In der Analyse der Anforderungen an die lokalen VDS-Dienste in Kapitel 2 wurden für den Query-Server und die LDS wichtige Funktionalitäten beschrieben. Da der Replikations- und Caching-Dienst in einer parallelen Diplomarbeit konzipiert wird, beschränkten sich die Untersuchungen zu diesem Dienst auf einige allgemeine Anforderungen, die sich aus der Zusammenarbeit der VDS Dienste untereinander ergaben. Die wesentlichen Aussagen seien in dieser Zusammenfassung kurz aufgeführt.

Query-Server. Für den Query-Server wurden in einer ersten Untersuchung vier Hauptaufgaben identifiziert. Während die Verwaltung der LDS eines VDS, die Metadaten-Verwaltung sowie die Kommunikation der Dienste untereinander nur ansatzweise erläutert wurde, ergab sich aus der Diskussion zur globalen Anfragebearbeitung ein wesentlicher Teil dieser Arbeit, da aufbauend auf den gewonnenen Erkenntnissen das Zusammenwirken der lokalen VDS-Dienste dargestellt und die Architektur abgeleitet werden konnte. Es wurden Betrachtungen zum Verhalten des Systems bei mangelnder Verfügbarkeit der einzelnen VDS-Dienste vorgenommen sowie einige Ansätze skizziert, wie auf diese Probleme durch das System reagiert werden könnte, um die Verfügbarkeit des Gesamtsystems zu erhöhen.

LDS/Wrapper. Die Untersuchungen zum LDS erstreckten sich im Wesentlichen auf die vom Wrapper des LDS zu exportierenden Daten, die für andere Dienste bereitgestellt werden, und die Schnittstellen für die Anfragebearbeitung. Nicht berücksichtigt wurden beispielsweise vorläufig die Probleme, die aus dem gleichzeitigen Update (das durch den Replikations- und Caching-Dienst kontrolliert wird) von Dokumenten resultieren und somit ein Transaktionskonzept erfordern. Die für die LDS spezifizierten Schnittstellen stellen nur einen Teil der insgesamt bereitzustellenden Funktionalität dar, da sich aus der Zusammenarbeit der LDS mit dem Replikations- und Caching-Dienst weitere wesentliche Dienstschnittstellen des LDS ergeben. Für weitere Informationen diesbezüglich sei daher auf [Jon99] verwiesen.

Schnittstellenbeschreibungen. Für den Query-Server und die LDS wurden erste Spezifikationen der externen Dienstschnittstellen vorgestellt und die durch die Implementierung der Schnittstelle bereitzustellende Funktionalität und das Verhalten im Fehlerfall beschrieben.

Anfragen. Die an einen VDS adressierten IRQL-Anfragen wurden in Metadaten-Retrieval-Anfragen und Dokument-Retrieval-Anfragen unterschieden. Erstere gestatten die Recherche in den Dokumenten sowie deren Metadaten, liefern aber nur die vom Nutzer angeforderten frei zugänglichen Dokument-Metadaten sowie Referenzen für den Zugriff auf diejenigen Dokumente zurück, die die Anfrage erfüllen. Unter dem Begriff Dokument-Metadaten wurden dabei sowohl die strukturiert vorliegende Bestandteile eines Dokuments (Titel, Autor, Abstrakt usw.), als auch Daten verstanden, die aus dem Dokument abgeleitet werden (Prüfsumme, Änderungsdatum) bzw. dem Dokument zugeordnet sind.

Dokument-Retrieval-Anfragen liefern als Ergebnis ein Handle zum Zugriff auf die Dokumente, Teile der Dokumente bzw. die nicht frei zugänglichen Dokument-Metadaten zurück. Die Unterteilung in zwei Klassen von Anfragen erschien notwendig, da der Zugriff auf die Dokumente selbst bzw. Teile der Dokumente kostenpflichtig sein kann und somit eine Autorisierung oder Abrechnung erforderlich wird. Obwohl für Metadaten-Retrieval-Anfragen prinzipiell die gleiche Annahme zutreffen könnte, wurde davon ausgegangen, daß die Recherche in den Metadaten eines Dokumentes nicht kostenpflichtig ist, da dies im überwiegenden Teil der LDS der Fall sein wird. Sollte in einem konkreten Fall tatsächlich eine Autorisierungspflicht für Metadaten-Retrieval-Anfragen erforderlich sein, ist eine Anpassung der in dieser Arbeit vorgeschlagenen Form der Anfragebearbeitung erforderlich.

In Anlehnung an die differenzierten Formen der Anfragebearbeitung und die daraus resultierende Zusammenarbeit der VDS-Dienste wurde die Grobarchitektur des VDS und verfeinernd die des Query-Servers hergeleitet. Erwartungsgemäß zeigten sich für viele der zu treffenden Entwurfsentscheidungen diverse Ansätze, die teilweise ausführlich diskutiert wurden. Beispielhaft sei die Modellierung und der Zugriff auf Ergebnislisten und Dokumente durch einen Handle-Mechanismus genannt.

Anfragebearbeitung. Eine wesentliche Aufgabe des Query-Servers bildet die Anfragebearbeitung, die sich aus der Analyse der Anfrage, der Zerlegung in Teilanfragen, die durch die lokalen bzw. externen Dienste bearbeitet werden, und der Nachbearbeitung der Anfrageergebnisse zusammensetzt. Da nicht alle diese Teilbereiche im Rahmen dieser Arbeit abgedeckt werden konnten, beschränkten sich die vorgenommenen Untersuchungen im Kapitel 4 auf die Umsetzbarkeit der in der IRQL definierten IR-Prädikate in die Anfragesprache der betrachteten Systeme *Fulcrum SearchServer*, *Harvest* und *DB2 TextExtender*.

Keines der Systeme konnte die Prädikate und Klauseln der IRQL vollständig umsetzen, so daß eine Simulation der fehlenden Funktionalitäten durch die Wrapper für die Systeme erforderlich wurde. Aus diesem Grund wurden Abschätzungen vorgenommen, die beurteilen, wie gut ein System die Simulation der fehlenden Funktionalität unterstützt. Als Ergebnis dieser Untersuchungen wurde festgestellt, daß der *Fulcrum SearchServer* Schwächen im Bereich der DB-Funktionalität besitzt, das *Harvest*-System für die Umsetzung ungeeignet ist, da kaum DB- und IR-Konzepte umgesetzt werden, und der *DB2 TextExtender* starke Einschränkungen hinsichtlich der Verwendbarkeit der bereitgestellten Funktionen aufweist.

Implementierung. Die Implementierung des Prototypen baut auf den zuvor entwickelten Schnittstellenspezifikationen auf. Da es sich bei dem VDS um eine verteilte Client-Server-Anwendung handelt, wurden geeignete Implementierungstechniken vorgestellt, evaluiert und eingesetzt. Im Verlauf der Implementierung wurden die anfangs spezifizierten Schnittstellen entsprechend der sich aus der gewählten Implementierungstechnik ergebenden Anforderungen verfeinert bzw. erweitert und Klassen bereitgestellt, die die Funktionalität des Query-Servers und der Wrapper prototypisch bereitstellen. Somit steht mit Abschluß dieser Arbeit ein Basissystem zur Verfügung, das in folgenden Arbeiten schrittweise erweitert werden kann.

Erwartungsgemäß wirft ein erster Architekturentwurf eines komplexen Systems und die Beschreibung der Funktionalitäten mehr Fragen auf, als im Rahmen einer Diplomarbeit beantwortet werden können. Deshalb wird abschließend auf offene Probleme und Erweiterungsmöglichkeiten eingegangen, die einen möglichen Ausgangspunkt für weitere Arbeiten bilden.

6.2 Opportunities For Growth (OFG)

Der Ausblick wird dazu genutzt, bestehende Einschränkungen in der Funktionalität des Systems aufzuzeigen und weitere bisher noch nicht durchgeführte Untersuchungen aufzuzählen und Erweiterungen vorzuschlagen.

Einschränkungen

Im vorliegenden Architekturentwurf wurde eine Annahme getroffen, die eine Einschränkung der Funktionalität des Systems nach sich zieht. Diese Einschränkung könnte durch eine Modifikation der Architektur und der Anfragebearbeitung aufgehoben werden. Da dies jedoch eine komplexe Aufgabe mit weitreichenden Auswirkungen (OFG) darstellt, wurde in der vorliegenden Arbeit auf eine Umsetzung verzichtet.

Eingeschränkte Metadaten-Retrieval-Anfragebearbeitung. In den Betrachtungen zur Anfragebearbeitung wurde angenommen, daß Metadaten-Retrieval-Anfragen generell nicht autorisierungs- bzw. kostenpflichtig sind. Die Anbindung eines LDS, bei dem dies jedoch obligatorisch ist, kann in einem VDS mit der hier verwendeten Form der Metadaten-Retrieval-Anfragebearbeitung zur Zeit nicht erfolgen.

Ein möglicher Lösungsansatz ergibt sich, wenn der Query-Server die Wrapper der LDS nicht direkt kontaktiert, sondern indirekt über den Replikations- und Caching-Dienst die Anfragebearbeitung der Wrapper auslöst. Da die Autorisierung innerhalb des Replikations- und Caching-Dienstes angesiedelt ist, kann dieser prüfen, ob es sich bei den kontaktierten LDS um frei verfügbare oder autorisierungs- oder kostenpflichtige Dienste handelt und gegebenenfalls die erforderlichen Informationen vom Nutzer anfordern bzw. die Abrechnung durchführen. Der negative Aspekt dieses Ansatzes zeigt sich darin, daß damit alle Anfragen über den Replikations- und Caching-Dienst vermittelt werden, so daß dieser einen Engpaß bilden kann bzw. im Fehlerfall die Verfügbarkeit des gesamten VDS nicht mehr gewährleistet werden kann.

Ein anderer Lösungsansatz ergibt sich, wenn der Query-Server über die Autorisierungs- bzw. Kostenpflichtigkeit der LDS bei der Anmeldung der Datenquellen informiert wird und von sich aus sicherstellt, daß die erforderlichen Maßnahmen zur Abrechnung oder Autorisierung getroffen werden. Dies erfordert jedoch eine Interaktion zwischen dem Autorisierungsdienst des Replikations- und Caching-Dienstes und dem Query-Server, die zur Zeit nicht vorhanden ist.

Generell kann die Bearbeitung von Anfragen, bei denen auf geschützte Datenquellen zugegriffen wird, wesentlich aufwendiger sein, da wesentlich mehr Entscheidungen getroffen werden müssen. Dies gilt beispielsweise in Bezug auf finanzielle Aufwendungen, da in der Regel eine Kostenminimierung durchgeführt werden muß.[1] Der Nutzer muß im Fall der kostenpflichtigen Dienste auch darüber informiert werden, wie hoch die Kosten sind und entscheiden, ob er bereit ist, für die gewünschten Informationen den entsprechenden Preis zu zahlen. Erst nachdem diese Voraussetzungen erfüllt sind, kann die eigentliche Anfragebearbeitung durchgeführt werden. Die kostenpflichtige Recherche in Datenquellen beeinflußt demzufolge die gesamte Anfragebearbeitung wesentlich mehr, als dies für Datenquellen der Fall ist, bei denen lediglich für den Zugriff auf das gesamte Dokument bzw. Teile davon eine Autorisierung bzw. Abrechnung erfolgen muß.

Diese Ausführungen verdeutlichen, warum in dieser Arbeit die vorläufige Entscheidung gegen die Integration von LDS getroffen wurde, in denen die Recherche autorisierungs- bzw. kostenpflichtig ist.

Erweiterungen und offene Untersuchungen

Da es sich bei der im Rahmen dieser Arbeit vorgenommenen Implementierung um einen Prototypen handelt, wird nur die grundlegende Funktionalität bereitgestellt (Verwaltung der Ressourcen, eingeschränkte Anfragebearbeitung und -übersetzung, Kommunikation mit den lokalen VDS-Diensten). Neben diesen prototypisch umgesetzten Funktionen gibt es einige weitere OFGs, die sich positiv auf das Verhalten des Systems auswirken. Eine solche Funktion soll stellvertretend kurz dargelegt werden.

Das vorliegende Basissystem bietet bereits einige Funktionen zur Administration der Ressourcen. Die Ressourcen-Schnittstelle des Query-Servers könnte um einige Funktionen erweitert werden, die das Wiederanmelden von Ressourcen im Query-Server automatisieren. Nachdem eine Ressource erstmalig in einem VDS angemeldet wurde, kann der Wrapper über die Zuordnung informiert werden und von da an selbständig nach einer Deaktivierung der ihm zugeordneten Datenquelle die Wiederanmeldung automatisch vornehmen, so daß ein Polling durch den Query-Server entfallen kann.

Offene Untersuchungen

In den vergangenen Kapiteln wurde an einigen Stellen darauf hingewiesen, daß weitere Untersuchungen zu bestimmten Problemen (OFGs), insbesondere bei der Anfragebearbeitung, durchzuführen sind. Diese Untersuchungen umfassen beispielsweise

[1] So könnten Daten, die in einer Datenquelle kostenpflichtig sind, in einer anderen frei verfügbar sein. In diesem Fall wird es im Interesse des Nutzers sein, bevorzugt die kostenlosen Informationen abzufragen.

- die Zerlegung komplexer Anfragen in Teilanfragen, die an die LDS weitergeleitet werden,
- die Optimierung gegebener IRQL-Anfragen,
- die Nachbearbeitung im Query-Server bzw. im Wrapper, d.h. das Normalisieren[2], das Mischen[3], das Verknüpfen[4] und das globale Ranking der Anfrageergebnisse[5] sowie
- die vollständige Umsetzung der IRQL-Prädikate in die Anfragesprache der LDS.

Die Durchführung dieser Untersuchungen und die Implementierung der zur Umsetzung erforderlichen Funktionalitäten überführt den vorliegenden Prototypen in einen vollständig nutzbaren VDS, der dem Nutzer die einfache Recherche in verteilten und heterogenen Datenquellen erlaubt.

[2] In den Betrachtungen zu den Anfragesprachen wurde bereits deutlich, daß von Systemen bereitgestellte Prädikate gleicher Grundfunktionalität unterschiedliche Ergebnisse liefern können und diese demzufolge normalisiert werden müssen.

[3] Im vorliegenden Prototyp wird ein einfaches Round-Robin-Verfahren eingesetzt. Dieses liefert jedoch in der Praxis kein zufriedenstellendes Ergebnis.

[4] Das Verknüpfen von Teilanfragen, deren Ergebnisse mittels JOIN-Operator das globale Ergebnis bilden, findet zur Zeit noch nicht statt.

[5] Das ermittelte Ranking der zugrunde liegenden Datenquellen ist qualitativ sehr weit gestreut — von unbrauchbar im Fall von *Harvest* bis sehr gut im Fall des *Fulcrum SearchServers*, so daß eine globale Bewertung sehr schwer fällt.

Literaturverzeichnis

[CGMP96] Chang, C.-C. K.; Garcia-Molina, H.; Paepcke, A.: Boolean Query Mapping Across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, Band 8, Nr. 1, S. 515–521, Februar 1996. `http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0044`.

[Cha98] Chamberlin, D.: *DB2 Universal Database.* Morgan Kaufman Publishers, Inc., 1998.

[cor] CORBA/IIOP Specification. `http://www.omg.org/corba/corbaiiop.html`.

[cor98] CORBA/IIOP Specification - IDL Syntax and Semantics, 1998. `ftp://www.omg.org/pub/docs/formal/98-02-08.ps`.

[ots98] Transaction Service Specification, 1998. `ftp://www.omg.org/pub/docs/formal/97-12-17.ps`.

[db2sqlj] SQLJ: IBM DB2 SQLJ Home Page. `http://www.software.ibm.com/data/db2/java/sqlj/index.html`.

[dc99] Dublin Core Metadata Initiative, 1999. `http://purl.oclc.org/dc/`.

[Dic97] Dicken, H.: *JDBC - Internet-Datenanbindung mit Java.* International Thomson Publishing, 1997.

[dig98] The Stanford Digital Library Project, 1998. `http://www-diglib.stanford.edu/`.

[Dow98] Downing, T. B.: *Java RMI.* IDG Books Worldwide, Inc., 1998.

[Fla98] Flanagan, D.: *JAVA in a Nutshell.* O'Reilly, 2. Auflage, 1998.

[Fuh97] Fuhr, N.: Information Retrieval, Skriptum zur Vorlesung, 1997. `http://ls6-www.informatik.uni-dortmund.de/ir/doc/courses/ir/irskall.ps.gz`.

[garlic] The Garlic Project. `http://www.almaden.ibm.com/cs/garlic/`.

[GCGMP97] Gravano, L.; Chang, K.; Garcia-Molina, H.; Paepcke, A.: STARTS: Stanford Proposal for Internet Meta-Searching, 1997. `http://www-db.stanford.edu/pub/gravano/1996/sigmod97.ps`.

[gi98] BMBF-Projekt Global-Info, 1998. `http://www.global-info.org/`.

[GMY98] Garcia-Molina, H.; Yerneni, R.: Coping with Limited Capabilities of Sources. Technischer Bericht, Stanford University, CS Dept., 1998. `http://www-db.stanford.edu/pub/papers/limcap.ps`.

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

[harvest] Harvest Web Indexing. `http://www.tardis.ed.ac.uk/harvest/`.

[Har97] Harold, E. R.: *Java Network Programming.* O'Reilly, 1997.

[HCF97] Hamilton, G.; Cattell, R.; Fisher, M.: *JDBC Database Access with Java.* Addison-Wesley, 1997.

[Heu99] Heuer, A.: GLOBAL-INFO: Fachübergreifende Digitale Bibliotheken für die Wissenschaft. In: *Tagungsband der 2. IUK-Tage Mecklenburg-Vorpommern, Rostock*, Juni 1999.

[HHSW97] Hughes, M.; Hughes, C.; Shoffner, M.; Winslow, M.: *Java Network Programming.* Manning Publications Co., 1997.

[HP99] Heuer, A.; Priebe, D.: IRQL— Yet another language for querying semistructured data? Preprint CS-01-99, Universität Rostock, Fachbereich Informatik, 1999.

[HSW96] Hardy, D. R.; Schwartz, M. F.; Wessels, D.: *Harvest Users's Manual.* University of Colorado at Boulder, Januar 1996. Version 1.4, Patchlevel 2, Technical Report CU-CS-743-94.

[IBM97] IBM Corporation: *DB2 Universal Database Text Extender Administration and Programming*, September 1997.

[ilu] Inter-Language Unification.
`ftp://ftp.parc.xerox.com/pub/ilu/ilu.html`.

[Inf97] Informix Software Inc.: *Excalibur Text Search DataBlade Module*, Juli 1997.

[jdbc] The JDBC(tm) Universal Data Access API.
`http://java.sun.com/products/jdbc/index.html`.

[Jon99] Jonas, M.: Ein Replikationsdienst für komplexe Dokumente. Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1999.
`http://wwwdb.informatik.uni-rostock.de/Lehre/Arbeiten/da-jonas.html`.

[KSS96] Kleiman, S.; Shah, D.; Smaalders, B.: *Programming with THREADS*. SunSoft Press, 1996.

[LBS98] Lewis, G.; Barber, S.; Siegel, E.: *Programming with Java IDL*. John Wiley & Sons Inc., 1998.

[mathnet] Math-Net Entry Point.
`http://www.math-net.de/`.

[md5] RFC 1321 - The MD5 Message-Digest Algorithm.
`http://www.ietf.org/rfc/rfc1321.txt`.

[med98] *Digital Libraries in Computer Science: The MeDoc Approach*, Lecture Notes in Computer Science 1392, 1998.
Homepage: `http://medoc.informatik.tu-muenchen.de/`.

[Mey96] Meyer, J.: Entscheidungsgrundlage für die Auswahl von Volltextdatenbanksystemen für den MeDoc-Dienst. Technischer Bericht, OFFIS Oldenburg, Juli 1996.
`http://www-is.informatik.uni-oldenburg.de/~meyer/medoc/papers/vtdbms.ps`.

[mime] MIME Multipurpose Internet Mail Extensions.
`http://www.ietf.org/rfc/rfc1521.txt`.

[MW94] Manber, U.; Wu, S.: GLIMPSE: A Tool to Search Through Entire File Systems. In: *Proceedings of the USENIX Winter 1994 Technical Conference*, S. 22–32, Januar 1994.

[odbc] Microsofts ODBC Web Page.
`http://www.microsoft.com/data/odbc/`.

[OH98] Orfali, R.; Harkey, D.: *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons Inc., 2. Auflage, 1998.

[Pae96] Paepcke, A.: Digital Libraries: Searching Is Not Enough. *D-Lib Magazine*, Jahrgang 1996, 1996.
`http://www.dlib.org/dlib/may96/stanford/05paepcke.html`.

[physnet] PhysNet-Worldwide Physics Department Network.
`http://www.physik.uni-oldenburg.de/EPS/PhysNet/`.

[Por99] Porst, B.: Untersuchung zu Datentyperweiterungen für XML-Dokumente und ihre Anfragemethoden am Beispiel von DB2 oder Informix. Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1999.
`http://wwwdb.informatik.uni-rostock.de/Lehre/Arbeiten/da-porst.html`.

[Rah94] Rahm, E.: *Mehrrechner-Datenbanksysteme*. Addison-Wesley, 1994.

[rmia] Java Remote Method Invocation.
`http://java.sun.com/products/jdk/rmi/index.html`.

[rmib] Java Remote Method Invocation Specification.
`http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html`.

[RS97] Roth, M. T.; Schwarz, P.: Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In: *Proceedings of the 23rd VLDB Conference, Athens*, 1997.

[sbb98] Fulcrum Technologies Inc.: *Fulcrum SearchBuilder for Visual Basic / Visual C++ : Developer's Guide*, 1998.

[sbc98] Fulcrum Technologies Inc.: *Fulcrum SearchBuilder for C : API Reference*, 1998.

[sbj98] Fulcrum Technologies Inc.: *Fulcrum SearchBuilder for Java : Developer's Guide*, 1998.

[sbt98] Fulcrum Technologies Inc.: *Fulcrum SearchBuilder : Text Reader Developer's Guide*, 1998.

[Sch98] Schmiede, R.: Global Info: Ein neuer Fokus für Library-Entwicklungen in Deutschland, 1998.
`http://www.global-info.org/doc/schmiede-9801.pdf`.

[Sie96] Siegel, J.: *CORBA - Fundamentals and Programming.* John Wiley & Sons Inc., 1996.

[sql3] SQL3 ISO-ANSI Working Draft. `http://ftp.digital.com/pub/standards/sql/`.

[sqlj] What is SQLJ and Who is supporting SQLJ. `http://www.sqlj.org/`.

[sqlja] An Overview of SQLJ: Embedded SQL in Java. `http://www.oracle.de:80/orcl/owai/sqlj.pdf`.

[sql89] International Organization for Standardization (ISO): *Database Language SQL. Document ISO/IEC 9075:1989*, 1989.

[sql92] International Organization for Standardization (ISO): *Database Language SQL. Document ISO/IEC 9075:1992*, 1992.

[sql98a] SQLJ: SQL Types using the Java Programming Language, 1998. `ftp://ftp.calweb.com/business/sqlj/SQLJ-P2.pdf`.

[sql98b] SQLJ: SQL Routines using the Java Programming Language, 1998. `ftp://ftp.calweb.com/business/sqlj/SQLJ-P1.pdf`.

[ssd98] Fulcrum Technologies Inc.: *Fulcrum SearchServer : Data Preparation and Administration*, 1998.

[ssi98] Fulcrum Technologies Inc.: *Fulcrum SearchServer : Introduction to SearchServer*, 1998.

[sss98] Fulcrum Technologies Inc.: *Fulcrum SearchServer : SearchSQL Reference*, 1998.

[threads] Doing two or more tasks at once: threads. `http://java.sun.com/docs/books/tutorial/essential/threads/index.html`.

[Tit98] Titzler, P.: Realisierungsvorschlag für die Implementierung einer verteilten Suchmaschine im WWW. Studienarbeit, Universität Rostock, Fachbereich Informatik, 1998. `http://wwwdb/Lehre/Arbeiten/sa-titzler.ps.gz`.

[Tit99] Titzler, P.: BlueRose / Implementierung eines Virtuellen Dokumenten-Servers. Preprint CS-02-99, Universität Rostock, Fachbereich Informatik, 1999.

[VD98] Vogel, A.; Duddy, K.: *JAVA Programming with CORBA.* John Wiley & Sons Inc., 2. Auflage, 1998.

[Vsu98] Vsudevan, V.: A Reference Model for Trader-Based Distributed Systems Architectures, 1998.
`http://www.objs.com/survey/trader-rm.html`.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Spezifikation der Dienstschnittstellen

In diesem Anhang sind die Schnittstellenbeschreibungen des Query-Servers (Abschnitt A.1) und die Schnittstellen der Wrapper der LDS (Abschnitt A.2) formal spezifiziert. Die Spezifikation der Dienstschnittstellen erfolgt mit Hilfe der *Interface Definition Language* (IDL). Die IDL deklariert öffentliche Methoden und Attribute sowie von den Methoden ausgelöste Ausnahmen. Durch die IDL wird die Funktionalität der Methoden, jedoch nicht ihre konkrete Implementierung beschrieben. Die Syntax und Semantik der IDL sind in [cor98] festgelegt.

Eine Schnittstellenbeschreibung des Replikations- und Caching-Dienstes wird nicht in diesem Anhang sondern in [Jon99] gegeben.

Allgemeine Hinweise

Aus Gründen der Übersichtlichkeit wird der für die Darstellung der Schnittstellendefinition verwendete Font gegenüber dem normalen Text um eine Schriftgröße verkleinert. Aus dem gleichen Grund werden für einige häufig verwendete lange Begriffe Abkürzungen eingeführt, um die Lesbarkeit der Kommentare zu erhöhen:

- Metadaten-Retrieval-Anfragen : M-R-Anfragen
- Dokument-Retrieval-Anfragen : D-R-Anfragen
- Replikations- und Caching-Dienst: RuC-Dienst

Die in der vorliegenden Arbeit spezifizierten Schnittstellen besitzen den nachfolgend skizzierten Aufbau, wobei nicht für jede Schnittstelle alle Definitionen erforderlich sind, da sie die entsprechenden Bestandteile nicht besitzen.

```
interface SCHNITTSTELLENNAME
 {
   // Attribute, auf die von außerhalb zugegriffen werden kann.
    ...
   // Definition der Ausnahmen, die durch die Methoden geworfen
   // werden können.
    ...
   // Definition von Hilfsdatentypen, aus denen sich die
   // komplexen Datentypen der Signatur zusammensetzen.
    ...
   // Definition von Datentypen, die in der Signatur der Methoden
   // auftauchen.
    ...
   // Definition der Methoden, die der Dienst nach außen zur
   // Verfügung stellt.
    ...
 };
```

An einigen Stellen der Definition konnte keine Festlegung auf eine konkrete Datenstruktur erfolgen, da Entscheidungen von externen Projektpartnern abhüngen bzw. durch die dortige konkrete Implementierung beeinflußt werden. In Definitionen, die dies betrifft sind die entsprechenden Datentypfestlegungen durch

```
// BEGIN IMPLEMENTATION_DEPENDENT_PART
// END IMPLEMENTATION_DEPENDENT_PART
```

gekennzeichnet.

Spezifikation der verwendeten Handles

Für den Zugriff auf Ergebnisse von Metadaten-Retrieval-Anfragen und den Zugriff auf den Inhalt eines Dokumentes werden durch die VDS-Dienste Handles bereitgestellt. Die Spezifikation der grundlegenden Methoden, die diese Handles bereitstellen, erfolgt in den folgenden Abschnitten. Die Handles werden von den Schnittstellen des Query-Servers und des Replikations- und Caching-Dienest erzeugt und an den Aufrufer der Methoden zurückgeliefert.

Ergebnislisten-Handle

Der Query-Server und die Wrapper der LDS liefern auf eine Anfrage ein Handle zurück, daß zum Zugriff auf die Ergebnisse verwendet wird. In diesem Abschnitt wird spezifiziert, wie dieses Handle aufgebaut ist. Die Spezifikation enthält nur die grundsätzlichen Methoden zum Zugriff auf die Elemente und zur Navigation innerhalb der Ergebnisliste.

```
interface ResultSetHandle
 {
   // ----------------------------------------------------------
   // Attribute, auf die von außerhalb zugegriffen werden kann.
   // ----------------------------------------------------------
   // Anzahl der zurückgelieferten Ergebnistupel
     readonly attribute short numberOfResultItems;

   // Aktuelle Position innerhalb der Ergebnismenge
   // Da die automatisch generierte SET-Methode zum Setzen des
   // eines Attributes keine Exception werfen würde, wenn ein
   // Wert < 1 oder > numberOfResultItems angegeben werden würde,
   // muß dieses Attribut als READONLY markiert werden und eine
   // eigene Methode zum Setzen bereitgestellt werden.
     readonly attribute short currentItemPosition;

   // ----------------------------------------------------------
   // Definition der Ausnahmen, die durch die Methoden geworfen
   // werden können.
   // ----------------------------------------------------------
   // Es wurde versucht, den Zeiger des aktuellen Elements auf
   // einen Wert zu setzen, der kleiner als das Minimum oder
   // größer als das Maximum ist.
     exception outOfBoundsException{};

   // Allgemeiner Fehler beim Laden eines Ergebnistupels.
     exception miscHandleException {string reason;};

   // ----------------------------------------------------------
   // Definition von Datentypen, die in der Signatur der Methoden
   // auftauchen.
   // ----------------------------------------------------------
   // Eine Typbeschreibung des Objektes, das ein Element der
   // Liste ListItems beschreibt soll nicht gegeben werden, da
   // dies zu umfangreich wäre. Deshalb wird hier eine Dummy-
   // definition gegeben, um eine gültige IDL-Beschreibung zu
   // erhalten.
    struct ListItemObject
     {
      // nothing in here yet
     };

   // ----------------------------------------------------------
   // Definition der Methoden, die der Dienst nach außen zur
   // Verfügung stellt.
```

```
// ----------------------------------------------------------
// Funktion  : Methode wird aufgerufen, um das nächste Ele-
//             ment der Ergebnisliste anzufordern. Inkremen-
//             tiert die currentItemPosition um eins.
// Parameter : position
// Rückgabe  : keine
// Ausnahmen : outOfBoundsException
//             miscHandleException
 void setCurrentItemPosition(in short position)
  raises(outOfBoundsException);

// Funktion  : Methode wird aufgerufen, um das Element der
//             Ergebnisliste anzufordern, auf dem
//             currentItemPosition steht. Verändert den Wert
//             der currentItemPosition nicht.
// Parameter : keine
// Rückgabe  :
// Ausnahmen : outOfBoundsException
//             miscHandleException
 void currItem(out ListItemObject ListItem)
  raises(outOfBoundsException,
         miscHandleException);

// Funktion  : Methode wird aufgerufen, um das vorherige Ele-
//             ment der Ergebnisliste anzufordern. Dekremen-
//             tiert die currentItemPosition um eins. Enthält
//             die Liste keine Elemente, so wird
//             currentItemPosition auf numberOfResultItems
//             (also 0) gesetzt.
// Parameter : keine
// Rückgabe  :
// Ausnahmen : outOfBoundsException
//             miscHandleException
 void prevItem(out ListItemObject ListItem)
  raises(outOfBoundsException,
         miscHandleException);

// Funktion  : Methode wird aufgerufen, um das erste Ele-
//             ment der Ergebnisliste anzufordern. Setzt die
//             currentItemPosition auf eins. Enthält die Liste
//             keine Elemente, so wird currentItemPosition auf
//             numberOfResultItems (also 0) gesetzt.
// Parameter : keine
// Rückgabe  :
// Ausnahmen : outOfBoundsException
```

```
  //            miscHandleException
   void firstItem(out ListItemObject ListItem)
    raises(outOfBoundsException,
           miscHandleException);

  // Funktion  : Methode wird aufgerufen, um das letzte Ele-
  //             ment der Ergebnisliste anzufordern. Setzt die
  //             currentItemPosition auf numberOfResultItems(0).
  // Parameter : keine
  // Rückgabe  :
  // Ausnahmen : outOfBoundsException
  //             miscHandleException
   void lastItem(out ListItemObject ListItem)
    raises(outOfBoundsException,
           miscHandleException);
}; // Ende der Schnittstellenbeschreibung eines Ergebnislisten-Handles
```

Dokument-Handle

Das Dokument-Handle wird vom Query-Server und vom RuC-Dienst als Ergebnis einer Dokument-Retrieval-Anfrage zurückgeliefert und vom auslösenden Dienst dazu genutzt, um auf den Inhalt des Dokumentes zuzugreifen. Weitere Methoden könnten den Zugriff auf einzelne Teile des Dokumentes realisieren.

```
interface DocumentHandle
 {
  // -----------------------------------------------------------
  // Die folgenden Ausnahmen können durch das Query-Interface in
  // einer geeigneten Form an den Initiator der Anfrage zurück-
  // gegeben werden.
  // -----------------------------------------------------------
  // Beim Laden des Dokumentinhaltes ist ein Fehler aufgetreten.
   exception miscHandleException {string reason;};

  // -----------------------------------------------------------
  // Definition von komplexen Hilfsdatentypstrukturen, die in
  // Definitionen der komplexen Datentypen auftreten können.
  // -----------------------------------------------------------
  // die Bytes, aus denen sich das Dokument zusammensetzt
   typedef sequence<octed> documentContentData;

  // -----------------------------------------------------------
  // Öffentliche Methoden, die der Dienst bereitstellt.
  // -----------------------------------------------------------
  // Funktion  : Fordert den Inhalt eines Dokumentes an.
  // Parameter : keine
```

```
 // Rückgabe  : die Bytes, aus denen sich das Dokument zusammen-
 //             setzt
 // Ausnahmen : miscHandleException
  void getContent(out documentContentData documentContent)
   raises (miscHandleException);
}; // Ende der Schnittstellenbeschreibung eines Dokument-Handles
```

A.1 Anfrage- und Retrieval-Dienst

Der Query-Server stellt Schnittstellen

- zum Empfang der Anfragen (Query-Interface),
- zur Verwaltung der LDS,
- zur Kommunikation mit den Wrappern der LDS,
- zur Kommunikation mit dem Replikations- und Caching-Dienst sowie
- zur Kommunikation mit dem Föderationsdienst

bereit. Diese sind in den folgenden Abschnitten formal spezifiziert.

A.1.1 Query-Interface

In diesem Abschnitt wird die Schnittstelle spezifiziert, die der Query-Server für den Empfang von Anfragen bereitstellt.

```
interface QueryInterface
 {
  // ------------------------------------------------------------
  // Öffentliche Attribute.
  // ------------------------------------------------------------
  // Die maximale Anzahl gleichzeitig bearbeiteter Anfragen.
  // Ist die Anzahl der eingehenden Anfragen größer, werden
  // diese in einer Warteschlange verwaltet, bis das Maximum
  // wieder unterschritten wird.
   readonly attribute short maxOpenRequests;

  // ------------------------------------------------------------
  // Die folgenden Ausnahmen können durch das Query-Interface in
  // einer geeigneten Form an den Initiator der Anfrage zurück-
  // gegeben werden.
  // ------------------------------------------------------------
  // Das angeforderte Dokument konnte nicht gefunden werden.
   exception documentNotFoundException{};

  // Beim Laden des Dokumentes ist ein Fehler aufgetreten, der
```

```
// durch den Rückgabewert näher beschrieben wird.
 exception miscRetrieveException {string reason;};

// Beim Laden des angeforderten Dokumentes wurde festgestellt,
// daß eine Autorisierung erforderlich ist aber der D-R-Anfrage
// keine Autorisierungsinformation beigefügt wurde.
 exception authorizationRequiredException{};

// Die Verifizierung der Autorisierungsinformation ist fehlge-
// schlagen. Das angeforderte Dokument wird nicht ausgeliefert.
 exception authorizationFailedException {string reason;};

// Bei der Bearbeitung einer M-R-Anfrage ist
// ein Fehler aufgetreten, der durch den Rückgabewert näher
// beschrieben wird.
 exception miscFindException {string reason;};

// ------------------------------------------------------------
// Definition von komplexen Hilfsdatentypstrukturen, die in
// Definitionen der komplexen Datentypen auftreten können.
// ------------------------------------------------------------
// eine Menge von Strings
 typedef sequence<string> listOfString;

// ein Attribut-Wert-Paar
 struct nameValuePair
  {
   string name;    // Attributname
   string value;   // dazugehöriger Wert
  };

// eine Menge von Attribut-Wert-Paaren
 typedef sequence<nameValuePair> listOfNameValuePairs;

// ------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// ------------------------------------------------------------
// Festlegung der Struktur, in der ein angefordertes Dokument
// zurückgeliefert wird. Wird von der Methode retrieveDocument
// zurückgeliefert.
 struct documentType
  {
```

```
    string documentIdentifier; // DOI
    string dateCreated;        // Erstellungsdatum
    string dateModified;       // letztes Änderungsdatum
    string md5;                // Prüfsumme (RFC1321)
    long   size;               // Größe in Bytes
    documentHandle complete_content; // ein Handle zum Zugriff auf
                               // den Inhalt des Dokumentes
    string mimeType;           // Format (RFC1521)
    string xmlDTD;             // DTD, falls Dok. ein XML-Dokument
   };

 // Festlegung, wie sich die Autorisierungsinformation zusammen-
 // setzt. Wird der Methode retrieveDocument als
 // Parameter übergeben.
  struct authorizationInformationType
   {
    string id;      // Identität des Nutzers
    string key;     // Schlüssel zur Autorisierung
    // BEGIN IMPLEMENTATION_DEPENDENT_PART
      // An dieser Stelle sind Entscheidungen zu berücksichtigen,
      // die von externen Projektpartnern getroffen werden.
    // END IMPLEMENTATION_DEPENDENT_PART
   };

 // Festlegung, wie sich die zu einer Anfrage gehörenden Prä-
 // ferenzen einer Anfrage zusammensetzen. Wird der Methode
 // findRelevantResults als Parameter übergeben.
  struct preferencesType
   {
    short maxResults;  // Anzahl der maximal zurückzuliefernden
                       // Ergebnistupel
    listOfString defaultAttributeList; // Attribute, die standard-
                                       // mäßig bei M-R-Anfragen
                                       // zurückgeliefert werden
    listOfString defaultResourceList;  // Datenquellen, die
                                       // eine M-R-Anfrage standard-
                                       // mäßig bearbeiten sollen
   };

 // Festlegung, wie sich die Ergebnistupelliste einer M-R-Anfrage
 // zusammensetzt, die vom Query-Interface an die aufrufende Instanz
 // (Nutzersystem, externer VDS) zurückgeliefert wird.
  struct resultListTupelType
   {
    string position;            // Rang in der Ergebnisliste
```

```
    string documentIdentifier;   // DOI des Dokumentes
    string source;               // Wo wurde das Dokument gefunden ?
    listOfNameValuePairs attributeValues; // Selektionsattribute und
                                          // deren Wert(e)
   };

// Eine geordnete Liste von Ergebnistupeln. Wird von der Methode
// findRelevantResults als Ergebnis zurückgeliefert.
 typedef sequence<resultListTupelType> resultListType;

// Festlegung, welche Optionen bei einer D-R-Anfrage über-
// geben werden können.
 struct optionType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind Optionen aufgeführt, die
     // nicht durch eine IRQL-Anfrage ausgedrückt werden können.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// -------------------------------------------------------------
// Öffentliche Methoden, die der Dienst bereitstellt.
// -------------------------------------------------------------
// Funktion  : Methode wird aufgerufen, wenn eine D-R-Anfrage an
//             das Query-Interface gestellt wurde
// Parameter : documentIdentifier - DOI des Dokumentes
//             options - Optionen, die die D-R-Anfrage beein-
//             flussen
//             authorizationInformation - Informationen, die
//              zur Autorisierung verwendet werden; dieser Para-
//              meter kann einen bestimmten Wert (entspr. NIL oder
//              NULL) annehmen, der kennzeichnet, das keine
//              Autorisierungsinformation übergeben wird
// Rückgabe  : document - das Dokument (inkl. Metadaten)
// Ausnahmen : documentNotFoundException
//             miscRetrieveException
//             authorizationRequiredException
//             authorizationFailedException
 void retrieveDocument(in string documentIdentifier,
        in optionType options,
        in authorizationInformationType authorizationInformation,
        out documentType document)
  raises (documentNotFoundException,
          miscRetrieveException,
          authorizationRequiredException,
```

```
             authorizationFailedException);

    // Funktion  : Auslösen der Anfragebearbeitung für
    //             M-R-Anfragen
    // Parameter : query - der IRQL-Anfragestring
    //             preferences - die Präferenzen des Nutzers
    // Rückgabe  : resultList - eine geordnete Liste von
    //              relevanten Dokumenten
    //             statusInfo - Fehler, die die Ergebnismenge
    //              beeinflußt haben, aber nicht zum Abbruch
    //              führen
    // Ausnahmen : miscFindException
     void findRelevantResults(in string query,
                              in preferencesType preferences,
                              out resultListType resultList,
                              out string statusInfo)
      raises (miscFindException);

  }; // Ende der Schnittstellenbeschreibung "QueryInterface"
```

A.1.2 Ressourcen-Verwaltung-Interface

In diesem Abschnitt ist die Schnittstelle spezifiziert, die für die Verwaltung der lokalen LDS im Query-Server zuständig ist.

```
  interface ResourceManager
   {
    // -----------------------------------------------------------
    // Die folgenden Ausnahmen können durch die Ressourcen-
    // Verwaltung in einer geeigneten Form an den Administrator, der
    // die Operationen auslöst, zurückgegeben werden.
    // -----------------------------------------------------------
    // Bei der Neuanmeldung einer Ressource (LDS) wurde festgestellt,
    // daß der Wrapper dieser Ressource bzw. die darunter liegende
    // Datenquelle nicht angesprochen werden konnte. Die Neuanmeldung
    // gilt damit als gescheitert.
     exception resourceNotFoundException{};

    // Eine Ressource mit der spezifizierten ID ist nicht im Repository
    // der Ressourcen-Verwaltung vorhanden.
     exception resourceIdNotFoundException{};

    // Bei der Durchführung einer Verwaltungsoperation ist ein
    // Fehler aufgetreten, der näher erläutert wird.
     exception resourceManagementException {string reason;};
```

```
// ------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// ------------------------------------------------------------
// Beschreibung der Daten, die bei der Anmeldung einer neuen
// Ressource durch die Ressourcenverwaltung gespeichert werden.
 struct resourceDescriptionType
  {
   string informalDescription; // verbale Beschreibung des
                               // Inhaltes der Ressource
   string classification;      // formale Einordnung in eine
                               // oder mehrere Kategorien
   string administrator;       // Administrator der Ressource
   string emailContact;        // E-Mail Adresse des Administrators
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind implementierungsspezifische
     // Informationen anzugeben, die nicht allgemein beschrieben
     // werden können. Dazu gehört die Beschreibung, wie der
     // Wrapper angesprochen werden kann. Beispielsweise würde
     // dies bei einer einfachen Socket-Verbindung der Name des
     // Hosts sein, auf dem der Wrapper läuft und die Port-
     // nummer, an dem der Wrapper lauscht.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// ------------------------------------------------------------
// Öffentliche Methoden, die der Dienst bereitstellt.
// ------------------------------------------------------------
// Funktion  : Neuanmeldung eines Wrappers/LDS. Anmeldung ist
//             erfolgreich, wenn die Verwaltung eine Verbindung
//             mit dem Wrapper aufbauen konnte.
// Parameter : resourceDescription
// Rückgabe  : resourceIdentifier - für den VDS eindeutige ID,
//              die der Ressource (LDS) zugewiesen wird
// Ausnahmen : resourceNotFoundException
//             resourceManagementException
 void addResource(in resourceDescriptionType resourceDescription,
                  out string resourceIdentifier)
  raises (resourceNotFoundException,
          resourceManagementException);

// Funktion  : Abmeldung eines Wrappers/LDS. Konsequenzen: Die
//             Datenquelle kann durch die VDS-Dienste nicht mehr
```

```
  //             angesprochen werden. Die Metadaten des LDS werden
  //             aus dem Repository entfernt.
  // Parameter : resourceIdentifier - die bei der Anmeldung zuge-
  //              wiesene eindeutige Identität der Ressource
  // Rückgabe  : keine
  // Ausnahmen : resourceIdNotFoundException
  //             resourceManagementException
   void removeResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            resourceManagementException);

  // Funktion  : Zeitlich begrenzte Abmeldung eines Wrappers/LDS.
  //             Konsequenzen: Die Datenquelle kann durch die
  //             VDS-Dienste bis zur Wiederanmeldung nicht mehr
  //             angesprochen werden. Die Metadaten des LDS
  //             verbleiben im Repository.
  // Parameter : resourceIdentifier - die bei der Anmeldung ver-
  //              gebene eindeutige Identität des LDS
  // Rückgabe  : keine
  // Ausnahmen : resourceIdNotFoundException
  //             resourceManagementException
   void deactivateResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            resourceManagementException);

  // Funktion  : Wiederanmeldung eines Wrappers/LDS nach einer
  //             zeitlich begrenzten Abmeldung. Konsequenzen:
  //             Die Datenquelle kann durch die VDS-Dienste
  //             wieder angesprochen werden.
  // Parameter : resourceIdentifier - die bei der Anmeldung ver-
  //              gebene eindeutige Identität des LDS
  // Rückgabe  : keine
  // Ausnahmen : resourceIdNotFoundException
  //             resourceNotFoundException
  //             resourceManagementException
   void reactivateResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            resourceNotFoundException,
            resourceManagementException);

}; // Ende der Schnittstellenbeschreibung "ResourceManager"
```

A.1.3 Ressourcen-Interface

In diesem Abschnitt ist die Schnittstelle spezifiziert, die der Query-Server zum Zugriff auf den Wrapper eines LDS nutzt. Sie stellt praktisch eine Abstraktion eines Wrappers dar. Die Implementierung dieser Schnittstelle nutzt die Schnittstelle, die der Wrapper bereitstellt.

```
interface Resource
 {
  // ------------------------------------------------------------
  // Die folgenden Ausnahmen können durch das Ressourcen-Inter-
  // face ausgelöst werden.
  // ------------------------------------------------------------
  // Der Verbindungsaufbau zum Wrapper des LDS ist fehlgeschlagen.
   exception connectFailedException{};

  // Bei der Bearbeitung einer Anfrage durch den Wrapper ist ein
  // Fehler aufgetreten, der näher erläutert wird. Dieser Fehler
  // kann dem Wrapper von der darunterliegenden Datenquelle mit-
  // geteilt worden sein, der diesen an den Aufrufer weiterleitet.
   exception miscWrapperException {string reason;};

  // ------------------------------------------------------------
  // Definition von komplexen Hilfsdatentypstrukturen, die in
  // Definitionen der komplexen Datentypen auftreten können.
  // ------------------------------------------------------------
  // Festlegungen, die nicht durch die IRQL ausgedrückt werden
  // konnten aber das Anfrageergebnis beeinflussen. Diese Prä-
  // ferenzen setzen sich aus einem Teil der Präferenzen, die
  // vom Nutzersystem an das Query-Interface übergeben wurden,
  // und weiteren Festlegungen zusammen, die durch den Query-
  // Server getroffen wurden.
  // Bsp.: <"RankingAlgorithm","TermsCount">
   struct preferenceType
    {
     string preferenceName;  // Für einen VDS eindeutiger Name
                             // einer Festlegung.
     string preferenceValue; // Wert dieser Festlegung.
    };

  // ein Attribut-Wert-Paar
   struct nameValuePair
    {
     string name;    // Attributname
     string value;   // dazugehöriger Wert
    };
```

```
// eine Menge von Attribut-Wert-Paaren
 typedef sequence<nameValuePair> listOfNameValuePairs;


// Festlegung, wie sich die Ergebnistupelliste zusammensetzt, die
// von einem Wrapper an den Query-Server zurückgeliefert wird.
 struct resultListTupelType
  {
   string position;             // Rang in der Ergebnisliste
   string documentIdentifier;   // DOI des Dokumentes
   listOfNameValuePairs attributeValues; // Selektionsattribute und
                                         // deren Wert(e)
   string localRankingValue;    // durch die Datenquelle ermittel-
                                // ter Ranking-Wert
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung können weitere
     // Daten aufgeführt werden. Ein Beispiel bilden Daten, anhand
     // derer von der Datenquelle der Ranking-Wert bestimmt wurde.
     // Diese können genutzt werden, um einen globalen Wert zu
     // ermitteln.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// ---------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// ---------------------------------------------------------------
// Eine Menge von Festlegungen, die bei der Anfragebearbeitung
// durch den Wrapper zu berücksichtigen sind. Wird von der
// Methode processQuery als Parameter übergeben.
 typedef sequence<preferenceType> preferencesListType;

// Eine geordnete Liste von Ergebnistupeln. Wird von der Methode
// processQuery als Ergebnis zurückgeliefert.
 typedef sequence<resultListTupelType> resultListType;

// Festlegung, in welcher Form die dynamischen Metadaten des
// LDS im Repository zwischengespeichert und an die entsprechenden
// externen VDS-Dienste weitergeleitet werden. Daten diesen Typs
// werden von der Methode getDynamicMetadata als Ergebnis
// zurückgeliefert.
 struct dynamicMetaDataType
```

```
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung wird die Struktur
     // der zurückgelieferten Metadaten festgelegt.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Festlegung, in welcher Form die statischen Metadaten des
// LDS im Repository gespeichert und an die entsprechenden
// externen VDS-Dienste weitergeleitet werden. Daten diesen Typs
// werden von der Methode getStaticMetadata als Ergebnis
// zurückgeliefert.
 struct staticMetaDataType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung wird die Struktur
     // der zurückgelieferten Metadaten festgelegt.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// ------------------------------------------------------------
// Öffentliche Methoden, die der Dienst bereitstellt.
// ------------------------------------------------------------
// Funktion  : Auslösen der Anfragebearbeitung durch den
//             Wrapper.
// Parameter : query - die IRQL-Anfrage
//             preferences - weitere Festlegungen zur Anfrage,
//              die nicht durch die IRQL ausgedrückt werden
//              können
//             resultList - geordnete Liste der Ergebnistupel
// Rückgabe  : keine
// Ausnahmen : connectFailedException
//             miscWrapperException
 void processQuery(in string query,
                   in preferencesListType preferences,
                   out resultListType resultList)
  raises (connectFailedException,
          miscWrapperException);

// Funktion  : Löst das Laden der statischen Metadaten der
//             Datenquelle durch den Wrapper aus. Die Metadaten
//             werden von der Metadatenverwaltung des Query-
//             Servers gespeichert.
// Parameter : keine
// Rückgabe  : metaData - statische Metadaten der Datenquelle
```

```
   // Ausnahmen : connectFailedException
   //             miscWrapperException
    void getStaticMetadata(out staticMetaDataType metaData)
     raises (connectFailedException,
             miscMetadataImportException);

   // Funktion  : Löst das Laden der dynamischen Metadaten der
   //             Datenquelle durch den Wrapper aus. Die Metadaten
   //             werden von der Metadatenverwaltung des Query-
   //             Servers zwischengespeichert.
   // Parameter : keine
   // Rückgabe  : metaData - dynamische Metadaten über den
   //              Datenbestand der Datenquelle
   // Ausnahmen : connectFailedException
   //             miscMetadataImportException
    void getDynamicMetadata(out dynamicMetaDataType metaData)
     raises (connectFailedException,
             miscMetadataImportException);

   // Funktion  : Testet, ob eine Verbindung zum Wrapper und der
   //             darunterliegenden Datenquelle aufgebaut werden
   //             konnte. Wird implizit bei der Anmeldung einer
   //             neuen Ressource aufgerufen.
   // Parameter : keine
   // Rückgabe  : alive - konnte zum der Wrapper und der
   //              darunterliegenden Datenquelle eine Verbindung
   //              aufgebaut werden ? (ja = true, nein = false)
   // Ausnahmen : keine
    void ping(out boolean alive);

  }; // Ende der Schnittstellenbeschreibung "Resource"
```

A.1.4 Replikations- und Caching-Dienst-Interface

In diesem Abschnitt wird die Schnittstelle spezifiziert, die der Query-Server nutzt, um auf den Replikations- und Caching-Dienst zuzugreifen. Sie stellt einen vereinfachte Schnittstellenspezifikation der Methoden des Replikations- und Caching-Dienstes dar, die vom Query-Server genutzt werden. Die Implementierung dieser Schnittstelle nutzt die Schnittstelle des eigentlichen Replikations- und Caching-Dienstes.

```
 interface RuCAccess
  {
   // -------------------------------------------------------------
   // Die folgenden Ausnahmen können durch die Methoden des Inter-
   // faces zum Zugriff auf den Replikations- und Caching-Dienst
```

```
// ausgelöst werden.
// ------------------------------------------------------------
// Das angeforderte Dokument konnte nicht gefunden werden.
 exception documentNotFoundException{};

// Beim Laden des Dokumentes ist ein Fehler aufgetreten, der
// näher beschrieben wird.
 exception miscRetrieveException {string reason;};

// Der RuC-Dienst lieferte auf eine D-R-Anfrage zurück, daß vor
// der Auslieferung eine Autorisierung erfolgen muß.
 exception authorizationRequiredException{};

// Der RuC-Dienst lieferte bei auf eine D-R-Anfrage zurück, daß
// die Autorisierung fehlgeschlagen ist. Der Grund dafür wird in
// der Ausnahme näher beschrieben.
 exception authorizationFailedException {string reason;};

// Bei der Bearbeitung durch die Ressourcenverwaltung des RuC-
// Dienstes ist ein Fehler aufgetreten, der näher beschrieben
// wird.
 exception miscManagementException {string reason;};

// Eine Ressource mit der angegebenen Identität ist der
// Ressourcen-Verwaltung des RuC-Dienstes nicht bekannt.
 exception resourceIdNotFoundException{};

// ------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// ------------------------------------------------------------
// Festlegung der Metadaten, die neben dem angeforderten Dokument
// zurückgeliefert werden. Wird von der Methode retrieveDocument
// zurückgeliefert.
 struct documentType
  {
   string documentIdentifier; // DOI
   string dateCreated;        // Erstellungsdatum
   string dateModified;       // letztes Änderungsdatum
   string md5;                // Prüfsumme (RFC1321)
   long   size;               // Größe in Bytes
   documentHandle complete_content; // ein Handle zum Zugriff auf
                              // den Inhalt des Dokumentes
```

```
   string mimetype;          // Format (RFC1521)
  };

// Festlegung, wie sich die Autorisierungsinformation zusammen-
// setzt. Wird der Methode retrieveProtectedDocument als
// Parameter übergeben.
 struct authorizationInformationType
  {
   string id;     // Identität des Nutzers
   string key;    // Schlüssel zur Autorisierung
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind Entscheidungen zu berücksichtigen,
     // die von externen Projektpartnern getroffen werden.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Festlegung, welche Optionen bei einer D-R-Anfrage über-
// geben werden können.
 struct optionType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind Festlegungen zu treffen, welche
     // sinnvollen Möglichkeiten es gibt, die D-R-Anfrage
     // zu beeinflussen um bspw. nur einen Teil des Dokumentes
     // anzufordern.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Beschreibung der Daten, die bei der Anmeldung einer neuen
// Ressource bei der Ressourcen-Verwaltung des RuC-Dienstes
// übergeben werden.
 struct resourceAccessDescriptionType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind implementierungsspezifische
     // Informationen anzugeben, die nicht allgemein beschrieben
     // werden können. Dazu gehört die Beschreibung, wie der
     // Wrapper angesprochen werden kann. Beispielsweise würde
     // dies bei einer einfachen Socket-Verbindung der Name
     // des Hosts sein, auf dem der Wrapper läuft und die Port-
     // nummer, an dem der Wrapper lauscht. Weiterhin könnten
     // weitere Informationen, die für den Administrator von
     // Bedeutung sein können, in dieser Beschreibung enthalten
     // sein.
   // END IMPLEMENTATION_DEPENDENT_PART
```

```
  };

// -----------------------------------------------------------
// Öffentliche Methoden, die der Dienst bereitstellt.
// -----------------------------------------------------------
// Funktion  : Ruft die für das Anfordern von Dokumenten zu-
//             ständige Schnittstelle des RuC-Dienstes auf
//             und erhält ein Dokument und dazugehörende
//             Metadaten als Ergebnis zurück.
// Parameter : documentIdentifier - die DOI des Dokumentes
//             authorizationInformation - Informationen, die
//              zur Autorisierung verwendet werden; dieser Para-
//              meter kann einen bestimmten Wert (entspr. NIL oder
//              NULL) annehmen, der kennzeichnet, das keine
//              Autorisierungsinformation übergeben wird
//             options - Optionen, die die D-R-Anfrage beein-
//              flussen
// Rückgabe  : document - das Dokument (inkl. Metadaten)
// Ausnahmen : documentNotFoundException
//             miscRetrieveException
//             authorizationRequiredException
//             authorizationFailedException
 void retrieveDocument(in string documentIdentifier,
         in authorizationInformationType authorizationInformation,
         in optionType options,
         out documentType document)
  raises (documentNotFoundException,
          miscRetrieveException,
          authorizationRequiredException,
          authorizationFailedException);

// Funktion  : Eine Ressource wurde beim Query-Server neu
//             angemeldet. Dieser registriert durch Aufruf
//             dieser Methode die Ressource beim RuC-Dienst.
//             Daraufhin kann der RuC-Dienst auf die Ressource
//             zugreifen.
// Parameter : resourceIdentifier - die bei der Erstanmeldung
//              durch die Ressourcen-Verwaltung des Query-
//              Servers vergebene eindeutige Identität des
//              LDS
//             resourceAccessDescription - Informationen, wie
//              der RuC-Dienst auf den Wrapper zugreifen kann
// Rückgabe  : keine
// Ausnahmen : miscManagementException
 void registerResource(in string resourceIdentifier,
```

```
        in resourceAccessDescriptionType resourceAccessDescription)
    raises (miscManagementException);

  // Funktion  : Eine Ressource wurde beim Query-Server endgültig
  //             abgemeldet. Durch Aufruf dieser Methode wird der
  //             RuC-Dienst darüber informiert. Dieser darf
  //             daraufhin keine Zugriffe mehr auf die Ressource
  //             durchführen.
  // Parameter : resourceIdentifier - die bei der Erstanmeldung
  //              durch die Ressourcen-Verwaltung des Query-
  //              Servers vergebene eindeutige Identität des
  //              LDS
  // Rückgabe  : keine
  // Ausnahmen : resourceIdNotFoundException
  //             miscManagementException
   void deleteResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            miscManagementException);

  // Funktion  : Eine Ressource wurde beim Query-Server für einen
  //             begrenzten Zeitraum abgemeldet. Durch Aufruf dieser
  //             Methode wird der RuC-Dienst darüber informiert.
  //             Dieser sollte daraufhin keine Zugriffe mehr auf die
  //             Ressource durchführen.
  // Parameter : resourceIdentifier - die bei der Erstanmeldung
  //              durch die Ressourcen-Verwaltung des Query-
  //              Servers vergebene eindeutige Identität des
  //              LDS
  // Rückgabe  : keine
  // Ausnahmen : resourceIdNotFoundException
  //             miscManagementException
   void deactivateResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            miscManagementException);

  // Funktion  : Eine zeitlich begrenzt deaktivierte Ressource
  //             wurde beim Query-Server wieder freigegeben. Durch
  //             Aufruf dieser Methode wird der RuC-Dienst darüber
  //             informiert und darf somit wieder auf die Ressource
  //             zugreifen.
  // Parameter : resourceIdentifier - die bei der Erstanmeldung
  //              durch die Ressourcen-Verwaltung des Query-
  //              Servers vergebene eindeutige Identität des
  //              LDS
  // Rückgabe  : keine
```

```
  // Ausnahmen : resourceIdNotFoundException
  //             miscManagementException
   void reactivateResource(in string resourceIdentifier)
    raises (resourceIdNotFoundException,
            miscManagementException);

 }; // Ende der Schnittstellenbeschreibung "RuCAccess"
```

A.1.5 Föderationsdienst-Interface

In diesem Abschnitt ist die Schnittstelle spezifiziert, die der Query-Server dem Föderationsdienst bereitstellt bzw. die er verwendet, um die Funktionalität des Föderationsdienstes in Anspruch zu nehmen.

```
interface FederationAccess
 {
  // ------------------------------------------------------------
  // Die folgenden Ausnahmen können durch die Föderations-
  // dienstschnittstelle in einer geeigneten Form an den Initiator
  // zurückgegeben werden.
  // ------------------------------------------------------------
  // Beim Laden des Metadatenbestandes durch die Metadaten-Ver-
  // waltung ist ein Fehler aufgetreten, der näher beschrieben
  // wird.
   exception miscFetchException {string reason;};

  // Beim Import des dynamischen Metadatenbestandes der LDS durch
  // die Metadaten-Verwaltung ist ein Fehler aufgetreten, der
  // näher beschrieben wird.
   exception miscVerifyException {string reason;};

  // Die Benachrichtigung des Föderationsdienstes über Ände-
  // rungen des Metadatenbestandes ist aus einem näher beschrie-
  // benen Grund fehlgeschlagen.
   exception miscNotifyException {string reason;};

  // Bei der Bearbeitung der weitergeleiteten Anfrage durch den
  // Föderationsdienst bzw. den Broker ist ein Fehler aufge-
  // treten, der näher beschrieben wird.
   exception miscForwardException {string reason;};

  // ------------------------------------------------------------
  // Definition von komplexen Hilfsdatentypstrukturen, die in
  // Definitionen der komplexen Datentypen auftreten können
  // ------------------------------------------------------------
  // ein Attribut-Wert-Paar
```

```
 struct nameValuePair
  {
   string name;    // Attributname
   string value;   // dazugehöriger Wert
  };

// eine Menge von Attribut-Wert-Paaren
 typedef sequence<nameValuePair> listOfNameValuePairs;

// Festlegung, wie sich die Ergebnistupelliste einer M-R-Anfrage
// zusammensetzt, die vom Föderationsdienst zurückgeliefert
// wird.
 struct resultListTupelType
  {
   string position;             // Rang in der Ergebnisliste
   string documentIdentifier;   // DOI des Dokumentes
   string source;               // Wo wurde das Dokument gefunden ?
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // Neben den Attributen <position>, <documentIdentifier> und
     // <source> könnten weitere Standardattribute (z.B. zur
     // Verwaltung) notwendig sein, die bei einer lokalen Anfrage
     // durch die Wrapper nicht zurückgeliefert werden würden.
     // Diese sind bei Bedarf in diese Struktur einzufügen.
   // END IMPLEMENTATION_DEPENDENT_PART
   listOfNameValuePairs attributeValues; // Selektionsattribute und
                                         // deren Wert(e)
  };

// -------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// -------------------------------------------------------------
// Festlegung, wie sich die zu einer Anfrage gehörenden Prä-
// ferenzen einer Anfrage zusammensetzen. Wird der Methode
// findRelevantResults als Parameter übergeben.
 struct preferencesType
  {
   short maxResults;  // Anzahl der maximal zurückzuliefernden
                      // Ergebnistupel
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung wird die Struktur
     // der zurückgelieferten Metadaten festgelegt.
   // END IMPLEMENTATION_DEPENDENT_PART
```

```
    };

  // Eine geordnete Liste von Ergebnistupeln. Wird von der Methode
  // forwardQuery als Ergebnis zurückgeliefert.
   typedef sequence<resultListTupelType> resultListType;

  // Festlegung, in welcher Form die dynamischen Metadaten, die von
  // der Metadaten-Verwaltung zwischengespeichert sind, an die
  // externen VDS-Dienste weitergeleitet werden. Daten diesen Typs
  // werden von der Methode fetchDynamicMetadata als Ergebnis
  // zurückgeliefert.
   struct dynamicMetaDataType
    {
     // BEGIN IMPLEMENTATION_DEPENDENT_PART
      // In Abhängigkeit von der Implementierung wird die Struktur
      // der zurückgelieferten Metadaten festgelegt.
     // END IMPLEMENTATION_DEPENDENT_PART
    };

  // Festlegung, in welcher Form die statischen Metadaten des
  // LDS im Repository gespeichert und an die entsprechenden
  // externen VDS-Dienste weitergeleitet werden. Daten diesen Typs
  // werden von der Methode getStaticMetadata als Ergebnis
  // zurückgeliefert.
   struct staticMetaDataType
    {
     // BEGIN IMPLEMENTATION_DEPENDENT_PART
      // In Abhängigkeit von der Implementierung wird die Struktur
      // der zurückgelieferten Metadaten festgelegt.
     // END IMPLEMENTATION_DEPENDENT_PART
    };

  // -------------------------------------------------------------
  // Öffentliche Methoden, die der Dienst bereitstellt.
  // -------------------------------------------------------------
  // Funktion  : Exportiert den dynamischen Metadatenbestand der LDS.
  //             Dabei kann in Abhängigkeit von der Implementierung
  //             der vom Query-Server zur Zeit zwischengespeicherte
  //             Datenbestand ausgeliefert werden oder aber eine
  //             Neuanforderung der aktuellen dynamischen Metadaten
  //             bei den LDS erfolgen.
  //             Ein Kriterium für die Entscheidung, ob die aktu-
  //             ellen Metadaten oder die zuletzt importierten Meta-
  //             daten exportiert werden, ist die Zeit, die seit dem
  //             letzten Import verstrichen ist.
```

```
// Parameter : keine
// Rückgabe  : metaData - die dynamischen Metadaten des gesamten
//                        VDS (also der zugeordneten LDS)
// Ausnahmen : miscFetchException
 void fetchDynamicMetadata(out dynamicMetaDataType metaData)
  raises (miscFetchException);

// Funktion  : Exportiert den statischen Metadatenbestand des
//             Repositorys im Query-Server. Dabei werden die Meta-
//             daten der LDS nicht explizit importiert, sondern die
//             zur Zeit im Repository vorhandenen ausgeliefert.
// Parameter : keine
// Rückgabe  : metaData - die statischen Metadaten des gesamten VDS
//                        (also der zugeordneten LDS)
// Ausnahmen : miscFetchException
 void fetchStaticMetadata(out staticMetaDataType metaData)
  raises (miscFetchException);

// Funktion  : Löst das Laden der dynamischen Metadaten der LDS
//             eines VDS aus, die daraufhin im Repository zwischen-
//             gespeichert werden.
// Parameter : keine
// Rückgabe  : keine
// Ausnahmen : miscVerifyException
 void verifyMetadata()
  raises (miscVerifyException);

// Funktion  : Informiert den Föderationsdienst, daß sich der vom
//             Query-Server verwaltete Metadatenbestand geändert
//             hat. Daraufhin sollte der Föderationsdienst die
//             aktualisierten Daten vom Query-Server anfordern.
// Parameter : keine
// Rückgabe  : keine
// Ausnahmen : miscNotifyException
 void notify()
  raises (miscNotifyException);

// Funktion  : Weiterleiten der Anfrage an den Broker (via
//             Föderations-Dienst), der eine Liste von Ergebnis-
//             tupeln zurückliefert.
// Parameter : query - die IRQL-Anfrage
//             preferences - Einstellungen, die bei der Bearbeitung
//                           zu berücksichtigen sind
// Rückgabe  : resultList - eine Liste, die alle Anfrageergebnisse
//                          enthält
```

```
    // Ausnahmen : miscForwardException
     void forwardQuery(in string query,
                       in preferencesType preferences,
                       out resultListType resultList)
      raises (miscForwardException);

  }; // Ende der Schnittstellenbeschreibung "FederationAccess"
```

A.2 Lokaler Dokumenten-Server

In diesem Abschnitt wird die Schnittstelle eines Wrappers spezifiziert, der einen LDS anbindet. Diese Schnittstelle wird vom Ressourcen-Interface des Query-Servers und vom Replikations- und Caching-Dienst genutzt.

```
interface Wrapper
 {
    // ------------------------------------------------------------
    // Die folgenden Ausnahmen können durch das Wrapper-Interface
    // in einer geeigneten Form an den Initiator der Anfrage zurück-
    // gegeben werden.
    // ------------------------------------------------------------
    // Das angeforderte Dokument ist nicht im LDS gespeichert.
     exception documentNotFoundException{};

    // Bei der Bearbeitung der D-R-Anfrage durch den Wrapper ist ein
    // Fehler aufgetreten, der näher beschrieben wird.
     exception wrapperException {string reason;};

    // Bei der Bearbeitung der D-R-Anfrage durch die Datenquelle ist
    // ein Fehler aufgetreten, der näher beschrieben wird.
     exception sourceException {string reason;};

    // ------------------------------------------------------------
    // Definition von komplexen Hilfsdatentypstrukturen, die in
    // Definitionen der komplexen Datentypen auftreten können
    // ------------------------------------------------------------
    // ein Attribut-Wert-Paar
     struct nameValuePair
      {
       string name;    // Attributname
       string value;   // dazugehöriger Wert
      };

    // Festlegungen, die nicht durch die IRQL ausgedrückt werden
    // konnten aber das Anfrageergebnis beeinflussen. Diese Prä-
```

```
// ferenzen setzen sich aus einem Teil der Präferenzen, die
// vom Nutzersystem an das Query-Interface übergeben wurden,
// und weiteren Festlegungen zusammen, die durch den Query-
// Server getroffen wurden. Dabei ist eine Normalisierung der
// Bezeichner erforderlich, damit alle Wrapper die Bezeichner
// gleich interpretieren.
// Bsp.: <"RankingAlgorithm","TermsCount">
 struct preferenceType
  {
   string preferenceName;  // Für einen VDS eindeutiger Name
                           // einer Festlegung.
   string preferenceValue; // Wert dieser Festlegung.
  };

// Festlegung, wie sich die Ergebnistupelliste einer M-R-Anfrage
// zusammensetzt, die ein Wrapper zurückliefert.
 struct resultListTupelType
  {
   string position;            // Rang in der Ergebnisliste
   string documentIdentifier;  // DOI des Dokumentes
   listOfNameValuePairs attributeValues; // Selektionsattribute und
                                         // deren Wert(e)
   string localRankingValue;   // durch die Datenquelle ermittel-
                               // ter Ranking-Wert
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung können weitere
     // Daten aufgeführt werden, anhand derer u.a. von der Daten-
     // quelle der Ranking-Wert bestimmt wurde. Diese können
     // gegebenenfalls vom Query-Server dazu genutzt werden, um
     // einen globalen Wert zu ermitteln.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// --------------------------------------------------------------
// Definition komplexer Datentypen, die in den Methoden dieser
// Schnittstelle als Parameter bzw. als Rückgabewert auftreten.
// Diese Definitionen sind als vorläufig anzusehen und sollten
// entsprechend der tatsächlichen Anforderungen geändert werden.
// --------------------------------------------------------------
// eine Menge von Attribut-Wert-Paaren
 typedef sequence<nameValuePair> listOfNameValuePairs;

// Festlegung der Struktur, in der ein angefordertes Dokument
// zurückgeliefert wird. Wird von der Methode getDocument
// zurückgeliefert. Die hier bereits festgelegten Attribute
```

```
// werden bei der Modellierung eines Dokument-Objektes im Query-
// Server gefordert (siehe entsprechende Spezifikation). Diese
// sind deshalb unbedingt vom Wrapper zu liefern und vom RuC-
// Dienst an den Query-Server weiterzuleiten.
 struct documentType
  {
   string documentIdentifier; // DOI
   string dateCreated;        // Erstellungsdatum
   string dateModified;       // letztes Änderungsdatum
   string md5;                // Prüfsumme (RFC1321)
   long   size;               // Größe in Bytes
   octed  complete_content;   // der Inhalt des Dokumentes (kein
                              // Handle !)
   string mimeType;           // Format (RFC1521)
   string xmlDTD;             // DTD, falls Dok. ein XML-Dokument
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind bei Bedarf weitere Attribute
     // einzufügen, die beim Entwurf der Schnittstellen des
     // RuC-Dienstes als erforderlich eingestuft werden
     // (aber zur Zeit noch nicht vorliegen).
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Festlegung, welche Optionen bei einer D-R-Anfrage an den
// Wrapper übergeben werden können.
 struct optionType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind bei Bedarf Optionen einzufügen,
     // die neben der DOI erforderlich sein könnten, um auf
     // das Dokument zugreifen zu können.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Festlegung, welche Art Autorisierung erforderlich ist (z.B.
// Gruppe, Nutzer) usw. Diese Information nutzt der Autorisie-
// rungsdienst des RuC-Dienstes, um die Autorisierung durchzu-
// führen.
 struct authInformationType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // An dieser Stelle sind Entscheidungen zu berücksichtigen,
     // die von externen Projektpartnern getroffen werden.
   // END IMPLEMENTATION_DEPENDENT_PART
  };
```

```
// Eine Menge von Festlegungen, die bei der Anfragebearbeitung
// durch den Wrapper zu berücksichtigen sind. Wird der
// Methode documentMetaDataQuery als Parameter übergeben.
 typedef sequence<preferenceType> preferencesListType;

// Eine geordnete Liste von Ergebnistupeln. Wird von der Methode
// documentMetaDataQuery als Ergebnis zurückgeliefert.
 typedef sequence<resultListTupelType> resultListType;

// Festlegung, in welcher Form die dynamischen Metadaten des
// LDS durch den Wrapper an den Query-Server zurückgeliefert
// werden. Daten diesen Typs werden von der Methode
// getResourceDynamicMetaData als Ergebnis zurückgeliefert. Die
// Struktur muß äquivalent zu der Struktur sein, die im
// Ressourcen-Interface des Query-Servers spezifiziert wurde, da
// ansonsten eine Konvertierung erforderlich ist.
 struct dynamicMetaDataType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung wird die Struktur
     // der zurückgelieferten Metadaten festgelegt.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// Festlegung, in welcher Form die statischen Metadaten des
// LDS durch den Wrapper an den Query-Server zurückgeliefert
// werden. Daten diesen Typs werden von der Methode
// getResourceStaticMetaData als Ergebnis zurückgeliefert. Die
// Struktur muß äquivalent zu der Struktur sein, die im
// Ressourcen-Interface des Query-Servers festgelegt wurde, da
// ansonsten eine Konvertierung erforderlich ist.
 struct staticMetaDataType
  {
   // BEGIN IMPLEMENTATION_DEPENDENT_PART
     // In Abhängigkeit von der Implementierung wird die Struktur
     // der zurückgelieferten Metadaten festgelegt.
   // END IMPLEMENTATION_DEPENDENT_PART
  };

// ------------------------------------------------------------
// Öffentliche Methoden, die der Dienst bereitstellt.
// ------------------------------------------------------------
// Funktion  : Laden eines Dokumentes aus der darunterliegenden
//             Datenquelle und Anforderung der Metadaten, die bei
```

```
//             D-R-Anfragen standardmäßig mitgeliefert
//             werden
// Parameter : documentIdentifier - die DOI des angeforderten
//              Dokumentes
//             options - Optionen, die die D-R-Anfrage beein-
//             flussen
// Rückgabe  : document - das Dokument (inkl. Metadaten)
//             requiredAuthorizationInformation - Festlegung,
//              welche Art von Autorisierung erfolgen muß, um auf
//              das Dokument zugreifen zu dürfen
// Ausnahmen : documentNotFoundException
//             wrapperException
//             sourceException
 void getDocument(in string documentIdentifier,
        in optionType options,
        out documentType document,
        out authInformationType requiredAuthorizationInformation)
  raises (documentNotFoundException,
          wrapperException,
          sourceException);

// Funktion  : Auslösen der ‘Anfrageübersetzung, Weiterleitung
//             der Anfrage an die Datenquelle, Nachbearbeitung‘
//             im Wrapper
// Parameter : query - die IRQL-Anfrage
//             preferences - weitere Festlegungen zur Anfrage,
//              die nicht durch die IRQL ausgedrückt werden
//              können
// Rückgabe  : resultList - nach Ranking-Wert geordnete Liste der
//              Ergebnistupel einer Anfrage
// Ausnahmen : wrapperException
//             sourceException
 void documentMetaDataQuery(in string query,
                            in preferencesListType preferences,
                            out resultListType resultList)
  raises (wrapperException,
          sourceException);

// Funktion  : Auslösen der Methode, die die statischen Metadaten
//             der Datenquelle einliest.
// Parameter : keine
// Rückgabe  : metaData - die statischen Metadaten der Datenquelle
// Ausnahmen : wrapperException
//             sourceException
 void getResourceStaticMetaData(out staticMetaDataType metaData)
```

```
     raises (wrapperException,
             sourceException);

   // Funktion  : Auslösen der Methode, die die dynamischen Metadaten
   //             der Datenquelle einliest.
   // Parameter : keine
   // Rückgabe  : metaData - die dynamischen Metadaten der Datenquelle
   // Ausnahmen : wrapperException
   //             sourceException
    void getResourceDynamicMetaData(out dynamicMetaDataType metaData)
     raises (wrapperException,
             sourceException);

   // Funktion  : Testweiser Aufbau einer Verbindung zur Datenquelle.
   //             Wird vom Ressourcen-Interface des Query-Servers aus-
   //             gelöst, um zu überprüfen, ob Wrapper und Daten-
   //             quelle ansprechbar sind.
   // Parameter : keine
   // Rückgabe  : alive - konnte zur darunterliegenden Datenquelle
   //              eine Verbindung aufgebaut werden ? (ja = true, nein
   //              = false)
   // Ausnahmen : keine
    void pong(out boolean alive);

}; // Ende der Schnittstellenbeschreibung "Wrapper"
```

Thesen

I Die in der IRQL vorgenommene Verschmelzung von Metadaten-Retrieval-Anfragen und Dokument-Retrieval-Anfragen wird im VDS aufgehoben, um den differenzierten Anforderungen bei der Bearbeitung gerecht zu werden.

II Die Nutzung eines Handles für den Zugriff auf die Ergebnisse und den Inhalt eines Dokumentes führt zu einer flexiblen Handhabung von umfangreichen Datenmengen und verbessert so die Robustheit des Systems.

III Die Verwendung von Java als Implementierungssprache, der IDL zur Schnittstellenspezifikation sowie CORBA als Middlewareschicht zur Verteilung der VDS-Dienste führt zur flexiblen Einsetzbarkeit, Umsetzbarkeit und Erweiterbarkeit des VDS.

IV Keines der betrachteten Systeme kann mit seiner Anfragesprache die Prädikate und Klauseln einer IRQL-Anfrage vollständig umsetzen.

V Der *Fulcrum SearchServer* bietet eine gute Basis für die Umsetzung der IR-spezifischen IRQL-Prädikate. Die Datenbankfunktionalität des Systems muß aber wesentlich erweitert und die fehlenden Prädikate sowie Operatoren des SQL-Anfrageteils implementiert werden, damit es auch als vollwertiges DB-System genutzt werden kann.

VI Das *Harvest*-System ist für die Umsetzung der IRQL ungeeignet, da es nicht über die erforderlichen DB- und IR-Konzepte verfügt.

VII Der *DB2-TextExtender* bietet für alle betrachteten IR-spezifischen IRQL-Prädikate eine entsprechende Funktion an. Die Einschränkung auf bestimmte Indexformen und die mangelnde Kombinierbarkeit der Funktionen führen jedoch zu einer stark eingeschränkten Nutzbarkeit, wodurch die Simulation durch den Wrapper unnötig erschwert wird.

VIII Die Kombination der DB-Funktionalität von *DB2* mit der Text-Retrieval-Funktionalität des *Fulcrum SearchServers* würde eine sehr gute Basis für die Umsetzung der IRQL-Anfragesprache bieten.