# Kopplung von objektrelationalen Datenbanksystemen mit externen Textretrieval-Werkzeugen

Diplomarbeit

Universität Rostock, Fachbereich Informatik


vorgelegt von Raiko Nitzsche
geboren am 25.04.1975 in Bergen/Rügen

Betreuer: Prof. Dr. Andreas Heuer
Dr.-Ing. Holger Meyer

Abgabedatum: 30. Juni 2000

## Zusammenfassung

Für die Speicherung und Verwaltung von Textdokumenten unterschiedlichster Art existieren heutzutage zwei verschiedene Ansätze. Auf der einen Seite haben sich spezialisierte Dokumentverwaltungssysteme entwickelt, welche über ausgefeilte Indizierungs- und Retrieval-Techniken verfügen. Auf der anderen Seite stehen die bekannten Datenbanksysteme, welche durch neue Entwicklungen multimediale Daten speichern können (ORDBMS). Leider sind die Retrieval-Techniken der ORDBMS nicht so leistungsfähig wie die der spezialisierten Systeme. Deren Schwächen wiederum liegen in der Organisation der Zugriffe auf die Dokumente sowie deren Speicherung. Viele Anwendungen wie z.B. digitale Bibliotheken benötigen nicht nur Zugriff auf den eigentlichen Textinhalt, sondern auch auf zusätzliche Meta-Daten (strukturierte Daten). Der Zugriff auf strukturierte Daten wiederum ist eine der Stärken der DBMS. Diese Arbeit beschreibt einen Ansatz zur Kombination der beiden Systeme, um ihre jeweiligen speziellen Fähigkeiten für die Dokumentverwaltung nutzbar zu machen. Es wird eine Architektur vorgestellt, mit der existierende Textretrieval-Werkzeuge zur Indizierung und Suche in einem herkömmlichen ORDBMS eingesetzt werden können.

## Abstract

Today, there exist two different approaches to store and manage all kinds of text documents. On one hand, specialized document management systems were developed, which provide sophisticated indexing and retrieval techniques. On the other hand, there are the well known database systems, which are now able to handle new multimedia data types due to new developments such as object-relational and object-oriented enhancements. Unfortunately, the retrieval techniques of these ORDBMS are far less efficient than those of the specialized systems. However, these systems have a poor or no access management as well as weak storage capabilities. Many application, e.g. digital libraries, not only need access to the text content but also to some meta information (structured data). Access to structured data is one of the strengths of DBMSs. This paper describes an approach to combine both systems in a way that their special features can be used for document retrieval. An architecture will be introduced which allows the use of existing text-retrieval tools for indexing and search in a conventional ORDBMS.

## CR-Klassifikation

**H.2.2** [Database Management]: Physical Design — Access Methods

**H.2.3** [Database Management]: Languages — Query Language

**H.2.4** [Database Management]: Systems — Query Processing, Textual Databases

**H.3.3** [Information Storage and Retrieval]:
Information Search and Retrieval — Query formulation, Search Process

## Key Words

Objektrelationale Datenbanken, Suchmaschinen
Text Retrieval, Information Retrieval

# Inhaltsverzeichnis


**1 Einführung** **1**
- 1.1 Zielsetzung . . . 3
- 1.2 Gliederung . . . 3

**2 Textretrieval-Werkzeuge** **5**
- 2.1 Überblick . . . 5
- 2.2 Ausgewählte Systeme und deren Zuordnung . . . 6
  - 2.2.1 Fulcrum SearchServer . . . 6
  - 2.2.2 Managing Gigabytes - MG . . . 10
  - 2.2.3 Weitere Suchmaschinen . . . 12
- 2.3 Zusammenfassung . . . 14

**3 ORDBMS mit Textretrieval-Erweiterungen** **15**
- 3.1 Speicherung großer Daten in ORDBMS . . . 16
- 3.2 IBM DB2 Universal Database . . . 17
  - 3.2.1 Erweiterungsmöglichkeiten von DB2 UDB . . . 17
  - 3.2.2 Der Text-Extender . . . 19
- 3.3 INFORMIX Universal Server . . . 22
  - 3.3.1 Die DataBlade-Technologie . . . 23
  - 3.3.2 Das Excalibur Text-DataBlade . . . 25
- 3.4 Zusammenfassung . . . 28

**4 Kopplungsarchitekturen** **29**
- 4.1 Suchmaschine mit Datenbankanbindung . . . 29
- 4.2 Datenbanken mit Suchmaschinenunterstützung . . . 31
- 4.3 Middleware-Ansatz . . . 34
- 4.4 Zusammenfassung . . . 36

**5 Architekturvorschlag** **37**
- 5.1 Anforderungen an die Architektur . . . 37
- 5.2 Bewertung der Architekturen . . . 38
- 5.3 Entwicklung einer neuen Architektur . . . 41
  - 5.3.1 Die Schnittstellen . . . 43
  - 5.3.2 Zugriff der Suchmaschine auf die Dokumente . . . 52
  - 5.3.3 Unterstützte Produkte . . . 53
- 5.4 Eigenschaften und Erweiterungsmöglichkeiten . . . 55
  - 5.4.1 Verwaltung mehrerer Suchmaschinen . . . 57
- 5.5 Zusammenfassung . . . 59

**6 Prototypische Implementierung** **61**
6.1 Ergebnisse und Erfahrungen . . . 61
6.2 Datenbankerweiterungen . . . 63
6.2.1 DB2 UDB . . . 63
6.2.2 INFORMIX . . . 64
6.3 Suchmaschinen-Module . . . 65
6.3.1 FULCRUM . . . 66
6.3.2 MG System . . . 69
6.4 Konkrete Kopplungen . . . 71
6.5 Zusammenfassung . . . 73

**7 Schlußbetrachtung** **75**
7.1 Ergebnisse . . . 76
7.2 Ausblick . . . 77

**A Definition der Schnittstellen** **79**
A.1 Die Textretrieval-Erweiterung . . . 79
A.2 Der Wrapper . . . 80
A.3 Das Steuerprogramm . . . 83
A.3.1 Indexerzeugung mit Fulcrum . . . 85

**B Tabellarischer Vergleich** **88**

**Abkürzungen** **90**

**Abbildungen** **91**

**Tabellen** **92**

**Literatur** **93**

# Kapitel 1

# Einführung

Die zunehmende Leistungsfähigkeit der Computer sowie die stärker werdende Vernetzung eröffnen heutzutage immer neue Möglichkeiten für den Einsatz von Computern. Vor einiger Zeit noch stellte das Speichern von sehr vielen Kundendaten mit Namen und Telefonnummern und vor allem der schnelle Zugriff auf ausgewählte Daten eine große Herausforderung für die Rechentechnik und die Datenbanksysteme dar. Heutzutage wird darüber hinaus noch der Schriftverkehr, Musik und sogar Filme in digitaler Form von Computern verwaltet. Der große Vorteil der digitalen Technik liegt dabei nicht nur in der kompakten Speicherung, so daß z.B. alle Bücher einer Bibliothek in nur einem Rechner abgespeichert werden können, sondern vor allem in einer wesentlich schnelleren Suche bestimmter Dokumente und einem schnellerem Zugriff auf diese.

Für die Speicherung und Suche multimedialer Daten wurden spezielle Programme entwickelt. So entstanden z.B. reihenweise Dokumentverwaltungsprogramme, wie WAIS, Fulcrum, OpenText und andere (siehe [Mey96]), mit deren Hilfe große und sehr viele Dokumente in einem Computer gespeichert und wiedergefunden werden können. Da die Suche in Multimedia-Dokumenten wie Texten oder Videos auch mit einem Computer noch sehr viel Zeit beansprucht, wurden neue Indizierungsmethoden erarbeitet und in diesen Systemen eingesetzt, was die Suche extrem beschleunigte. Auf diese Weise entwickelten sich beispielsweise *Digitale Bibliotheken*, Online-Nachschlagewerke und andere computergestützte Verwaltungssysteme. Die unzähligen Dokumente des World Wide Web stellen selbst eine Art digitale Bibliothek dar, obwohl es sich hierbei nicht um eine zentrale, sondern verteilte Datenhaltung handelt, deren Zugriffsformen ebenso verschieden sind wie deren Datenquellen. Die Suche und Navigation erfolgt hier über die zahlreichen, bekannten Web-Suchmaschinen, wobei aber keine den gesamten Inhalt des Webs erfaßt.

Während früher solch große Datenmengen von ausgereiften Datenbanksystemen verwaltet wurden, werden heutzutage die meisten Dokumente in vielen verschiedenen, spezialisierten Systemen gehalten. Selbst die noch in Datenbanken gespeicherten multimedialen Daten werden von "Fremdsystemen" verwaltet, in Bezug auf Indizierung und Suche. Die Datenbanken dienen dabei, genau wie das Filesystem, nur noch als persistenter Speicherplatz. Dadurch gehen die wichtigsten Datenbankeigenschaften und damit ihre Vorzüge verloren und lassen sie für die Anwendung im Multimediabereich in einem schlechten Licht erscheinen, da sie eher als kompliziert gelten. Auf der anderen Seite wird mit neuen Programmen, als Aufsatz auf die bestehende Systeme, versucht, wieder eine Integration der in vielen verstreuten und heterogenen Systemen gespeicherten Daten zu erzielen — eine Eigenschaft, die DBMS von sich aus bieten.

In den letzten Jahren allerdings konnte die Datenbankforschung und -entwicklung die traditionellen Datenbanksysteme dahingehend erweitern, daß sie ihrerseits

die neuen, semistrukturierten Datentypen, wie Geospatial-Daten, Text, Video oder Audio verwalten können (siehe [Car86, Cha98, Loh91, Sch86, Sto96]). Aus diesen Entwicklungen gingen neue Datenbankgenerationen, wie die *objektrelationalen* und *objektorientierten* Datenbanken, hervor. Diese Datenbanksysteme sind in ihrem Typsystem und ihrer Anfragesprache erweiterbar und bieten dem Nutzer somit die Möglichkeit, neue Datentypen und Funktionen (oder Methoden) zu erzeugen, um das System an die neuen Daten anzupassen und gleichzeitig diese Daten zu manipulieren und zu durchsuchen. Dadurch lassen sich die neuen Daten in das Datenbanksystem integrieren und ermöglichen deren Verwaltung bei gleichzeitiger Beibehaltung aller Vorzüge eines DBMS.

Durch diese Entwicklung stehen sich heutzutage zwei Lager von Produkten zur Verwaltung multimedialer Dokumente gegenüber. Auf der einen Seite stehen die Spezialprogramme zur Dokumentverwaltung und auf der anderen die erweiterbaren Datenbanksysteme. Die Stärken jedes dieser Lager liegen in unterschiedlichen Bereichen, was sich durch ihre Herkunft begründet. Die spezialisierten Dokumentverwaltungsprogramme haben ihren Ursprung im ***I**nformation **R**etrieval* (IR) und bieten daher alle oder zumindest sehr viele Fähigkeiten in sehr ausgereifter Form aus diesem Bereich (z.B. verschiedene Retrieval-Modelle, inkrementelle Suche, Relevance Feedback, usw.). Dagegen wurde bisher die Notwendigkeit von Funktionen zur eigentlichen Dokumentverwaltung, wie parallele Anfragebearbeitung, Transaktionszugriffe und Sicherung der Datenintegrität, unterschätzt und kaum implementiert.

Die Stärken der erweiterbaren Datenbanksysteme liegen in ihren ererbten Datenbankeigenschaften, die die eben aufgezählten Funktionen schon seit Jahren beinhalten. Für die multimedialen Datentypen wurden neue Zugriffsmethoden bzw. Indexmechanismen entwickelt, wie z.B. der R-Baum ([Gut84]) und der P-Baum ([Jag90]) für Geospatial Daten, oder der D-Baum ([Dav96]) für Textdaten. Allerdings ist die Integration neuer Indexstrukturen in ein Datenbanksystem sehr aufwendig und wurde wegen der zu speziellen Funktion der Zugriffsmethoden (im Gegensatz zu B-Bäumen, die für alle bisher unterstützten Datentypen genutzt werden können) nur in sehr wenigen Systemen umgesetzt. Andere Systeme geben dem Nutzer die Möglichkeit, eigene Indexstrukturen in das DBMS zu integrieren, was allerdings einer gewissen Vorkenntnis des Systems bedarf und zudem recht aufwendig ist (siehe [Nit99] für Arten der Erweiterung von Zugriffsmethoden in DBMS). Allerdings konnte durch diese erweiterten Zugriffsmechanismen nicht die Funktionalität der spezialisierten Programme im Bereich des IR erreicht werden.

Die unterschiedlichen Stärken der Systeme spaltet die Anwender von solchen Programmen ebenfalls in zwei Lager. Je nachdem wo ihre Prioritäten bei der Dokumentverwaltung liegen, in einer ausgereiften Datenverwaltung oder einer effizienten Such- und Anfragemöglichkeit, müssen sie sich für die eine oder andere Seite entscheiden. Bisher ist es nicht gelungen, ein System auf den Markt zu bringen, das beide Anforderungen in ausreichendem Maße erfüllt. Es wurde aber von beiden Seiten erkannt, daß die Fähigkeiten des jeweils anderen Systems sehr wohl von den Anwendern benötigt wird. Die Systeme entwickeln sich mehr und mehr aufeinander zu; Dokumentverwaltungsprogramme bekommen mehr Datenbankfunktionalität und die Erweiterungsmodule für die Datenbanken werden immer komplexer und bieten weitere IR-Funktionalität.

Viele Arbeiten, die sich mit der Evaluierung von Systemen zur Verwaltung multimedialer Dokumente, insbesondere Textdokumente, befassen, stellen am Ende fest, daß ein optimales System eine Kombination aus einem spezialisierten Dokumentverwaltungsprogramm und einem erweiterbaren Datenbanksystem wäre. Dadurch würde der Nutzer in den Genuß aller benötigten Funktionen kommen und müßte nicht zu einer Kompromißlösung greifen, bei der die eine oder andere Funktionalität gar nicht oder nur in begrenztem Maße verfügbar ist.

## 1.1 Zielsetzung

Durch die intensive Arbeit sowohl mit Dokumentverwaltungsprogrammen und mit erweiterbaren Datenbanksystemen entstand die Idee, daß es theoretisch möglich ist, beide Systeme für die Dokumentverwaltung zu nutzen. Die Systeme würden dabei so miteinander gekoppelt, daß ihre jeweiligen speziellen Fähigkeiten ausgenutzt werden. Die Arbeit soll in zwei Aufgaben unterteilt und jede dieser Aufgaben dem System zugeteilt werden, das diese Aufgabe am besten erfüllen kann, d.h. die Speicherung und Zugriffsverwaltung der Dokumente wird von einer Datenbank übernommen, und den Bereich der Indizierung und Suche der Dokumente übernimmt ein spezielles Dokumentverwaltungsprogramm.

In dieser Arbeit soll eine Architektur entwickelt werden, die diese Arbeitsteilung ermöglicht. Dazu sollen bestehende Produkte aus beiden Lagern untersucht werden, um die optimale Art der Kopplung der Systeme zu finden. Eine Zielsetzung dabei ist, daß die bestehenden Systeme möglichst einfach und ohne Eingriff in deren Quelltext (der in den meisten Fälle nicht zur Verfügung steht) in der Architektur eingesetzt werden können. Dadurch sollen die Anwender auf einfache Weise verschiedene Systemkombinationen testen können, um eine optimale Kombination für ihre Anforderungen zu finden. Darüber hinaus soll die Möglichkeit einer engeren Integration bestehen, um die Performance des Gesamtsystems im Bedarfsfall weiter zu verbessern. Die Architektur muß dabei offen für bestehende und zukünftige Systeme sein, was bedeutet, daß Verbesserungen in einem der beiden Systeme auch dem Gesamtsystem zu Gute kommen.

Diese Arbeit befaßt sich nicht mit Techniken der Textanalyse und der Textindizierung, wie z.B. [WMB99], oder mit der Erstellung neuer Datenmodelle für Datenbanksysteme, wie [Fuh94]. Dadurch können bestimmte Einschränkungen der relationalen Algebra und den darauf basierenden Anfragesprachen in Bezug auf die Verwendung im Information Retrieval nicht komplett ausgeräumt werden. So müssen Features, wie die Navigation in Hypertexten, inkrementelle Anfragen und Relevance Feedback, auch weiterhin von spezialisierten Anwendungsprogrammen übernommen werden.

Hauptsächlich konzentriert sich diese Arbeit auf die Integration von bestehenden Werkzeugen zum Textretrieval und zur Datenverwaltung. Im einzelnen bedeutet das: Wie kann ein Textretrieval-Werkzeug auf die Daten einer Datenbank zugreifen und wie können im Gegenzug die Ergebnisse aus dem Retrieval-Prozeß von einer Datenbank zur Dokumentselektion verwendet werden?

## 1.2 Gliederung

Die Arbeit entwickelt schrittweise die genauen Anforderungen für eine solche Kopplung und daraus folgend die eigentliche Architektur, welche durch eine prototypische Implementation getestet wird. Dazu ist die Arbeit wie folgt gegliedert:

Im Kapitel 2 werden unterschiedliche Werkzeuge für das Information Retrieval vorgestellt und deren Aufbau und Funktionsweise untersucht. Dabei werden die einzelnen Produkte klassifiziert, um später Aussagen über bestimme Gruppen von Textretrieval-Werkzeugen in Bezug auf die Tauglichkeit für den Einsatz in der neuen Architektur treffen zu können.

Das Gleiche erfolgt im Kapitel 3 für die objektrelationalen Datenbanksysteme. Hierbei wird jedoch keine Gruppeneinteilung vorgenommen, sondern unterschiedlich leistungsfähige Erweiterungsschnittstellen der ORDBMS vorgestellt und deren Verwendbarkeit und Auswirkung auf die Kopplungsarchitektur untersucht.

Nach der Vorstellung alle Produkte, die in dieser Arbeit untersucht wurden, widmet sich Kapitel 4 den verschiedenen Möglichkeiten einer Kopplung der Systeme.

Die Lösungen stammen meist von einigen der untersuchten Produkte, die ihrerseits bereits eine Ankopplung der Systeme anstreben. Dabei werden vor allem die Vor- und Nachteile der einzelnen Architekturen beleuchtet.

Im Kapitel 5 werden schließlich aus den gewonnenen Erfahrungen die Anforderungen an die neue Lösung abgeleitet und schrittweise eine Architektur konzipiert, die diesen Anforderungen entspricht. Es werden die einzelnen Module mit ihrer Funktion und ihren Schnittstellen vorgestellt. Danach werden die genauen Eigenschaften der Architektur erläutert und die Erweiterungsmöglichkeiten aufgezeigt.

Die Entwicklung der Architektur wurde von Testimplementationen begleitet und wesentlich von ihnen beeinflußt. Die dabei entstandenen Prototypen von gekoppelten Datenbanken und Suchmaschinen werden in Kapitel 6 vorgestellt. Es werden Probleme mit den einzelnen Produkten aufgezeigt und die Erfahrungen bei der Implementation zusammengefaßt.

Zum Ende erfolgt im Kapitel 7 eine Schlußbetrachtung der Arbeit. Es werden die Ergebnisse der Arbeit zusammengefaßt und abgeschätzt, in wie weit die erarbeitete Lösung der Anfangsidee eines gekoppelten Dokumentverwaltungssystems entspricht. Am Schluß erfolgt ein Ausblick auf mögliche Erweiterungen der Architektur, um weiteren Anforderungen gerecht zu werden.

Der Anhang A enthält eine Auflistung der Prototypdateien und Schnittstellendefinitionen dieser Architektur, wie sie auch von den Testimplementationen verwendet wurden. Außerdem wird an Beispielprogrammen dieser Implementationen die konkrete Funktionsweise spezieller Module der Architektur erläutert.

# Kapitel 2

# Textretrieval-Werkzeuge

Dieses Kapitel stellt vorhandene Textretrieval-Werkzeuge und deren Architekturen vor. Zuerst wird ein Überblick über die Produkte gegeben und Merkmale für die Unterscheidung und Einteilung erläutert. Danach werden konkrete Systeme vorgestellt und auf Grund ihrer Architektur und Funktionalität einer Gruppe zugeordnet. Den Abschluß des Kapitels bildet die Zusammenfassung der gewonnenen Erkenntnisse.

## 2.1 Überblick

Die Palette der verfügbaren Textretrieval-Produkte ist sehr groß und reicht von einer Websuchmaschine bis hin zur Dokumentverwaltung in einem Intranet. Dabei unterscheiden sich die Systeme nicht nur in der gebotenen Retrieval-Technik und -Qualität, sondern vor allem auch in ihrem Aufbau und der unterstützten Funktionalität. Man kann die Systeme grob in folgende Klassen einteilen:

- Indexmaschine
- Suchmaschine
- Volltextdatenbank

Die Einteilung erfolgt dabei anhand der unterstützten Funktionen. Es ist keinesfalls so, daß die Systeme der einzelnen Klassen komplett verschieden sind. So enthalten z.B. alle Suchmaschinen auch einen Indizierer und eine Volltextdatenbank nutzt eine Suchmaschine für die Indizierung und Anfragebearbeitung. Die Übergänge zwischen den einzelnen Klassen sind sehr vage und unterliegen in vielen Fällen subjektiven Einschätzungen.

#### Indexmaschinen

Indexmaschinen bilden die Grundlage für die meisten Textretrieval-Systeme. Da das komplette Durchsuchen der Dokumente bei einer Anfrage zu zeitaufwendig ist, werden die Texte vorher indiziert. Es gibt die unterschiedlichsten Methoden für die Indizierung von Texten — die bekannteste davon ist die *invertierte Liste*. Einen guten Überblick über die verschiedenen Techniken bietet [WMB99].

Die Indexmaschinen implementieren jeweils eine solche Indizierungsmethode. Sie zeichnen sich durch eine sehr einfache Schnittstelle aus. Im Prinzip unterstützen sie nur zwei Funktionen — die Textindizierung und die Suchanfrage. Daher beschränkt sich die Schnittstelle auf das Anlegen eines Indexes für eine Dokumentensammlung, die in einem systemabhängigen Format bereitgestellt werden muß. Unterstützt eine Indexmaschine mehrere Indexe gleichzeitig (was für die meisten gilt), wird jedem

Index eine Bezeichnung zugeordnet. Über diese Bezeichnung kann der Indizierer bei einer Anfrage auf den richtigen Index zugreifen.

Die zweite wichtige Funktion ist das Stellen einer Anfrage an den Indizierer. Dieser bearbeitet die Anfrage mit Hilfe des angegebenen Indexes und liefert eine Liste mit denjenigen Dokumenten bzw. Dokumentbezeichnungen (Identifikatoren) zurück, die sich für die Anfrage qualifizieren.

Damit ist die Leistungsfähigkeit einer Indexmaschine erschöpft.

#### Suchmaschinen

Funktionen, wie die Verwaltung von Dokumenten, das Bereitstellen einer Nutzerschnittstelle zum Indizieren und Suchen von Dokumenten, sowie die Koordination von mehreren Indexen und Anfragen, werden von den Suchmaschinen angeboten. Sie bilden ein eigenständiges Produkt für die Suche in Textsammlungen. Für die Dokumentensuche stehen wie auch bei den Indizierern nur reine Retrieval-Kommandos zur Verfügung.

Suchmaschinen können in Bezug auf die Anwenderfreundlichkeit unterschiedlich weit ausgebaut sein. Sie können z.B. nur einfache Textkommandos oder aber eine grafische Benutzeroberfläche bieten. Im allgemeinen bieten Suchmaschinen eine eigene Schnittstelle für den Anwender, über die benutzerspezifische Anwendungsprogramme auf die Suchmaschinen zugreifen können.

#### Volltextdatenbanken

Volltextdatenbanken bilden das obere Ende der Textretrieval-Systeme. Sie vereinen die Funktionen eines DBMS zur Datenverwaltung mit den Funktionen eines Textretrieval-Systems. Die Datenverwaltung ist zwar auf das Speichern von Texten spezialisiert, es können aber durchaus weitere Datentypen, wie z.B. Datum und Zahlen, gespeichert werden, die als Attribute den Texten zugeordnet werden. Dadurch bieten Volltextdatenbanken im Gegensatz zu den beiden oben genannten Systemen neben Retrieval-Kommandos auch attributierte Anfragen und Mehr-Variable-Anfragen.

In der Praxis finden sich solche Systeme leider sehr selten. Genaugenommen untersucht diese Arbeit neue Möglichkeiten der Kopplung von Textretrieval-Werkzeugen mit herkömmlichen Datenbanken, gerade weil die vorhandenen Systeme den Anforderungen (entweder beim Textretrieval oder bei der Datenbankfunktionalität) nicht ausreichend gerecht werden.

## 2.2 Ausgewählte Systeme und deren Zuordnung

Im folgenden werden spezielle Textretrieval-Systeme vorgestellt und eine Klassifikation entsprechend den Kategorien aus dem vorigen Abschnitt vorgenommen. Die Systeme FULCRUM SearchServer und MG System lagen beide zu Testzwecken vor. Daher können zu diesen Systemen genauere Aussagen getroffen werden als zu den im Abschnitt 2.2.3 angesprochenen Produkten. Die Informationen zu diesen beruhen ausschließlich auf Literaturrecherchen der Produktdokumentation und Onlinepräsentationen der entsprechenden Hersteller. Im Anhang B befindet sich ein tabellarischer Überblick über die Systeme FULCRUM und MG (Tabelle B.1).

### 2.2.1 Fulcrum SearchServer

Das Fulcrum Retrieval-System ist eines der bekanntesten Retrieval-Werkzeuge. In vielen Evaluierungen (z.B. [Rah98, Tit99]) werden seine hervorragenden Retrieval-Qualitäten hervorgehoben.

Der SearchServer ist ein Client/Server-basiertes Information Retrieval-System. Es ist als eine Art Dokumentensuchmaschine innerhalb des Intranets einer Firma konzipiert und unterstützt eine Vielzahl von Dokumentformaten und -quellen.

In einem Unternehmen werden unzählige Dokumente an den verschiedensten Orten und in unterschiedlicher Form gespeichert. So kann es z.B. einen WWW-Server mit Präsentationen und Ankündigungen oder auch Online-Hilfen geben. Daneben existiert noch eine Datenbank, in welcher der komplette Schriftverkehr, ein- und ausgehende Briefe, verwaltet werden. Für die interne Kommunikation wird vielleicht ein Lotus Notes oder Microsoft Exchange Server verwendet. Der Quellcode der Programmierer schließlich wird ganz einfach im Filesystem gespeichert. Die Suche nach bestimmten Informationen ist also sehr aufwendig, wenn man nicht schon vorher genau weiß, wo man mit der Suche beginnen soll.

Diesem Problem widmet sich Fulcrum mit dem SearchServer (wie auch einige andere Hersteller), indem eine einheitliche Oberfläche für die Suche von Dokumenten bereitgestellt wird. Über diese Anwenderschnittstelle bietet Fulcrum einen transparenten Zugriff auf alle vom System verwalteten Datenquellen. Das bedeutet, der Anwender muß nicht wissen, wo die von ihm gesuchten Dokumente evtl. gespeichert sein könnten.

Natürlich bietet der SearchServer nicht nur einen einheitlichen Zugriff auf alle Dokumente einer Firma, sondern auch, oder vor allem, alle Features eines modernen Information Retrieval-Systems. Zur Suche nach Dokumenten mit entsprechenden Informationen bietet das System drei Retrieval-Modelle und vier verschiedene Ranking-Algorithmen, die vom Nutzer festgelegt werden können. Weitere Informationen zu den Retrieval-Eigenschaften findet man in [SSI98, Rah98, Tit99].

### Die Architektur des SearchServers

Der SearchServer ist nach dem Client/Server Prinzip aufgebaut. Die Kommunikation zwischen den Anwendungsprogrammen und dem SearchServer erfolgt über eine von zwei genormten Schnittstellen für den Zugriff auf Datenquellen — ODBC oder JDBC. Demnach präsentiert sich der SearchServer als einheitliche Datenquelle für alle Dokumente im Intranet. Mit Hilfe von speziellen ODBC Driver Managern kann Fulcrum damit homogen in eine bestehende Anwendungsumgebung eingebunden werden und ist sogar über das Netzwerk verfügbar. Bei der JAVA-Anbindung sind diese Features auch ohne zusätzliche Manager gegeben. In Abbildung 2.1 ist der Aufbau des gesamten Systems schematisch dargestellt.

Das System kann grob in drei Hauptkomponenten eingeteilt werden:

- Client Application (z.B. Fulcrum FIND! Client)
- SearchServer
- Textreader

### Die Client Applikation

Sie übernimmt die Kommunikation mit dem Anwender. Fulcrum stellt beispielsweise das Programm FIND! zur Verfügung. Dieses soll als einheitliche Oberfläche für die Suche nach entsprechenden Dokumenten dienen. Das Programm gibt es auch als WebFIND!, wodurch es praktisch auf jeder Workstation mit einem Webbrowser verfügbar ist. Es ist aber auch möglich, beliebige benutzerspezifische Anwendungen, sowohl als Standardprogramm oder als Webanwendung, zu implementieren. Existierende Anwendungen, die bisher über ODBC oder JDBC auf andere Datenbanken zugegriffen haben, können mit geringem Aufwand an den SearchServer angepaßt werden.

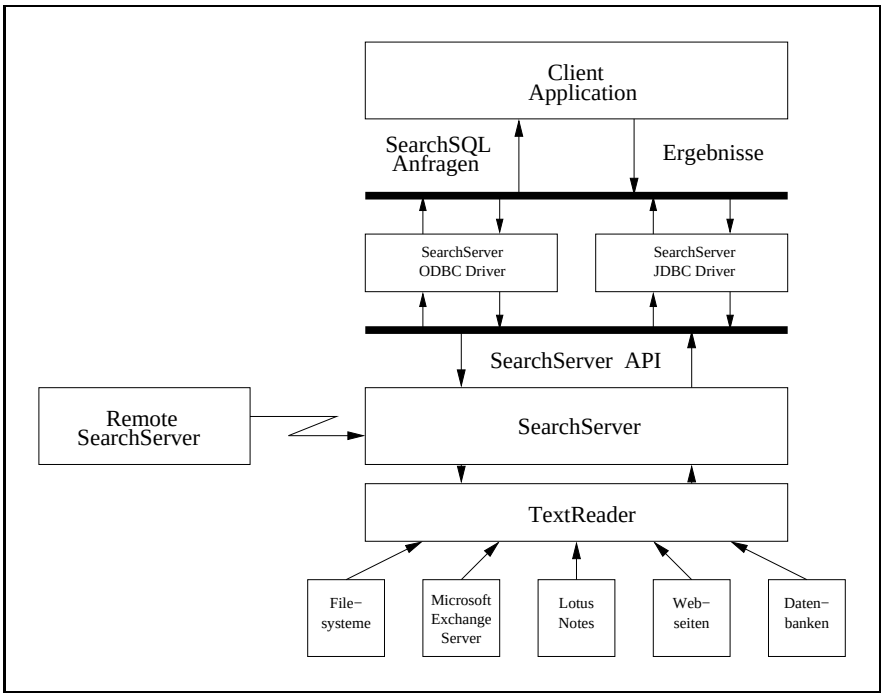


Abbildung 2.1: Architektur des Fulcrum SearchServers

### Der SearchServer

Der SearchServer übernimmt die eigentliche Verwaltung der indizierten Dokumente. Dabei werden die Daten ähnlich wie in einer relationalen Datenbank abgespeichert. In Fulcrum können Tabellen angelegt werden, in denen die entsprechenden Dokumente und evtl. weitere Informationen eingefügt werden. Dabei ist zu beachten, daß nicht die Dokumente selbst, sondern nur Referenzen auf sie im System gespeichert werden. Die referentielle Integrität der Daten kann von Fulcrum dabei nicht zugesichert werden. Das bedeutet, es kann unter Umständen auf Dokumente zugegriffen werden, die bereits gelöscht wurden. Das Update der Tabelle erfolgt in festgelegten Zyklen oder durch Anforderung durch den Benutzer.

Zu jedem eingefügten Dokument können nach Bedarf noch weitere Informationen in Extraspalten gespeichert werden, z.B. der Autor und das Erstellungsdatum des Dokumentes. Diese Informationen können in eine Suchanfrage einbezogen werden. Da Fulcrum als Volltextsystem keine Strukturen innerhalb von Texten beachtet, z.B. bei HTML- oder XML-Dokumenten, ist es so möglich, die benötigten Daten außerhalb als Attribute dem Text zuzuordnen.

Die Anfrage an den SearchServer erfolgt über einen SQL Dialekt (SearchSQL), der zum einen im Bereich der strukturierten Anfrage eingeschränkt, zum anderen aber um IR-spezifische Anteile erweitert wurde (Einzelheiten in [Tit99]).

Eine weitere Aufgabe für den SearchServer ist die Netzunterstützung. Wie in Abbildung 2.1 zu sehen war, ist es möglich, auf andere entfernte SearchServer im Netz zuzugreifen und deren Daten mit zu nutzen. Dadurch werden auch Datenquellen nutzbar, die keine eigene Netzanbindung bieten, wie z.B. ein lokales Filesystem. Der SearchServer wird hierzu auf der gleichen Maschine wie die Datenquelle installiert und hat somit Zugriff auf diese Daten. Über die Netzfähigkeiten des SearchServers sind diese Daten dann auch über das Netz verfügbar.

### Die Textreader

Fulcrum benutzt für das Einlesen von Dokumenten, zu deren Indizierung und Ausgabe sogenannte Textreader. Dabei handelt es sich um Filterprogramme, die einen Text von einem Format in ein anderes konvertieren. Dem System liegt ein Standard-Textreader bei, der etwa 250 (nach Angaben des Herstellers) verschiedene Textformate unterstützt. Die Formate werden dabei automatisch beim Einlesen erkannt.

Die Schnittstelle der Textreader ist von Fulcrum offengelegt, damit bei Bedarf eigene Reader vom Benutzer geschrieben werden können, z.B. um ein anwendungsspezifisches Format zu unterstützen.

Eine weitere Spezialität der Reader und des Fulcrum Systems ist es, daß mehrere Filter hintereinander ausgeführt werden können, ähnlich dem "pipe" Prinzip unter UNIX. So reicht die Konvertierungsroutine der eigenen Textverarbeitung vom eigenen Format nach beispielsweise ***Rich Text Format*** (RTF) als Filter für Fulcrum. Da der Standard-Textreader RTF-Dokumente verarbeiten kann, wird der eigene Filter einfach davor ausgeführt und liefert somit das Dokument in einem dem System verständlichen Format.

Am Schluß muß immer der Fulcrum Textreader ausgeführt werden, da er das Dokument in ein Fulcrum-eigenes Format wandelt, auf welchem die Indizierung stattfindet.

Da es dem Fulcrum System praktisch egal ist, woher der Textreader seine Dokumente bezieht, ist es auch möglich, z.B. einen Datenbank-Reader zu implementieren. Und tatsächlich liegt dem System ein Datenbank-Reader für relationale DBMS mit ODBC-Anbindung bei ([SSD98]). Eine Konfigurationsdatei beschreibt, welche Attribute aus der Datenbank in welche Spalten der SearchServer Tabelle zu speichern sind. Dabei ist es sogar möglich, Attribute aus virtuellen Tabellen, sprich VIEWs oder auch JOINs, in Fulcrum zu speichern.

Da die Texte in der Datenbank meist in ihrem Originalformat oder zumindest in ASCII vorliegen, muß nach dem Datenbank-Reader ein weiterer Filter zur Textkonvertierung aufgerufen werden.

Mit dieser Architektur steht dem Fulcrum SearchServer praktisch jede denkbare Datenquelle zur Verfügung.

### Einordnung

Das Fulcrum System bietet eine Menge Funktionalität, die über die reine Textindizierung hinausgeht. Das Speichern der Daten in relationalen Tabellen, die SQL ähnliche Anfragesprache, sowie die ODBC-Schnittstelle zum System haben Fulcrum den Status eines Volltextdatenbanksystems eingebracht. Allerdings sei dazu angemerkt, daß Fulcrum die Texte gar nicht selbst speichert, sondern nur die Referenzen auf Texte, die mittels Textreader bei Bedarf gelesen werden. Die eigentliche Datenhaltung übernehmen also andere Systeme.

Durch dieses Prinzip fehlen Fulcrum wesentliche Datenbankeigenschaften, vor allem die Datenintegrität — Fulcrum hat keine wirkliche Kontrolle über die Daten. Auch eine Nutzerverwaltung fehlt dem System gänzlich. Es gibt keine Transaktionen, welche aber in einem reinen Anfragesystem weniger Bedeutung haben; trotzdem sind auch Anfragen während eines Indexupdates möglich. Andererseits unterstützt der SearchServer den parallelen Zugriff und die verteilte Datenhaltung.

Die starken Einschränkungen bei den Datenbankeigenschaften sind der Grund für die teils negativen Ergebnisse bei Datenbankevaluierungen zum Thema Information Retrieval. Es ist eben kein richtiges Datenbanksystem, dafür aber ein überaus gutes Textretrieval-Werkzeug. Es kann als der Beginn der Skala für Volltextdatenbanken betrachtet werden.

### 2.2.2 Managing Gigabytes - MG

Das MG System ist ein Public Domain Textretrieval-System, welches auf verschiedenen UNIX-Plattformen läuft. Es wurde als Forschungsprototyp zum Buch *Managing Gigabytes* ([WMB99]) entwickelt, um die darin vorgestellten Verfahren und Algorithmen praktisch zu demonstrieren. Außerdem wurde damit eine Reihe von Tests zur Performance und Qualität verschiedener Komprimierungs- und Indizierungsverfahren durchgeführt.

Da es sich vorwiegend um ein Demonstrations- und Testsystem handelt, bietet es nur eine sehr simple Nutzerschnittstelle. Die Hauptarbeit wurde in die Retrieval-Maschine investiert. Zur Zeit arbeitet MG nur auf statischen Dokumentensammlungen, d.h. es können keine Dokumente zur Laufzeit hinzugefügt oder gelöscht werden. Die komplette Sammlung muß einmal indiziert werden, bevor eine Suche möglich ist. Werden Dokumente geändert, hinzugefügt oder gelöscht, muß die Sammlung neu indiziert werden.

Trotzdem stellt MG ein sehr mächtiges und schnelles Werkzeug für die Volltextsuche dar. Es wird z.B. von der Melbourner Forschungsgruppe, die am *TREC* Projekt beteiligt war, eingesetzt und bildet die Suchmaschine für die *New Zealand Digital Library* (NZDL).

#### Die Architektur von MG

Das MG System besteht aus einer Sammlung verschiedener Shell-Skripte und C-Programme. Die wichtigsten Programme sind `mgbuild` (Anlegen eines Indexes), `mg_get` (Bereitstellen der Dokumente im korrekten Format) und `mgquery` (Nutzerinterface für Suchanfragen). In Abbildung 2.2 ist der Zusammenhang zwischen den Programmen schematisch dargestellt.

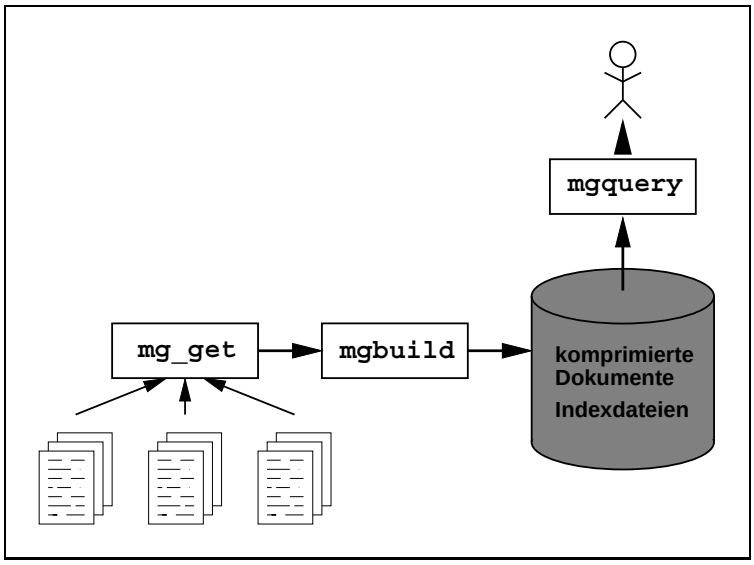


Abbildung 2.2: Funktionsweise des MG Systems

Diese Programme greifen dabei noch auf weitere kleinere Programme zurück, z.B. zum Komprimieren von Texten, zur Indexerzeugung, zur Indexkomprimierung, usw. In diesen Modulen sind die entsprechenden Algorithmen aus dem Buch umgesetzt.

Mittels `mgbuild` werden die einzelnen Unterprogramme beim Aufbau eines Indexes koordiniert. Dies umfaßt das Einlesen der zu indizierenden Texte, das Erfassen und Indizieren aller auftretenden Wörter und anschließendes Komprimieren der

Texte. Optional können noch weitere Zusatzindexfiles erzeugt werden, die die Suche weiter beschleunigen.

Am Ende liegen eine komprimierte Textdatei, die alle Dokumente enthält, und eine Reihe von Index- und Hashtabellen vor. Alle Files zusammen bilden im MG System eine sogenannte "Textdatenbank", auf welche gezielt Suchanfragen gestellt werden können, um Dokumente aus der Datenbank zu extrahieren. Die originalen Textdaten werden nach dem Anlegen dieser Datenbank nicht mehr benötigt.

### Der Textreader

Ein wichtige Rolle bei der Indexerzeugung spielt das Programm `mg_get`. Es ist sozusagen der Textreader des MG Systems. Dieses Programm ist ebenfalls ein Skript und kann nach Belieben des Nutzers geändert werden. Die Fähigkeiten dieses Programms bestimmen, welche Dokumente mit MG indiziert werden können. Es gibt eine festgelegte Schnittstelle zum MG System, die dieses Programm erfüllen muß. Woher es die Dokumente bezieht, ist praktisch frei wählbar. Die Testimplementation bietet allerdings nur die Unterstützung für bestimmte Dokumentensammlungen. Das können mehrere Dokumente in einer einzigen Datei sein (z.B. eine UNIX-Maildatei), wobei einstellbar ist, wie die Dokumente voneinander getrennt sind. Außerdem können mehrere Dateien in einem Unterverzeichnis als Dokumentensammlung indiziert werden.

Auch wenn die Fähigkeiten des bestehenden Systems nicht sehr beeindruckend sind, eröffnet doch die Möglichkeit der Erweiterung ungeahnte Quellen. So können in Datenbanken gespeicherte Text über einen entsprechenden Datenbank-Reader mittels ODBC/JDBC oder auch eine systemspezifische Schnittstelle ausgelesen und indiziert werden. WEB Inhalte können durch eine Art WEB-Crawler (Programm zum Extrahieren von Dokumenten aus dem WWW) und die verschiedensten Dokumentformate durch entsprechende Filterprogramme dem System zugänglich gemacht werden.

### Das Anfrageprogramm

Nachdem die entsprechenden Indexfiles angelegt und die Dokumente komprimiert wurden, können mittels `mgquery` Anfragen an diese Sammlung gestellt werden. Das Programm liest die zuvor erzeugten Tabellen und Indexfiles in den Speicher und beantwortet dann die Nutzeranfragen.

Mit Hilfe der Indexfiles werden die entsprechenden Dokumente und deren Position innerhalb der komprimierten Textdatei ermittelt. Für die Datenkompression wurde extra ein Algorithmus verwendet, der es erlaubt, gezielt entsprechende Teilabschnitte wieder zu entpacken. Dadurch wird verhindert, daß bei einem Zugriff auf ein Dokument die gesamte Kollektion wieder dekomprimiert werden muß.

Mit MG können nur reine Retrieval-Anfragen an die Textsammlung gestellt werden. Bis auf die Anzahl der indizierten Dokumente sind keine zusätzlichen Metainformationen verfügbar.

### Einordnung

Das MG System ordnet sich in die Gruppe der Indexmaschinen ein. Es bietet keine Nutzerverwaltung, Parallelverarbeitung oder Transaktionskonzepte. Die Dokumentverwaltung beschränkt sich auf die Zusammenfassung aller Dokumente in einer Datei. Der Zugriff ist danach allein über Textsuchanfragen möglich; ein direkter Zugriff auf einzelne Dateien wird nicht unterstützt, genausowenig wie das Verändern von Dokumenten. Es eignet sich also nur zur Nutzung mit statischen Textsammlungen. Dafür bietet es aber sowohl gute Retrieval-Qualität als auch gute Performance.

Die Eigenschaften von MG sind durch seine Entstehungsgeschichte und die Einsatzgebiete zu erklären. Durch seine offene Architektur und die freie Verfügbarkeit der Quellen bietet es eine ideale Testumgebung für neue und bekannte Algorithmen. Darüber hinaus kann es aber auch für praktisches Textretrieval eingesetzt werden (siehe NZDL).

Wie im ersten Abschnitt des Kapitels beschrieben, ist der Übergang vom Indizierer zur vollständigen Suchmaschine sehr verschwommen. Durch die Zusatzprogramme und die Möglichkeiten, z.B. dem Textreader (`mg_get`) mehr Funktionalität zu verleihen, kann sich das System durchaus zu einer Suchmaschine entwickeln.

### 2.2.3 Weitere Suchmaschinen

In diesem Abschnitt werden weitere Suchmaschinen kurz vorgestellt. Diese Systeme wurden nicht in der Praxis getestet, da sie bis auf die IBM Textsuchmaschine nicht zur Verfügung standen. Im Falle der Textsuchmaschine wurde diese mit in die Untersuchung zum DB2 Text-Extender einbezogen, da sie nicht als alleinstehendes Produkt genutzt werden kann.

#### IBM Textsuchmaschine

Unter dem simplen Namen Textsuchmaschine vertreibt IBM sein Produkt zur Unterstützung der Textsuche. Die Textsuchmaschine bietet keine eigene Benutzeroberfläche, sondern lediglich eine Programmierschnittstelle. Über spezielle Funktionen kann die Suchmaschine von anderen Anwendungsprogrammen angesprochen werden. Abbildung 2.3 zeigt schematisch die Nutzung der Textsuchmaschine.

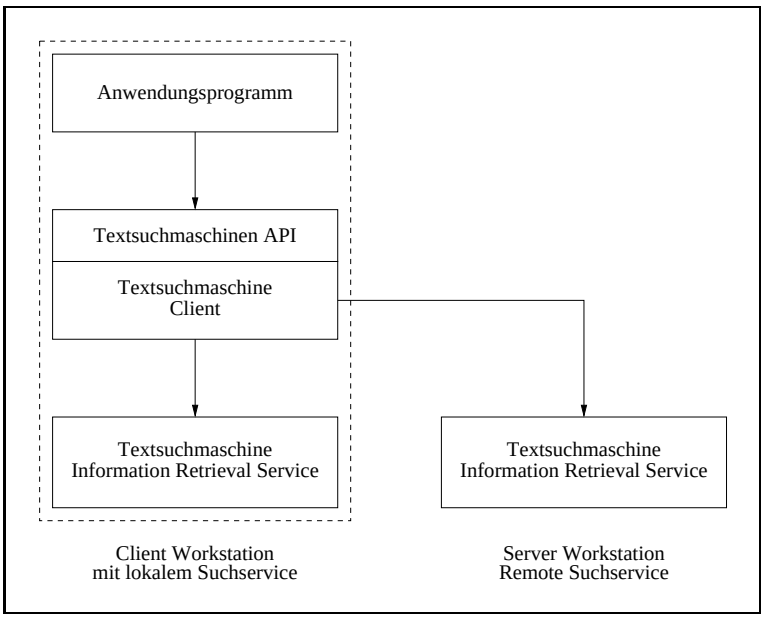


Abbildung 2.3: Beispielkonfiguration der Textsuchmaschine

Die Suchmaschine wird von IBM selbst in vielen eigenen Textverwaltungsapplikationen eingesetzt — z.B. im *Intelligent Miner for Text* oder beim *DB2 Text-Extender*, der im Kapitel 3 noch näher vorgestellt wird.

Die Textsuchmaschine unterstützt vier verschiedene Indexvarianten, sowie Stopwortlisten und Thesauri. Die Indexe werden unabhängig voneinander verwaltet und angesprochen, so daß es möglich ist, mehrere Indexe für ein und dieselbe Dokumentensammlung anzulegen. Das ist vor allem deshalb sinnvoll, da jede Indexart

nur bestimmte Anfragen unterstützt. Durch die Verwendung mehrerer Indizes stehen dem Nutzer mehr Varianten der Textsuche zur Verfügung.

Das System entspricht ganz seinem Namen und ist daher der Gruppe der Suchmaschinen zuzuordnen. Durch die vier verschiedenen Indexarten wird deutlich, daß hier vier Indexmaschinen zum Einsatz kommen, die jeweils einen bestimmten Index unterstützen.

### Websuchmaschinen

Eine sehr bekannte und verbreitete Art von Suchmaschinen sind die Websuchmaschinen. Dazu zählen allseits bekannte Produkte wie AltaVista, Excite, Lycos und viele andere. Bei allen handelt es sich um Suchmaschinen mit einer Webschnittstelle. Jedes System implementiert seine eigene Indexvariante und Retrieval-Methode. Abbildung 2.4 gibt einen groben Überblick über die Funktionsweise einer Websuchmaschine.

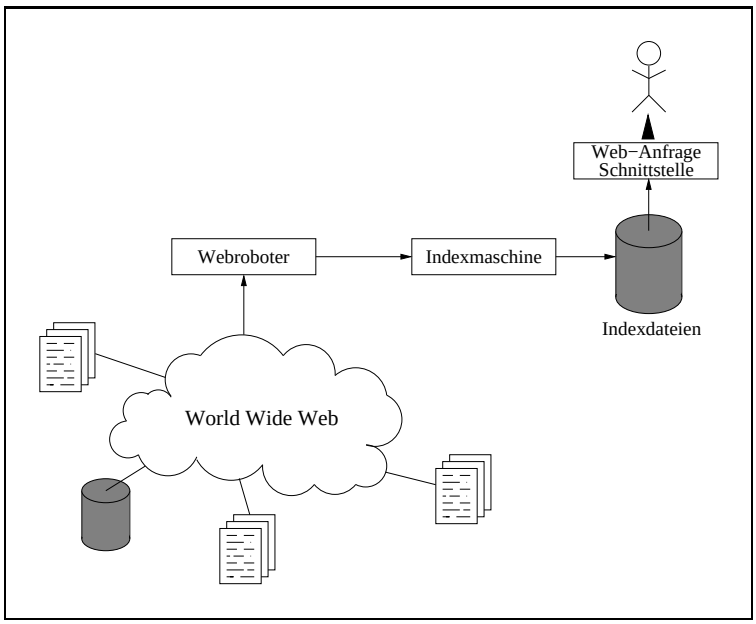


Abbildung 2.4: Funktionsweise einer Websuchmaschine

Die Dokumente werden über einen Webroboter (Tool zum automatischen Lesen und Verfolgen von Webseiten) aus dem World Wide Web (WWW) extrahiert und dann indiziert. Dieses Tool stellt also den Textreader bei dieser Architektur dar. Bei allen Websuchmaschinen werden nur die Links zu den gefundenen Dokumenten gespeichert, nicht die eigentlichen Dokumente. Dadurch kommt es häufig vor, daß als Treffer angezeigte Links auf nicht mehr vorhandene Webseiten zeigen oder aber die Seiten bereits verändert wurden und die gewünschte Information nicht mehr enthalten.

Ein weiteres Problem bei der Suche nach Dokumenten im Web stellen die sogenannten dynamischen HTML-Seiten dar. Sie werden erst bei einer Anfrage erzeugt, meistens mit Inhalten aus einer Datenbank. Einige Suchmaschinen haben spezielle Techniken entwickelt, um solche Datenbanken zu “plündern” — wie es in [Sch99] genannt wurde. Dazu stellen sie so viele verschiedene Anfragen wie möglich und erhalten immer neue Dokumente mit Inhalten aus der Datenbank. Ein ziemlich aufwendiges und unbefriedigendes Verfahren — durch fehlende Anfrageschnittstellen im WWW aber momentan das einzig funktionsfähige.

Eine weitere Entwicklung bei den Websuchmaschinen sind die sogenannten Meta-Suchmaschinen (z.B. MetaCrawler). Sie stellen eine höhere Anwenderschicht dar, die mehrere "normale" Websuchmaschinen zur Anfragebearbeitung benutzt. Diese Meta-Suchmaschinen besitzen weder eigene Indizes noch speichern sie Dokumente oder deren Links. Sie verteilen lediglich die Anfrage auf verschiedene Suchmaschinen und fügen die Resultate wieder zusammen. Eine guten Überblick über Meta-Suchmaschinen gibt [Tit98].

## 2.3 Zusammenfassung

In diesem Kapitel wurde das breite Anwendungsspektrum für das Textretrieval und unterschiedliche Produkte für die einzelnen Anwendungsfälle vorgestellt. Wie man sehen konnte, existieren Abhängigkeiten zwischen den einzelnen im ersten Abschnitt vorgestellten Gruppen in der Art, daß die Systeme sich gegenseitig nutzen.

Interessant ist der äquivalente Aufbau aller vorgestellten Systeme. Die Dokumente werden von externen Systemen verwaltet und über sogenannte Textreader den Suchmaschinen zugänglich gemacht. Die Suchmaschine indiziert mit Hilfe von Indexmaschinen die Dokumentensammlung und bietet dem Anwender eine mehr oder minder komplexe Anfrageschnittstelle zur Suche in der Dokumentensammlung. Dabei kann es sich um eine einfache Programmierschnittstelle wie bei der IBM Textsuchmaschine handeln oder aber um ein graphisches Tool wie z.B. bei Fulcrum.

Welches Produkt am besten geeignet ist, hängt von der Anwendung ab. Sollen verschiedene Datenquellen innerhalb eines Intranets über eine einheitliche Oberfläche den Mitarbeitern zugänglich gemacht werden, so empfehlen sich Systeme wie Fulcrum SearchServer. Für spezielle Nutzerapplikationen die eine Textsuche anbieten müssen oder wollen, eignen sich Indexmaschinen wie MG oder Suchmaschinen wie die IBM Textsuchmaschine. Internetnutzer werden für ihre Suche nach Informationen weiterhin die Websuchmaschinen verwenden.

Die einzige Anwendung, die von existierenden Systemen nicht ausreichend unterstützt wird, ist die wirklich effiziente Speicherung und Verwaltung großer Textmengen, kombiniert mit den guten Retrieval-Fähigkeiten der heutigen Werkzeuge. Für diesen Bereich kämen die Volltextdatenbanken in Frage. Leider fehlen dem Markt noch gute Produkte aus diesem Bereich.

# Kapitel 3

# ORDBMS mit Textretrieval-Erweiterungen

Datenbanken spielen auch im multimedialen Zeitalter nach wie vor eine bedeutende Rolle bei der Speicherung und Verwaltung großer Datenmengen. Durch die Anforderungen der "neuen" Datentypen, wie Video, Audio, Bilder und Text, mußten die herkömmlichen Datenbanksysteme erweitert werden. Dabei wurden die unterschiedlichsten Strategien verfolgt. Als die zwei bedeutendsten Richtungen entwickelten sich die ***O**bjekt**r**elationalen **D**aten**b**anken* (ORDB) und die ***O**bjekt**o**rientierten **D**aten**b**anken* (OODB).

Die ORDB stellen dabei eine Erweiterung der bekannten ***R**elationalen **D**aten**b**anken* (RDB) dar. So bleiben prinzipiell alle Vorteile dieser Systeme, wie z.B. die ausgefeilten Optimierer, weiterhin bestehen. Zusätzlich verfügen die ORDB aber auch über objektorientierte Eigenschaften wie typisierte Attribute und Tabellen, sowie über Typ- und Tabellenhierarchien.

Die OODB sind eine komplett neue Entwicklung. Sie implementieren objektorientierte Datenmodelle. Einige OODB speichern ihre Daten allerdings trotzdem noch in einem unterliegenden RDB oder ORDB und bieten nur nach außen dem Anwender eine objektorientierte Sicht auf seine Daten. Trotz der großen Euphorie in Bezug auf die OODB Anfang der 90er konnten sich diese Systeme bisher noch nicht in dem erwarteten Maße durchsetzen. Während man sie damals noch als eine Ablösung für die "überholten" RDB betrachtete, findet man sie heute lediglich vereinzelt in Spezialanwendungen. Ein Grund dafür kann die fehlende Standardisierung der Anfragesprache sein. Außerdem sind die Optimierer für OODB heute noch lange nicht so effektiv wie die der RDB. Dadurch leidet die Qualität und Performance in Bezug auf Anfrage und Retrieval. Die Zukunft wird zeigen, welche Verbesserungen im Bereich der OODB noch möglich sind, und ob sie eines Tages doch noch die bestehenden ORDB ablösen werden.

In diesem Kapitel werden ausschließlich die objektrelationalen DBMS und ihre Erweiterungsmöglichkeiten betrachtet. [Heu97] beschäftigt sich ausführlich mit den Konzepten objektorientierter Datenbanken und untersucht spezielle Produkte auf die Umsetzung dieser Konzepte.

Als konkrete Systeme werden hier die objektrelationalen Datenbanksysteme DB2 UDB von IBM und INFORMIX Universal Server und ihre entsprechenden Volltexterweiterungen untersucht. Die Auswahl der Systeme erfolgte sowohl aufgrund der Verfügbarkeit als auch der Leistungsfähigkeit. Diese Arbeit untersucht nicht die Retrieval-Qualitäten der einzelnen Volltexterweiterungen — dazu sei auf [Por99] für speziell diese beiden Systeme und [Rah98] für weitere Volltextsysteme verwiesen. Vielmehr untersucht diese Arbeit die Art und Weise der Integration von herkömm-

licher Datenbankarchitektur mit der neuen Textretrieval-Funktionalität. Nach einer getrennten Betrachtung der einzelnen Systeme erfolgt am Ende des Kapitels eine Zusammenfassung und Gegenüberstellung der wichtigsten Merkmale.

## 3.1 Speicherung großer Daten in ORDBMS

Datenbankmanagementsysteme waren bisher im wesentlichen auf schnellen Zugriff und Manipulation von einfachen alphanumerischen Daten spezialisiert. Durch die rasante Entwicklung im Bereich der Computerhardware und der Computervernetzung über das Internet, haben sich die Anforderungen der Anwender verändert. Zunehmend gewannen Bilder, Sprache und sogar Videos an Bedeutung. Für die Speicherung und Verwaltung solch riesiger Daten waren herkömmliche DBMS nicht vorgesehen, weswegen diese anderweitig verwaltet wurden — evtl. unter Zuhilfenahme spezieller Software. Andererseits konnte auch nicht auf Datenbanken verzichtet werden, da wichtige Daten der Unternehmen in großem Umfang darin gespeichert waren. So kam es zu einer Situation, in der Anwendungsprogramme entstanden, welche die Informationen von den verschiedensten Datenquellen zusammensammelten und sie dann dem Nutzer präsentierten. Für die Programmierer dieser Programme kam erschwerend hinzu, daß jede dieser Datenquellen eine eigene Schnittstelle für den Zugriff auf die Daten mitbrachte. Dieser Umstand verkomplizierte und verteuerte die Software.

Die Datenbankhersteller reagierten darauf, indem sie ihre Systeme mit der Fähigkeit ausstatteten, auch sehr große Daten speichern zu können. Dazu wurde ein neuer Datentyp namens ***B**inary **L**arge **Ob**ject* (BLOB) eingeführt ([Cha98, INF00]). Dieser Datentyp kann je nach System bis zu 2 Gigabyte Daten zusammenhängend aufnehmen. Außerdem kann jede Art von Information darin gespeichert werden, da sich das Datenbanksystem nicht um den Inhalt kümmert. D.h. Bilder, Audio und Videos können gleichermaßen von diesem Datentyp aufgenommen werden.

Allerdings blieb ein Problem ungelöst. Traditionelle RDBMS bieten die Möglichkeit, eine begrenzte Anzahl von Datentypen — Integer, Fließkommazahlen, Datum und Zeichenketten — sehr effizient zu verwalten. Dazu bieten die Systeme eine Sammlung nützlicher Funktionen für die eingebauten Datentypen:

- inhaltsbasierte Anfragen
- Vergleichsoperatoren
- intelligente Anfrageoptimierung
- effiziente Speichermöglichkeiten
- Indizierungsmethoden angepaßt an die Daten
- einfache Aggregationsfunktionen

Durch die Speicherung der neuen Daten in dem Datentyp BLOB werden diese in einen "zweite Klasse"-Status verwiesen. Da BLOBs nicht-interpretierte Bit-Muster sind, weiß ein herkömmliches DBMS nicht, wie es eine inhaltsbasierte Anfrage auf diesen Typ ausführen soll. Durch das Fehlen sinnvoller Vergleichsoperatoren gibt es auch keine Möglichkeit, einen optimalen Anfrageplan zu erstellen, wodurch eine effiziente Speicherung und Bereitstellung von BLOB-Daten erschwert wird.

Die Interpretation der Daten muß deshalb von den Anwendungsprogrammen übernommen werden. Das bedeutet, für jedes individuelle Programm muß die Funktionalität immer wieder neu implementiert werden. Außerdem müssen die kompletten BLOB-Daten über das Netz zum jeweiligen Programm geschickt werden, was ein zusätzliche Belastung des Netzwerkes darstellt.

Einige Hersteller haben versucht, ihre Datenbanken den Bedürfnissen ihrer Kunden anzupassen, indem sie einen oder mehrere der neuen Datentypen unterstützten ([INF00]). Aber keiner hat es geschafft, eine umfassende Unterstützung für alle Daten in einem Server bereitzustellen, wie z.B. aus dem Vergleich der Retrieval-Eigenschaften verschiedener Datenbankerweiterungen in [Por99] deutlich wird. Vor allem dauert es zu lange, bis die entsprechende Unterstützung verfügbar ist. Bei der heutigen Schnellebigkeit und Konkurenz können die Firmen nicht auf ihre Datenbankhersteller warten. Sie benötigen Datenbankserver, die sie jederzeit an ihre Bedürfnisse und Anforderungen anpassen können.

Die meisten heute vorherrschenden Datenbanksysteme haben dieser Anforderung Rechnung getragen und bieten unterschiedliche Möglichkeiten zur Erweiterung ihrer Fähigkeiten. Zwei Systeme werden im folgenden näher betrachtet.

## 3.2 IBM DB2 Universal Database

DB2 Universal Database (UDB) ist ein objektrelationales Datenbankmanagementsystem der Firma IBM. Es ist auf einer Vielzahl von Hardwareplattformen, vom Hostrechner bis hin zur Workstation, und einer ebenso großen Anzahl an verschiedenen Betriebsystemen verfügbar. Ursprünglich als relationales Datenbanksystem entwickelt, wurden seit der ersten UDB Version ständig neue objektrelationale Features in DB2 verwirklicht. Das System wird in [Cha98] im Detail vorgestellt.

Zur Unterstützung von multimedialen Daten gehören zum Umfang des DB2 UDB Paketes die sogenannten DB2 Extender. Für die verschiedenen Datentypen existieren jeweils eigenständige Extender. Die DB2 Extender-Familie umfaßt mittlerweile:

- Text-Erweiterungen
- Geospatial-Erweiterungen
- Multimedia-Erweiterungen für Audio, Video und Bilder
- Verschlüsselungs- und Sicherheits-Erweiterungen
- Business- und Finanz-Erweiterungen
- Netzwerk-Erweiterungen

Die Extender werden entweder von IBM selbst oder von Fremdfirmen, die am sogenannten DB2 Extender Partners Programm teilnehmen, angeboten. Mit Hilfe dieser Extender können sogar kombinierte Suchanfragen über die verschiedenen Datentypen in einer SQL-Anfrage realisiert werden. Auch wenn sich die untersuchten Ansätze und Methoden prinzipiell auf alle diese Extender anwenden lassen, wird hier gemäß dem Thema der Arbeit nur auf den Text-Extender eingegangen.

### 3.2.1 Erweiterungsmöglichkeiten von DB2 UDB

Wie in Abschnitt 3.1 schon erläutert, bietet UDB grundsätzlich die Möglichkeit, große multimediale Daten in BLOBs zu speichern. Neben den *Binary* Large Objects (BLOB) bietet UDB aber auch noch sogenannte *Character* Large Objects (CLOB). Wie der Name schon andeutet, dient dieser Typ vornehmlich der Speicherung von Textdaten und ist damit für die hier untersuchten Texterweiterungen interessant. CLOBs sind also eine Art Spezialisierung der allgemeinen BLOBs. Das Datenbanksystem hat damit schon nähere Informationen zu der Art der gespeicherten Daten. Jedem CLOB ist z.B. eine Codepage zugeordnet, die den verwendeten Zeichensatz beschreibt. Eine eingehendere Beschreibung dieser Datentypen findet man in [Cha98].

### Datentyperweiterung

Um von einer Erweiterung des Datenbanksystems um neue Datentypen sprechen zu können, reicht dies aber noch nicht aus. Deshalb bietet UDB seit Version 5.0 die Möglichkeit, *nutzerdefinierte* Datentypen (***U**ser **D**efined **T**ype* – UDT) anzulegen. Bei den älteren Versionen gab es noch die Einschränkung, daß jeder UDT von einem vorhandenen Basistyp abgeleitet sein mußte und nicht komplex strukturiert sein durfte. Seit der Version 6.1 ist dieses Manko beseitigt und praktisch beliebige neue Datentypen definierbar. Damit wird es dem Anspruch, ein objektrelationales DBMS zu sein, besser gerecht.

Durch die UDTs ist der Anwender in der Lage, eigene Datentypen für Text, Bilder, Audio, Video oder andere Daten zu kreieren. Der neu erzeugte Datentyp wird vollständig in das Typsystem aufgenommen, wodurch unter anderem eine Typüberprüfung automatisch vom Datenbanksystem vorgenommen werden kann. So wird sichergestellt, daß in einer Tabelle mit Bildern auch nur Bilder enthalten sind und nicht zufällig ein Video auftaucht, nur weil es auch als BLOB gespeichert wird.

Wie von einem ORDBMS erwartet, können mit Hilfe der UDTs auch Typhierarchien erzeugt werden. Ein Obertyp `TEXT` kann z.B. weiter untergliedert werden in `Bücher`, `Artikel` und `Gedichte`. `Bücher` könnten dann weiter untergliedert werden in `wissenschaftliche_Bücher` und `literarische_Bücher`, usw. Das integrierte Typsystem achtet auch hierbei wieder auf korrekte Zuordnung der Daten zu ihren Typen. Außerdem werden auch objektorientierte Aspekte, wie z.B. die *Substitution* (ein Obertyp kann durch einen spezielleren repräsentiert werden), unterstützt. So können in einer Spalte vom Typ Bücher sowohl wissenschaftliche als auch literarische Bücher abgespeichert werden, aber keine Gedichte (es sei denn, es handelt sich um ein Buch, das Gedichte enthält).

Die verwalteten Daten werden durch UDTs klassifiziert und erhalten eine Bedeutung. Trotzdem weiß das Datenbanksystem noch nicht, wie es auf den eigentlichen Inhalt zugreifen kann, wodurch inhaltsbasierte Anfragen an diese Typen weiterhin nicht möglich sind. Sie müßten noch immer komplett an die Anwendung geschickt werden, wo das benötigte Dokument herausgefiltert wird. Um dieses Manko zu beseitigen, bedarf es zusätzlicher Funktionalität im Datenbanksystem.

### Funktionserweiterung

Da der Hersteller einer Datenbank nicht wissen kann, welche Daten seine Kunden mit dem System verwalten werden, ist es nicht möglich, die Funktionalität für die zukünftigen Datentypen im voraus zu integrieren — dies wäre wieder nur für eine gewisse Auswahl an Daten realisierbar.

Aus diesem Grund bietet UDB folgerichtig auch sogenannte *nutzerdefinierte* Funktionen (***U**ser **D**efined **F**unctions* – UDF) an. Die eingebauten Funktionen, wie Vergleichs-, Aggregat- und andere mathematische Funktionen, unterstützen nur die Grunddatentypen. Funktionen für die neuen Datentypen können vom Benutzer selbst definiert und implementiert werden. So kann man sich eine Funktion vorstellen, die z.B. die Anzahl der Worte in einem Text liefert, oder die Sprache, in der der Text verfaßt wurde. Die nutzerdefinierten Funktionen können genauso verwendet werden wie die vordefinierten auch.

Die Routinen selbst können fast in jeder beliebigen Programmiersprache geschrieben werden. UDB unterstützt z.B. C/C++, JAVA, COBOL, FORTRAN und REXX. Da der Nutzer die Daten kennt, die in dem vom ihm definierten Datentyp gespeichert werden, kann er auch entsprechende Funktionen schreiben, die den Inhalt repräsentieren.

Auf diese Art und Weise erfahren die neuen Daten keine Behandlung "zweiter Klasse" mehr, denn jetzt kann auf sie genauso zugegriffen werden wie auf die

Standarddatentypen — Objekte können verglichen und analysiert werden. Sogar inhaltsbasierte Anfragen sind nun möglich. Speichert man z.B. geographische Daten in einer Datenbank, so sind Funktionen zum Größenvergleich sinnvoll, und Funktionen, die die Überlappung von Objekten erkennen oder den Abstand von Objekten berechnen. Im Bereich der Textverwaltung sind natürlich Inhaltsanfragen besonders wichtig, d.h., welcher Text enthält Informationen zu einem gewissen Thema.

Diese Architektur bietet somit eine solide Grundlage zur Erweiterung von UDB und der Anpassung an immer neue Anforderungen an die Datenbank. Allerdings ist damit auch die Erweiterungsmöglichkeit von UDB erschöpft. Das bedeutet, Anfrage- und Speicherstrukturoptimierung werden nicht berücksichtigt — Punkte, die eigentlich Stärken eines ORDBMS ausmachen. Aber zumindest die Nutzung datenbezogener Indexmethoden ist möglich, ohne welche ein effizienter Zugriff auf die Daten kaum vorstellbar wäre. Wie dies geschieht, wird im nächsten Abschnitt erläutert.

### 3.2.2 Der Text-Extender

Der DB2 Text-Extender ist ein Mitglied der DB2 Extender-Familie. Er ist, wie es der Name schon sagt, für die Unterstützung von Textdaten in UDB gedacht. Im Prinzip besteht ein Extender aus einer Sammlung von nutzerdefinierten Datentypen und den dazugehörigen Funktionen für eine spezielle Art von Daten. Dadurch muß nicht jeder Anwender die gleiche Arbeit in die Unterstützung der Standardmultimediatypen investieren.

Desweiteren beinhalten die Extender auch die im vorigen Abschnitt angesprochenen Indexmechanismen für ihren jeweiligen Datentyp. Ein Zugriff auf Multimediadaten ohne eine Indexunterstützung ist in der Praxis viel zu zeitaufwendig. Man stelle sich die Beispiel-Textsammlung *TREC* vor, welche mehr als 700.000 Dokumente enthält und über 2 Gigabyte groß ist. Die Suche nach Dokumenten, die eine gewisse Phrase beinhalten, würde auch mit den besten Pattern-Matching-Algorithmen auf heutigen Rechnern noch mehr als eine Stunde dauern. Wie man im Abschnitt 2 sehen konnte, gibt es im Bereich des Information Retrieval aber ausgereifte Methoden der Indizierung solcher Dokumentenmassen. Mit Hilfe dieser Methoden kann die gleiche Anfrage im Sekundenbereich beantwortet werden.

Der Text-Extender besteht aus drei Hauptbestandteilen, der Suchmaschine, den Verwaltungsroutinen und den Anfrageroutinen.

#### Die Indexunterstützung

Eine der Stärken eines ORDBMS ist die Unterstützung von speziellen Zugriffspfaden für die Daten, die es verwaltet. Unterschiedliche Daten verlangen aber auch unterschiedliche Indexstrukturen und ein DBMS kann nicht jeden denkbaren Index unterstützen. Vor allem für die nutzerdefinierten Daten wird es schwer, passende Methoden im Vorfeld zu integrieren. Da UDB keine Möglichkeit bietet, *nutzerdefinierte* Indexe zu definieren, mußte eine andere Möglichkeit gefunden werden, um den Zugriff auf die Nutzerdaten zu verbessern.

Die naheliegendste Lösung ist, einen Index außerhalb des Datenbanksystems zu halten und über entsprechende Kommunikation mit der Datenbank, die Selektion der Dokumente zu beschleunigen. Genau dieser Lösung bedienen sich auch die Extender von UDB.

Für die Textindizierung und die Bearbeitung von Suchanfragen nutzt der Text-Extender die *Textsuchmaschine*, ebenfalls ein Produkt aus dem Hause IBM. Diese Suchmaschine wird auch noch in weiteren IBM Anwendungen, z.B. *Intelligent Miner for Text*, verwendet. Die Suchmaschine wird als Extraprogramm gestartet und über gewisse Interprozeß-Kommunikationsmechanismen angesprochen.

### Die Verwaltungs- und Überwachungsfunktionen

Die Verwaltung der eigentlichen Textdaten und der dazugehörigen Indexdaten bringt aber auch eine Reihe von Problemen mit sich, die es zu lösen gilt.

Da der externe Index nicht als solcher im Datenbanksystem registriert ist, muß eine eigene Verwaltung der indizierten Tabellen und Spalten aufgebaut werden. Der Text-Extender erzeugt dazu Systemtabellen, in denen alle wichtigen Daten abgespeichert werden, z.B. welcher Index für eine gewisse Spalte einer Tabelle zuständig ist. Diese Daten werden beim Anlegen und Löschen der Indexe aktualisiert.

Das Anlegen und Löschen von Indexen sowie das Einrichten der Textunterstützung wird übrigens von einem externen Frontend durchgeführt, da es sich um eine Anweisung an die Suchmaschine handelt. Auf eine Erweiterung der SQL-Syntax zur Unterstützung solcher Aufgaben wurde verzichtet.

Als weiteres muß die Integrität und Aktualität der Daten in beiden Systemen gewährleistet werden. Die externe Suchmaschine muß informiert werden, welche Dokumente indiziert werden sollen, ob neue Dokumente hinzugekommen sind oder gar bereits indizierte Dokumente gelöscht wurden. Für diese Aufgaben stellt der Text-Extender spezielle Triggermechanismen und -funktionen bereit. Diese überwachen alle Änderungen auf den vom Extender verwalteten Tabellen und geben entsprechende Informationen an die Suchmaschine weiter.

### Die Anfragefunktionen

Ein weiteres Problem ist die Kommunikation mit der Datenbank, welche die Auswahl der passenden Dokumente beschleunigen soll. Die einzigen Möglichkeiten, von außen mit dem DBMS zu kommunizieren, sind die SQL-Schnittstelle und die UDFs. Die SQL-Schnittstelle scheidet aus, da genau die Anfragen über diese Schnittstelle beschleunigt werden sollen. Was bleibt, sind also die UDFs.

UDFs wurden entwickelt, um den Zugriff auf den Inhalt von großen Datenblöcken zu unterstützen. So bietet der Text-Extender z.B. eine Routine `contains()`, mit deren Hilfe festgestellt werden kann, ob ein Text etwa bestimmte Worte oder eine Phrase beinhaltet. Diese Funktion müßte dazu theoretisch den kompletten Text scannen, um eine Anfrage nach dem Enthaltensein mit JA oder NEIN zu beantworten. Um dies zu beschleunigen, gibt es aber den Index. Folgerichtig muß also nur die `contains()`-UDF mit der Suchmaschine kommunizieren, welche dann eine indizierte und damit schnellere Suche durchführt. Abbildung 3.1 veranschaulicht die Architektur des Text-Extenders und die eben beschriebene Kommunikation zwischen Datenbank und Suchmaschine.

Die UDF liefert am Ende nur einen WAHR oder FALSCH Wert zurück, an dem das DBMS erkennt, ob das entsprechende Dokument selektiert bzw. aus der Datenbank geladen werden muß. Die Abarbeitung dieser UDFs erfolgt noch vor dem eigentlichen Zugriff auf den Text. Auf diese Weise wird eine Art Index simuliert. Ein Index dient unter anderem vor allem dazu, gewisse Zeilen einer Tabelle vor dem eigentlichen Zugriff auf die Datenbank auszuwählen. Dadurch wird der sogenannte komplette *Tablescan* vermieden, der immer sehr aufwendig ist.

Ein Suchanfrage mit Hilfe der `contains()`-Funktion lautet z.B.:

```
SELECT resume
FROM   applicants
WHERE  contains( resume_id, '"db2" AND "optimizer"') = 1
```

Das `'= 1'` ist bei UDB notwendig, da es keinen Boolean-Typ gibt, den die Funktion zurückliefern könnte. So gibt die Funktion 0 oder 1 zurück und der Vergleich erzeugt den entsprechenden Boolean-Wert.

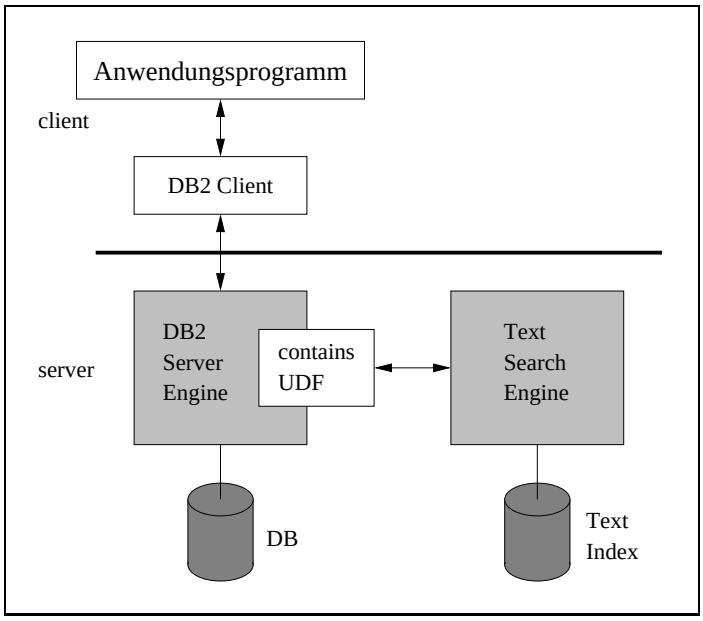


Abbildung 3.1: Architektur des Text-Extenders

Es ist sogar möglich, die Suchanfrage auf verschiedenen Textspalten zu kombinieren. Dies geschieht einfach über einen weiteren Aufruf der `contains()`-Funktion in der `WHERE`-Klausel. Diese Bedingungen können in beliebiger Weise mit den zur Verfügung stehenden Operatoren verknüpft werden:

```
SELECT name
FROM   applicants
WHERE  contains( resume_id, '"db2" AND "optimizer"') = 1
   AND contains( recomd_id, '"workaholic" AND "best"') = 1
```

Die Abarbeitung der Suchanfrage wurde noch dahingehend optimiert, daß nicht bei jedem Aufruf der entsprechenden UDF ein Zugriff auf die Suchmaschine durchgeführt wird. Die Suchmaschine arbeitet eigentlich wie ein richtiger Index und liefert auf eine einmalige Anfrage ein Liste mit Identifikatoren der Dokumente, die sich qualifizieren. Da die UDF für jeden Eintrag in der Tabelle aufgerufen wird, reicht es, wenn die entsprechende Liste nur beim ersten Aufruf von der Suchmaschine angefordert und dann im Speicher gehalten wird. Nachfolgende UDF Aufrufe müssen dann nur noch in der Ergebnisliste nach dem gerade übergebenen Identifikator suchen.

Der Text-Extender bietet noch weitere Funktionen, die im Bereich des Textretrieval eine wichtige Rolle spielen, wie z.B. `rank()` und `no_of_matches()`. Damit können die Dokumente entsprechend ihrer Wichtigkeit bzgl. einer Anfrage sortiert werden. Diese Funktionen greifen auf die gleiche Art und Weise auf die externe Suchmaschine zurück, um die entsprechenden Daten zu ermitteln. Die Nutzung dieser Funktionen ist allerdings etwas gewöhnungsbedürftig:

```
WITH TEMPTABLE( resume, rank)
    AS (SELECT  resume,
                rank( resume_id,'"db2" AND "optimizer"')
        FROM applicants)
    SELECT *
        FROM  TEMPTABLE
        WHERE rank > 0
        ORDER BY rank DESC
```

Für sehr große Tabellen ist das häufige Aufrufen der UDF noch immer ein großer Overhead, denn die Liste liegt ja schon beim ersten Aufruf bereit. Um die Zugriffsgeschwindigkeit weiter zu verbessern, wurde in neueren Versionen des Text-Extenders eine *Tablefunction* (eine UDF, die als Ergebnis eine Tabelle liefert) für das Textretrieval eingeführt. Als Ergebnis liefert diese Funktion eine Tabelle mit den Identifikatoren der Dokumente, die sich bei der Suche qualifiziert haben. Außerdem liefert die Ergebnistabelle auch noch den *Rank-Wert* und die *Number of Matches*, was den Zugriff auf diese Daten vereinfacht und vor allem beschleunigt.

Alles in allem ist es eine gute Lösung für die Unterstützung von Textdaten mit UDB. Durch die eingeschränkten Möglichkeiten bei der Anbindung der externen Suchmaschine geht zwar einiges an Performance verloren, aber wenn man den Zeitgewinn durch die Verwendung eines Indexes berücksichtigt, fällt dies kaum ins Gewicht. Außerdem soll das künftige UDB auch Erweiterungsmöglichkeiten für das Indexsystem bieten, welche die Integration von externen Indexen deutlich verbessern werden.

## 3.3 INFORMIX Universal Server

Das Datenbanksystem der Firma Informix Software Inc. wurde seit der Version 9.1 unter dem Namen Universal Server vertrieben (die neuesten Versionen wurden mittlerweile in Dynamic Server umbenannt). Es war eines der ersten Datenbanksysteme, das objektorientierte Konzepte unterstützte, und ist heute eines der führenden Produkte im Bereich der objektrelationalen DBMS. Im Vergleich mit DB2 UDB unterstützt es schon einige objektorientierte Features mehr, z.B. Typkonstruktoren für Arrays. Der Universal Server ist eine Kombination aus dem früheren relationalen DBMS von Informix und dem DBMS ILLUSTRA, welches von Stonebreaker entwickelt und später von Informix aufgekauft wurde. Die meisten objektrelationalen Features wurden von ILLUSTRA übernommen und dann im Universal Server weiterentwickelt.

Von ILLUSTRA stammt auch das Feature der funktionalen Skalierbarkeit des Systems durch die sogenannte DataBlade-Technologie. Diese Technologie ermöglicht es dem Benutzer, eigene Datentypen und Funktionen zu implementieren. Dadurch werden anwendungsspezifische Systemerweiterungen des Datenbanksystems auf einfache Weise unterstützt. Es gibt inzwischen eine Vielzahl von DataBlades zur Verwaltung der unterschiedlichsten Datentypen, wie z.B.:

- Text Module
- Digital Media Module (Image, Video)
- Geospatial Data Module
- Data Warehouse Module
- Web/E-Com Module
- Financial Module
- Bioinformatics Module

Auch die hier untersuchte Texterweiterung, Excalibur, basiert auf dieser Technologie. Deshalb beschäftigt sich der erste Teil des Kapitels näher mit der Erweiterbarkeit des Informix Universal Servers. Der zweite Teil untersucht dann die Anwendung dieses Feature zur Unterstützung von Textdaten am Beispiel des Excalibur-DataBlades etwas genauer.

### 3.3.1 Die DataBlade-Technologie

Unter dem Begriff der DataBlade-Technologie sind praktisch alle Möglichkeiten der Erweiterung des Universal Servers zusammengefaßt. Im Gegensatz zu DB2 UDB beschränken sich diese aber nicht nur auf nutzerdefinierte Datentypen und Funktionen (die hier Routinen genannt werden, ***U**ser **D**efined **R**outines* – UDR). Vielmehr ist das komplette DBMS an bestimmten Schnittstellen erweiterbar. Dazu gehören z.B. auch der SQL-Parser, der Anfrageoptimierer und der Zugriffspfadmanager. Über die sogenannte *DataBlade-API* können die Module der Anwender auf die entsprechenden Teile des DBMS zugreifen. Abbildung 3.2 zeigt schematisch den Aufbau des Informix DBMS.

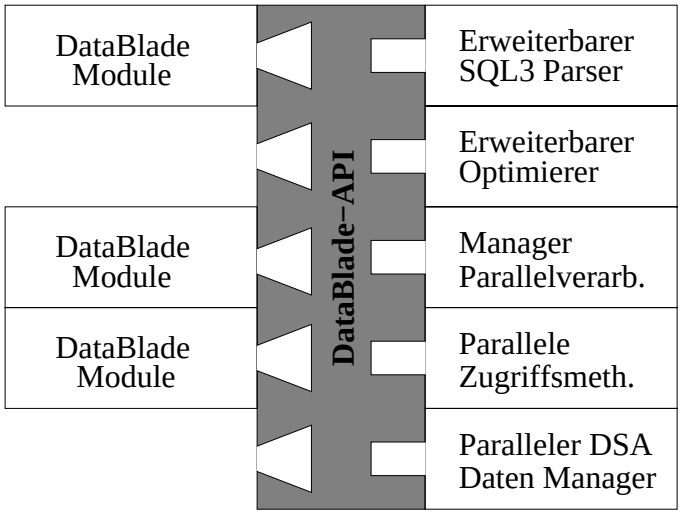


Abbildung 3.2: Schema der DataBlade-Technologie

Ein DataBlade-Modul kann mit einem *Package* in einer objektorientierten Programmiersprache verglichen werden, das einen speziellen Datentyp kapselt. Ein DataBlade besteht aus einem oder mehreren der folgenden Komponenten:

- Datentypen
- Routinen
- Schnittstellen
- Tabellen und Indexen
- Zugriffsmethoden
- Client Code

#### Nutzerdefinierte Datentypen

Alle Datentypen werden gleichermaßen vom System gehandhabt, d.h. es macht für das System keinen Unterschied, ob es sich um einen Standarddatentyp handelt oder einen nutzerdefinierten. Eigentlich sind die Standarddatentypen auch nichts anderes, nur daß der Nutzer in diesem Fall der Datenbankhersteller war. Das System unterstützt verschiedene Arten von nutzerdefinierten Datentypen.

Der *Row Type* faßt mehrere Spalten in einem neuen Typ zusammen. Ein Row Type kann dabei sowohl aus Standard- als auch nutzerdefinierten Datentypen bestehen.

Ein *Distinct Type* ist ein an eine Anwendung angepaßter Standarddatentyp. Zum Beispiel kann ein Datentyp `Schuhgröße` vom Typ `INTEGER` erzeugt werden. Dieser Typ entspricht den Distinct Types der ersten DB2 UDB Versionen — damals die einzigen nutzerdefinierten Typen.

Ein *Opaque Type* wird durch den C, C++ oder JAVA Code definiert, der zum Speichern, Indizieren und Zugreifen dieses Typs geschrieben wird. Dieser Typ bietet die maximale Flexibilität für DataBlade-Programmierer. Meistens wurden die Datentypen schon für andere Anwendungen implementiert und können so einfach in Opaque Types eingebettet werden. Da der Programmierer die genaue Struktur dieser Typen kennt, kann er auch sehr effiziente Routinen für die Handhabung dieser Daten schreiben.

Die nutzerdefinierten Datentypen unterstehen komplett der Kontrolle des DBMS und profitieren dadurch auch von dessen Transaktions- und Recover-Mechanismus, sowie von dessen Integritätssicherung.

### Routinen

Eine weitere Komponente eines DataBlades ist eine Sammlung von Routinen, die auf den Datentypen operieren. Dabei können die Routinen auf alle Datentypen, die dem Server bekannt sind, zugreifen, auch auf solche, die von anderen DataBlade-Modulen definiert wurden. Die Routinen erweitern somit die Verarbeitungs- und Aggregationsfunktionen des DBMS. Bei der Verwendung von Opaque-Datentypen müssen gewisse Routinen bereitgestellt werden, die es dem Server erlauben, mit diesen Typen zu arbeiten. Weitergehende Routinen können optional angeboten werden.

Die Routinen können sowohl in der Informix eigenen Programmiersprache ***S**tored **P**rocedure **L**anguage* (SPL) oder in einer Programmiersprache der 3. Generation wie C, C++ und JAVA geschrieben werden. Routinen, die in einer dieser Sprachen geschrieben sind, werden vorcompiliert und bei Bedarf aus einer Programmbibliothek vom System nachgeladen. Solche Routinen können auf Grund der Mächtigkeit der Programmiersprachen sehr komplexe Aufgaben übernehmen. SPL ist bei weitem nicht so mächtig, aber meistens sind Routinen in SPL einfacher zu schreiben.

### Schnittstellen

Eine Schnittstelle ist eine Sammlung von Routinen, die einem Standard entsprechen und einen definierten Service bereitstellen. Dadurch wird die gemeinsame Benutzung von Diensten, die DataBlade-Module benötigen oder anbieten, möglich.

Ein Bildervergleich-Modul z.B. könnte dem Anwender erlauben, Textbezeichnungen für Bilder anzugeben. Wenn eine Schlüsselwort-Suchschnittstelle im Datenbankserver registriert ist, könnte das Bildvergleich-Modul diese Schnittstelle nutzen, um nach der Bildbezeichnung zu suchen.

Informix standardisiert verschiedene DataBlade-Module, z.B. für den Bild- und Textvergleich, damit diese einheitliche Schnittstellen bieten. Dadurch ist es dem Anwender möglich, sich das für seine Bedürfnissen am besten geeignete Modul auszuwählen.

### Tabellen und Indexe

Ein DataBlade-Modul kann seine eigenen Daten direkt in der Datenbank speichern. Zum Beispiel könnte das Bildmodul alle unterstützten Bildformate und die Konvertierungsprogramme, die sie umwandeln, in einer Tabelle in der Datenbank speichern. Diese Tabellen können genau wie die Nutzertabellen indiziert werden, um den Zugriff auf häufig benutzte Daten zu verbessern.

#### Zugriffsmethoden

Zugriffsmethoden arbeiten auf Tabellen und Indexen, die vom Datenbankserver verwaltet werden. DataBlade-Programmierer können ihre neuen Datentypen mit Hilfe der eingebauten Methoden indizieren oder aber neue implementieren.

Zugriffsmethoden sind durch die Bereitstellung entsprechender Routinen definiert, die der Datenbankserver an den entsprechenden Stellen während der Abarbeitung eines Anfrageplans aufruft. Die Routinen einer neuen Zugriffsmethode führen verschiedene Operation durch, wie z.B. Öffnen eines Indexscans, Anfordern des nächsten Records im Scan, Einfügen eines neuen Records, Löschen eines Records, Ersetzen eines Records und Schließen eines Scans.

Da die neuen komplexen Datentypen nicht mehr ohne weiteres durch Standardmethoden wie den B-Tree unterstützt werden können, verbessert die Möglichkeit, neue Zugriffsmethoden zu definieren, wesentlich die Performance beim Zugriff auf diese Daten. So nutzt beispielsweise das Geodetic-DataBlade von Informix den wesentlich besser geeigneten R-Tree Index für die geographischen Daten.

Diese Fähigkeit ist überaus bedeutend für die Optimierung des Zugriffes auf die neuen Datentypen und stellt daher auch einen Vorteil gegenüber DB2 UDB dar.

#### Client Code

Der Client Code kann in das DataBlade integriert sein oder auch separat geliefert werden. Der Client Code greift auf die Modul-API-Bibliothek zu und stellt dem Benutzer ein entsprechendes Interface zur Verfügung. Ein solches Interface ermöglicht es dem Benutzer, die neuen Datentypen anzufragen, darzustellen und zu manipulieren.

#### DataBlade Developer's Kit

Im Gegensatz zu DB2 UDB bietet Informix für seine Datenbank ein Entwicklungswerkzeug für DataBlades, welches den Programmierer bei der Erstellung eines DataBlades unterstützt. Trotzdem bleibt es sehr aufwendig, ein solches Modul zu implementieren.

Informix bietet auch die Möglichkeit für Drittanbieter, ihre Module zertifizieren zu lassen. Das bedeutet, daß Informix sie auf die Zusammenarbeit mit dem kompletten System und korrekte Umsetzung der Schnittstellen untersucht. Das gibt dem Anwender die Gewißheit, ein entsprechendes DataBlade ohne Gefahr für die Stabilität des bestehenden Systems einsetzen zu können.

### 3.3.2 Das Excalibur Text-DataBlade

Das Text-DataBlade von Excalibur ist ein solches von Informix zertifiziertes Modul, das quasi als "Standard"-Textmodul mit dem Universal Server geliefert wird. Im Prinzip handelt es sich hierbei um eine Programmbibliothek, die alle Routinen für die Textsuche und -verwaltung enthält. Bei einer Textsuchanfrage — `SELECT`-Statement enthält `etx_contains()`-Operator — wird diese Bibliothek vom Datenbankserver nachgeladen und die entsprechenden Teilanfragen an die Excalibur-Suchmaschine weitergeleitet.

Die Suchmaschine liefert daraufhin eine Liste von Zeilen, welche die Bedingungen im `etx_contains()`-Operator erfüllen. Falls weitere Bedingungen in der `WHERE`-Klausel angegeben wurden, werden diese vom Server genutzt, um die Ergebnisliste der Suchmaschine weiter einzuschränken. Am Ende liefert Universal Server eine Liste von Zeilen, die allen Bedingungen in der `WHERE`-Klausel genügen. Eine detaillierte Untersuchung der Retrieval-Eigenschaften und -Qualität findet man in [Por99].

Das Excalibur TextSearch-DataBlade setzt sich aus vier Grundkomponenten zusammen, den ***etx-Zugriffsmethoden***, dem ***etx_contains()-Operator***, den *Filterfunktionen* und den *Systemroutinen*.

### Die etx-Zugriffsmethode

Mit Hilfe des Text-DataBlades ist es möglich, einen spezialisierten *secondary* Index namens **etx** auf Spalten zu definieren, die Texte enthalten. Um von der Zugriffsmethode unterstützt werden zu können, muß die Spalte vom Typ IfxDocDesc, BLOB, CLOB, CHAR, VARCHAR, LVARCHAR sein. Dabei ist der erste Typ speziell für den Einsatz mit den Textzugriffsmethoden ausgelegt.

Für jeden dieser Datentypen stellt das DataBlade eine sogenannte *Operator Class* bereit. Dabei handelt es sich um spezielle Funktionen, die der Server mit einer **etx**-Zugriffsmethode verknüpft und sowohl für die Anfrageoptimierung als auch für die Generierung des Indexes selbst benötigt.

Das Anlegen des Indexes erfolgt einfach über die SQL-Schnittstelle. Durch einen Zusatz zum Standard-SQL (Schlüsselwort `USING`) kann das entsprechende Modul ausgewählt werden, welches für den Index verantwortlich ist.

```
CREATE INDEX resume_idx ON applicants( resume etx_doc_ops)
    USING etx in sbsp1;
```

Der Parameter `etx_doc_ops` bezeichnet hierbei die weiter oben schon erwähnte Operatorklasse. Es gibt noch eine Reihe weiterer Parameter, mit deren Hilfe Einfluß auf den zu erzeugenden Index genommen werden kann, z.B. Unterstützung von *Exact* oder *Pattern* Matching. Nähere Informationen zum Anlegen eines Indexes und zu weiteren Parametern für das Tuning findet man in [INF97].

### Der etx_contains()-Operator

Der `etx_contains()`-Operator dient dazu, eine Suche auf **etx**-Indexen durchzuführen. Dieser Operator entspricht von der Funktion her der `contains()`-UDF des DB2 Text-Extenders, obwohl sich seine Parameter doch etwas unterscheiden. Eine Suchanfrage mit diesem Operator lautet z.B. so:

```
SELECT resume
FROM   applicants
WHERE  etx_contains( resume, 'informix optimizer');
```

Wie hierbei auffällt, unterstützt Informix auch einen Boolean-Datentyp, den die UDR `etx_contains()` als Rückgabetyp benutzt. Außerdem bietet das TextSearch-DataBlade auch keine Funktion `etx_rank()` an, um die Ergebnisse entsprechend zu wichten. Für diesen Zweck gibt es einen weiteren optionalen Parameter in der `etx_contains()`-Routine. Dieser Parameter stellt einen Rückgabewert dar und enthält die entsprechenden Ranking-Informationen. Eine gerankte Anfrage lautet dann so:

```
SELECT rc.score, resume
FROM   applicants
WHERE  etx_contains( resume, 'informix optimizer',
                     rc # etx_Return_Type)
ORDER BY 1;
```

Um die Art der Suche beeinflussen zu können, stellt `etx_contains()` noch weitere Parameter bereit, wie z.B `SEARCH_TYPE` oder `MAX_MATCHES`. Mit dem `SEARCH_TYPE`

wird festgelegt, ob eine boolesche Suche, Proximity-Suche oder gar eine Fuzzy-Suche durchgeführt werden soll. Einzelheiten zu den unterstützten Sucharten und den weiteren Parametern findet man in [INF97]. Eine boolesche Suche würde man etwa so formulieren:

```
SELECT resume
FROM   applicants
WHERE  etx_contains( resume,
                     ROW( 'informix optimizer',
                          'SEARCH_TYPE = BOOLEAN_SEARCH'));
```

Der `ROW`-Operator ist eine Spezialität des Universal Servers und kreiert einen neuen Typ durch das Zusammenfassen von mehreren Attributen. Der zweite Parameter in der `ROW`-Funktion ist der "Tuningparameter". Damit kann der Nutzer die Suchanfrage in gewissen Grenzen steuern, vorausgesetzt es wurde ein entsprechender Index erzeugt.

Eine Besonderheit des `etx_contains()`-Operators ist die, daß er nur ein einziges Mal in einer `WHERE`-Klausel auftauchen darf. Es können zwar weitere Bedingungen angegeben werden, aber keine weiteren Suchanfragen. Dies schränkt die Kombinationsmöglichkeiten etwas ein, da im Gegensatz zu DB2 UDB nicht in verschiedenen Spalten gleichzeitig gesucht werden kann.

### Die Filterfunktionen

Das DataBlade benötigt zum Indizieren die Texte in reinem ASCII-Format, frei von allen Formatinformationen, wie sie z.B. Textverarbeitungen oder PostScript benutzen. Das bedeutet, daß Dokumente, die in BLOB, CLOB, CHAR, VARCHAR oder LVARCHAR gespeichert sind, vor dem Einfügen in die Tabelle vom Benutzer in das ASCII-Format konvertiert werden müssen.

Allerdings gibt es die Möglichkeit, Texte auch in ihrem Originalformat zu speichern. Dazu muß das Dokument in dem IfxDocDesc-Datentyp gespeichert werden. Für diesen Datentyp stellt Excalibur sogenannte Filterroutinen bereit, welche den Text automatisch in reines ASCII konvertieren, wann immer es nötig ist. Das Modul erkennt selbständig eine Reihe verschiedener Textformate, wie HTML, Microsoft Word, WordPerfect und andere. Wenn es ein Textformat nicht unterstützt, gibt das DataBlade einen Fehler zurück.

### Die Systemroutinen

Das TextSearch-Modul bietet noch eine Reihe weiterer nützlicher Funktionen für die Textsuche. Eine genaue Beschreibung der Funktionen gibt es wieder in [INF97].

**etx_CreateStopWlst()** Diese Funktion erlaubt dem Benutzer, eine eigene Stopwortliste zu erzeugen. Wie die meisten Suchmaschinen indiziert Excalibur alle Wörter in einem Text und schließt die Stopworte nur bei der Anfrage aus, um die Suche etwas zu beschleunigen. Zum Löschen der Stopwortliste dient die Funktion `etx_DropStopWlst()`.

**etx_CreateSynWlst()** Mit Hilfe dieser Funktion ermöglicht Excalibur die Nutzung eigener Synonymlisten für die Suchanfragen. Gelöscht werden diese Listen mit der Funktion `etx_DropSynWlst()`.

**etx_GetHilite()** Diese Funktion gibt die genaue Position der von der Suchmaschine gefundenen Worte in dem Text zurück. Damit wird es z.B. möglich, die entsprechenden Wörter in einem Betrachter hervorzuheben, um sie schneller zu erkennen. Diese Funktion kann nur im Zusammenhang mit dem `etx_contains()`-Operator aufgerufen werden, wobei man immer darauf achten muß, daß beide Funktionen mit den gleichen Dokumenten aufgerufen werden.

## 3.4 Zusammenfassung

Obwohl die beiden Lösungen zur Erweiterung von Datenbanken für die Unterstützung von Textdaten auf den ersten Blick sehr unterschiedlich erscheinen, sind sie sich doch ähnlich. Während UDB nur die Erweiterung der verwalteten Datentypen und der zur Verfügung stehenden Funktionen erlaubt, geht die DataBlade-Technologie sehr viel weiter mit den angebotenen Erweiterungsmöglichkeiten.

Im Prinzip stellt das UDB System eine Vorstufe der DataBlades dar, in dem nur ein erster Teil des Datenbanksystems erweiterbar ist. Mit den neueren Versionen von UDB wird auch das Indexsystem erweiterbar sein und später wahrscheinlich auch der Anfrageoptimierer. Die Datenbankhersteller haben erkannt, daß sie alleine nicht mehr schnell genug qualitativ hochwertige Unterstützungen für die ständig neu auftauchenden Daten bereitstellen können. Außerdem macht es keinen Sinn, in einem System alle denkbaren Datentypen zu unterstützen. Das macht ein DBMS unnötig schwerfällig und teuer. Eine Überblick über Erweiterbarkeit der ORDBS gibt die Tabelle B.2.

Die DataBlade-Technologie stellt eine sehr gute Lösung für ein vollkommen erweiterbares Datenbanksystem dar. Vor allem die Erweiterbarkeit der Zugriffsmethoden und des Anfrageoptimierers sind von größter Wichtigkeit für die Unterstützung neuer Datentypen durch eine bestehendes Datenbanksystem. Allerdings erkauft man sich diese Flexibilität mit einem großen Implementierungsaufwand ([Nit99]). Trotzdem gibt es für das Informix System bei weitem die meisten Erweiterungsmodule verglichen mit den anderen führenden Datenbanken, wie ORACLE und DB2 UDB. So gibt es alleine für die Textunterstützung etwa vier Zusatzmodule und für die Unterstützung von Bildern gar neun.

Doch trotz der vielen Texterweiterungen kann der Universal Server bei Evaluierungen zum Textretrieval weder gegen die spezialisierten Systeme ([Rah98]) noch gegen DB2 UDB ([Por99]), das nur eine einzige Erweiterung bietet, bestehen. Die meisten Limitierungen der Erweiterungen ergeben sich direkt aus den Limitierungen der verwendeten Suchmaschine. Die Lösung liegt also offensichtlich nicht in einer weiteren Verbesserung der Datenbankerweiterungsschnittstellen, sondern bei einer Verbesserung der Suchmaschinen. Es gibt aber schon bessere Suchmaschinen, wie aus vielen Evaluierungen hervorgeht. So wird in [Tit99] von der Kombination aus der FULCRUM-Suchmaschine und dem UDB Datenbanksystem gesprochen, die viele Anforderungen besser bewältigen könnte als bestehende Systeme.

Allerdings gibt es auch andere Einschränkungen. So ist es wenig verständlich, warum Suchanfragen mittels Excalibur auf eine Textspalte beschränkt sind. Bei DB2 UDB stören z.B. der fehlende "Tuningparameter" für die Suchanfrage (man muß einen anderen Identifikator übergeben, welcher einem anderen Index zugeordnet ist) und die fehlenden Rückgabewerte bei den Anfragefunktionen. Dadurch sind die Anfragen unnötig kompliziert und für den Nutzer wenig transparent.

Als Fazit bleibt festzustellen, daß beide Datenbanksysteme eine relativ gute Unterstützung für Textdaten bieten. Trotzdem reicht die Qualität des Textretrieval noch nicht aus, um den hohen Anforderungen in heutigen IR-Anwendungen gerecht zu werden.

# Kapitel 4

# Architekturen zur Kopplung von ORDBMS und Textretrieval-Werkzeugen

In den vorangegangenen Kapiteln wurden einige existierende Produkte zum Textretrieval und zur Textverwaltung vorgestellt. In diesem Kapitel werden die verschiedenen Architekturen analysiert, die diesen Produkten zugrunde liegen.

So vielfältig wie die Produkte selbst sind auch die Möglichkeiten der Kopplung von Datenbanken und Information Retrieval-Werkzeugen. Welche Variante die bessere ist, hängt von der Priorisierung einzelner Schwerpunkte ab, wie Performance, Erweiterbarkeit, Austauschbarkeit, usw. Im folgenden werden die Vor- und Nachteile der einzelnen Kopplungsvarianten näher untersucht.

Die verschiedenen Techniken lassen sich in drei großen Gruppen zusammenfassen. Das Unterscheidungsmerkmal dieser Gruppen ist die "Hauptkomponente" des kompletten Systems. Als Hauptkomponente wird dabei der Teil bezeichnet, der für die Steuerung und Kontrolle, sowie für die Kommunikation mit dem Benutzer oder den Anwendungsprogrammen zuständig ist. Der komplette Datenaustausch zwischen Anwendung und dem Textretrieval- und -verwaltungssystem erfolgt über diese Hauptkomponente.

Neben der Hauptkomponente existieren noch weitere "Hilfskomponenten", die für spezielle Teilaufgaben herangezogen werden. Die Unterscheidung erfolgt also danach, welches System zur Unterstützung an ein anderes gekoppelt ist.

Die nachfolgenden Abschnitte widmen sich jeweils einer Architekturvariante. Dabei werden keine genauen Detaillösungen für die einzelnen Kopplungsmechanismen vorgestellt, sondern nur die Technik und ihre Besonderheiten erläutert. Am Ende werden die unterschiedlichen Ansätze noch einmal miteinander verglichen.

## 4.1 Suchmaschine mit Datenbankanbindung

Diese Variante der Kopplung von Suchmaschinen und Datenbanken stammt von den Suchmaschinen selbst. Nachdem die ersten Suchmaschinen zunächst nur Dokumente suchen konnten, die frei in einem Filesystem verfügbar waren, wurde ständig nach Möglichkeiten gesucht, auch andere Datenquellen zu erschließen. Eine der bedeutendsten Datenquellen dabei ist das World Wide Web (WWW).

Aufgrund der extrem großen Menge von Daten werden diese aber häufig in Datenbanken gepeichert. Viele Webseiten beispielsweise werden erst beim Betrachten mit den Inhalten einer Datenbank gefüllt. Aber auch komplette Dokumente

werden in ORDB gespeichert, z.B. der Briefverkehr einer Firma oder alle Dokumente bei einer Versicherung.

Um auch solche Dokumente mit Hilfe von Suchmaschinen durchsuchen zu können, mußten Lösungen gefunden werden, um die Dokumente aus einer Datenbank direkt einer Suchmaschine zugänglich zu machen. Am meisten werden hierfür die erweiterbaren *Text-Reader* genutzt. Diese Technik kommt z.B. bei dem Textretrieval-System Fulcrum zum Einsatz. Mit Hilfe der Text-Reader werden Dokumente eingelesen und in ein dem Suchsystem verständliches Format gewandelt. Abbildung 4.1 zeigt den Aufbau einer Suchmaschine mit verschiedenen Text-Readern.

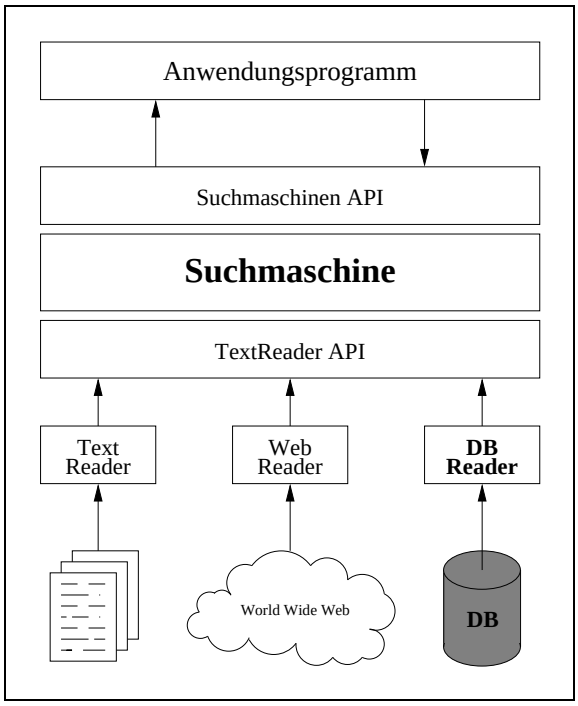


Abbildung 4.1: Textretrieval-System mit Suchmaschine

### Der Datenbank-Reader

Wie in Abbildung 4.1 zu sehen, besteht eine Suchmaschine aus mehreren Modulen: den Text-Readern, dem Indizierer und dem Anfragemodul. Bei den Text-Readern setzt die Erweiterung zur Anbindung an die Datenbanken an.

Wenn die Schnittstelle zwischen Suchmaschine und Text-Reader eindeutig und abstrakt genug ist, z.B. einfacher, unformatierter ASCII-Text als Datenstrom, dann kann der Text-Reader praktisch eine beliebige Anwendung sein und die Dokumente sowohl von einem Filesystem, aus dem World Wide Web und auch aus einer Datenbank extrahieren.

Unter diesen Voraussetzungen kann eine Datenbankanwendung geschrieben werden, die entsprechende Dokumente aus einer Datenbank ausliest und sie als reinen Textstrom ausgibt. Diese Datenbankanwendung kann dann in der Suchmaschine anstelle des Text-Readers eingesetzt werden und würde diese mit den entsprechenden Dokumenten versorgen. In diesem Fall handelt es sich dann um einen *Datenbank-Text-Reader*.

Die Steuerung des Datenbank-Text-Readers erfolgt meistens statisch über Steuerdateien, die vom Administrator angelegt werden müssen. In ihnen wird unter anderem festgelegt, in welchen Tabellen und Attributen die gewünschten Dokumente

zu finden sind. Eine dynamische Steuerung über die Suchmaschine ist zwar denkbar, aber nicht angebracht. Da die Text-Reader auf unterschiedlichste Weise gesteuert werden, ist es schwer, eine gemeinsame Schnittstelle dafür zu finden, welche auch zukünftige Text-Reader unterstützen soll.

Auf der anderen Seite ist ein Index meistens statisch an eine Textsammlung gebunden, weshalb die statische Steuerung der Datenbank-Reader keine wirkliche Einschränkung darstellt.

Durch die zunehmende Standardisierung des Zugriffes auf DBMS mittels ODBC und JDBC ist es sogar möglich, einen "Standard"-Datenbank-Text-Reader zu implementieren. Dieser kann auf alle Datenbanken zugreifen, die einen dieser Standards unterstützen. Dem Volltextsystem Fulcrum liegt z.B. ein Datenbank-Reader für ODBC-Datenbanken bei. Für alle anderen proprietären DBMS muß der Reader entsprechend angepaßt werden.

#### Eigenschaften

Bei dieser Architektur verbleiben die Dokumente an ihren Ursprungsorten. Die Suchmaschine erzeugt einen Textindex über alle Dokumente und speichert nur eine Referenz auf das Dokument. Wird das Dokument bei einer Anfrage als Treffer ermittelt, dann wird es mit Hilfe der Text-Reader wieder eingelesen und an den Benutzer weitergeleitet.

Auf diese Art und Weise wird die Funktionalität der Datenbank für die Speicherung großer Dokumente genutzt und gleichzeitig eine Suche nach relevanten Dokumenten ermöglicht. Die Daten können außerdem dort bleiben, wo sie bereits gespeichert sind und müssen nicht in ein neues System eingespeist werden.

Der große Nachteil besteht hierbei aber in der sehr losen Kopplung der beiden Systeme, wodurch die Integrität der Daten nicht gewährleistet werden kann. Dadurch kommt es bei sehr dynamischen Textsammlungen (z.B. im WWW) häufiger vor, daß ein Dokument, welches als Treffer ermittelt wurde, schon längst nicht mehr an seinem Ursprungsort existiert und damit die Referenz in der Suchmaschine ungültig ist. Diesem Problem kann man zwar mit einem regelmäßigen Update des Indexes begegnen, doch wird man durch den sehr hohen Zeit- und Rechenaufwand immer einen Kompromiß zwischen Aktualität und Aufwand eingehen müssen.

Die meisten Produkte, die auf diese Lösung aufbauen, haben noch einen weiteren Nachteil. Da es prinzipiell noch Suchmaschinen sind, steht dem Anwender auch nur die Schnittstelle einer Suchmaschine zur Verfügung. Durch die einfache Art der Datenbank-Reader-Anbindung geht dem Benutzer der mächtige Anfragemechanismus der Datenbanken verloren, wie z.B. Anfragen auf mehreren Attributen oder JOINs mehrerer Tabellen. Es handelt sich hier also lediglich um eine Ressourcenkopplung und nicht um eine Funktionskopplung.

## 4.2 Datenbanken mit Suchmaschinenunterstützung

Bei dieser Lösung handelt es sich um das genaue Gegenteil zu der des letzten Abschnittes. Wie zu erwarten, stammt die Architektur von den Datenbanksystemen ab, genauer von den objektrelationalen Datenbanksystemen (ORDBS).

Durch die fortschreitende Entwicklung der multimedialen Datentypen waren die Datenbankhersteller gezwungen, ihre Systeme für die Unterstützung solcher Daten zu erweitern. Mit der Einführung der ***L**arge **O**bject* Datentypen (LOB) waren die ORDBS in der Lage, sehr große Objekte, wie Bilder, Videos oder Audiodaten zu speichern. Allerdings fehlte die Unterstützung dieser Daten durch entsprechende Zugriffspfade und spezielle Zugriffsoperationen. Dadurch war der Umgang mit diesen Daten sehr umständlich und beschränkte sich im wesentlichen auf das Ein- und

Auslesen der kompletten Objekte.

Für eine weitere Unterstützung der LOBs sind tiefe Eingriffe in die Indexverwaltung, den Optimierer und den Anfrageparser notwendig. Da man nicht für alle Datentypen (Text, Video, Geospatial-Daten) einen neuen Zugriffspfad in das System implementieren kann, müßte man sich auf einen Kompromiß einigen.

Aufgrund dieser Umstände griffen die Datenbankhersteller zu einem Trick. Die Aufgabe der Indizierung wird von speziellen Retrieval-Maschinen vorgenommen und über spezielle nutzerdefinierte Funktionserweiterungen erfolgt die Kommunikation zwischen den beiden Systemen. Diese Technik findet z.B. beim Text-Extender von DB2 UDB Anwendung. In Abbildung 4.2 ist der Aufbau eines solchen Retrieval-Systems dargestellt.

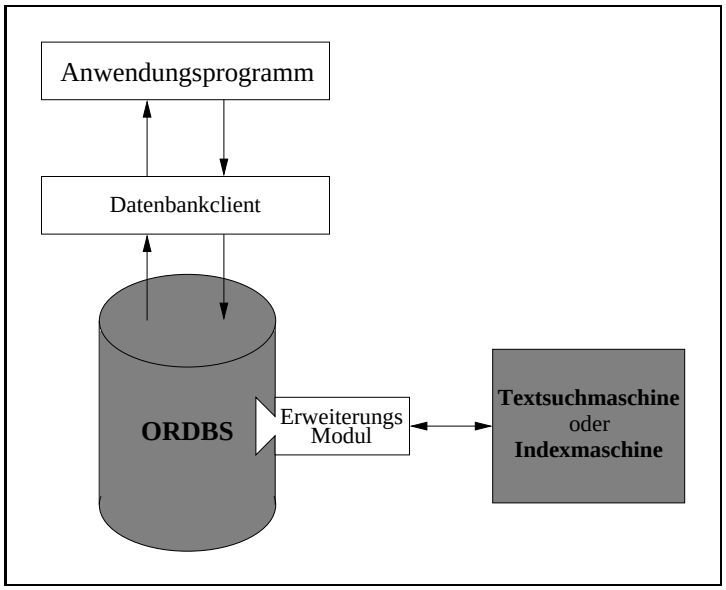


Abbildung 4.2: Textretrieval-System mit ORDBMS

Ein Problem bei dieser Architektur stellt die Verbindung von DBMS und Suchmaschine dar. Wie im Abschnitt 3.2.2 am Beispiel des DB2 UDB Text-Extenders gesehen, kann die Einbindung der externen Suchen kompliziert und unhandlich werden, wenn das ORDBMS nicht schon entsprechende Erweiterungsmöglichkeiten bietet. Vor allem die Performance ist unbefriedigend, wenn die Ergebnisse der Suchmaschine nicht frühzeitig im Anfrageplan berücksichtigt werden.

Genau in diesen Ankopplungsmechanismen unterscheiden sich die Textsucherweiterungen für die verschiedenen ORDBMS. Ebenso unterscheidet sich der Ort, wo die Suchmaschine ihre eigenen Daten speichert. Der DB2 UDB Text-Extender z.B. speichert alle zur Verwaltung nötigen Daten in der Datenbank und die erzeugten Indexe in einem speziellen Ordner im Filesystem.

Je enger die Einbindung der Suchmaschine in das ORDBMS erfolgt, um so komfortabler wird die Handhabung der Suche und um so schneller können die Anfragen vom System beantwortet werden. Auf der anderen Seite bedeutet eine engere Ankopplung aber auch einen erhöhten Programmieraufwand und eine Verminderung der Austauschbarkeit der Suchmaschine oder Datenbank.

### Komponenten-Datenbanken

Die engste Art der Ankopplung findet man in den sogenannten *Komponenten-Datenbanken*. Einer der bekanntesten Vertreter dieser Gattung ist das Informix

ORDBMS. Durch seine DataBlades läßt sich das System für die verschiedensten Datentypen anpassen.

Die Idee dieser Systeme ist die totale Erweiterbarkeit der einzelnen Bestandteile des DBMS. Es existiert nur ein Kernsystem, welches entsprechende Schnittstellen zu anderen Modulen bereitstellt. Diese Module übernehmen dann die Arbeit für den jeweiligen Datentyp. Auf diese Weise ist es möglich, praktisch jeden beliebigen Datentyp zu unterstützen.

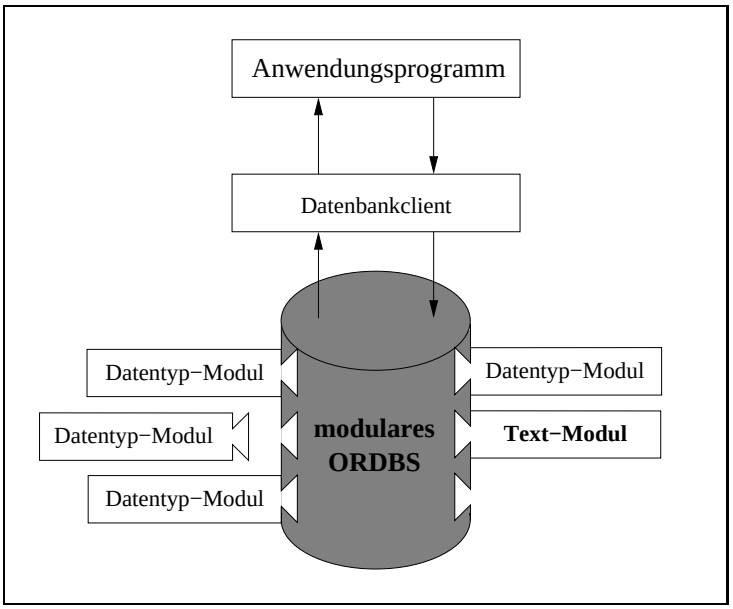


Abbildung 4.3: Textretrieval-System mit Komponenten-DBMS

Die Suchmaschine ist in diesem Fall nicht mehr ein eigenständiges Programm, sondern dessen Funktionen sind in dem Erweiterungsmodul integriert. Dieses Modul wird vom Kernsystem verwaltet und die entsprechenden Funktionen zum gegebenen Zeitpunkt aufgerufen.

Der größte Vorteil im Vergleich mit "normalen" ORDBMS ist die Erweiterbarkeit der Speicherstrukturen, der Zugriffspfade und des Optimierers. Dadurch können die neuen Datentypen genauso effizient unterstützt werden wie die Standarddatentypen. Allerdings erfordert das von den Programmierern einer solchen Erweiterung genaue Kenntnisse über das Datenbanksystem und über die Programmierung von Speicherstrukturen und Zugriffspfaden.

### Eigenschaften

Bei diesem Ansatz ist die Hauptkomponente weiterhin die Datenbank, auf die mittels einer Datenbankanwendung zugegriffen werden kann. Die Suchmaschine ist nur indirekt über entsprechende Erweiterungen der Datenbankanfragesprache zu erreichen.

Die Dokumente werden weiterhin komplett von der Datenbank verwaltet und der Suchmaschine nur zum Zwecke der Indizierung verfügbar gemacht. Die Suchmaschine speichert ihrerseits nur Identifikatoren für die Dokumente in ihrem Index, damit es nicht zu einer unnötigen Doppelspeicherung kommt und dieErgebnisse der Suche einfach zur Selektion der entsprechenden Zeilen in der Datenbank genutzt werden können.

Der Vorteil dieser Lösung ist die mehr oder minder transparente Einbindung der Suchfunktionalität in herkömmliche ORDBMS, je nach verwendeter Erweite-

rung und Datenbank. Alle bestehenden Datenbankanwendungen können weiterhin mit dem ORDBMS genutzt werden. Außerdem bleibt die komplette DBMS-Funktionalität erhalten und nutzbar.

Auf der anderen Seite wird aber nicht unbedingt die komplette Funktionalität einer Suchmaschine verfügbar. Es handelt sich hier also um eine eingeschränkte Funktionskopplung. Von den Suchmaschinen wird nur der reine Textindex als Ersatz für die fehlenden Zugriffspfade genutzt. Daher reichen reine Indexmaschinen für diese Kopplungsvariante aus, was den Ressourcenbedarf des Gesamtsystems verbessert.

## 4.3 Middleware-Ansatz

Der Middleware-Ansatz unterscheidet sich von den vorhergehenden Ansätze insofern, daß hier weder die Suchmaschine noch die Datenbank die Hauptkomponente darstellt. Es wird also nicht eines der beider Systeme um Fähigkeiten des anderen erweitert. Dieser Ansatz stellt ein neues System dar, bei dem sowohl die Datenbank als auch die Suchmaschine nur Teilkomponenten für spezielle Aufgaben sind.

Die Hauptkomponente bei dieser Lösung ist eine sogenannte Middleware-Anwendung. Middleware deshalb, weil es sich nicht um ein Endanwendungsprogramm handelt, sondern um eine Software, die gewisse Funktionen verschiedener anderer Programme zusammenfaßt und dem eigentlichen Endanwendungsprogramm unter einer einheitlichen Schnittstelle zur Verfügung stellt. In Abbildung 4.4 ist der Aufbau einer solchen Middleware-Architektur dargestellt.

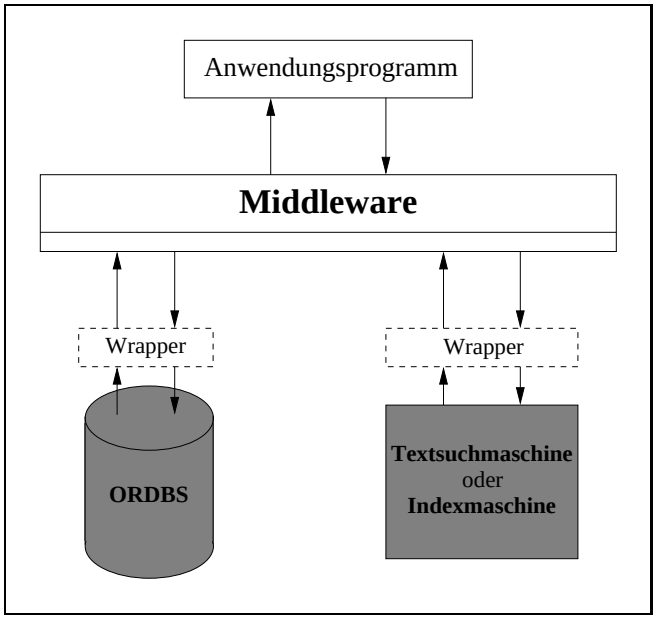


Abbildung 4.4: Textretrieval-System mit Middleware

### Funktionsweise

Die Middleware kapselt im wesentlichen die Funktionen von Suchmaschine und Datenbank und bietet dem Anwender eine eigene problemspezifische Schnittstelle. Weiterhin übernimmt die Software Kontroll- und Steueraufgaben für ihre Hilfskomponenten. Sowohl Suchmaschine als auch Datenbank kommunizieren über ihre Standardschnittstellen ausschließlich mit der Middleware.

Um nicht auf eine spezielle Datenbank und Suchmaschine festgelegt zu sein, implementieren die Middleware-Systeme feste Schnittstellen für den Zugriff. Für die konkreten Systeme müssen dann unter Umständen entsprechende Wrapper zur Verfügung gestellt werden, um die systemeigene Schnittstelle an die der Middleware anzupassen.

Bei den Datenbanken gibt es durch die Standards ODBC und JDBC eine einfache Möglichkeit, sehr viele Produkte zu unterstützen. Datenbanken ohne eine solche Unterstützung sind weiterhin auf einen Wrapper angewiesen. Bei den Suchmaschinen gibt es bisher keinen vergleichbaren Anfragestandard. Deshalb wird hier immer ein Wrapper benötigt.

Im Prinzip besteht die Middleware aus zwei Teilen. Der eine ist eine Art Broker, der den Zugriff auf die entsprechenden Ressourcen bereitstellt. Der andere beinhaltet die komplette Logik für die Anfragebearbeitung.

Bei der Indizierung übernimmt die Middleware die Aufgabe eines Datenbank-Text-Readers für die Suchmaschine. Sie extrahiert die Dokumente aus der Datenbank und übergibt sie der Suchmaschine zur Indizierung. Da die Systeme unabhängig voneinander sind, muß die Zuordnung einer Textspalte in der Datenbank zu einem Index in der Suchmaschine von der Middleware verwaltet werden.

Bei einer Suchanfrage wird die Middleware diese zunächst an die Suchmaschine weiterleiten. Für den Fall, daß kein Ergebnis gefunden wurde, ist die Anfrage beendet. Im anderen Fall muß festgestellt werden, welcher Index ein Ergebnis geliefert hat und eine Anfrage an die korrespondierende Textspalte der Datenbank gestellt werden. Mit Hilfe der Dokument-IDs werden die entsprechenden Texte selektiert und ausgelesen. Dann können sie dem Benutzer zur Verfügung gestellt werden. Sollten mehrere Indizes ein Ergebnis geliefert haben, dann werden alle Texte nacheinander aus den entsprechenden Tabellen extrahiert und als gemeinsames Ergebnis dem Benutzer präsentiert.

### Eigenschaften

Die Middleware-Architektur hat einige Vorteile. So können alle denkbaren Datenbanken und Suchmaschinen eingesetzt werden, ohne daß diese speziell erweitert oder angepaßt werden müssen (lediglich ein Wrapper wird benötigt). Das macht diese Lösung sehr flexibel in Bezug auf die nutzbaren Produkte. Will man beispielsweise eine bessere Suchmaschine nutzen, so kann die alte einfach gegen die neue ausgetauscht werden, vorausgesetzt das Middleware-System ist entsprechend modular aufgebaut.

Ein wichtiger Vorteil des Systems ist die Möglichkeit, eine eigene Anfrageschnittstelle bereitstellen zu können. Dadurch können problemspezifische Systeme entwickelt werden, die spezielle Aufgabenbereiche sehr gut unterstützen. Die teilweise umständliche Umsetzung auf Datenbank- oder Suchanfragen wird automatisch von der Middleware übernommen.

Zusätzlich kann die komplette Datenbankfunktionalität an den Nutzer weitergereicht werden. Dadurch würde sich das System nach außen wie ein Datenbanksystem mit zusätzlicher Suchfunktionalität verhalten. Hierbei handelt es sich also um volle Funktionskopplung.

Auch die Erweiterungsfähigkeiten sind bei dieser Architektur sehr gut. Wenn der Broker weiter ausgebaut wird, ist es möglich, mehrere verschiedene Datenbanken und Suchmaschinen (z.B. für Bilder und Videos) in einem System zu verwenden.

Als nachteilig kann der hohe Entwicklungs- und Implementierungsaufwand angesehen werden, da die meisten komplexen Funktionen in der Middleware integriert sind. Weiterhin müssen Wrapper für die verwendeten Komponenten geschrieben werden, die je nach Funktionalität der Komponenten mehr oder weniger komplex sind.

Die extreme Kapselung führt außerdem zu einem erhöhten Kommunikations- und Verwaltungsaufwand. Das wirkt sich negativ auf die Performance des gesamten Systems aus. Bei einfachen statischen Textsammlungen (z.B. auf einer CD-ROM) würde sich also der Aufwand für ein solches System kaum lohnen.

## 4.4 Zusammenfassung

Die beiden ersten Ansätze sind jeweils Versuche, das eine System um Funktionen des anderen zu erweitern. Aufgrund der großen Unterschiede von Textretrieval-Systemen und ORDBMS können dabei nicht alle Funktionen in das andere System integriert werden.

Aus Datenbanksicht sind an den Suchmaschinen die fehlenden Datenbankeigenschaften wie Datenintegrität, Transaktionen und Nutzerverwaltung zu kritisieren. Auf der anderen Seite bieten sie auf dem Gebiet des reinen Textretrievals und der Nutzerinteraktion wesentliche Vorteile gegenüber den ORDBMS mit Textretrieval-Unterstützung.

Das Problem sind die zugrunde liegenden Daten und Anfragen. Während Datenbanken stark strukturierte Daten und exakte Anfragen unterstützen, sind das bei den Textretrieval-Systemen große un- oder semistrukturierte Daten und vage Anfragen ([Fuh92]). Um dieses Mißverhältnis zu überwinden bedarf es neuer Konzepte und Modelle für Datenbanken und Information Retrieval. Tiefergehende Untersuchungen zu den Problemen der Integration und mögliche Lösungen sind in [Fuh93, Fuh94, Fuh96] zu finden.

Der Middleware-Ansatz bietet die beste Möglichkeit, beiden Ansprüchen gerecht zu werden. Allerdings kann es sich hier nur um eine Übergangslösung handeln, um vorhandene Systeme weiterhin nutzen zu können. Die Software muß die Unterschiede der beiden Systeme auf einer höheren Ebene ausgleichen. Dadurch wird sie sehr komplex und das Gesamtsystem kann keine optimale Performance bieten.

Im Anhang B befindet sich eine Gegenüberstellung der hier vorgestellten Architekturen (Tabelle B.3).

# Kapitel 5

# Architekturvorschlag

Trotz der verschieden Möglichkeiten zur Kopplung von IR-Systemen und ORDBMS, die im letzten Kapitel aufgezeigt wurden, gibt es noch keine Produkte, die den Anforderungen aus beiden Anwendungsgebieten in ausreichendem Maße gerecht werden. Andererseits gibt es auf beiden Seiten (IR und DB) ausgereifte Systeme, die in ihrer Klasse sehr gute Ergebnisse erzielen.

In diesem Kapitel soll eine Möglichkeit aufgezeigt werden, die verschiedenen Systeme so miteinander zu verbinden, daß sie sich gegenseitig mit ihren jeweiligen Funktionen unterstützen. Die Lösung soll vom Anwender selbst eingerichtet werden können, ohne daß direktes Wissen über den Aufbau oder gar der Zugriff auf die Interna der Datenbank- und Retrieval-Systeme benötigt wird.

Wenn man sich die in Kapitel 4 vorgestellten Architekturen anschaut, muß man feststellen, daß es eigentlich keine weiteren Varianten der Kopplung der beiden Systemen gibt. Die einzige noch verbleibende Möglichkeit ist die Integration aller Funktionen in einem neuen System. Allerdings kann man hierbei kaum noch von der Kopplung zweier Systeme sprechen. Doch wenn es all diese Techniken schon gibt, es aber trotzdem kein System gibt, das allen Anforderungen gerecht wird, müßte man davon ausgehen, daß es prinzipiell nicht möglich ist, IR- und DB-Systeme zufriedenstellend mit einander zu verbinden. Daß dem nicht so ist, soll hier gezeigt werden.

Dazu werden zuerst die Anforderungen an die Kopplungsarchitektur festgelegt und die im letzten Kapitel vorgestellten Methoden danach bewertet. Aus einer dieser Methoden wird dann eine erweiterte Architektur entwickelt. Die Details dieser Architektur werden näher erläutert und die genauen Schnittstellen zwischen den Komponenten definiert. Danach werden die Eigenschaften des Ansatzes bezüglich der festgelegten Kriterien und in Bezug auf die Erweiterungs- und Anpassungsmöglichkeiten untersucht. Am Ende erfolgt eine Zusammenfassung des Kapitels.

## 5.1 Anforderungen an die Architektur

Bei der Suche nach einer geeigneten Lösung für die Kopplung von IR-Werkzeugen und objektrelationalen Datenbanken sollen verschiedene Gesichtspunkte berücksichtigt werden, um eine einfache Handhabung und gleichzeitig volle Funktionskopplung zu ermöglichen.

Im einzelnen bedeutet das, daß vorhandene ORDBMS und IR-Werkzeuge genutzt werden können, ohne daß sie vorher vom Nutzer oder Hersteller speziell dafür angepaßt werden müssen. Allerdings soll die Architektur dadurch nicht auf eine minimale, allen Systemen gemeinsame Funktionalität beschränkt werden. Vielmehr soll die Kopplung so generisch erfolgen, daß jede gebotene Funktionalität auch ge-

nutzt werden kann. Außerdem soll es den Herstellern einfach möglich sein, ihre Produkte zu erweitern, um dem Gesamtsystem mehr Funktionalität oder Performance zu bieten.

Um einen geringen Implementierungsaufwand zu gewährleisten, ist eine Arbeitsteilung angebracht. Durch genau definierte Schnittstellen können die Hersteller zu ihren Systemen entsprechende Module zur Unterstützung dieser Schnittstellen anbieten — ähnlich wie ODBC- oder JDBC-Treiber. Gleichzeitig wird dadurch auch die Austauschbarkeit der benutzten Produkte verbessert.

Die Frage der Performance ist bei beiden Anwendungen ein Hauptkriterium. Eine einheitliche Schnittstelle wird immer mit einem Performanceverlust bezahlt werden, da gewisse Umsetzungen auf diese Schnittstelle nötig sind. Deshalb soll die Schnittstelle einfach und offen sein, um den Verlust möglichst gering zu halten. Weiterhin sollen die beteiligten Systeme über ihre Standardanfrageschnittstellen miteinander gekoppelt werden, was sich sowohl positiv auf die Breite der unterstützten Produkte als auch auf den notwendigen Implementierungsaufwand auswirkt.

Die eben beschriebenen Anforderungen sollen den Maßstab für die hier zu entwickelnde Architektur bilden. Die Punkte können wie folgt zusammengefaßt werden:

- Unterstützung von ORDBMS und IR-Werkzeugen über deren API
- geringer Implementierungsaufwand für die Anbindung der Systeme
- einfache Austausch- und Erweiterbarkeit der beteiligten Komponenten
- Vereinigung von DB- und IR-Funktionalität
- gute Performance des Gesamtsystems

## 5.2 Bewertung der Architekturen

In diesem Abschnitt werden die einzelnen Kopplungsvarianten nochmal in Bezug auf die oben aufgeführten Punkte untersucht und entsprechend bewertet.

### Suchmaschine mit Datenbankanbindung

Diese Methode koppelt ORDBMS über deren reguläre Anfragesprache (ODBC, JDBC oder eigene) an eine bestehende Suchmaschine. Auf Suchmaschinenseite muß ein entsprechender Datenbank-Reader zur Verfügung stehen. Die Schnittstellen der Textreader zu den Suchmaschinen sind sehr unterschiedlich und in vielen Fällen gar nicht verfügbar. Deshalb muß die Implementierung entsprechender Treiber von den Suchmaschinenherstellern übernommen werden. Im Fall von Fulcrum und MG ist die Schnittstelle der Textreader offengelegt. Damit ist es dem Nutzer möglich, beliebige Datenbanken seiner Wahl an die Suchmaschine zu koppeln.

Der Implementierungsaufwand variiert je nach Suchmaschine. Bei Fulcrum ist der Aufwand beispielsweise wesentlich höher als bei MG, da der Datenbank-Reader viel enger ins System eingebunden ist. Da Fulcrum die Dokumente nicht selbst speichert, erfolgen alle Zugriffe auf diese über verschiedene Aufrufe des Datenbank-Readers. MG dagegen benötigt den Reader nur einmal bei der Indexerzeugung, um die Dokumente in ein entsprechendes Format zu wandeln. Für ODBC/JDBC-Datenbanken kann von den Suchmaschinenherstellern ein entsprechender Reader bereitgestellt werden, da hierbei mit einmalig anfallendem Aufwand sehr viele Datenbanksysteme unterstützt werden.

Durch die Unterstützung der standardisierten Schnittstellen ODBC und JDBC können praktisch Suchmaschine und Datenbank beliebig gegeneinander ausgetauscht werden — ähnlich wie bei DB-Anwendungsprogrammen, die auf diese Schnittstellen aufsetzen.

Der Punkt der DB- und IR-Funktionalität wird von dieser Architektur nicht erfüllt. Wie im Abschnitt 4.1 schon erläutert, handelt es sich hier eher um eine Ressourcen- und nicht um eine Funktionskopplung. Eine solche Suchmaschine kann zwar Dokumente in einer Datenbank indizieren, bietet aber keinerlei Datenbankeigenschaften. Am Ende bleiben zwei getrennte Systeme: eines zur Verwaltung der Dokumente und eines zur Suche nach benötigten Dokumenten. Durch diese Trennung sind keinerlei Kombinationen von Anfragebedingungen möglich. Es sind entweder nur Suchanfragen an den Inhalt der Texte oder Datenbankanfragen an weitere gespeicherte Attribute möglich.

Der letzte Punkt bildet das K.o.-Kriterium für diese Variante der Kopplung. Systeme wie Fulcrum versuchen zwar, durch Erweiterungen der speicherbaren Zusatzinformationen mehr Datenbankfunktionalität zu bieten, reichen aber an die Anforderungen nicht heran. Außerdem bieten die meisten anderen Suchmaschine solche Funktionalität nicht einmal ansatzweise, wodurch die Zahl der benutzbaren Suchmaschinen drastisch eingeschränkt wird.

#### Datenbanken mit Suchmaschinenunterstützung

Diese Architektur zielt darauf ab, die Suchmaschinen als eine Art erweiterten Index zur Unterstützung neuer und komplexer Datentypen in das Datenbanksystem zu integrieren. Die Suchmaschine wird dabei über ihre Standardschnittstelle angesprochen. Da es keinen Standard für die Kommunikation mit Suchmaschinen gibt, ist hierbei die Verwendung einer festen Schnittstelle sinnvoll, um verschiedene Produkte unterstützen zu können. Die Anbindung der Suchmaschinen an diese Schnittstelle wird von speziellen Wrappern übernommen.

Diese Wrapper-Lösung erhöht zwar die Zahl der unterstützten Suchmaschinen, bedeutet aber gleichzeitig auch einen längeren Kommunikationsweg und einen zusätzlichen Implementierungsaufwand. Allerdings läßt sich dieser Aufwand wieder unter den entsprechenden Herstellern und Nutzern aufteilen. Die Datenbank muß nur noch eine virtuelle Suchmaschine unterstützen und die Suchmaschinen benötigen jeweils nur einen speziellen Wrapper für die entsprechenden Aufrufe. Wenn die Suchmaschinenschnittstelle bei allen ORDBMS-Erweiterungen gleich ist, reduziert sich so der Aufwand für die Suchmaschinenhersteller auf einen Wrapper.

Ein weiterer Vorteil einer einheitlichen Schnittstelle bei allen ORDBMS-Erweiterungen ist die beliebige Austauschbarkeit der einzelnen Komponenten. Das erhöht die Flexibilität und Erweiterbarkeit des Gesamtsystems. Der Nutzer kann sich die Komponenten nach seinen Bedürfnissen zusammenstellen oder einfach bereits vorhandene Systeme miteinander koppeln.

Eine Funktionskopplung ist durch diese Architektur größtenteils gegeben. Die Datenbankfunktionalitäten bleiben komplett erhalten und werden durch die Suchfunktionen der Textretrieval-Werkzeuge erweitert. Da die Kommunikation der Endanwendung aber immer mit dem Datenbanksystem erfolgt, können nur solche Funktionen genutzt werden, die von der Datenbankanfragesprache unterstützt werden. Je eingeschränkter also die Anfragesprache ist, desto weniger Funktionen können genutzt werden.

Die Performance spielt eine interessante Rolle in dieser Architektur. Es ist praktisch fast bedeutungslos, wie kompliziert die Wrapper-Anbindung ist und wie schnell die Ergebnisse der Suchmaschine zur Datenbank gelangen. Da es sich hier um einen Indexersatz für die Datenbank handelt und die Suche zur Laufzeit bei den betrachteten Daten extrem aufwendig ist, ergibt sich alleine durch die Verwendung einer Suchmaschine ein Performancegewinn bei den Datenbankanfragen. Der Anteil der Datenübertragung selbst ist nur einen Bruchteil der insgesamt benötigten Zeit für die Suche im Index und das Auslesen der Dokumente aus der Datenbank. Durch entsprechende Investitionen der Datenbankhersteller in die Erweiterungsmöglichkei-

ten ihrer Produkte können aber durchaus weitergehende Performancesteigerungen erzielt werden. Die Architektur bietet also genug Raum für neue Erweiterungen.

### Middleware-Ansatz

Dieser Ansatz ist in vieler Hinsicht sehr flexibel. Zum einen ist die Schnittstelle zum Anwender frei wählbar, wodurch sie sehr genau an die Anforderungen angepaßt werden kann. Zum anderen greift er auf die verwendeten Komponenten über deren Standardschnittstelle zu.

Bei Datenbanken handelt es sich wieder um ODBC und JDBC. Datenbanken, die keine solche Anbindung bieten, müssen einen entsprechenden Wrapper bereitstellen, der die Anfragen in das eigene Format umwandelt. Die Schnittstellen der Suchmaschinen sind leider nicht standardisiert. Um sich nicht auf eine Suchmaschine festzulegen, wird eine eigene Schnittstelle definiert, die alle Funktionen unterstützt, die vom System benötigt werden. Dadurch benötigt man für jede Suchmaschine einen eigenen Wrapper. Dieser kann entweder vom Anwender geschrieben, oder aber von den Herstellern mit den Suchmaschinen mitgeliefert werden — ähnlich wie bei den Datenbanken. Auf diese Art und Weise wird jede Suchmaschine unterstützt, die über alle nötigten Funktionen für diese Schnittstelle verfügt.

Durch die festgeschriebene Definition der Schnittstellen beschränkt sich der Implementierungsaufwand auf das Schreiben der entsprechenden Wrapper. Wie oben schon erwähnt, können diese sogar schon von den jeweiligen Herstellern bereitgestellt werden. Dies führt zu einer guten Arbeitsteilung und reduziert den Aufwand für den Nutzer des Systems.

Die vielen standardisierten Schnittstellen wirken sich allerdings auch negativ auf die Leistung des Gesamtsystems aus, da die Daten über viele Stufen (Treiber, Wrapper, etc.) weitergereicht werden. Trotzdem sollte auf die dadurch gewonnene Modularität nicht verzichtet werden, da dadurch das System erweiterbar bleibt.

Das Hauptproblem dieser Architektur liegt in der Middleware-Software selbst. Die große Flexibilität und Funktionalität dieses Ansatzes wird mit einem sehr hohen Implementierungsaufwand erkauft. Theoretisch ist zwar eine komplette Unterstützung aller Funktionen von Datenbank und Suchmaschine möglich, doch muß diese Unterstützung auch implementiert werden. Die Koordination von solch unterschiedlichen Systemen ist recht kompliziert. Außerdem muß die Kombination von Suchargumenten von der Middleware übernommen werden, so es unterstützt wird. Diese Auswertung und Verteilung der Anfragen durch ein drittes System kostet wiederum Performance.

### Fazit

Als Ergebnis dieser Gegenüberstellung der einzelnen Architekturen kann festgestellt werden, daß die *Datenbanken mit Suchmaschinenunterstützung* den hier gestellten Anforderungen am besten gerecht werden. Dies betrifft insbesondere das Verhältnis zwischen Implementierungsaufwand und erreichbarem Ergebnis.

Der Gesamtaufwand der Kopplung ist bei den ersten beiden Architekturen etwa vergleichbar, da hier das Hauptsystem schon vorliegt und nur die Kommunikation mit der anzubindenden Hilfskomponente entsprechend angepaßt werden muß. Der Middleware-Ansatz hingegen impliziert die Programmierung eines komplett neuen Systems und ist dadurch weitaus komplizierter zu realisieren. Sieht man die Middleware als reine Schnittstellenzusammenführung, so verringert sich zwar der Aufwand für die Middleware, bedeutet aber nur die Verschiebung der Arbeit in die Endanwendung.

Der Ansatz zur Erweiterung der Datenbank verspricht neben einem begrenzten Programmieraufwand gleichzeitig auch eine hohe Funktionalität des Gesamt-

systems. Weiterhin stellt diese Architektur nicht nur ein reine Lösung zur gemeinsamen Nutzung von DB- und IR-Funktionen dar, sondern beschreibt auch eine Möglichkeit zur Erweiterung von ORDBMS um die Unterstützung neuer Datentypen. Daher wird dieser Ansatz hier für die Entwicklung der eigenen Architektur favorisiert.

## 5.3 Entwicklung einer neuen Architektur

Es soll nun eine Architektur entwickelt werden, die den gestellten Anforderungen gerecht wird. Ausgangspunkt hierfür ist die Erweiterung von Datenbanken um Suchfunktionalität von externen Suchmaschinen, wie sie in Abschnitt 4.2 vorgestellt wurde.

In Kapitel 3 wurden zwei objektrelationale Datenbanksysteme mit entsprechenden Textretrieval-Erweiterungen vorgestellt. Sie unterschieden sich vor allem in der verwendeten Suchmaschine und in der konkreten Einbindung in das Datenbanksystem. Beide Systeme haben jedoch das Problem, daß sie auf eine spezielle Suchmaschine festgelegt sind. Da das Textretrieval-System nur so gut ist, wie die verwendete Suchmaschine, stellte sich das als größtes Handicap heraus. Anstatt aber nun einfach eine andere Suchmaschine zu verwenden — jedes der vier Text-DataBlades von Informix nutzt eine eigene Suchmaschine —, sollte das Ziel eher die Modularisierung der Architektur sein. Dadurch wäre sowohl die Datenbank als auch die Suchmaschine beliebig austauschbar und das Gesamtsystem kann dadurch besser an die Bedürfnisse des Nutzers angepaßt werden.

Um den Implementierungsaufwand gering zu halten, sollten die Schnittstellen fest definiert sein, so daß die Programmierung der entsprechenden Module an die verschiedenen Hersteller verteilt werden kann. Außerdem soll die Anbindung flexibel genug sein, um neue Erweiterungen der Datenbanksysteme und Suchmaschinen ausnutzen zu können. Es wird sich also um eine Art Template für Texterweiterungen handeln, das entsprechend an die verwendeten Systeme angepaßt wird.

Auf der anderen Seite muß man aber auch die Grenzen des Systems sehen. Ein Datenbanksystem muß nicht unbedingt die kompletten IR-Features implementieren und dadurch alle IR-Applikationen ersetzen. Eine Datenbank besitzt auch keine CAD- oder Kundenverwaltungsfunktionalität und wird trotzdem als Datenspeicher in solchen Applikationen eingesetzt. Die Datenbank dient nur der Verwaltung der Daten und der Kontrolle des Zugriffs auf diese. Um effizient mit einer Datenbank arbeiten zu können, muß diese natürlich gewisse Zugriffsmechanismen für die gespeicherten Daten bieten und genau um diese Zugriffsmechanismen geht es bei der Ankopplung von Textretrieval-Werkzeugen an Datenbanksysteme. Anders betrachtet wird das IR-System zu einem ausgelagerten Index für die Datenbank (vgl. dazu [Nit99]). Dieser Ansatz dient also einer verbesserten Unterstützung der neuen semi- und unstrukturierten Datentypen durch das ORDBMS. Darauf aufbauend können dann Datenbankapplikationen ein IR-System implementieren. In [HP00] wird dazu eine Architektur vorgestellt, die DB- und IR-Funktionalität auf einer höheren Ebene realisiert. Dazu werden die Datenbanken über spezielle Wrapper an das System angebunden, welches eine gemeinsame Anfragesprache für strukturierte und semi/unstrukturierte Daten bereitstellt (IRQL). Eine erweiterte Unterstützung von inhaltsbasierten Suchanfragen für diese Daten durch die ORDBMS würde die Anbindung an dieses System, in diesem Fall die benötigten Wrapper, deutlich vereinfachen.

### Die Architektur

Die Abbildung 5.1 zeigt die etwas abgewandelte und erweiterte Architektur für die Nutzung von ORDBMS als Textretrieval-System. Wie man sieht, besteht die Architektur aus drei Modulen. Diese Dreiteilung stellt eine klare Abgrenzung der Aufgaben in dem System dar und vereinfacht somit die Austauschbarkeit der Module, die nur über festgelegte Aufrufe miteinander kommunizieren.

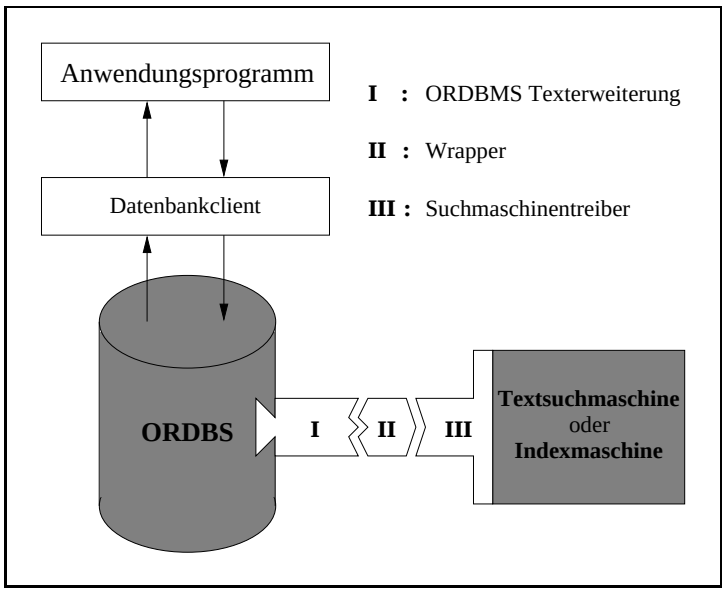


Abbildung 5.1: Suchmaschine als externer Index eines ORDBMS

### ORDBMS-Texterweiterung

Dieses Modul stellt die eigentliche Texterweiterung der Datenbank dar, entspricht also im wesentlichen dem Text-Extender oder dem Text-DataBlade. Dieses Modul kann vom Hersteller der Datenbank oder einem Drittanbieter geschrieben werden. Es beinhaltet alle Funktionen zur Erweiterung des ORDBMS um die Unterstützung von Textdaten. Dazu zählen unter anderem die Definition entsprechender Datentypen sowie der Routinen für den Zugriff und die Suche dieser Datentypen. Außerdem können Meta-Tabellen zur Speicherung zusätzlicher Informationen angelegt werden. Der Text-Extender von DB2 UDB verwaltet in solchen Tabellen z.B. alle vom ihm unterstützten Textspalten und deren Zuordnung zu einem entsprechenden Index.

Sofern die Datenbank keinen erweiterbaren Index anbietet, muß dieses Modul auch die Datenspeicherung in den entsprechenden Tabellen mittels Trigger überwachen. Ist die Möglichkeit eines erweiterten Indexes gegeben, so muß dieser angelegt und die benötigten Routinen dafür bereitgestellt werden.

Im Falle von Informix kann dieses Modul ein vereinfachtes DataBlade sein, das einen neuen Datentyp und entsprechende Zugriffsfunktionen bereitstellt. Die Suchfunktionalität wird von diesem Modul allerdings nicht implementiert, sondern nur die Aufrufe zur Suchmaschine weitergeleitet.

### Wrapper

Wie der Name es schon andeutet, dient dieses Modul der Anpassung der Schnittstellen. Dieser Wrapper stellt im Prinzip die Schnittstelle zwischen der Datenban-

kerweiterung und der Suchmaschine dar. Hier sind alle Funktionen definiert, die die Texterweiterung benötigt und von den Suchmaschinen implementiert werden müssen.

Der Wrapper muß nicht unbedingt ein eigenes Modul sein, da er mit zum Suchmaschinentreiber gehört. Der Grund für seine Existenz ist eher praktischer Natur und Ergebnis der ersten Testprototypen. Er dient der Vereinfachung bei der Entwicklung der beiden anderen Module, die jeweils nur den Wrapper zur Kommunikation benötigen.

Ein weiterer Grund für den Wrapper sind die unterschiedlichen Eigenheiten der Datenbanken bei der Funktionsprogrammierung. So ist z.B. die Art und Weise der Fehlerrückgabe ebenso unterschiedlich wie die Parameterübergabe an die Funktionen. Da die Funktionen des Suchmaschinentreibers ohne den Wrapper direkt mit den nutzerdefinierten Funktionen der Datenbank kommunizieren würden, müßten diese für jede Datenbank umgeschrieben werden. Der Wrapper vereinheitlicht also die Sicht der Suchmaschinen auf die Datenbanken.

Ein Wrapper ist immer für eine spezielle Kombination von ORDBMS und Suchmaschine ausgelegt. Beim Austausch eines der beiden Systeme muß er vom Anwender eventuell angepaßt oder ausgetauscht werden.

#### Suchmaschinentreiber

Dieser Treiber dient zur Kommunikation mit der Suchmaschine. Hier werden Standardaufrufe, wie das Anlegen, Aktualisieren, Löschen und vor allem das Durchsuchen eines Indexes mit Hilfe der API der Suchmaschine bereitgestellt. Dieser Treiber kann z.B. ein vereinfachtes Suchmaschinenanwendungsprogramm ohne interaktive Nutzerschnittstelle sein. Durch den Wrapper lassen sich bestehende Applikationen bequem an die Schnittstelle dieser Architektur anpassen. Der Wrapper ruft einfach die entsprechenden Routinen der Applikation auf.

Von den Anbietern der Suchmaschinen kann aber auch ein spezieller Treiber für diese Schnittstelle mitgeliefert werden. Da pro Suchmaschine nur ein Treiber benötigt wird, hält sich der Aufwand für die Hersteller in Grenzen.

### 5.3.1 Die Schnittstellen

Dieser Abschnitt befaßt sich mit einem der wichtigsten Punkte der Kopplungsarchitektur — den Schnittstellen. Das komplette System besitzt davon mehrere:

- Schnittstelle zum Anwendungsprogramm
- Schnittstelle zwischen Texterweiterung und Wrapper
- Schnittstelle zwischen Wrapper und Suchmaschinentreiber
- Schnittstelle zur Suchmaschine

Die Schnittstelle der Suchmaschine soll nicht verändert werden müssen, um eine einfache Unterstützung vieler Produkte zu ermöglichen. Notwendige Anpassungen werden von den Suchmaschinentreibern und den Wrappern erledigt.

Die Schnittstelle zwischen Wrapper und Suchmaschinentreiber dient der Anpassung des Treibers an die Forderungen dieser Architektur. Der Treiber ist eine Art Anwendungsprogramm der Suchmaschine und muß nicht zwangsläufig der Wrapper-API entsprechen. Er stellt eventuell fehlende Funktionalität für das Retrieval-System bereit, und der Wrapper paßt die Schnittstellen an. Demnach ist die Kombination aus beiden der eigentliche Wrapper (im herkömmlichen Sinne) für die Suchmaschine.

Über die Schnittstelle zwischen Texterweiterung und Wrapper erfolgt die Kommunikation zwischen den beiden zu koppelnden Systemen. Für alle Funktionen, die von der Texterweiterung benötigt oder zusätzlich von einer Suchmaschine bereitgestellt werden, muß hier ein entsprechender Funktionsaufruf definiert sein. Da es sich bei der Architektur nur um ein Template handelt, können jederzeit weitere Funktionsaufrufe hinzugefügt werden. Dadurch schränkt man zwar etwas die Möglichkeit ein, die Systeme auszutauschen, erlangt aber andererseits sehr viel Flexibilität bei der Erweiterung und Anpassung der Architektur.

Die letzte Schnittstelle dient der Anbindung an die Anwendungsprogramme. Dies sind im allgemeinen Datenbankapplikationen, welche über die Datenbanksprache mit dem System kommunizieren. Es müssen also Erweiterungen für diese Sprache definiert werden, um die neuen Fähigkeiten auch dem Anwendungsprogramm nutzbar zu machen.

Im folgenden werden die Schnittstellen genauer definiert und Standardfunktionen für das Template vorgestellt.

#### 5.3.1.1 Schnittstelle zum Anwendungsprogramm

Hierbei handelt es sich um eine Erweiterung der Datenbankanfragesprache um Textretrieval-Komponenten. Solche Erweiterungen existieren inzwischen in den verschiedensten Versionen. Die Textretrieval-Erweiterungen für DB2 UDB, Informix oder ORACLE definieren alle ihre eigenen Spracherweiterungen. Dabei handelt es sich im wesentlichen um neue, nutzerdefinierte Funktionen für den speziellen Textdatentyp.

Es gibt aber einige Bestrebungen, diese Spracherweiterungen zu standardisieren (Multimediaerweiterung von SQL; [SQLMM]) bzw. eine neue und erweiterte Sprache zu definieren (***I**nformation **R**etrieval **Q**uery **L**anguage* - IRQL; [HP00]). Eine neue Sprache bietet zwar konzeptionell eine bessere Integration von Datenbank- und IR-Anforderungen, benötigt aber erweiterte Funktionalität, die derzeit nicht alle bestehenden ORDBMS bieten. In [HP00] wird eine Wrapper-Architektur vorgeschlagen, mit deren Hilfe die fehlende Funktionalität nachgebildet werden kann. Die hier vorgestellte Architektur stützt sich nur auf bereits vorhandene Erweiterungsmöglichkeiten der ORDBMS, läßt sich aber auch an neue Sprachentwicklungen anpassen, sofern ihre Funktionalität nicht zu sehr von der hier vorgestellten abweicht. Im Falle eines Einsatzes von IRQL ist es sogar besser, bestimmte Funktionen für den eigentlichen Retrieval-Prozeß in dieser Datenbankerweiterung zu implementieren, anstatt sie im nachhinein in einem externen Aufsatz zu verwirklichen. Durch die Integration der Retrieval-Funktionalität in der Textretrieval-Erweiterung erhöht sich die Funktionalität des gesamten ORDBMS und die evtl. noch benötigten Wrapper für weitere Aufsätze werden kleiner und einfacher.

Der SQL/MM-Standard hingegen setzt auf den Spracherweiterungsfähigkeiten der ORDBMS auf, d.h. auf nutzerdefinierte Datentypen und Funktionen und standardisiert nur deren Syntax und Semantik. Er ist bereits verabschiedet und sollte sich in Neuentwicklungen vermehrt wiederfinden. Für die hier vorgestellte Architektur wird der Standard soweit unterstützt, wie es mit Hilfe der verwendeten Datenbanksysteme und Suchmaschinen möglich ist. Dies gilt insbesondere für die ORDBMS-Texterweiterung in Bezug auf die neu definierten Textdatentypen und die dafür bereitgestellten Funktionen bzw. Methoden.

Im Anhang A.1 befindet sich die Definition der Prototypen der im Template vorgesehenen Funktionen. Nachfolgend werden einige Gesichtspunkte diskutiert, die zu speziellen Entscheidungen, die Funktionen und Parameter betreffend, geführt haben.

### Suchfunktionen

Die wahrscheinlich wichtigste Funktion ist die sogenannte `contains()`-Funktion. Sie ermöglicht die Suche in den gespeicherten Textdokumenten mit Hilfe des Indexes einer Suchmaschine. Die Funktionen der Texterweiterungen unterscheiden sich allerdings im Namen, in den Übergabeparametern und den Rückgabewerten (siehe Abschnitt 3.2.2 und 3.3.2).

Die Funktionalität der `contains()`-Funktion bestimmt in großem Maße die Funktionalität und Flexibilität des gesamten Textretrieval-Systems. Deshalb müssen hierbei viele Punkte abgewogen werden. Als erstes kann die `contains()`-Routine eine nutzerdefinierte Funktion nach der Art der herkömmlichen ORDBMS sein oder aber eine Methode des neuen Textdatentyps, wie sie von neueren Versionen der ORDBMS unterstützt werden. Die Entscheidung darüber bleibt dem Programmierer der Texterweiterung überlassen, der sich mit dem ORDBMS am besten auskennen sollte.

Ein weiterer Punkt betrifft die Übergabe der Suchparameter an die Funktion. Die `contains()`-Funktion oder Methode dient nur der Einbindung der Suchanfrage in SQL. Die Interpretation der Eingabeparameter sollte den Suchmaschinen vorbehalten sein, um nicht von vornherein spezielle Suchmaschinen auszuschließen, die eine komplett eigene Anfragesprache besitzen. (Die Auswertung der Suchanfragen ist im übrigen Aufgabe des Suchmaschinentreibers, da wohl kaum eine Suchmaschine solche Anfragen von sich aus interpretieren kann.) Auf diese Weise erhält man einen generischen `contains()`-Aufruf, der für die unterschiedlichsten Suchmaschinen genutzt werden kann. Da eine Trennung der Implementierung angestrebt wird (Texterweiterung durch Datenbankhersteller, Suchmaschine durch Drittanbieter), ist eine derart generische Schnittstelle praktisch notwendig.

Allerdings garantiert eine generische `contains()`-Funktion noch keine Kompatibilität von IR-Anwendungen mit verschiedenen Suchmaschinen. Daher ist es sinnvoll, sich auch bei den Suchanfragen auf einen Standard zu einigen. Dadurch wird es möglich, eine andere Suchmaschine zu nutzen ohne das Anwendungsprogramm umschreiben zu müssen. Einen Ansatz hierfür bietet wieder der SQL/MM-Standard (siehe [SQLMM]), der genaue Vorgaben für die zu verwendenden Suchstrings gibt. Eine Suchmaschine kann sich dann als "SQL/MM-kompatibel" bezeichnen, wenn sie diesen Standard unterstützt, und wäre somit für alle Anwendungsprogramme auf SQL/MM-Basis geeignet. Andere Suchmaschinen müßten dann wie bisher eine genaue Beschreibung ihrer Anfragesprache mitliefern und sind somit auf die Unterstützung durch den Anwendungsprogrammierer bzw. Nutzer angewiesen.

In Spezialfällen ist ein Abweichen vom Standard durchaus sinnvoll. Bietet eine Suchmaschine z.B. eine natürlichsprachliche Anfrage, so wäre es wenig sinnvoll und auch nur schwer realisierbar, diese Anfrage in eine SQL/MM-konforme Anfrage umzuwandeln. Stattdessen sollte diese bequeme Anfragemöglichkeit dem Nutzer direkt verfügbar gemacht werden. Bei einer solchen Anwendung würde die Suchmaschine auch nur gegen eine verbesserte Version oder eine andere Suchmaschine mit natürlichsprachlicher Anfrage ausgetauscht werden.

Genausowenig wie der Inhalt der Übergabeparameter festgelegt sein sollte, dürfen auch keine zu strengen Festlegungen über deren Datentypen getroffen werden. In welcher Form die Werte an die `contains()`-Funktion übergeben werden, sollte der Programmierer der Texterweiterung, in den meisten Fällen also der Datenbankhersteller selbst, nach den gegebenen Fähigkeiten des Datenbanksystems festlegen. Mit Fähigkeiten ist hier insbesondere die Unterstützung von Objekten als Datentypen gemeint. So bietet z.B. Informix schon seit längerem die Möglichkeit, ein Textobjekt immer als Ganzes zu referenzieren, auch wenn man nur Informationen zu einzelnen Attributen benötigt.

Desweiteren kann die `contains()`-Funktion auch zu einer speziellen Methode

des Text Datentyps gemacht werden, was verhindert, daß diese Methode mit anderen Typen aufgerufen wird. Methoden haben noch einen weiteren Vorteil. Es können nämlich mehrere Methoden mit gleichem Namen unterschiedlichen Datentypen zugeordnet werden (*Overriding*). So kann es in einer Datenbank dann `contains()`-Methoden sowohl für Text, Bilder oder auch für Audio geben. Das System wählt selbständig immer die richtige Methode in Abhängigkeit vom Datentyp.

Ältere Datenbanksysteme unterstützen solche komplexen Datentypen noch nicht. So bietet z.B. IBM mit DB2 UDB diese Fähigkeit erst ab der Version 7.1 an. Aber auch solche Systeme können durch die Texterweiterungen unterstützt werden, sofern sie die sogenannten ***User Defined Types*** (UDT) und ***User Defined Functions*** (UDF) unterstützen. Dazu wird jeder Spalte mit Texten eine weitere Zeile mit dazugehörigen Handles zugeordnet. In diesen Handles werden dann alle nötigen Information zu dem dazugehörigen Text kodiert. Die Handles verwalten sozusagen die Attribute des Textes. Bei einem Aufruf einer zugehörigen Funktion wird dann nicht der Text selbst, sondern das entsprechende Handle übergeben. Auf diese Weise wird der Kommunikationsoverhead deutlich reduziert, da die Texte unter Umständen sehr groß werden können. Beim Zugriff auf den Index wird sowieso nicht der komplette Text benötigt, sondern nur der Name des Indexes. Diese Form der Textunterstützung wird unter anderem beim Text-Extender für DB2 UDB eingesetzt. Auch das bei den Methoden angesprochene *Overriding* kann simuliert werden, wenn die Handles der verschiedenen Datentypen, welche alle LOBs sind, als unterschiedliche UDTs definiert werden. Dadurch kann die jeweils richtige Funktion anhand ihrer Signatur ermittelt werden.

Als letztes fehlt noch die Festlegung der Rückgabewerte. Die beiden in Kapitel 3 untersuchten Texterweiterungen beschreiten auch hierbei unterschiedliche Wege. Die Funktion muß für jedes Dokument einen Wahrheitswert entsprechend der Qualifizierung für das Suchargument liefern. Bei Informix geschieht dies über den eingebauten `Boolean`-Typ; bei DB2 UDB wird dies über eine Vergleichsoperation simuliert. Allerdings hat man sich beim SQL/MM-Standard auf die bei DB2 UDB und dem Text-Extender praktizierte Methode geeinigt, so daß sie auch hier übernommen werden soll. Das Excalibur Text-DataBlade bietet darüber hinaus noch die Möglichkeit, einen `RETURN`-Parameter als dritten Parameter zu definieren. Mit Hilfe dieses Parameters können Informationen zum *Ranking* und *Highlighting* zurückgeliefert werden. Auch hierbei geht der Standard wieder den Weg von DB2 UDB. Für Anfragen, die einen Wert für die Relevanz des Suchargumentes im Dokument zurückliefern, gibt es die spezielle Funktion/Methode `rank()`. Auch sie wird von dieser Schnittstelle unterstützt, da kein ein anderes Datenbanksystem außer Informix die Möglichkeit von Rückgabewerten in nutzerdefinierten Funktionen/Methoden bietet.

Nach den hier aufgezeigten Punkten würde das Template für eine generische `contains()`-Funktion so aussehen:

```
CREATE FUNCTION contains(
          Textobjekt      Texttyp,
          Suchargument    Suchtyp
       )
       RETURNS INTEGER
```

Für *gerankte* Anfragen ist die folgende Funktion zuständig:

```
CREATE FUNCTION rank(
          Textobjekt      Texttyp,
          Suchargument    Suchtyp
       )
       RETURNS DOUBLE
```

Falls es sich um Methoden für den Textdatentyp handeln soll, muß `FUNCTION` durch `METHOD` ersetzt werden und der erste Parameter entfällt, da er durch `SELF` innerhalb der Routine referenziert werden kann. Die konkrete Natur des `Textobjekts` ist nicht festgelegt. Es kann sowohl ein kompletter Text in einem abstrakten Datentyp (wie beim Excalibur Text-DataBlade) oder einfach nur ein Handle (wie bei DB2 UDB Text-Extender) sein.

Der DB2 UDB Text-Extender bietet noch die Funktion `no_of_matches()`, welche natürlich auch hier definiert werden kann. Prinzipiell sollen alle existierenden Erweiterungen mit Hilfe dieser Architektur implementiert werden können — mit Rücksicht auf den neuen SQL/MM-Standard.

### Konfiguration der Suche

Ein weiteres Problem, das es zu lösen gilt, ist die Konfiguration der Suche. So unterstützt der Text-Extender von DB2 UDB z.B. verschiedene Arten der Indizierung, die ebenfalls alle unterschiedliche Anfragen unterstützen. Für jeden dieser Indexe wird ein eigenes Handle der Textspalte zugewiesen. Durch die Übergabe des richtigen Handles wird dann der passende Index zur Suche ausgewählt.

Die meisten Suchmaschinen bieten verschiedene Arten der Suche an, z.B. *Boolean-*, *Proximity-* oder *Fuzzy-Suche*. Weiterhin bieten einige die Möglichkeit, dynamische Parameter (z.B. das verwendete Retrieval-Modell) für die Suche zu verändern. All diese Auswahlen über verschiedene Handles zu treffen, ist kompliziert und ineffizient, da für jedes Handle eine neue Spalte erzeugt werden muß. Fulcrum bietet zum Ändern der Parameter den speziellen Aufruf `SET` und MG das Kommando `.set`.

Ein solcher `SET`-Befehl läßt sich allerdings sehr schlecht in SQL direkt einbinden. Der Grund dafür ist, daß er kein Ergebnis liefert, sondern nur Einstellungen ändert. Eine solche Funktion muß vor der eigentlichen Anfrage (Query) ausgeführt werden, da die Reihenfolge der Funktionsabarbeitung innerhalb einer Query nicht festgelegt ist, und dadurch die Parameter zu spät gesetzt werden könnten. Nicht alle ORDBMS bieten aber die Möglichkeit, eine Funktion oder gar Methode eigenständig außerhalb einer Query aufzurufen (wie z.B. Informix mit dem `exec`-Kommando). Außerdem muß dadurch eine externe Zuordnung der `SET`-Kommandos zu nachfolgenden Anfragen erfolgen.

Das Excalibur Text-DataBlade löst dieses Problem eleganter, indem die entsprechenden Parameter mit in das Suchargument eingebunden werden. Diese Methode soll hier aber nicht direkt übernommen werden, da sie nicht konform zum SQL/MM-Standard ist und auch der benötigte `ROW`-Operator nur in den neueren ORDBMS verfügbar ist. Anstatt die optionalen Parameter in das Suchargument zu schreiben, wird einfach ein zusätzlicher und optionaler Konfigurationsparameter eingeführt.

```
CREATE FUNCTION contains(
          Textobjekt      Texttyp,
          Suchargument    Suchtyp
        [,Konfig          Konfigtyp]
       )
       RETURNS INTEGER
```

Der Inhalt dieses Parameter ist suchmaschinenabhängig und sollte daher genau wie das Suchargument von der Texterweiterung nicht interpretiert, sondern nur an den Suchmaschinentreiber weitergleitet werden. Der Typ ist im einfachsten Fall eine Zeichenkette. Durch entsprechende Schlüsselwörter kann dann ein entsprechender Index ausgewählt oder aber ein dynamischer Parameter für die Suche gesetzt werden — und das für jede Suchanfrage getrennt, auch wenn mehrere `contains()`-Aufrufe

in einer Query auftauchen. Da der Parameter optional ist, beeinträchtigt er nicht die Kompatibilität zum SQL/MM-Standard.

### Beschränkung der Ergebnisse

Ein weiterer wichtiger Punkt, vor allem in Bezug auf die Optimierung der Anfrageabarbeitung, ist die Beschränkung der zurückgelieferten Ergebnisse. Viele Arbeiten beschäftigen sich mit diesem als *Top-k Query* bezeichneten Problem, wie z.B. [CK97, CK98, CG99, DR99]. Bei diesen Untersuchungen wird allerdings von Anfragen auf exakten Attributwerten ausgegangen und nicht von Anfragen auf Texte. Aus diesem Grund sind die Lösungen, die DB-Statistiken des Optimierers benötigen, hier nicht ohne weiteres anwendbar. Außerdem bietet sich mit der Verwendung von Textretrieval-Werkzeugen eine elegantere Lösung an.

Bei einer *gerankten* Query werden normalerweise alle Dokumente entsprechend ihrer Gewichtung ausgegeben. (Dokumente, die sich gar nicht qualifizieren, haben das Gewicht 0.) Selten sind aber alle Dokumente relevant, und der Nutzer beschränkt sich in der Regel auf z.B. die ersten 10, 50 oder 100 Dokumente. Dieses Verhalten kann zur Optimierung der Anfrage ausgenutzt werden, in dem nicht alle Dokumente aus der Datenbank geladen werden.

Solch eine Einschränkung wird am besten auf der Anfrageebene vorgenommen. Die am meisten verwendete Variante ist eine zusätzlichen `WHERE`-Klausel wie z.B. `WHERE rank > 0.5`. Damit kann die Ergebnismenge zumindest grob auf Dokumente einer gewissen Wichtigkeit eingeschränkt werden. In einer umfangreichen Textsammlung wie z.B. *TREC* kann es allerdings vorkommen, daß trotzdem noch zu viele Dokumente in der Ergebnismenge enthalten sind. Es ist von vornherein nicht abzuschätzen, welcher Ranking-Wert die Ergebnismenge ausreichend weit einschränkt, um nicht unnötig viel Rechnerressourcen für Ergebnisdokumente zu verbrauchen, die sich der Anfragende mit einiger Sicherheit nicht mehr anschaut.

Eine Lösung hierfür wäre natürlich, mit einem sehr hohen Ranking-Wert, z.B. 1, anzufangen und diesen dann in weiteren Anfragen immer weiter zu verkleinern, bis eine überschaubare Anzahl von Dokumente gefunden wurde. Allerdings kann es hierbei wiederum dazu kommen, daß sich bei kleiner Verringerung des Ranking-Wertes die Ergebnismenge plötzlich drastisch vergrößert, wenn die Verteilung der relevanten Dokumente in der Textsammlung ungünstig ist. Zur Veranschaulichung stelle man sich folgende Konstellation vor: 10 Dokumente haben eine Relevanz von 0.8 - 1 bzgl. einer Anfrage, aber bereits 1000 Dokumente werden bei einer Relevanz > 0.7 gefunden.

Das Problem liegt hierbei eindeutig darin, daß nicht explizit angegeben werden kann, wieviele Dokumente die Ergebnismenge maximal enthalten soll. Textretrieval-Systeme wie *MG* bieten hierfür die Option `maxdocs`, mit deren Hilfe man die Ergebnismenge explizit beschränken kann. Wenn man eine solche Option auch auf der Anfrageebene für die Datenbank hätte, kann der Nutzer die Ergebnismenge entsprechend begrenzen.

Diese Option läßt sich auch mit einem Ranking-Wert verbinden. Wird die Ergebnismenge im obigen Beispiel auf 100 Dokumente eingeschränkt, dann erhält man 10 Dokumente mit einer Relevanz zwischen 0.8 und 1 und 90 Dokumente mit einer Relevanz von 0.7. Ist man nun an genau diesen Dokumenten interessiert, dann verwendet man die `WHERE`-Klausel wie folgt: `WHERE rank = 0.7` und kann sich so diese Dokumente in entsprechend großen Blöcken, mittels `maxdocs`, anschauen. Ein weiteres Szenario ist ebenfalls denkbar. Man stellt eine Suchanfrage an eine Datenbank, ohne vorher zu wissen, wie die Ergebnisse ungefähr ausfallen, z.B. weil man die Texte in der Datenbank nicht kennt. Man kann dann die Ergebnismenge dahingehend einschränken, daß nicht mehr als 100 Dokumente als Ergebnis geliefert werden und andererseits aber auch keine Dokumente, die weniger relevant sind als

z.B. 0.8. Damit werden also 2 Beschränkungen festgelegt, eine nach der Anzahl und eine nach der Relevanz. Dies scheint viel sinnvoller als die Beschränkung der Anzahl über die Relevanz steuern zu wollen, denn dies setzt immer die Kenntnis der Dokumente voraus und bleibt trotzdem ein Glücksspiel.

Die Frage ist nun, wie die Option `maxdocs` umgesetzt wird. Es sind zwei Varianten denkbar. Zum einen kann es in die Datenbankabarbeitung eingebaut werden, welche die Anzahl der Ergebnisse beschränkt (siehe [CK97, CK98]). Zum anderen kann diese Beschränkung von der Suchmaschine umgesetzt werden.

Der erste Vorschlag hat, genau wie die Variante der Einschränkung über den Ranking-Wert, den Nachteil eines großen Overheads an Daten bei der Kommunikation zwischen Suchmaschine und Datenbank sowie bei der Auswertung der Daten durch das Datenbanksystem. Die meisten Daten sind dabei nutzlos, da sie durch die Beschränkungen wieder verworfen werden. Bei einer Textsammlung wie *TREC* würden über 700.000 Handles in absteigender Ordnung an das Datenbanksystem geliefert, welches diese dann entsprechend des Ranking-Wertes ausgibt oder nicht. Dabei würde das Datenbanksystem ohne weiteres nicht einmal wissen, daß die Daten vorsortiert sind und ab einem gewissen Handle der Vergleich eingestellt werden könnte. Zumindest die Zugriffe auf die eigentlichen Dokumente werden bei dieser Variante aber gespart.

Der zweite Ansatz bietet in vieler Hinsicht die besseren Eigenschaften. Als erstes wird dieses Feature schon von existierenden Suchmaschinen angeboten (siehe *MG*) und muß deshalb nicht neu in das System implementiert werden. Dadurch können mehr existierende Produkte unterstützt werden. Außerdem ergeben sich durch die Reduzierung des unnötigen Overheads Vorteile für die Performance. Überläßt man die Einschränkung der Ergebnisse der Suchmaschine, so verringern sich die zum einen die Kommunikationskosten bei der Übertragung der Ergebnis-Handles und zum anderen die Rechenzeit im Datenbanksystem. Im Gegensatz zu [CG99, DR99] sind hierfür keine aufwendigen Berechnungen und Statistikauswertungen nötig.

Die zweite Variante läßt sich einfach in das Modell integrieren, in dem über den suchmaschinenspezifischen Konfigurationsparameter die entsprechende Anweisung an die Suchmaschine übergeben wird — es handelt sich hierbei also um einen der oben angesprochenen dynamischen Parameter.

#### 5.3.1.2 Schnittstelle zwischen Texterweiterung und Wrapper

Während die eben vorgestellte Schnittstelle eher der Applikationsanbindung dient und jederzeit an neue Standards von SQL oder neuentwickelte Anfragesprachen angeglichen werden kann, stellt diese Schnittstelle die eigentliche Verbindung zwischen Datenbankkern und Suchmaschine her. Ihre Funktionalität wird sowohl durch das IR als auch durch die Anforderungen einer Indexunterstützung in dem Datenbanksystem bestimmt.

Die Einführung einer zusätzlichen Schnittstelle an diesem Punkt ist der bedeutendste Unterschied dieser Architektur zu den bisher existierenden Textretrieval-Erweiterungen für ORDBMS. Diese sind alle auf eine spezielle Suchmaschine festgelegt, welche mehr oder weniger stark in die eigentliche Erweiterung integriert war. Die Qualität der Texterweiterung ist entscheidend davon abhängig, wie gut sie die Anforderungen des Nutzers erfüllt. Ist die Retrieval-Qualitäten nicht ausreichend, dann muß eine komplett neue Textretrieval-Erweiterung oder im schlimmsten Fall sogar ein neues Datenbanksystem benutzt werden. Der letzte Fall ist auch noch mit einem extremen Portierungsaufwand der Daten von einer Datenbank in die andere verbunden. Die Extraschnittstelle zwischen der Textretrieval-Erweiterung und der Suchmaschine ermöglicht hingegen einen relativ einfachen Austausch der Suchmaschine.

Die Verwendung von Standardsuchmaschinen und die Kommunikation über deren Programmierschnittstelle, wie es eine der Anforderungen an die Architektur ist, bedingt unter anderem eine schwache Kopplung der beiden Systeme. Die Suchmaschine wird als eigenständiges Programm neben der Datenbank laufen und über entsprechende Bibliotheken mit der Retrieval-Erweiterung der Datenbank kommunizieren; eine Technik, die auch beim Text-Extender von DB2 UDB verwendet wird. Eine engere Kopplung wie bei den DataBlades von Informix benötigt Zugriff auf den Code der Suchmaschine, um diesen in das Modul zu integrieren. Selbst wenn dieser vorliegt (wie im Falle von MG), sind noch weitere Programmierarbeiten nötig, da der Aufbau der Suchanwendung abweichend ist. Bei der ersten Variante kann das System als solches bestehen bleiben und benötigt lediglich einen Wrapper zur Anpassung der Kommunikation.

In den nächsten Abschnitten werden nun einige Funktionen dieser Schnittstelle vorgestellt. Es ist hier deutlich die Beziehung zu den Funktionen/Methoden der Anwenderschnittstelle zu erkennen. Die genauen Prototypen-Definitionen sind im Anhang A.2 aufgelistet.

### Suchfunktionen

Jede von der Textretrieval-Erweiterung definierte Funktion braucht ein entsprechendes Äquivalent in der Schnittstelle zur Suchmaschine. Praktisch gesehen sind die Routinen der Erweiterung nur für die Parameterübergabe und -rückgabe im entsprechenden Format der Datenbank zuständig. Unter Umständen werden die Parameter noch aufbereitet, wie z.B. der Textparameter. Dieser enthält in der Regel einen Handlewert mit einkodiertem Namen des Indexes (wie beim DB2 UDB Text-Extender) oder aber ein Textobjekt mit entsprechenden Attributen (wie beim Excalibur Text-DataBlade). Für die Suchmaschinen ist nur der Name des Indexes wichtig, in dem die Suche stattfinden soll, und natürlich das Suchargument. Die Routine der Textretrieval-Erweiterung muß also den Namen des Indexes aus dem Textparameter extrahieren und dann die entsprechende Suchfunktion aufrufen.

Die hier vorgestellte Schnittstelle ist keinesfalls vollständig. Sie kann durch neue Fähigkeiten von Suchmaschinen oder Datenbanken jederzeit erweitert werden. Die einzige Funktion, die zur Suche mindestens verfügbar sein muß, ist `IR_Search()`. Daneben sollten noch die Funktionen `IR_RankSearch()` und `IR_SearchCount()` bereitgestellt werden. Diese Funktionen werden normalerweise im Wrapper implementiert und rufen ihrerseits die entsprechenden Routinen der Suchmaschine auf. Da der Wrapper speziell an die Suchmaschine angepaßt ist, können hier auch entsprechende Formatanpassungen des Suchstrings und der Fehlermeldungen vorgenommen werden. Beim Suchstring betrifft das in erster Linie das Quoten der Suchmuster, das je nach Datenbank und Suchmaschine mit Einfach- oder Doppelquotes erfolgt.

**IR_Search()** Sie erhält als Parameter den Namen eines Indexes in der Suchmaschine und den Suchstring. Wie weiter oben schon angesprochen wurde, sollte bis zu hier noch keine Interpretation des Suchstrings stattgefunden haben, um keine Annahmen über die Fähigkeiten der Suchmaschine zu machen und dadurch bestimmte Funktionalität auszuschließen. Als Ergebnis liefert die Funktion eine Struktur, die einen Zähler für die Anzahl der Ergebnisse und eine Liste der Dokumentidentifikatoren enthält. Liefert die Suchmaschine mehr Daten als nötig, so werden sie hier herausgefiltert.

**IR_RankSearch()** Diese Funktion entspricht im wesentlichen der `IR_Search()`-Funktion, bietet aber ein Ranking der Ergebnisse an. Dazu gibt sie eine erweiterte Ergebnisliste zurück, die zu jedem Dokumentidentifikator den entsprechenden Ranking-Wert angibt. Die entsprechenden Strukturen sind in Anhang A.2

beschrieben. `IR_RankSearch()` kann unter Umständen die gleiche Routine der Suchmaschinen-API aufrufen wie `IR_Search()`, nachdem z.B. vorher der entsprechende Suchmodus eingestellt wurde.

**IR_SearchCount()** Diese Funktion ist ebenfalls sehr nützlich. Sie liefert die Anzahl der Dokumente, die sich für eine Anfrage qualifizieren. Damit kann z.B. die `count()`-Funktion in SQL für Textobjekte überschrieben werden. Die beiden anderen Funktionen liefern zwar ebenfalls einen Wert für die Anzahl der Ergebnisse, aber viele Suchmaschinen unterstützen eine schnellere Suche, wenn als Ergebnis nur die Anzahl der gefundenen Dokumente erwartet wird. Um diesen Vorteil zu nutzen, gibt es diese Funktion.

Genaugenommen handelt es sich hierbei nicht um Funktionen, sondern um Prozeduren. Ihr Rückgabewert ist ein Fehlercode, der über Erfolg oder Mißerfolg und dessen Ursache Auskunft gibt. Die eigentliche Ergebnisliste wird über einen Parameter der Prozedur zurückgeliefert. Dieses Vorgehen entspricht weitestgehend dem Standard der Programmierung von Anwenderroutinen bei den verbreiteten Datenbanksystemen.

### Konfiguration der Suche

Zur Konfiguration der Suche wird den Routinen der Textretrieval-Erweiterung ein optionaler Parameter übergeben. Dieser ist suchmaschinenabhängig und muß deshalb uninterpretiert an die Suchmaschine weitergeleitet werden. Auf dieser Ebene nun wäre eine spezielle Funktion für die Konfiguration der Suche durchaus realisierbar. Allerdings ergibt sich hierbei ein Problem mit den meisten Suchmaschinen.

Eine Umkonfiguration der Suchparameter ist meistens nicht statisch, sondern auf eine bestehende Verbindung beschränkt. Dieses Verhalten ergibt sich aus der parallelen Unterstützung mehrerer Suchanfragen. Dadurch wären die Einstellungen durch eine getrennte Konfigurationsfunktion bei der nachfolgenden Verbindung zur eigentlichen Suche verloren. Aus diesem Grund wird zur Übergabe der Konfigurationswerte das gleiche Verfahren wie bei der Anwenderschnittstelle verwendet. Jeder Suchfunktion ist ein weiterer Parameter zugeordnet, der den eventuell angegebenen Konfigurationsparameter enthält, andernfalls ist er leer.

### Verwaltungsfunktionen

Neben den Routinen zur eigentlichen Suche sind auch noch Verwaltungsfunktionen denkbar. Das sind insbesondere die Funktionen zum Anlegen, Aktualisieren und Löschen eines Indexes (`IR_Create()`, `IR_Update()` und `IR_Delete()`). Hierbei gibt es aber weitaus größere Probleme als bei einer Vereinheitlichung der Suchanfrage. Das Erstellen eines Indexes ist sehr komplex in Bezug auf die benötigten Informationen und Auswahlkriterien und variiert dazu stark bei den verschiedenen Suchmaschinen. Alle Parameter in einer langen Zeichenkette zu übergeben, stellt keine befriedigende Alternative dar.

In der von Informix gebotenen DataBlade-Technologie stehen auch hierfür genaue Schnittstellen fest. Dies ermöglicht es, einen neuen Index genauso wie einen Standardindex mit `CREATE INDEX` zu erstellen und ihn mit `DROP INDEX` wieder zu löschen. Durch die Verwendung autonomer Produkte läßt sich solch eine Lösung nicht befriedigend realisieren. Daher soll wieder auf eine vom DB2 UDB Text-Extender genutzte Technik zurückgegriffen werden. Hier wird zur Erstellung und Verwaltung des Indexes ein externes Programm genutzt. Werden eigenständige Suchmaschinen verwendet, können deren (meist graphischen) Werkzeuge für diese Aufgabe genutzt werden. Da ein Index in der Regel nur einmal angelegt wird, stellt

diese Lösung keine sehr große Einschränkung dar, obwohl sie wenig transparent für den Nutzer ist.

Bei der Update-Funktion sieht es da schon anders aus. Als Parameter sollte der Name des Indexes ausreichen, damit dieser erneuert werden kann. Diese Erneuerung kann z.B. in regelmäßigen Zeitabständen automatisch erfolgen. Bei einigen Anwendungen ist das allerdings eine schlechte Lösung. Ändern sich Daten beispielsweise sehr unregelmäßig, dann wäre eine ständige Indexerneuerung unnötig. Will man dagegen zu einem bestimmten Zeitpunkt eine Anfrage auf einem garantiert aktuellen Index durchführen, muß es die Möglichkeit eines `UPDATE INDEX` geben. Ein solcher Befehl ist nicht Bestandteil des SQL, da Indexe bisher immer aktuell waren. Aber die Textretrieval-Erweiterung könnte eine solche Funktion bieten. Der DB2 UDB Text-Extender bietet z.B. auch die Möglichkeit, einen Index zu erneuern, wenn sich eine bestimmte Anzahl von Dokumenten geändert hat. Zur Überwachung der Änderungen dienen Triggerfunktionen und zum Aktualisieren des Indexes ist eine entsprechende Funktion nötig. Daher sollte die Funktion `IR_UPDATE()` von der Schnittstelle bereitgestellt werden.

#### 5.3.1.3 Schnittstelle zwischen Wrapper und Suchmaschinentreiber

Hierbei handelt es sich um eine zusätzlich eingeführte Schnittstelle, die den eigentlichen Wrapper für die Suchmaschine in zwei Module spaltet, einen zur Angleichung der Kommunikation und einen zur Erweiterung der Funktion. Das erhöht die Freiheit der Treiberprogrammierung. Bei der Entwicklung der Prototypen hat sich gezeigt, daß für den Treiber bereits vorhandene Anwendungen oder Treibermodule genutzt werden können. Durch den getrennten Wrapper entfällt das Umprogrammieren dieser Anwendungen für die neue Schnittstelle. Somit verringert sich der Implementierungsaufwand, da das Wrapper-Modul praktisch durch einfache Namensänderung und neue Parameteranpassungsroutinen immer wieder verwendet werden kann und auch für das Treibermodul bestehende Routinen genutzt werden können.

Erst die Funktionen im Suchmaschinentreiber führen eine Interpretation der übergebenen Parameter durch. Das bedeutet zwar, daß etwaige Syntaxfehler erst an dieser Stelle auffallen, sichert aber die Unterstützung aller Funktionen der Suchmaschine. Da die Entwickler der Textretrieval-Erweiterung nicht wissen, welche Suchmaschine einmal genutzt wird, sollten sie auch nicht deren Parameter überprüfen. Im Falle einer SQL/MM-kompatiblen Suchmaschine, wie sie weiter oben erwähnt wurde, bedeutet dies allerdings einen gewissen Aufwand für den Treiber. Hierbei kann der Suchstring nicht einfach weitergereicht werden, sondern muß vorher geparst und analysiert werden.

Diese Schnittstelle bietet keine festgelegten Funktionen mehr. Allerdings muß natürlich jede Funktion des Wrappers mit den hier gebotenen Funktionen ausgeführt werden können. Wie hierbei die verschiedenen Optionen zur Suche unterstützt werden, ist implementierungsabhängig. Diese Schnittstelle unterliegt keiner Regulierung, solange die Wrapper-API vollständig unterstützt wird.

### 5.3.2 Zugriff der Suchmaschine auf die Dokumente

Im vorhergehenden Abschnitt wurden die Architektur und die Schnittstellen für den Informations- und Datenfluß bei einer Suchanfrage beschrieben. Ein weiterer wichtiger Punkt ist der Datenfluß beim Erzeugen eines Indexes.

Die bisherigen Funktionen waren alle datenbankgesteuert, d.h. sie werden vom Datenbanksystem aufgerufen und greifen letztendlich auf die Standardschnittstelle der Suchmaschine als eine Art Suchmaschinenanwendungsprogramm zu. Auf Grund der Eigenarten der UDFs (man kann UDFs nicht von außerhalb des DBMS zum

Zugriff auf dessen Daten aufrufen) kann dieser Weg nicht für die andere Richtung genutzt werden.

Zur Erzeugung eines Indexes benötigt die Suchmaschine aber die Dokumente aus der Datenbank. In diesem Falle invertiert sich die Situation, die Suchmaschine erfragt Daten von der Datenbank und wird somit zu einer Datenbankanwendung. Dieses Szenario entspricht dem normalen Erzeugen eines Indexes bei *Suchmaschinen mit Datenbankanbindung* mit Hilfe der Datenbank-Textreader, kann also hier genauso eingesetzt werden. In Abbildung 5.2 ist diese Art der Anbindung dargestellt.

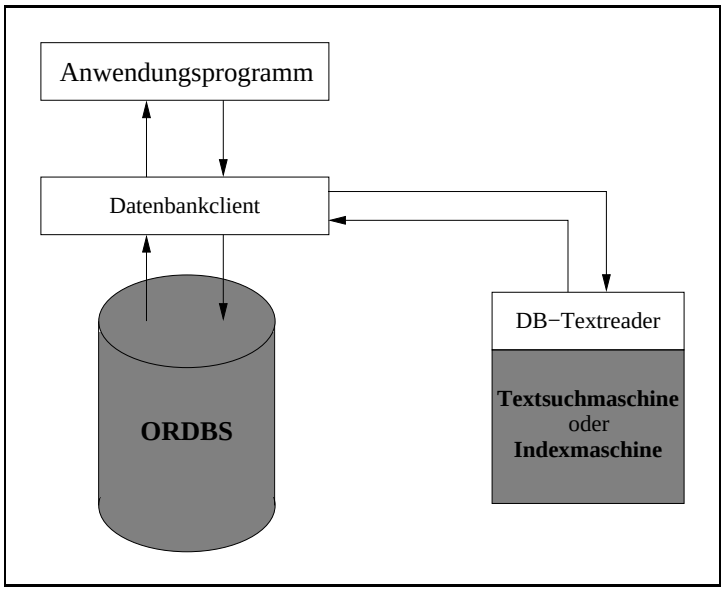


Abbildung 5.2: Suchmaschine als DB Anwendung bei der Indexerzeugung

Glücklicherweise ist die Schnittstelle für Datenbankanfragen durch ODBC und JDBC besser standardisiert als die für Suchmaschinenanfragen. Bei Suchmaschinen mit Datenbank-Textreader ist keine zusätzliche Programmierung notwendig, es sei denn, die entsprechende Datenbank wird nicht unterstützt. Bei Indexmaschinen und einfacheren Suchmaschinen muß ein entsprechendes Modul für den Datenbankzugriff geschrieben werden. Die Art und Weise des Zugriffs auf die Datenbank unterliegt keinerlei Einschränkungen, muß also nicht unbedingt über eine der Standardschnittstellen erfolgen. Eine Anbindung über ODBC/JDBC verbessert aber die Austauschbarkeit der Komponenten, in diesem Falle der Datenbank.

### 5.3.3 Unterstützte Produkte

Die Modularisierung dieser Architektur ermöglicht die Verwendung fast alle existierenden objektrelationalen Datenbankprodukte mit ihren entsprechenden Erweiterungen, angefangen von der einfachen Definition von nutzerdefinierten Datentypen und Funktionen bis hin zu Komponenten-Datenbanken. Auch bei den Textretrieval-Werkzeugen sind praktisch kaum Grenzen gesetzt. Welche Systeme sich am besten eignen bzw. die besten Ergebnisse und Leistungen erzielen, hängt vom speziellen Anwendungsfall an.

#### Objektrelationale Datenbanksysteme

Es werden prinzipiell alle Datenbanken unterstützt, die eine Erweiterung der Datentypen und vor allem der Funktionen bieten. Dazu zählen praktisch alle objekt-

relationalen Datenbanken seit ihrer frühesten Form.

**DB2 UDB** bot schon in der Version 5.0 eine Textretrieval-Unterstützung durch den sogenannten Text-Extender, obwohl nur einfache Typumbenennungen als Datentyperweiterung möglich waren. Allein durch die Möglichkeit der nutzerdefinierten Funktionen war eine solche Anbindung möglich. Mittlerweile hat sich dieses System weiterentwickelt und bietet freidefinierbare, strukturierte Objekttypen, was die Abspeicherung der Dokumente mit allen Attributen vereinfacht und das Arbeiten mit Textobjekten für den Nutzer transparenter werden läßt. Darüber hinaus bietet UDB auch erweiterbare Indexe (siehe [Nit99, WFC98]) an, was die Handhabung der Ergebnisse der Suchmaschine im Datenbankkern verbessert und dadurch eine bessere Performance des Gesamtsystems ermöglicht.

**Informix** bietet mit der DataBlade-Technologie die Möglichkeit der modularen Erweiterung aller wichtigen Komponenten des Datenbanksystems. Der Vorteil der DataBlades in Informix ist weniger die Textretrieval-Geschwindigkeit. Die meiste Zeit wird für die eigentliche Suche in der Suchmaschine und für das Laden der Ergebnisdokumente benötigt, nicht für die Kommunikation zwischen den beiden Systemen. Der wesentliche Vorteil liegt in der engen Ankopplung an das Datenbanksystem, speziell an Queryplaner und Optimierer. Hierdurch kann eine optimalere Bearbeitung der Datenbankanfrage erreicht werden und daraus folgend eine höhere Geschwindigkeit beim Daten-Retrieval.

Die hier vorgestellte Lösung stellt in gewisser Weise eine eingeschränkte Version der DataBlade-API dar, speziell den Anteil der SQL-Erweiterung. Die API kann jederzeit auch um Aufrufe für den Optimierer erweitert werden, so diese von dem verwendeten ORDBMS unterstützt werden. Neuere Versionen der heutigen Systeme werden solche Erweiterungsmöglichkeiten für den Optimierer bieten. Mit jeder Erweiterung der API bewegt man sich auf die Funktionalität der DataBlade-API zu, vergrößert damit aber gleichzeitig den erforderlichen Implementierungsaufwand.

### Textretrieval-Werkzeuge

Bei den Textretrieval-Werkzeugen gibt es wie in Kapitel 2 beschrieben drei verschiedene Systeme. Bei dieser Architektur kann praktisch jedes dieser Systeme eingesetzt werden. Allerdings ergibt sich hierbei ein unterschiedlicher Programmieraufwand für die erforderlichen Wrapper, Treiber und Administrations-Tools (Werkzeuge zum Erzeugen, Verwalten und Löschen von Indexen), je nachdem, wieviel Funktionalität das verwendete Produkt selbst bietet. Leider beinhaltet eine größerer Funktionalität auch einen größeren Kommunikationsoverhead, da solche Systeme nicht für den Einsatz als Index einer Datenbank gedacht waren.

**Indexmaschinen** sind die einfachste Form der Textretrieval-Werkzeuge. Sie bieten keine Programme für die Verwaltung der Indexe und auch die gebotene Anfrageschnittstelle ist sehr begrenzt. Das erhöht der Aufwand für die Treiberprogrammierung und die Verwaltungsprogramme. Andererseits entfällt bei diesen Systemen praktisch jeder Overhead, da sie nur den nötigsten eines externen Index bieten. Indexmaschinen sind daher schlank und schnell, und eignen sich aus Sicht der Performance und des Speicherplatzverbrauchs am besten für den Einsatz in dieser Architektur.

**Suchmaschinen** sind die am weitesten verbreitete Form der Textretrieval-Werkzeuge. Sie bieten nicht nur die reine Suchfunktionalität, sondern auch Verwaltungs-

werkzeuge (in vielen Fällen auch graphische) und vor allem eine komfortable und leistungsfähige Anfragesprache. Das vereinfacht die Programmierung und ermöglicht eine schnelle Anpassung vieler Suchmaschinen an die Textretrieval-Erweiterung einer Datenbank. Die Suchperformance ist mit der der Indexmaschinen vergleichbar, aber die Systeme sind größer und deshalb schwerfälliger. Aus Sicht des Programmieraufwands und der Einfachheit des Ankoppelns eignen sich Suchmaschinen für diese Architektur am besten.

**Volltextdatenbanken** sind für den Einsatz als Index in einem anderen Datenbanksystem weniger geeignet. Zum einen sind sie zu groß und schwerfällig und zum anderen verwalten sie die zu indizierenden Daten selbständig, was zu einer Doppelspeicherung führen würde. Prinzipiell können aber auch solche System verwendet werden, wenn man die Nachteile in Kauf nimmt. Aufgrund der gebotenen Leistungsfähigkeit fällt der nötige Programmieraufwand eher gering aus. Da eine Volltextdatenbank praktisch schon die Funktionalität dieser Architektur von sich aus verwirklicht, macht es, von Testzwecken abgesehen, wenig Sinn, sie als reinen Index zu verwenden, anstatt die kompletten Daten direkt damit zu verwalten.

## 5.4 Eigenschaften und Erweiterungsmöglichkeiten

In diesem Abschnitt sollen Eigenschaften und Erweiterungsmöglichkeiten der hier vorgestellten Architektur untersucht werden. Dabei geht es vor allem um eine Analyse, wie bestimmte Merkmale die Funktionalität des Datenbanksystems beeinflussen und wie sich das System an neue Funktionalität anpassen läßt.

### Index außerhalb der Datenbank

Normalerweise sind Indexe fest in den Datenbankkern integriert oder, wie im Falle der DataBlades bei Informix, mit aufwendigen Routinen an das System angekoppelt. Das DBMS ist dafür zuständig, daß die Indexe zu jeder Zeit auf dem aktuellen Stand sind. So wird die Grundforderung an Zugriffspfade in DBMS garantiert, daß das Ergebnis einer Anfrage auf derselben Datenmenge unabhängig vom verwendeten Zugriffspfad immer das gleiche ist. Es darf also nicht vorkommen, daß ein Datum nicht ausgegeben wird, weil es noch nicht in den Index aufgenommen wurde.

Diese Forderung nach der Konsistenz des Indexes mit dem Datenbankinhalt muß bei den neuen Datentypen und den Indexen auf ihnen neu überdacht werden. Im Bereich des IR existieren andere Forderungen als bei herkömmlichen Datenbanksystemen. Dies ergibt sich in direkter Folge aus dem hohen Rechenaufwand für die Erstellung eines Indexes für Textdaten und der Retrieval-Qualität der Indexe. Ein Index im Bereich des IR ist nicht mit denen der Datenbanken zu vergleichen. Es gibt keinen IR-Index, der eine hundertprozentig korrekte Antwort auf eine Anfrage liefert, so wie es bei Datenbanken gefordert wird. Dies liegt vor allem an den vagen Anfragen, wie sie im IR-Bereich auftreten ([Fuh92]). Es existieren also nur mehr oder weniger gute Retrieval-Maschinen.

Neben der schon eingeschränkten Qualität der Retrieval-Ergebnisse muß weiterhin noch ein Kompromiß zwischen Aktualität der gelieferten Ergebnisse und der benötigten Rechenleistung gefunden werden. Die Indexe eines IR-Systems haben also im allgemeinen eine gewisse Nachlaufzeit (z.B. einmal pro Tag aktualisieren). Da Datenbanken mit einer Textretrieval-Erweiterung für Aufgaben im IR-Bereich eingesetzt werden, gelten in diesem Fall auch die Forderungen des IR. Das bedeutet, daß die Konsistenz des Indexes nicht unbedingt mit hohem Rechenaufwand ständig gewährleistet werden muß. Nimmt man einen gewissen Nachlauf des Indexes in Kauf, kann man dadurch der Leistungsbedarf des Systems in Grenzen halten.

Durch einen Nachlauf des Indexes kann es aber zu Problemen bei der Anfragebearbeitung kommen. Man stelle sich vor, ein Dokument hat sich für eine Anfrage qualifiziert, wurde aber kurz zuvor gelöscht. Liefert ein herkömmlicher Index einen Identifikator für ein nicht mehr vorhandenes Dokument, so kommt es zu einem Laufzeitfehler. Damit solche Inkonsistenzen nicht zu Fehlern führen, wird ein sogenanntes ID-Mapping eingeführt (siehe [Nit99]), was im Prinzip einem JOIN der Menge aller existenten Dokument IDs mit der Menge der Ergebnis IDs entspricht. Dies verhindert den Zugriff auf nicht mehr vorhandene Datensätze. Daß neue Datensätze nicht sofort vom System erkannt werden, hat lediglich Auswirkungen auf die Retrieval-Qualität, nicht aber auf die Lauffähigkeit.

Das Speichern der Indexdateien außerhalb der Datenbank im Filesystem, stellt grundsätzlich keine Einschränkung für die Datenbankfunktionalität dar. Sie liegen genau wie die Datenbankdateien im Systemordner der Datenbankinstanz und dieser sollte vor willkürlichem Zugriff durch Unbefugte geschützt sein. Das Argument, die Indexdatei könnte ohne Wissen des Datenbanksystems gelöscht werden, trifft damit genauso auf die Datenbankdateien zu.

### Dokumente außerhalb der Datenbank

Die meisten Texterweiterungen bieten neben dem Standardverfahren, die Texte in einer entsprechenden Spalte abzulegen, auch die Möglichkeit, die Dokumente auf dem Filesystem zu belassen und nur einen Verweis darauf in der Datenbank zu speichern. Dieses Vorgehen bedeutet allerdings eine Einschränkung der Integritätssicherung durch das Datenbanksystem, da hierbei Daten gelöscht oder verändert werden können, ohne daß das DBMS davon Kenntnis hat. Es gibt aber auch spezielle Filesystemerweiterungen, die dem Datenbanksystem eine Kontrolle über spezielle Verzeichnisse ermöglichen.

Um dieses Problem mit der ausgelagerten Suchmaschine nicht noch weiter zu verschärfen, darf die Suchmaschine nur über entsprechende Datenbank-Reader auf die Texte zugreifen. Dadurch wird gewährleistet, daß die Suchmaschine nur auf den Dokumenten arbeitet, die auch in der Datenbank verwaltet werden. Auf diese Weise wird eine etwaige Kontrolle des Datenbanksystems, wie oben angesprochen, nicht hintergangen.

### Einbindung der Ergebnisliste in die Anfragebearbeitung

Die Auswertung der Suchergebnisse ist abhängig von der Textretrieval-Erweiterung und dem verwendeten ORDBMS. Bei einfachen Systemen, die nur skalare nutzerdefinierte Funktionen bieten, erfolgt die Auswertung innerhalb der Suchfunktion der Retrieval-Erweiterung. Die erhält nacheinander die IDs aller in der Datenbank gespeicherter Dokumente und durch den Aufruf der Suchmaschine die Liste der IDs aller qualifizierten Dokumente. Ein einfacher Vergleich der Listen ergibt ein WAHR oder FALSCH für das aktuelle Dokument. Diese Methode ist bei praktisch allen ORDBMS anwendbar.

Darüber hinaus gibt es aber noch Verbesserungsmöglichkeiten, die die Performance deutlich steigern können. Zum einen kann die skalare Funktion durch eine Tabellenfunktion ersetzt werden (siehe [IBM99]), wodurch der Vergleich der Mengen durch das JOIN des DBMS ersetzt wird und die vielen Einzelaufrufe einer skalaren Funktion gespart werden. [DM97] bietet eine Lösung, die die Handhabbarkeit der Anfragesprache beim Einsatz von Tabellenfunktionen verbessert.

Die neueste und beste Lösung sind erweiterbare Indexe, bei denen die Ergebnisliste direkt in die Indexverwaltung eingespeist wird. Diese Lösung kann bei Informix durch die DataBlade-Technologie und bei DB2 UDB ab Version 7.1 durch die Index Extension-Technologie genutzt werden. Eine genaue Beschreibung dieser

Lösung gibt [Nit99]. All diese Lösungen liegen im Aufgabenbereich der Textretrieval-Erweiterung und sollten unabhängig von der verwendeten Suchmaschine funktionieren.

#### Relevance Feedback

Das Relevance Feedback sollte nach Möglichkeit nicht zu tief in das Datenbanksystem integriert werden. Es entspricht nicht dem Anfrageschema einer Datenquelle und sollte deshalb besser in der Datenbankapplikation umgesetzt werden. Diese kann dabei auch noch entsprechende Nutzerprofile verwalten und so das Retrieval auf den einzelnen Nutzer abstimmen. Die Datenbankapplikation paßt die Suchanfrage entsprechend des Nutzer-Feedback an und erhält dadurch verbesserte Ergebnisse.

Dazu muß die Datenbank ihrerseits natürlich Informationen zu den entsprechenden Dokumenten liefern können, die es der Anwendung ermöglichen, die Anfragen umzuformulieren, um die Suche in eine gewisse Richtung zu lenken. Denkbar ist z.B. die Abfrage der relevanten Wörter eines Dokumentes, welche in eine erneute Suchanfrage aufgenommen werden können, um mehr Dokumente dieser Art zu finden. Dadurch kann ein Anwendungsprogramm interaktive Anfragen, wie in [WMB99] beschrieben, unterstützen.

#### Thesauri

Eine weitere Möglichkeit zur Verbesserung der Anfrageergebnisse ist die Nutzung von Thesauri (siehe [WMB99]). Diese müssen nicht extra außerhalb von der Suchmaschine verwaltet werden, sondern können auf die in [SQLMM] vorgestellte Art und Weise zur Verfügung gestellt werden. Dazu werden die entsprechenden Tabellen von der Datenbankerweiterung angelegt und mit Werten gefüllt. Die Textretrieval-Maschine kann diese Werte auf die gleiche Weise auslesen wie die zu indizierenden Dokumente.

### 5.4.1 Verwaltung mehrerer Suchmaschinen

Eine besondere Erweiterung der Architektur stellt die Unterstützung von mehreren Suchmaschinen dar. Wenn man die Suchmaschinen gegeneinander austauschen kann, dann können auch mehrere gleichzeitig zur Suche genutzt werden.

Durch einfache Umbenennung der nutzerdefinierten Funktionen für die SQL-Anbindung und der Texterweiterungsmodule, können mehrere Erweiterungen gleichzeitig an einer Datenbank angemeldet werden (genau wie die verschiedenen Module der DB2 Extender-Familie). Die entsprechenden Suchfunktionen können beliebig in einer Anfrage kombiniert werden. Bei der Implementation der Prototypen zum Test dieser Architektur wurden beispielsweise gleichzeitig drei Suchmaschinen über drei Textretrieval-Erweiterungen an das ORDBMS DB2 UDB gekoppelt — die IBM Textsuchmaschine über den DB2 Text-Extender, den Fulcrum SearchServer und die MG Suchmaschine jeweils über einen Extender dieser Architektur. Damit standen drei Suchmaschinen für die Volltextsuche auf den Dokumenten zur Verfügung, die über `contains()`, `FULcontains()` und `MGcontains()` angesprochen wurden.

Nachteil dieser Art der Erweiterung um mehrere Suchmaschinen sind die verschiedenen Namen für die Suchfunktionen. Diese müssen den Anwendungen bekannt sein, um sie zu nutzen. Verwendet jede Suchmaschine ihre eigenen Funktionsnamen, so bietet sich der Anwendung keine einheitliche Anfrageschnittstelle und ein Austausch der Suchmaschine würde auch den Austausch oder zumindest eine Umprogrammierung der Anwendungen bedeuten.

Um das zu vermeiden, muß eine einheitliche Anfrageschnittstelle gewahrt bleiben. Dies kann auf zwei unterschiedliche Weisen geschehen. Für beide muß al-

lerdings entweder die Textretrieval-Erweiterung oder der Wrapper zu einer Art Suchmaschinen-Broker ausgebaut werden. Die Suchmaschinen werden dann nicht mehr fest mit der Textretrieval-Erweiterung verbunden, sondern müssen dort angemeldet werden, ähnlich wie bei einem ODBC Driver Manager, der mehrere Datenbanken verwaltet.

### Explizite Suchmaschinenauswahl

Als erste Möglichkeit kann eine weitere Meta-Schnittstelle eingeführt werden, mit deren Hilfe die angemeldeten Suchmaschinen, die von ihr unterstützten Funktionen und deren exakte Namen erfragt werden können. Die hier vorgestellten Suchfunktionen stellen dabei die Menge der möglichen Funktionen und deren Bedeutung dar. Mit Hilfe der Meta-Schnittstelle werden die konkreten Funktionen für eine entsprechende Suchmaschine bestimmt. Wichtig hierbei ist, daß sich alle Suchfunktionen der einzelnen Erweiterungen an die hier festgelegte Signatur halten, da ansonsten die Parameterübergabe verkompliziert oder unmöglich wird.

Für die Meta-Schnittstelle sind drei Funktionen notwendig:

- `extenders()` liefert die verfügbaren Sucherweiterungen
- `capabilities()` liefert die unterstützten Funktionen
- `map()` liefert die konkreten Namen der Funktionen

Mit Hilfe dieser Funktionen kann eine Anwendung die verfügbaren Suchmaschinen erfragen und explizit eine oder mehrere für die Suche auswählen. Der genaue Ablauf sieht wie folgt aus.

**extenders()** Zuerst wird diese Funktion mit der Bezeichnung des unterstützten Datentyps aufgerufen. Als Ergebnis erhält man eine Liste der am System angemeldeten Erweiterungsmodule.
(z.B. `extenders( TEXT) -> [SearchEngine, FULCRUM, MG]`)

**capabilities()** Um die von einer speziellen Erweiterung unterstützten Funktionen zu erfahren, wird diese Routine mit dem Namen der Erweiterung aufgerufen. Das Ergebnis ist wieder eine Liste der Namen der unterstützten Funktionen. Dies sollten nur Funktionen aus einem festgelegten Pool für den entsprechenden Datentyp sein, damit die Anwendung weiß, wofür die entsprechende Funktion genutzt wird.
(z.B. `capabilities( FULCRUM) -> [contains, rank, count]`)

**map()** Als letztes dient diese Funktion der Ermittlung der exakten Namen der Funktionen, da nicht alle Erweiterungen ihre Suchfunktionen unter dem gleichen Namen anmelden können. Mit Hilfe der hierbei ermittelten Funktionsnamen kann dann die eigentliche Suchanfrage, an eine oder mehrere Suchmaschinen, generiert werden.
(z.B. `map( FULCRUM, contains) -> [FULcontains()]`)

Diese Lösung überläßt die volle Kontrolle, aber auch die Verantwortung für die Steuerung der Suchmaschinen dem Anwendungsprogramm. Das erhöht zum einen den Entwicklungsaufwand, bietet aber gleichzeitig dem Programmierer größere Flexibilität bei der Erzeugung von Suchanfragen und der Auswertung der Ergebnisse. Bestehende Anwendungen können nicht ohne Veränderungen mit dieser Lösung genutzt werden, es sein denn, es wird eine Standardsuchmaschine festgelegt, zu der die Suchanfragen beim Aufruf der Originalfunktionsnamen weitergeleitet werden.

#### Automatische Suchmaschinenauswahl

Bei der zweiten Lösung wird die Verwaltung der Suchmaschinen und Weiterleitung von Anfragen an diese automatisch von einer Textretrieval-Erweiterung übernommen. Dies erleichtert die Programmierung neuer und die Verwendung bestehender Anwendungen.

Bei einer Suchanfrage mittels `contains()` oder einer anderen Suchfunktion wird diese hier automatisch von dem Broker (Textretrieval-Erweiterung oder Wrapper) an die angemeldeten Suchmaschinen weitergeleitet. Dabei kann automatisch an alle angemeldeten oder aber nur an ausgewählte Maschinen weitergeleitet werden. Die Auswahl bevorzugter Suchmaschinen könnte über den Konfigurationsparameter der Suchfunktionen erfolgen. Die Ergebnislisten der einzelnen Suchmaschinen können entweder einfach gemischt oder aber vorverarbeitet und gewichtet werden. Techniken zum Mischen mehrerer Suchergebnisse bieten die Meta-Suchmaschinen aus dem WWW (siehe Abschnitt 2.2.3).

Soll auch hier die explizite Auswahl von Suchmaschinen unterstützt werden, so benötigt man wieder eine Meta-Schnittstelle mit den Funktionen `extenders()` und `capabilities()` (falls eingeschränkte Funktionalität erlaubt wird).

Dieses Szenario läßt sich noch weiter verfolgen. In [LH00] wird eine erweiterte Architektur vorgestellt, mit der verschiedene IR-Werkzeuge für Text, Bild, Audio und Video von nur einer sogenannten Multimediaerweiterung für das ORDBMS verwaltet werden. Dadurch soll eine noch bessere Transparenz beim Zugriff auf Multimediadokumente und die Einbeziehung strukturierter Zusatzinformationen erreicht werden. Durch die erhöhte Integration der Funktionen steigen aber auch die Anforderungen an die Anfragesprache, so daß über den Einsatz einer neuentwickelten IR-DB-Anfragesprache wie in [Fuh92, HP00] nachgedacht werden muß.

## 5.5 Zusammenfassung

In diesem Kapitel wurde eine Möglichkeit aufgezeigt, mit der man herkömmliche ORDBMS und Textretrieval-Werkzeuge auf relativ einfache Weise miteinander verbinden kann, so daß sie sich in ihren Funktionen ergänzen.

Nach der Analyse der existierenden Textretrieval-Erweiterungen war zu sehen, daß die bestehende Technik durchaus funktionsfähig ist, die Produkte jedoch durch zu konservative Programmierung nicht den Anforderungen am Markt entsprechen konnten. Dabei lag das Problem nicht an den Datenbanken selbst, sondern an der Festlegung auf Textretrieval-Maschinen, die eine schlechtere Qualität als die führenden Produkte im IR-Bereich bieten.

Die offensichtliche Lösung für diese Situation ist eine modularere Programmierung der Textretrieval-Erweiterung, wie sie in der Software Entwicklung heutzutage weit verbreitet ist. Der Text-Extender von DB2 UDB lieferte die Idee, daß zur Suche auf den Datenbankinhalten eine externe Suchmaschine eingesetzt werden kann. Allerdings ist der Text-Extender zu sehr an diese Suchmaschine angepaßt, als daß sie einfach gegen eine andere ausgetauscht werden kann.

Bei der Kommunikation von verschiedenen Applikationen gibt es auf Betriebssystemebene verschiedenste Lösungen, wodurch z.B. ein Dateiverwaltungsprogramm einfach einen Bild-, Audio- oder Videobetrachter quasi als Anzeigekomponente aufrufen kann. Das gleiche Prinzip kann man sich auch bei den ORDBMS und Textretrieval-Werkzeugen zunutze machen.

Die hier vorgestellte Architektur wurde durch die Implementation von Testprototypen in der Praxis getestet und hat ihre Tauglichkeit unter Beweis gestellt. In Forschungseinrichtungen und Universitäten sind mit ihrer Hilfe sehr einfach verschiedene Kombinationen von Datenbanken und Suchmaschinen realisierbar. Für den

Einsatz im Endanwenderbereich bedarf es allerdings einer breiten Unterstützung durch die Hersteller der entsprechenden Produkte, damit alle notwendigen Module und Programme verfügbar sind.

Für die Zukunft bleibt noch viel Spielraum für die Erweiterung der Anfragesprache der Datenbanken, um die typischen Anfragen in IR-Anwendungen besser zu unterstützen. Ebenso benötigen noch viele der ORDBMS weitergehende Erweiterungsmechanismen für den Datenbankkern. Vor allem eine bessere Unterstützung der nutzerdefinierten Funktionen und damit der Suchanfragen durch den Anfrageoptimierer wird immer wichtiger. Die hier vorgestellte Methode der Kopplung ist offen für solche Erweiterungen und kann sie zur weiteren Leistungssteigerung nutzen.

# Kapitel 6

# Prototypische Implementierung

Die im letzten Kapitel vorgestellte Architektur wurde vorwiegend durch theoretische Überlegungen und die Analyse der Funktionalität von existierenden ORDBMS und Suchmaschinen erarbeitet. Gleichzeitig wurden die Ideen aber auch praktisch umgesetzt, um die Praxistauglichkeit der Lösungen zu testen. So ist z.B. die Dreiteilung der Verbindungsschnittstelle durch das zusätzliche Wrapper-Modul als direkte Konsequenz aus diesen Testimplementierungen hervorgegangen.

Das Ziel dieser Arbeit war es, möglichst viele der vorhandenen Datenbanken und Suchmaschinen miteinander zu koppeln. Dabei sollte der benötigte Aufwand eingeschätzt und Probleme mit konkreten Systemen festgestellt werden. Dieses Kapitel gibt zunächst einen Überblick über die gesammelten Erfahrungen und Ergebnisse bei der Anwendung der neuen Kopplungsarchitektur. Danach werden die Systeme, die hier untersucht wurden, konkret mit ihren Besonderheiten in Bezug auf die Anforderungen der Architektur vorgestellt und die Implementation genau beschrieben.

## 6.1 Ergebnisse und Erfahrungen

Bei der Implementierung lag der Schwerpunkt auf der Suchanbindung, d.h. auf dem Zugriff auf verschiedene Suchmaschinen über eine `contains()`-Funktion vom ORDBMS aus. Es wurde keine vollständige Textretrieval-Erweiterung mit zwei konkreten Systemen realisiert, da die Verwaltungs- und Überwachungsfunktionen zu aufwendig sind und außerdem von den bereits existierenden Text Erweiterungen übernommen werden können.

Die implementierten Lösungen bieten die Funktionalität einer statischen Dokumentensammlung. Das bedeutet, daß keine Inserts und Updates auf der Dokumententabelle durchgeführt werden können, bzw. daß diese nicht in den Index übernommen werden. Allerdings ist das Einrichten eines zyklischen Updates des Indexes mit Hilfe des Aufgabenmanagers des Betriebssystem (unter UNIX OS der `crond`) leicht zu realisieren. Ob dabei der Index komplett neu angelegt oder inkrementell erweitert wird, hängt davon ab, ob ein Indexupdate von der Suchmaschine unterstützt wird.

Die hier vorgestellten Lösungen bieten alle Funktionen, um eine Dokumentensammlung in die Datenbank einzulesen, die entsprechende Textspalte in der Datenbank für die Textretrieval-Erweiterung anzumelden und einen Index durch die Suchmaschine erzeugen zu lassen. Außerdem wurde eine entsprechende Suchfunktion bei der Datenbank angemeldet. Wie im letzten Kapitel bei den Erweiterungsmöglichkeiten schon angesprochen, wurden hier für jede Suchmaschine eigene Funktionsnamen

verwendet, um mehrere Systeme gleichzeitig testen zu können.

In Bezug auf den Arbeitsaufwand für die Ankopplung einer Suchmaschine an ein ORDBMS kann positiv festgestellt werden, daß dieser sehr gering ist. Das gilt vor allem dann, wenn schon entsprechende Routinen für den Zugriff auf die Suchmaschine und gewisse nutzerdefinierte Funktionen für die Datenbank vorhanden sind. Solche Routinen fanden sich bei allen hier betrachteten Systemen als Beispielanwendung bzw. Programmierbeispiel. Einzig Informix machte hier eine Ausnahme, da man eher mit der Programmierung kompletter DataBlades als mit einfachen UDRs rechnet.

### Das Template

Das Programmier-Template besteht aus der Vorgabe der zu installierenden nutzerdefinierten Funktionen im ORDBMS und aus dem Wrapper-Modul nebst zugehörigen Header-Dateien. Konkret sind das die Dateien:

- extender.ddl
- IRapityp.h
- IRapi.h
- wrapper.c

Diese sind im Anhang A aufgelistet und erklärt. Bei der ersten Datei handelt es sich um ein Datenbankskript mit Beispiel-DDL-Anweisungen zum Anmelden von nutzerdefinierten Datentypen und Funktionen. Diese Datei wird von der Textretrieval-Erweiterung der Datenbank verwendet und muß an diese entsprechend angepaßt werden. Alle anderen Dateien werden für das Wrapper-Modul benötigt. Der Wrapper wird zusammen mit dem Suchmaschinentreiber zu einer Bibliothek gelinkt. Bei der Programmierung der Datenbankerweiterung dienen die Header-Dateien und das Wrapper-Template nur der Symboldereferenzierung bei der Übersetzung, da normalerweise kein Suchmaschinentreiber zur Verfügung steht.

Den größten Implementierungsaufwand beanspruchen die Textretrieval-Erweiterungen und die Suchmaschinentreiber. Erfahrene Programmierer von Datenbank- oder Suchmaschinenanwendungen haben hier jedoch keine Probleme und auch weniger erfahrene gelangen mit Hilfe von Beispielroutinen und -anwendungen der entsprechenden Produkte schnell zum Ziel. Außerdem müssen genau diese Module nur einmal pro System (ORDBMS oder Suchmaschine) programmiert werden und können dann für die Kombination mit einem anderen System wiederverwendet werden. Das verringert den Gesamtaufwand bei der Kombination mehrerer Systeme deutlich.

Das einzige Modul, das spezifisch für jede Kombination von Datenbank und Suchmaschine ist, ist der Wrapper. Dieser erledigt aber nur im Bedarfsfall kleinere Konvertierungsaufgaben und ist deshalb meistens auch wiederzuverwenden bzw. einfach umzuprogrammieren. Bei den Testimplementationen handelte es sich hierbei um die Konvertierung der Doppelquotes, welche die Suchargumente bei der SQL-Anfrage umschließen, in Einfachquotes für den Fulcrum SearchServer. Für das MG System mußten sie komplett entfernt werden.

Nach der Implementation der entsprechenden Texterweiterungen für die einzelnen Datenbanken und die Treiber für die Suchmaschinen reduziert sich der Arbeitsaufwand für die eigentliche Ankopplung zweier Systeme auf etwa einen Tag und ist somit mehr als vertretbar.

## 6.2 Datenbankerweiterungen

Hierbei handelt es sich um das wohl aufwendigste Modul der gesamten Architektur. Dabei entfällt der geringste Teil auf die Programmierung der nutzerdefinierten Suchfunktionen für die Einbindung der Suche in SQL (d.h. `contains()`, etc.). Den größten Anteil der Erweiterung macht die Verwaltung und Überwachung der Dokumente innerhalb der Datenbank aus. Hierzu gehören die Triggerfunktionen für die Überwachung der Dokumententabellen und die Meta-Tabellen für die Speicherung von Informationen der angemeldeten Textspalten.

Diese Funktionalität wird allerdings von den heute verfügbaren Textretrieval-Erweiterungen bereits zur Verfügung gestellt und kann durchaus weiterverwendet werden (das gilt natürlich nur für die Hersteller der entsprechenden Erweiterungen). Für die Testimplementationen wurde auf diese Funktionalität aus Kostengründen (hierbei ist der Arbeitsaufwand für die Implementation gemeint) verzichtet, was darin resultierte, daß nur statische Dokumentensammlungen verwaltet werden können. Für das Testen von Qualität und Geschwindigkeit des Retrievals der verschiedenen Suchmaschinen ist die gebotene Funktionalität jedoch zureichend.

Im einzelnen wurden Erweiterungen für die beiden objektrelationalen DBMS DB2 UDB und Informix Universal Server programmiert. Die verfügbaren Funktionen umfassen das Einlesen einer Dokumentensammlung in die Datenbank, das Anlegen einer entsprechenden Zusatzspalte für die Dokumentidentifikatoren und natürlich das Generieren dieser Werte.

### 6.2.1 DB2 UDB

DB2 UDB bildete mit seinem Text-Extender die Vorlage für die neue Architektur. In vielen Teilen wurde die Funktionalität des Text-Extenders nachprogrammiert. Das wichtigste Merkmal, welches übernommen wurde, ist das Zuweisen eines eindeutigen Handles zu jedem Dokument in der Datenbank. Der Grund hierfür sind die begrenzten objektorientierten Fähigkeiten der bisherigen UDB Versionen. So kann mit den vorliegenden Systemen kein komplexer Datentyp zum Speichern von Textdokumenten mit all ihren Zusatzinformationen wie Sprache, Format, Indexname, usw. erzeugt werden. Aus dem gleichen Grund führte der Text-Extender das Dokument-Handle ein und speicherte die Zusatzinformationen kodiert in diesem Handle ab. Diese Methode funktioniert bei allen bekannten ORDBMS, auch wenn neuere Systeme erweiterte Möglichkeiten bieten.

Der ***O**bject **Id**entifier* (OID) kann zwar zur eindeutigen Bestimmung des jeweiligen Textes genutzt werden und wird auch von den vorliegenden UDB Versionen unterstützt (wie auch von den meisten anderen ORDBMS), doch lassen sich darin keine Zusatzinformationen einkodieren. Dadurch entfällt die Möglichkeit, den Indexnamen implizit über den Identifikator an die Suchfunktion übergeben zu können; was einen zusätzlichen Parameter erforderlich macht. Die OID kann erst bei der Verwendung der Index Extension der neuen UDB Versionen als Dokumentidentifikator genutzt werden (siehe [Nit99]). Die Index Extension steht in UDB allerdings erst ab der Version 7.1 zur Verfügung.

#### Anmelden der Erweiterung

Das Anmelden der Erweiterung beschränkt sich bei UDB auf das Definieren des neuen Textdatentyps und der neuen Suchfunktionen. Der Datenbankkern weiß nichts über verfügbare Erweiterungen.

In Kombination mit dem Fulcrum SearchServer gab es leider Komplikationen mit dem Zugriff auf nutzerdefinierte Datentypen, was dazu führte, daß auf das Mapping des Typs `VARCHAR( 30)` auf `TEXTHD` zum Speichern des Handles verzichtet

wurde. Dies hat zwar keine Auswirkungen auf die Funktionstüchtigkeit der Erweiterung, macht aber eine Typüberprüfung des Handleparameters durch das Datenbanksystem unmöglich. Die Suchfunktion kann somit mit jedem beliebigen Wert des Typs `VARCHAR( 30)` aufgerufen werden, auch wenn es sich nicht um ein Text-Handle handelt. Dadurch könnte es unter Umständen zu Laufzeitfehlern kommen. Daß diese Einschränkung nicht durch DB2 bedingt ist, beweist der Text-Extender, der den Typ `DB2TEXTH` für die Handles definiert und so Typsicherheit für seine Suchroutinen erreicht.

Bei diesem Prototyp wurde als Suchfunktion lediglich `contains()` angemeldet. Wie in Abschnitt 5.4.1 schon erwähnt, wurden für jede getestete Suchmaschine verschiedene Funktionsnamen verwendet und mehrere Funktionsbibliotheken eingebunden. Dies geschah aus dem Grund, mehrere Suchmaschinen gleichzeitig testen zu können und beeinträchtigt nicht die Funktion der Erweiterung — bis auf den Fakt, daß sie nicht generisch für alle Suchmaschinen eingesetzt wird, sondern für jede speziell abgeändert wurde.

#### Laden der Dokumente

Das Laden der Dokumente erfolgt über den Standardweg von UDB mittels des `IMPORT` Befehls. Ein kleines Shellskript (`crtimp.sh`) hilft bei der Erzeugung der Importdatei und ein Datenbankskript (`dbload.clp`) lädt schließlich die Dokumente in die Datenbank.

#### Aktivieren der Textspalte

Das Aktivieren der Spalte mit den Dokumenten ist beim originalen Text-Extender ein komplexer Prozeß, bei dem unter anderem die Meta-Informationen in die entsprechenden Tabellen des Extenders geschrieben und Trigger für die Tabelle eingerichtet werden, um jede Inhaltsveränderung der Tabelle zu registrieren. Der wesentlichste Bestandteil ist jedoch das Hinzufügen einer neuen Spalte für die Dokument-Handles und das Generieren dieser Handles.

Die beiden letzten Aufgaben müssen auch von der neuen Erweiterung erfüllt werden. Dafür gibt es wieder zwei Programme: `alter_db` fügt die Handlespalte hinzu und `gen_hd` erzeugt die entsprechenden Werte und trägt sie in die neue Spalte ein. Zu diesem Zeitpunkt muß bereits der Name des Indexes für diese Dokumente bekannt sein, da er in das Handle einkodiert wird. Die Aufgabe des Namenszuweisung wird von einem Shellskript übernommen, das den kompletten Ablauf der Dokumentenindizierung steuert und auf die Suchmaschine abgestimmt wird. Dieses Skript wird bei den *konkreten Kopplungen* (Abschnitt 6.4) genauer erläutert.

### 6.2.2 INFORMIX

Obwohl Informix bereits mit der verfügbaren Version sehr viel mehr objektorientierte Möglichkeiten bietet, wurde die Textretrieval-Erweiterung ebenfalls nur auf dem Level der DB2 UDB Erweiterung implementiert. Für den Test der Suchmaschinen ist diese Lösung völlig ausreichend, schöpft aber keinesfalls das ganze objektrelationale Potential von Informix zur Steigerung der Performance und Anwenderfreundlichkeit aus.

Da die Implementation eines kompletten DataBlades aufgrund der vielen obligatorischen Funktionen zu aufwendig ist, wurden wie bei der Erweiterung für UDB nur Datentyp und Suchfunktionen implementiert. Obwohl Informix komplexe Datentypen unterstützt, wurde wieder auf die Handlelösung zurückgegriffen, um möglichst viel Quellcode wiederverwenden zu können. Kommerzielle Textretrieval-Erweiterungen sollten allerdings auf den existierenden Text-DataBlades aufsetzen.

Im Gegensatz zum Excalibur Text-DataBlade ändert sich somit die Nutzung der Erweiterung und entspricht der DB2 UDB Erweiterung. Hierbei wird deutlich, wie ähnlich sich die ORDBMS eigentlich sind und daß die gleiche Erweiterung mit nur kleinen Anpassungen auf verschiedene Systeme übertragbar ist.

#### Anmelden der Erweiterung

Wie schon angedeutet, wird die Erweiterung nicht über die DataBlade-API bei der Datenbank angemeldet, obwohl dies vom System unterstützt wird. Somit wissen andere DataBlades nichts von der Existenz einer Texterweiterung. Das ist aber deswegen unkritisch, da diese nicht dem Standard von Informix für Texterweiterungen entspricht. Wenn ein DataBlade als Texterweiterung am System angemeldet ist, sollte es auch die von Informix für eine solche Erweiterung vorgesehene Funktionalität bieten.

Die hier implementierte Erweiterung dient wieder nur der Evaluierung verschiedener Suchmaschinen mit diesem Datenbanksystem. Deshalb beschränkt sich das Anmelden auch hier auf den Datentyp für das Text-Handle und die nutzerdefinierte Routine (UDR, wie sie unter Informix heißt) zum Suchen. Im wesentlichen trifft hier das Gleiche zu, wie bei der DB2 UDB Erweiterung, da es sich hier "nur" um eine Portierung derselben auf das Informix Datenbanksystem handelt — eine Methode zur Senkung des Implementierungsaufwandes.

Auf das Erzeugen eines neuen Datentyps für das Text-Handle wurde zur Vereinfachung des Parameter-Handlings in den UDRs und dem Textreader von Fulcrum auch hier verzichtet. Es wurde der Standardtyp `VARCHAR( 30)` für die Speicherung der Text-Handles gewählt. Als Suchfunktion wurde lediglich `contains()` getestet, welche für jede Suchmaschine einen spezifischen Namen erhielt.

#### Laden der Dokumente

Zum Einlesen der Dokumente existieren wieder zwei Skripte: `crtimp.sh` erzeugt eine Importdatei und `dbload.sql` liest die Dokumente per `LOAD` Befehl in die Datenbank ein.

#### Aktivieren der Textspalte

Bei den originalen Text-DataBlades wird eine Textspalte nicht wie beim Text-Extender von DB2 UDB aktiviert. Sie nutzen das `CREATE INDEX`-Kommando und stellen die für die Indexverwaltung nötigen Routinen bereit. Auch auf diese Fähigkeit, die sehr zur Transparenz bei der Nutzung von Standard- und nutzerdefinierten Datentypen beiträgt, wurde hier wegen des Programmieraufwands verzichtet.

Das Hinzufügen einer neuen Textspalte für die Handles der Dokumente wird von dem Shellskript `alter_db` vorgenommen. Das Programm `gen_hd` erzeugt die Handlewerte und trägt sie in die neue Spalte ein. Der komplette Ablauf der Indizierung wird auch hierbei von einem externen Shellskript gesteuert, welches den Indexnamen für die Dokumentensammlung ermittelt.

## 6.3 Suchmaschinen-Module

Die Programmierung der Suchmaschinentreiber stellte die größte Herausforderung dar, denn für diese Treiber gibt es bislang keine Vorbilder. Sie stellen zusammen mit dem Wrapper-Modul die Neuheit dieser Architektur dar. Daher mußte hierbei alles neu konzipiert und programmiert werden. Eine weitere Schwierigkeit besteht darin, daß es keine einheitliche Zugriffsform für Suchmaschinen gibt und deshalb ein einfaches Portieren von einer Lösung auf eine andere Suchmaschine praktisch

unmöglich ist — die beiden hier vorgestellten Treiber könnten unterschiedlicher nicht sein. Diese völlig verschiedenen Ansätze verdeutlichen aber gleichzeitig die Flexibilität der neuen Architektur.

Es ist praktisch möglich, von einer einfachen Indexmaschine bis hin zu einer Volltextdatenbank alles über diese Architektur mit einem ORDBMS zu verbinden. Dies ist zum Testen der verschiedenen Retrieval-Eigenschaften und von neuen Anfragesprachen für Datenbanken mit IR-Qualitäten durchaus schätzenswert. Für eine Anwendungslösung sollte die Suchmaschine allerdings besser an diese Aufgabe angepaßt werden. Das betrifft zum einen den Umfang der Suchmaschine. Auf Funktionen, die nur für den stand-alone-Betrieb nötig sind, sollte verzichtet werden, genauso wie auf die Speicherung der Dokumente oder Dokumentteilen innerhalb der Suchmaschine. Zum anderen sollte die Schnittstelle zum Index möglichst kompakt sein und nur die für die Suche und die Konfiguration notwendige Aufrufe enthalten. Dadurch wird der Kommunikationsaufwand gesenkt und die Retrieval-Performance gesteigert.

### 6.3.1 FULCRUM

Fulcrum scheint durch seine extensive Unterstützung der Textreader und die mitgelieferten Datenbank-Textreader für ODBC-Datenbanksysteme bestens geeignet für den Einsatz mit einem ORDBMS. Auf der anderen Seite bietet Fulcrum auch für den eigenen Zugriff eine ODBC-Anfrageschnittstelle. Das entspricht einem fast idealen System mit ausschließlicher Verwendung von Standardschnittstellen. (Warum nur fast, wird im nächsten Abschnitt genauer erläutert.) Die Abbildung 6.1 verdeutlicht den kompletten Aufbau dieses Systems mit einer Suchmaschine wie Fulcrum. Würden alle Suchmaschinen eine solche standardisierte Anfrageschnittstelle bereitstellen, wäre auch die einfache Portierung der Suchmaschinentreiber möglich.

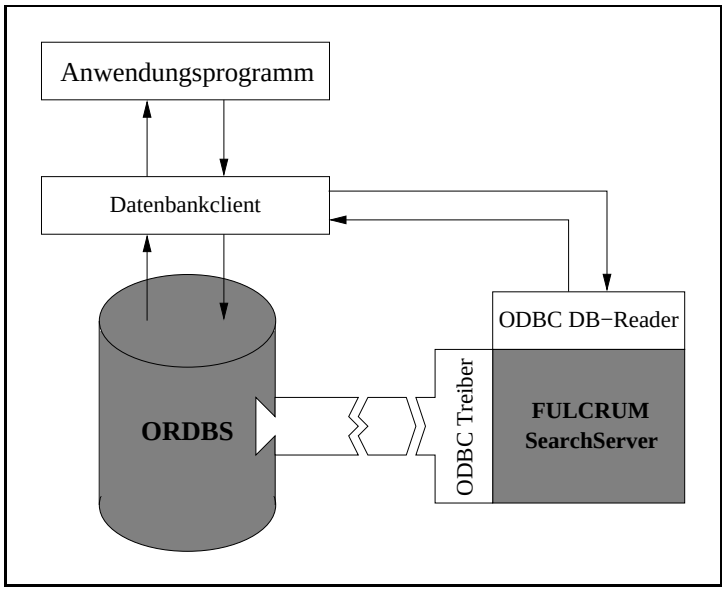


Abbildung 6.1: Textretrieval-Erweiterung mit Fulcrum Suchmaschine

Bei der Nutzung von Fulcrum als Suchmaschine muß darauf geachtet werden, daß die Installation in die Systemverzeichnisse der Datenbank erfolgt. Bei bereits installierten Systemen müssen zumindest die entsprechenden Verzeichnisse und Bibliotheken in die Datenbankverzeichnisse gelinkt werden. Der Grund hierfür ist, daß die Suchmaschine von einer gewissen Verzeichnisstruktur ausgeht, die nur über

Environment-Variablen geändert werden kann. Da die Suchfunktionen aber mit dem Datenbank-Environment aufgerufen werden, stehen nur die Datenbanksystemverzeichnisse und -variablen zur Verfügung.

### Der Suchmaschinentreiber

Leider birgt die Verwendung der ODBC-Schnittstelle für den Zugriff auf die Suchmaschine ein Problem in sich. Diese Schnittstelle wurde für Datenbanken konzipiert und wird von den hier verwendeten Systemen auch unterstützt. Da nun aber die Datenbank (von der die Suchfunktion aufgerufen wird) und die Suchmaschine (die von der Suchfunktion angesprochen werden soll) die gleiche Schnittstelle besitzen, kann das System die Aufrufe nicht mehr eindeutig zuordnen. Grund hierfür ist das dynamischen Laden der entsprechenden Treiberbibliotheken. In diesem Fall werden alle ODBC-Aufrufe der Suchfunktion von der Datenbank ausgewertet anstatt von der Suchmaschine.

Deshalb muß auf die Standardarchitektur für ODBC-Anbindungen von Datenquellen zurückgegriffen werden, um eine eindeutige Zuordnung der ODBC-Aufrufe zu den Datenquellen zu ermöglichen. Die Lösung ist der Einsatz eines ODBC-Treibermanagers (unter Windows-Betriebssystemen das Standardverfahren) und der genauen Angabe der Datenquelle beim Aufruf der ODBC-Funktionen, in diesem Falle also der Fulcrum SearchServer.

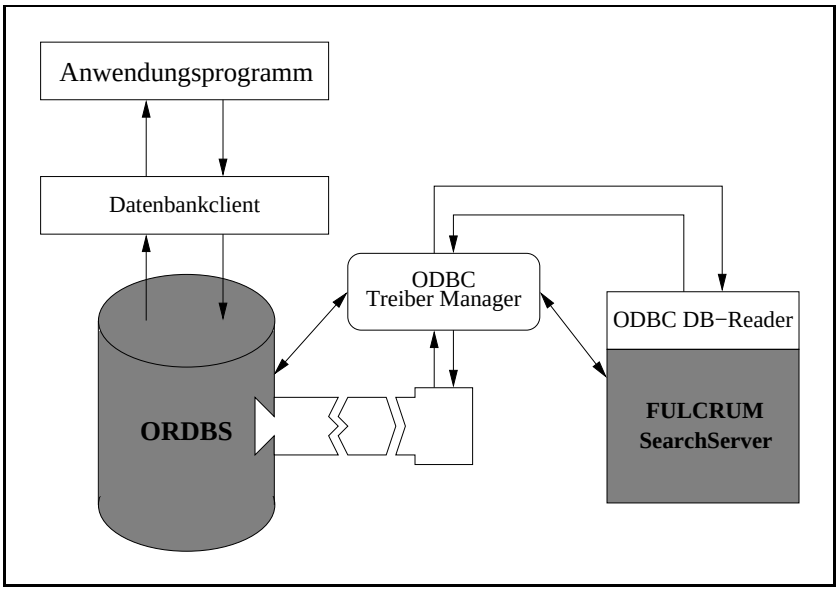


Abbildung 6.2: Textretrieval-Erweiterung mit Fulcrum Suchmaschine
reale Umsetzung mittels ODBC-Treibermanager

In Abbildung 6.2 ist die reale Implementation der Fulcrum Suchmaschine abgebildet. Wie hier zu sehen ist, greifen alle beteiligten Komponenten über den Treibermanager auf die gewünschte Datenquelle zu. Im Falle des Datenbank-Textreaders ist das nicht unbedingt erforderlich, da dieser direkt mit den Datenbankbibliotheken gelinkt werden kann. Da aber der Treibermanager in diesem Falle bereits installiert und dieser Weg unter Windows schon Standard ist, ist es besser, alle Zugriffe einheitlich zu gestalten.

Durch die Verwendung der Datenbankanfrageschnittstelle für die Suchmaschine werden also einige Komplikationen hervorgerufen, die zu dem Schluß gelangen lassen, daß die Suchmaschinen für diese Architektur besser einen eigenen, zumin-

dest optionalen, Schnittstellenstandard besitzen sollten. Neben den Komplikationen entspricht die ODBC-Schnittstelle auch nicht der Forderung nach einer kompakten Schnittstelle für die Kommunikation zwischen Datenbank und Index.

Bei der Implementation des Prototyps wurde ein Beispielprogramm für die Demonstration der Programmierung der SearchServer API genutzt (das Programm heißt `stsample` und liegt der Fulcrum-Distribution bei) und einige darin enthaltenen Routinen direkt für den Suchmaschinentreiber verwendet. Dadurch entfiel eine aufwendigere Einarbeitungszeit in die Programmierung des SearchServers. Es machte außerdem deutlich, daß für den Treiber die Anfrageroutinen von Standardanwendungen ausreichend sind und keine aufwendige Neuentwicklung notwendig ist.

### Der Datenbank-Textreader

Der Fulcrum SearchServer ist standardmäßig für die Integration von Datenbanken als Dokumentenquelle vorbereitet (siehe [SSD98]). Dem System liegt für die Windows-Betriebssystemumgebung ein ODBC-Datenbank-Reader bei, mit dessen Hilfe die Anbindung ohne große Schwierigkeiten möglich ist. Da der Prototyp unter einem UNIX-Betriebssystem entwickelt wurde, mußte allerdings ein eigener Reader programmiert werden.

Zu Demonstrationszwecken und zur Entwicklung von eigenen DB-Readern für Datenbanksysteme, die kein ODBC unterstützen, bietet die Fulcrum Distribution einen Beispiel-ODBC-Datenbank-Reader im Quelltext an. Dieser ist zwar auch für den Betrieb unter Windows gedacht, läßt sich aber mühelos auf UNIX portieren. Dieses Beispielprogramm ist zwar nicht so leistungsfähig wie der mitgelieferte Datenbank-Textreader, enthält aber alle nötigen Funktionen für das Auslesen von Dokumenten aus einer Tabelle in einer relationalen Datenbank. Die meisten zusätzlichen Funktionen, wie das JOIN von Tabellen und Zugriff auf VIEWs, werden in dieser Rolle der Suchmaschine nicht benötigt. Schließlich handelt es sich hier um die Simulation eines Datenbankindexes und diese sind nur auf Spalten von realen Tabellen definiert (abgesehen von neueren Formen der Indexe, wie *materialized views* o.ä.).

Zur Nutzung des Datenbank-Readers wird eine Steuerdatei benötigt, die angibt, aus welcher Spalte, welcher Tabelle und welcher Datenbank die Dokumente in welche interne Tabelle des SearchServers geladen werden sollen. Diese Datei wird in [SSD98] detailliert beschrieben und muß von dem Steuerprogramm der Textretrieval-Erweiterung, das zum Anlegen des Indexes dient, mit den richtigen Daten gefüllt und dem SearchServer zugänglich gemacht werden. Für jede Textspalte, die mit dem System indiziert wird, muß eine eigene Steuerdatei erstellt werden.

### Probleme des Fulcrum SearchServers

Bei der aufwendigen Programmierung der Such- und Anfragefunktionen über einen ODBC-Treibermanager wurde eine Eigenart des SearchServers festgestellt, die das korrekte Betreiben der kompletten Textretrieval-Erweiterung vereitelte.

Aus unbekannten Gründen greift der SearchServer bei einer Suchanfrage immer auf die originalen Dokumente zu. In diesem Fall befinden diese sich in der Datenbank, was bedeutet, daß über den Datenbank-Reader wieder ein Zugriff auf die Datenbank getätigt wird. Bei einer "normalen" Anfrage aus einem externen Programm stellt dies kein Problem dar (getestet durch stand-alone Suchfunktion). Da die Anfrage im Falle der Textretrieval-Erweiterung aber aus der Datenbankumgebung selbst und insbesondere von einer Funktion, die auf den angeforderten Dokumenten arbeitet, aufgerufen wird, kommt es anscheinend zu einem *Deadlock* —

die Anfrage hängt. Ob der Grund bei den verschachtelten Treiberaufrufen über den ODBC-Treibermanager oder bei Anfragebearbeitung der Datenbank mit nutzerdefinierten Funktionen liegt, konnte bisher nicht ermittelt werden.

Um dennoch einen Test der Suchmaschine und Anfragesprache durchführen zu können, wurden die Dokumente vor der Indizierung in ein Datenverzeichnis exportiert und diese dann vom SearchServer indiziert. Damit funktioniert die Suchanfrage an die Datenbank per SQL-Anweisung problemlos, allerdings bedeutet dies eine Doppelspeicherung der Dokumente und ist für ein Endanwendungsszenario nicht geeignet.

### 6.3.2 MG System

Das MG System ist durch die Verfügbarkeit der Quellen die ideale Testplattform für verschiedene Anbindungsvarianten. In seiner Grundausstattung besteht es nur aus einer Reihe von kleinen Programmen und Shellskripten und scheint weniger geeignet als Suchmaschine in einer Textretrieval-Erweiterung. Doch auch diese Form einer Suchmaschine kann mit der neuen Architektur genutzt werden.

Beim MG System läuft nicht ständig ein Server, der die Suchanfragen von einer Anwendung entgegennimmt, wie etwa bei der Textsuchmaschine von IBM. Das Suchprogramm wird hier für jede Suche neugestartet. Das spart zum einen Ressourcen, steigert zum anderen aber auch den Zeitaufwand bei einer Suchanfrage. Für Tests der Retrieval-Eigenschaften ist das aber akzeptabel und durch die Verfügbarkeit der Quellen kann jederzeit eine Variante entwickelt werden, die ständig aktiv ist und so nicht jedesmal nachgeladen werden muß.

Ein weiterer ungewöhnlicher Punkt ist die Anfrageschnittstelle des Systems. Hierbei handelt es sich nicht um eine Funktionsbibliothek, sondern um ein Nutzerinterface mit direkten oder indirekten (über eine Eingabedatei) Eingaben — gemeint ist das Programm `mgquery`. Auch wenn es ungewöhnlich ist, so reicht auch diese Art der Schnittstelle für eine schnelle Entwicklung und Anfragetests aus. Für eine Endanwenderlösung sollte allerdings das Programm `mgquery` in eine Programmierschnittstellen-Bibliothek umgewandelt werden.

#### Probleme des MG Systems

Das MG System ist in erster Linie zur Verwaltung einer statischen Dokumentensammlung, z.B. auf einer CD-ROM, gedacht. Dabei werden die Dokumente in einer eigenen "Datenbank" komprimiert abgespeichert und die originalen Dokumente werden nicht mehr benötigt. Konkret bedeutet das, daß keine Referenzen auf Dokumente außerhalb des Systems abgespeichert werden, sondern die Dokumente selbst. Eine Anfrage an das System liefert daher auch den konkreten Text oder Textausschnitt aus der Datenbank.

Diese Eigenschaft ist für den Einsatz als Index in einem Datenbanksystem sehr ungeeignet, da zum einen die Dokumente doppelt gespeichert werden und zum anderen keine Identifikatoren zu den Dokumenten gespeichert werden können, die zur Erstellung einer Ergebnisliste bei einer Suchanfrage notwendig sind. Für eine praktische Nutzung von MG als Suchmaschine für diese Architektur ist daher eine Umprogrammierung der Datenspeicherung dringend anzuraten, wodurch sowohl die Größe des erzeugten Indexes verringert als auch die Suchgeschwindigkeit deutlich erhöht wird.

Aus Gründen des erforderlichen Programmieraufwands und um die Nutzung beliebiger Suchmaschinen ohne Eingriff in ihren Programmcode über die gebotenen Schnittstellen zu demonstrieren, wurde hier allerdings auf diese Anpassung verzichtet und die Identifikatorenspeicherung extern vorgenommen.

Jedes gespeicherte Dokument erhält eine interne Nummer, die in aufsteigender Reihenfolge für jedes neueingelesene Dokument vergeben wird. Bei der Indexerzeugung wird zusätzlich zum eigentlichen Index noch eine weitere Datei mit der Zuordnung der internen Dokumentennummer zu den Dokument-Handles in der Datenbank gespeichert. Die Suchanfrage wird in zwei Abschnitte unterteilt: die eigentliche Suchanfrage an MG und das Mapping der Dokumentidentifikatoren. Diese Aufgaben werden von den beiden Shellskripten `rn_mgquery` und `rn_get_id` erledigt.

### Der Suchmaschinentreiber

Durch die ungewöhnliche Anfrageschnittstelle von MG mußte der Suchmaschinentreiber etwas anders gestaltet werden als allgemein üblich. Für jede Anfrageroutine existiert ein kurzes Shellskript (zur Zeit nur `db2_mgquery` für die `contains()`-Funktion), das die eigentliche Suche durchführt. Diese Shellskripte werden von den zugehörigen Funktionen des Treibers mittels des `System()`-Kommandos aufgerufen. Damit der Aufruf der Skripte problemlos erfolgen kann, müssen sie in einem erreichbaren Pfad stehen, am besten also wieder in einem Systempfad der Datenbank.

Wie im letzten Abschnitt schon angesprochen, liefert MG von sich aus nicht die richtigen Dokumentidentifikatoren für den Vergleich in der Datenbankfunktion. Aus diesem Grund ruft `db2_mgquery` die beiden Skripte `rn_mgquery` und `rn_get_id` nacheinander auf und erhält dadurch die richtigen Handles aus der Datenbank.

Die Parameterübergabe erfolgt auf zwei verschiedene Arten. Die Suchparameter, also Indexname und Suchargument, werden dem Shellskript direkt als Kommandozeilenparameter übergeben. Da das Ergebnis eines Skripts nur ein Fehlercode in Form eines numerischen Wertes ist, wird die Ergebnisliste über eine temporäre Datei zurück geliefert. Die Treiberroutinen lesen die Daten aus der Datei, übertragen sie in die interne Struktur der Ergebnisliste und übergeben diese abschließend an die jeweilige Datenbankfunktion.

### Der Datenbank-Textreader

Das MG System ist sehr modular aufgebaut und besitzt deshalb eine spezielle Textreader-Komponente (siehe auch Abschnitt 2.2.2). Der standardmäßige Textreader ist ein Shellskript, das Dokumente aus einer zusammenhängenden Datei, z.B. einer Maildatei, extrahieren oder aus verschiedenen Ordnern und Unterordnern zusammensammeln kann. Er kann allerdings keine Dokumente aus einer Datenbank auslesen.

Aber die Schnittstelle zwischen Textreader und Suchmaschine ist entsprechend dokumentiert, so daß die Implementation eines Datenbank-Textreaders einfach möglich ist. Dieser Datenbank-Reader kann zum einen eine ODBC-Anwendung sein, was den Aufwand der ersten Implementation erhöht, aber danach für jede ODBC-Datenbank wiederverwendet werden kann — ähnlich dem Fulcrum Datenbank-Reader. Eine andere und schneller zu implementierende Möglichkeit ist der Zugriff über die direkte Datenbankschnittstelle (z.B. per Shellskript). Diese Skripte müssen dann aber an jede Datenbank angepaßt werden. Der ersten Methode ist der Vorzug zu geben, wenn es sich um eine Endanwendung des Systems handeln soll, da man das letztendlich verwendete DBMS nicht kennt und es die kompatibelste Lösung ist. Die zweite Methode ist für das schnelle Prototyping von Testsystemen sehr brauchbar, da der Entwicklungsaufwand unter einer halben Stunde liegt und somit auch noch für mehrere Datenbanken vertretbar ist.

Die zweite Methode wurde in einer etwas abgewandten Form auch für diesen Prototypen genutzt. Der Textreader (`mg_get`) kann nur komplett ausgetauscht oder aber dessen Programmtext erweitert werden. Eine schon weiter oben angesprochene Forderung dieser Prototypen war die Verwendung der vorhandenen Systeme ohne

Eingriff in deren Quellcode. Um die bisherige Funktionalität des MG Systems für andere Anwendungen nicht zu beeinflussen, wurde die Datenbankfähigkeit als Aufsatz für `mg_get` entwickelt. Dazu werden die Dokumente aus der Datenbank ausgelesen und als Dokumentensammlung im späteren Indexverzeichnis abgelegt. Danach wird ein entsprechender Eintrag in die Steuerdatei des Textreaders (`.mg_getrc`) geschrieben und somit die Sammlung und dessen Typ angemeldet. Die erste Aufgabe erledigt das Shellskript `rn_export_tbl`, welches gleichzeitig auch die Mapping-Datei für die Dokument-Handles in der Datenbank und im MG System erzeugt. Die Anmeldung übernimmt `rn_mggetrc_add`.

Die Anmeldung der Datenquelle geschieht nur einmal beim Anlegen des Indexes. Wird die zugehörige Textspalte wieder durch die Textretrieval-Erweiterung deaktiviert, so sollte die Datenquelle auch bei MG wieder abgemeldet werden. Das Skript `rn_export_tbl` muß dagegen bei jeder Indexerzeugung und jedem Update wieder aufgerufen werden, um die aktualisierten Dokumente für die Indizierung bereitzustellen.

## 6.4 Konkrete Kopplungen

Bisher wurden die einzelnen Erweiterungsmodule für die verschiedenen Produkte vorgestellt. Zu einer Textretrieval-Erweiterung gehört aber noch etwas mehr. So muß der Administrator die Möglichkeit besitzen, eine Textspalte für den Index zu aktivieren und die eigentliche Indizierung durch die Suchmaschine zu starten. Wie bei der Vorstellung der Suchmaschinen bereits angesprochen, ist die Verfahrensweise zum Indizieren von Dokumenten bei den Systemen sehr unterschiedlich.

Um auch diesen Prozeß zu vereinheitlichen, wurde ein neuer Aufruf eingeführt: `crtindex`. Dieses Kommando entspricht in etwa dem `ENABLE TEXT COLUMN`-Befehl des Text-Extenders von DB2 UDB. Nach dem Aufruf dieses Programms mit den entsprechenden Parametern zur Identifizierung der Textspalte und der neuen Handlespalte ist das System bereit, Suchanfragen auf diese Textspalte zu beantworten. In Abhängigkeit des Umfangs der Dokumentensammlung muß eine gewisse Zeit zur Erstellung der Indexdatei eingerechnet werden.

### Das Steuerprogramm

Hinter dem einheitlichen Kommando zum Erzeugen eines Indexes auf einer Textspalte steht in der prototypischen Implementation wieder ein Shellskript namens `crtindex`. Dieses Skript ist das schon öfter erwähnte Steuerprogramm für den Ablauf des Indizierungsprozesses. Es arbeitet im wesentlichen die folgenden Aufgaben ab:

1. Überprüfung der nötigen Environment-Einstellungen
2. Ermittlung eines neuen, eindeutigen Indexnamens
3. Hinzufügen der Handlespalte zu der Tabelle
4. Berechnung der Handlewerte und Eintragen in die Handlespalte
5. Vorbereitung und Starten der Textindizierung

Damit das Steuerprogramm nicht für jede Kombination von Datenbank und Suchmaschine neu geschrieben werden muß, ruft es für jeden Punkt ein zugehöriges Programm auf, welches von der jeweiligen Erweiterung bereitgestellt werden muß.

Der erste Punkt ist unkompliziert und für die meisten Systeme bis auf die Namen der Variablen gleich. Beide Erweiterungen (datenbank- und suchmaschinenseitig) müssen jeweils ein Testprogramm für die von ihnen benutzten Environment-Variablen bereitstellen. Die Programme heißen `chk_env_db` bzw. `chk_env_se` und liefern als Ergebnis 0, wenn alle Variablen gesetzt sind, ansonsten 1.

Für die Zuweisung eines Namens für den neuen Index ist der Programmierer der Suchmaschinentreiber verantwortlich. Mit Hilfe des Programms `get_idx_name` wird ein eindeutiger Name für den anzulegenden Index erzeugt. Für diese Architektur haben die Indexnamen die Form `IX000000`, wobei die Nullen eine sechs stellige Nummer darstellen, die mit jedem neuen Index inkrementiert wird. Dies bedeutet, es sind eine Million verschiedener Indexe gleichzeitig möglich bzw. nach einer Million erzeugten und evtl. wieder gelöschten Indexen muß über eine Methode nachgedacht werden, wie nicht mehr verwendete Nummern herausgefiltert und neu vergeben werden können. Diese Werte dürften in der Praxis durchaus ausreichend sein (obwohl man nie weiß, wie lange solch eine Behauptung in der schnellebigen Computerwelt Bestand hat). Die Einschränkung auf insgesamt acht Zeichen wurde mit Rücksicht auf Filesysteme gewählt, die keine längeren Dateinamen erlauben, da die Indexdateien meist unter diesem Namen auf der Platte abgespeichert werden. Im übrigen entspricht diese Namensgebung der des Text-Extenders von DB2 UDB. Auch wenn mehr als acht Zeichen unterstützt werden, darf der Indexname nicht länger sein, da nicht mehr Platz in den Dokument-Handles reserviert ist.

Die Punkte 3 und 4 werden von der verwendeten Datenbankerweiterung erledigt. Sie stellen für das Aktivieren von Textspalten zwei Standardaufrufe bereit: `alter_db` und `gen_hd` (siehe Abschnitt 6.2). Diese Programme werden nacheinander aufgerufen; damit ist der Index datenbankseitig vorbereitet. Bei der Erläuterung von `gen_hd` wurde festgestellt, daß zum Erzeugen der Handlewerte der Indexname benötigt wird, da er in die Handles einkodiert wird. Zu diesem Zeitpunkt ist der Indexname bereits ermittelt worden (Punkt 2) und kann dem Programm als Parameter übergeben werden.

Der letzte Punkt fällt wieder in den Aufgabenbereich der Suchmaschinentreiber. Dazu muß ein Programm namens `create_idx` bereitgestellt werdem, das alle noch notwendigen Schritte für das Erzeugen des Indexes abarbeitet. Nach der Ausführung dieses Programms ist die Indizierung der Texte abgeschlossen und das Steuerprogramm ist beendet.

Da die Prozedur zum Indizieren von Dokumenten von der verwendeten Suchmaschine abhängig ist und daher sehr stark variiert, werden die einzelnen Vorgänge für die getesteten Suchmaschinen in den folgenden Abschnitten genauer aufgeschlüsselt.

### Fulcrum SearchServer

Die Prozedur zum Indizieren von Texten aus einer Datenbank ist beim SearchServer sehr komplex (Hinweise hierzu findet man in [SSD98]). Konkret sind folgende Schritte abzuarbeiten:

1. Erstellen einer passenden Steuerdatei für den Datenbank-Reader (DID-Datei)
2. Erstellen einer neuen Tabelle für die Textdokumente und deren Handles
3. Eintragen entsprechender Steuerkommandos in die Tabelle
4. Starten des Indizierungsprozesses

Für Punkt 1 muß eine sogenannte ***D**atabase **I**nterface **D**efinition*-Datei (DID) erstellt werden. Hierin werden die Beziehungen zwischen den Tabellen des SearchServers und der Datenbank beschrieben. (SearchServer verwaltet die Dokumente und Zusatzinformationen ebenfalls in Tabellen, siehe [SSI98].) Mit Hilfe dieser Datei

wird der Zugriff des Datenbank-Readers auf den Inhalt einer Datenbank gesteuert. Im vorliegenden Fall wird nur ein Bruchteil der Fähigkeiten des Datenbank-Readers benötigt, da es sich um einen einfachen Zugriff auf die Dokumenten- und Handlespalte handelt. Es existiert eine Art Template für eine einfache DID-Datei, das von dem Steuerprogramm mit den richtigen Werten für den Namen der Datenbank, Tabelle, Dokumenten- und Handlespalte gefüllt werden muß und dann unter dem Namen des neuen Indexes abgespeichert wird.

Alle anderen Schritte werden durch *SearchSQL*-Kommandos (Anfragesprache des SearchServers, siehe [SSS98]) in einem *ExecSQL*-Skript ausgeführt. Auch für dieses Skript existiert ein Template mit den Standardaufrufen, in das wieder die entsprechenden Werte eingetragen werden müssen. Nachdem das Kommandoskript erstellt wurde, wird es im letzten Schritt des Steuerprogramms für Fulcrum mittels `execsql` ausgeführt. Das Skript wird nur einmal zur Erzeugung der Tabellen und zum Starten des Indizierungsprozesses benötigt und wird deshalb nicht permanent mit dem Index gespeichert. Das Erneuern des Indexes erfolgt bei Fulcrum entweder automatisch zu festgelegten Zeiten oder aber durch einen expliziten SearchSQL-Aufruf.

Mit dem Ausführen des *ExecSQL*-Skripts ist die Indexerzeugung für die neue Textretrieval-Erweiterung mit dem Fulcrum SearchServer als Suchmaschine abgeschlossen. Die Templates des Steuerprogramms werden im Abschnitt A.3 im Detail vorgestellt.

#### MG System

Bei der Nutzung von MG als Suchmaschine ist der Aufwand für das Steuerprogramm geringer. Im Abschnitt 6.3.2 wurden die Schritte zur Erzeugung eines Indexes im Unterabschnitt *Datenbank-Reader* schon erläutert. Konkret handelt es sich hierbei um das:

1. Exportieren der Dokumente in ein Unterverzeichnis des Indexes
2. Eintragen der neuen Dokumentensammlung in die Konfigurationsdatei
3. Starten der Indizierung mit dem Befehl `mgbuild`
4. Löschen der originalen Dokumente aus dem Indexverzeichnis

Das Steuerprogramm muß die einzelnen Punkte abarbeiten. Für jede Aufgabe stehen allerdings entsprechende Shellskripte bereit, die einfach mit den richtigen Parametern aufgerufen werden müssen. Hierbei werden also keine vorgefertigten Templates benötigt. Das Steuerprogramm ruft nacheinander die entsprechenden Shellskripte auf: `rn_export_tbl` (Punkt 1), `rn_mggetrc_add` (Punkt 2) und `mgbuild` (Punkt 3). Damit ist die Indexerzeugung abgeschlossen.

Da MG die Dokumente in einer Art internen Datenbank komprimiert abspeichert, werden die exportierten Dokumente nicht mehr benötigt und belegen nur unnötig Speicher (dreifache Speicherung der Dokumente!). Deshalb werden sie zum Abschluß wieder gelöscht. Dabei muß allerdings darauf geachtet werden, daß die Erzeugung des Indexes eine gewisse Zeit benötigt und die Dokumente nicht zu früh, also noch während des Indizierungsprozesses entfernt werden. Deshalb wird erst auf die Beendigung des Programms `mgbuild` gewartet.

## 6.5 Zusammenfassung

Die prototypische Implementierung der in dieser Arbeit entwickelten Architektur diente zum einen dem Beweis der Anwendungsfähigkeit und war gleichzeitig wichtiger Bestandteil der Erarbeitung einiger Lösungsansätze.

Wie in diesem Kapitel deutlich wurde, lag das Augenmerk nicht auf der Programmierung einer vollständigen Textretrieval-Erweiterung mit einfach austauschbaren Suchmaschinen. Obwohl dies das Ziel für eine Endanwendung dieser Architektur ist, wurde hier mehr Wert auf die Einfachheit der Implementation gelegt. Das Hauptziel bestand darin, vorhandene ORDBMS und Suchmaschinen ohne Beeinträchtigung deren bisherigen Funktion und möglichst ohne Veränderung des Programmcodes so miteinander zu verbinden, daß in der Datenbank gespeicherte Dokumente direkt indiziert werden können und vor allem, eine inhaltsbasierte Suche in diesen Dokumenten direkt von der Datenbank mit dessen Anfragesprache zu ermöglichen.

Bei der Programmierung wurde nicht immer die effizienteste Lösung in Hinblick auf die Performance gewählt. Aber wie früher schon einmal erwähnt wurde, erreicht man allein durch die Verwendung eines externen Indexes einen beachtlichen Performancegewinn. Verbesserungen beim Aufruf der verschiedenen Module und effizientere Parameterübergabe erzielen nur noch eine geringe Performancesteigerung, selbst bei der Lösung für das MG System. Obwohl hier die meiste Arbeit von Shellskripten erledigt wird, ist das Gesamtsystem durchaus vergleichbar schnell in der Beantwortung der Suchanfragen, wie z.B. die originalen Texterweiterungen der verschiedenen Datenbanksysteme.

Der Grund für den geringen Einfluß der Effizienz der einzelnen Module ist die Häufigkeit des Aufrufs. Bei einer Suchanfrage über die Datenbank wird jedes Modul nur genau einmal aufgerufen, um die Suchanfrage an die Suchmaschine weiterzuleiten und die Ergebnisse zurück an die Datenbank zu liefern. Dabei ist es egal, wieviele Dokumente sich bei einer Suchanfrage qualifizieren. Da nur relativ kleine Handlewerte übermittelt werden anstatt der kompletten Dokumente, bleibt der Umfang der Ergebnisliste in einem vertretbaren Rahmen. Je größer also die Datenmenge ist, in der gesucht wird, und um so mehr Dokumente sich qualifizieren und damit aus der Datenbank geladen werden müssen, desto geringer wird der Anteil der Datenübermittlung durch die Module an der Gesamtbearbeitungszeit einer Suchanfrage.

Einen viel entscheidenderen Einfluß auf die Retrieval-Geschwindigkeit hat zum einen die Performance der verwendeten Suchmaschine und zum anderen die Ausnutzung der Ergebnisliste bei der Anfragebearbeitung in der Datenbank. Der erste Punkt ist offensichtlich, hat aber ebenso nur einen linearen Einfluß auf die Suchgeschwindigkeit. Eine größere Bedeutung kommt hier der Datenbank zu. Die hier verwendete Variante der Einbindung der Ergebnisliste ist zwar die einfachste, aber auch langsamste. Vor allem bei einer großen Anzahl von Dokumenten macht es sich bemerkbar, daß alle aus der Datenbank geladen werden müssen, um dann deren Handle in der Ergebnisliste zu suchen. Damit ist zwar praktisch ein Index auf den Dokumenten vorhanden, dieser wird aber in der Datenbank nicht als solcher verwendet, sondern als Filter bei der Ausgabe der Ergebnisse nach einem Tablescan. Deutlich günstiger ist hier die direkte Nutzung der Ergebnisliste im Indexmanager. Diese Verbesserungen sind allerdings unabhängig von der hier vorgestellten Architektur und wurden deshalb nicht weiter untersucht. Einen genauen Einblick in die Möglichkeiten der Verbesserung von Anfrage und Performance bei Textretrieval-Erweiterungen erhält man in [DM97, Nit99, WFC98].

# Kapitel 7

# Schlußbetrachtung

In dieser Arbeit wurde nach einer Möglichkeit gesucht, eine Brücke zwischen zwei Lagern von Systemen zur Verwaltung von multimedialen Dokumenten zu schlagen — den modernen Datenbanksystemen und den Information Retrieval-Werkzeugen. Da die etablierten Produkte jeweils nur einen Teil der gewünschten Funktionalität bieten, die für eine Kombination von Dokumentverwaltung und Information Retrieval nötig ist, sollten durch eine Kopplung der Systeme aus beiden Lagern alle Funktionen verfügbar gemacht werden.

Die Hersteller von IR-Werkzeugen haben die Techniken und Indexstrukturen immer weiter verbessern können und bieten heutzutage hochspezialisierte Systeme zur Indizierung und Suche bestimmter Datentypen, wie z.B. Texte, Bilder, Audio und Video, an. Die Datenbankhersteller versuchen durch die Integration bekannter Indexmechanismen in ihre Systeme bzw. die Möglichkeit, ihre Systeme um neue Indexmechanismen erweitern zu können, den Vorsprung der spezialisierten Werkzeuge zu verringern. Durch die neuen Datentypen der ORDBMS können die multimedialen Dokumente zwar gespeichert werden, aber der Zugriff auf diese Dokumente und die inhaltsbasierte Suche stellen ein Problem für die Datenbanken dar.

Der bisherige Ansatz zur Lösung dieses Problems war die Suche nach neuen Varianten, wie die von den IR-Werkzeugen gewonnenen Erfahrungen und Methoden (speziell neue Indexformen) in die DBMS integriert werden können (siehe [Aok91, DDS95, GFH97, LS88]). Wie aber schon in [DM97, Nit99] festgestellt wurde, sind diese Ansätze meist sehr aufwendig und beinhalten einige Einschränkungen, z.B. wenn hochspezialisierte Indexstrukturen auf relationale Tabellen abgebildet werden. Daher wurde hier ein Ansatz untersucht, der beide Systeme einbezieht. Durch eine Aufgabenverteilung verrichtet jede Anwendung den Teil der Arbeit, auf den sie spezialisiert ist — Datenbanken übernehmen Dokumentspeicherung und Zugriffssteuerung, IR-Werkzeuge sind für Indizierung und Suche zuständig. Durch eine offene Architektur wird erreicht, daß die verwendeten Produkte austauschbar sind, um das Gesamtsystem flexibel an die jeweiligen Aufgaben anpassen zu können oder einfach ein neues, besseres Produkt zu nutzen.

In der Arbeit wurden existierenden IR-Werkzeuge, ORDBMS mit ihren entsprechenden Texterweiterungen sowie bereits umgesetzte Kopplungsvarianten gründlich analysiert. Darauf aufbauend konnte eine neue Architektur nach den hier aufgestellten Anforderungen entwickelt werden, welche durch Beispielimplementationen getestet und verbessert wurde. Durch die dabei entstandenen Prototypen konnten gleichzeitig Erfahrungen mit den verwendeten Produkten gesammelt und die Funktionstüchtigkeit der Gesamtarchitektur unter Beweis gestellt werden. Im folgenden sollen nun die Ergebnisse der Arbeit zusammengefaßt und ein Ausblick auf mögliche Erweiterungen gegeben werden.

## 7.1 Ergebnisse

Das wichtigste Ergebnis dieser Untersuchungen ist, daß es durchaus möglich ist, die Fähigkeiten der verschiedenen Dokumentverwaltungs- und -Retrieval-Programme miteinander zu vereinen. In Zukunft sollten bei einer Evaluierung von Produkten zur Dokumentverwaltung die beiden Bereiche (Datenhaltung und Daten-Retrieval) getrennt betrachtet und die entsprechenden Produkte auf die jeweiligen Anforderungen hin überprüft werden. Aussagen wie aus [Tit99], daß eine Kombination aus dem speziellen IR-Werkzeug zur Indizierung und Suche und diesem speziellen Datenbanksystem zur Dokumentspeicherung und Zugriffssteuerung die beste Lösung darstellt, sollten also zum Regelfall werden und nicht nur ein Wunschsystem mit den kombinierten Eigenschaften beschreiben.

Natürlich kann eine Kopplungsarchitektur von zwei selbständigen Programmen nicht die Performance oder ausgereifte Integration aller Funktionen bieten, wie es ein speziell nach einem solchen Anforderungskatalog entwickeltes System vermag. Dafür bietet sie aber die Vorteile modularer Systeme, wie Anpassung des Systems an spezielle Aufgaben und einfache Austauschbarkeit und Erweiterbarkeit der verwendeten Komponenten. Der große Vorteil dieser Architektur ist der minimale Implementierungsaufwand, da die wichtigsten Module bereits als fertige Produkte vorhanden sind. Anhand der Prototypen konnte gezeigt werden, daß die IR-Werkzeuge und ORDBMS, so wie sie heute bereits existieren, einfach in diese Architektur eingebunden werden können. Darüber hinaus kann durch zukünftige Verbesserungen bzw. Anpassungen an die Architektur eine weitere Steigerung der Performance und Benutzbarkeit erzielt werden.

Auch wenn die Prototypen der Architektur zeigen, daß praktisch alle Gruppen von IR-Werkzeugen (Indexmaschinen, Suchmaschinen und Volltextdatenbanken) ohne Veränderungen am System verwendet werden können, haben sich doch Probleme gezeigt, die einige Produkte für eine Endanwendung in der Architektur weniger geeignet machen. Vor allem Volltextdatenbanken sind zu groß und schwerfällig, um effektiv eingesetzt zu werden. Sie beinhalten zu viel Funktionalität, die in dieser Umgebung nicht benötigt wird. Größtes Problem ist hier wohl die doppelte Datenhaltung.

Obwohl MG nicht zu den Volltextdatenbanken gehört, sondern zu der am besten geeigneten Gruppe, den Indexmaschinen, werden auch hier die Dokumente doppelt abgespeichert. Trotzdem ist der Speicherverbrauch des Indexes mit den Dokumenten kleiner als bei den meisten Indexen der heutigen Textretrieval-Erweiterungen der ORDBMS. Bei einigen Indizierungstechniken wird außerdem der eigentliche Text für die endgültige Lokalisierung des Suchbegriffs benötigt, so daß sich diese Doppelspeicherung kaum vermeiden läßt. In diesem Fall sollten die bei MG vorgestellten Komprimierungsverfahren genutzt werden, um Speicherplatz zu sparen. Ein weiterer Punkt, der durch das MG System aufgeworfen wurde, ist die Zuweisung von Dokumentidentifikatoren durch den Nutzer. Diese müssen unbedingt von dem verwendeten IR-Werkzeug unterstützt werden, da sie die einzige semantische Verbindung zwischen den Daten in der Datenbank und denen in der Suchmaschine sind. Durch die offenen Quellen des MG Systems läßt sich diese Funktionalität nachträglich hinzufügen. Generell ist das offene System von MG gut dazu geeignet, neue Techniken und engere Ankopplungsvarianten zu testen.

Der Fulcrum SearchServer bietet zwar mehr Funktionalität bei der Datenverwaltung, ist aber bei Problemen, wie dem Zugriff auf die Datenbank während der Suche, nicht so einfach anzupassen. Dafür lassen sich Identifikatoren direkt von der Datenbank übernehmen und alle Zugriffe erfolgen über standardisierte und gut dokumentierte Schnittstellen. Die genaue Art der Indizierung und Suche ist nicht bekannt, aber für den Fall, daß der Zugriff auf die originalen Dokumente wirklich notwendig ist, kann die Lösung des MG Systems adaptiert werden. Die Dokumente

werden aus der Datenbank extrahiert, komprimiert und über einen Textreader mit einem Online-Entpacker vom SearchServer indiziert. Wie man hieran sehen kann, ist die neue Architektur offen für die verschiedensten Arten des Datenaustauschs.

Die in dieser Arbeit entwickelte Architektur erfüllt alle zu Beginn gesteckten Ziele. Bestehende Produkte werden genutzt und gemäß ihrer Spezialisierung im Gesamtsystem eingesetzt. Die Lösung präsentiert sich als ein erweitertes Datenbanksystem, da alle Anfragen und der Datenaustausch über eine Datenbank abgewickelt werden. Die Suchmaschine verrichtet ihre Arbeit unbemerkt vom Nutzer. Suchanfragen werden vom Datenbanksystem an die Suchmaschine weitergereicht und die Ergebnisse in der Anfragebearbeitung der Datenbank berücksichtigt. Die Architektur stellt gleichzeitig eine Lösung für die Erweiterung von ORDBMS um neue Zugriffsmethoden dar, obwohl diese Methoden nicht direkt in die Datenbank integriert werden. Es kann auf spezialisierte Programme zurückgegriffen werden, wodurch der Implementierungsaufwand gesenkt wird und die Effektivität der verschiedenen Indizierungstechniken und -strukturen unbeeinflußt bleibt.

## 7.2 Ausblick

Die hier vorgestellte Architektur und die implementierten Prototypen stellen nur einen ersten Schritt dar. Es konnte gezeigt werden, wie einfach eine Kopplung der Systeme möglich ist. Dadurch soll Interesse an weiteren Testimplementationen mit neuen Systemen und an Verbesserungen bestehender Prototypen geweckt werden.

Eine Verbesserung der bestehenden Implementationen ist in vielen Bereichen möglich. Die wichtigste Erweiterung betrifft die Einbindung der Suchergebnisse in die Anfragebearbeitung des Datenbanksystems. In [Nit99] wird dazu eine Möglichkeit unter Verwendung sogenannter Indexerweiterungen beschrieben. Solch eine Erweiterung bietet z.B. DB2 UDB ab der Version 7.1 an. Auch mit Informix kann diese Variante angewendet werden, da die DataBlade-Technologie praktisch eine Zusammenfassung aller Erweiterungen des Datenbankkerns ist und somit auch einen erweiterbaren Index umfaßt. Durch die Nutzung der erweiterbaren Indexe wird die Integration der Suchfunktionalität in die Datenbankanfragesprache noch transparenter und die Anfrageauswertung durch das Datenbanksystem wesentlich effizienter. Eine Anpassung an diese Form der Datenbankankopplung ist ohne großen Aufwand möglich, da die bereits implementierten Funktionen weiter- bzw. wiederverwendet werden können.

Eine andere Möglichkeit betrifft weniger die Geschwindigkeit des Retrievals, als vielmehr dessen Funktionsumfang. Durch eine Erweiterung der Architektur zu einer Art Brokersystem ist es möglich, mehrere Suchmaschinen gleichzeitig an eine Datenbank zu koppeln. (Eine Suchmaschine mit verschiedenen Datenbanken zu nutzen, ist bereits jetzt möglich.) Dadurch kann zum einen die Retrieval-Qualität verbessert werden, oder aber der Umfang der Retrieval-Methoden (z.B. Fuzzy-Suche, exakte Suche, natürlichsprachliche Suche). Der letzte Schritt ist dann die Verwendung verschiedener Suchmaschinen in einer Erweiterungsarchitektur. So können neben Texten auch Bilder, Audio, Video oder Geospatial-Daten indiziert und durchsucht werden. Dieser Ansatz wird z.B. in [LH00] weiter verfolgt.

Für Testimplementationen und Prototypen dieser Architektur eigenen sich im besonderen Suchmaschinen, deren Quellen frei verfügbar sind, wie z.B. freeWAIS, MG System oder Harvest. Diese Systeme können beliebig angepaßt werden und ermöglichen so eine enge Anbindung an die Datenbank. Außerdem können sie einfacher von nicht benötigten Funktionen befreit und so die Ressourcen der Rechentechnik geschont werden.

Die Definition der Such- und Steuerfunktionen für die Textretrieval-Erweiterung und das Wrapper-Modul bilden ebenfalls nur einen Grundstein. Sie ermöglichen dem

Nutzer, Suchanfragen zu stellen und dem Datenbanksystem, mit der Suchmaschine zu kommunizieren. Sie können und sollen durch neue Funktionen zur Ausnutzung aller Fähigkeiten der Suchmaschinen und Datenbanken erweitert werden. Für die Anfragefunktionalität bietet der SQL/MM-Standard eine gute Basis. Dadurch werden gleichzeitig weitere Schnittstellen im Wrapper, z.B. zur Thesaurusverwaltung, nötig.

Sollten die Hersteller von Datenbanken und Suchmaschinen diese Architektur unterstützen, so würde der Programmieraufwand für den Endanwender komplett entfallen. Dieser könnte sich dann aus den angebotenen Modulen, die für seine Aufgaben passendsten auswählen und könnte sie einfach zusammenfügen. Danach stünde ihm ein speziell zugeschnittenes Dokumentverwaltungssystem zur Verfügung, daß alle Anforderungen zufriedenstellend erfüllt.

# Anhang A

# Definition der Schnittstellen

In diesem Abschnitt werden die Prototypdateien, Schnittstellendefinitionen und Beispielimplementationen der verschiedenen Module aufgeführt und erläutert.

## A.1 Die Textretrieval-Erweiterung

Jede Datenbankerweiterung muß die neuen nutzerdefinierten Suchfunktionen bei

```
--------------------------------------------------------------------------------
-- DDL satements of the text retrieval extension
--
-- Purpose:
--    creates a handle type for text dokuments and some search functions for
--    full-text search
--
--------------------------------------------------------------------------------

-- drop old types and functions
drop type TEXT;

drop function contains;
drop function rank;
drop function count;

-- create TEXT type, actually only a handle type for texts
create type TEXT as VARCHAR(30);

-- create the search functions
create function contains(
        txt_hd  TEXT,
        s_arg   LONG VARCHAR,
        opt_arg LONG VARCHAR)
        returns INTEGER
        language c
        external name 'irlib.so!IR_contains';

create function rank(
        txt_hd  TEXT,
        s_arg   LONG VARCHAR,
        opt_arg LONG VARCHAR)
        returns DOUBLE
        language c
        external name 'irlib.so!IR_rank';

create function count(
        txt_hd  TEXT,
        s_arg   LONG VARCHAR,
        opt_arg LONG VARCHAR)
        returns INTEGER
        language c
        external name 'irlib.so!IR_count';
```

Abbildung A.1: extension.ddl

```
/*****************************************************************************\
 * Filename : IRapityp.h
 * Author   : Raiko Nitzsche (raiko@informatik.uni-rostock.de)
 * Date     : 03/02/2000
 * Version  : 1.1
 *
 * Purpose  :
 *      DB-IR connection datatypes
 *
 * History  :
 *       | Who Ver Date      | Comment
 *       |-------------------|-------------------------------------------------
 *       | rns 1.0 02/16/00  | initial version containing struct result
 *       | rns 1.1 03/02/00  | added struct docrank for rank function
 *       |                   | changed handle_list to (void *) to hold different
 *       |                   | types of lists (plain ids, ids + rank value)
 *
\*****************************************************************************/

#define IDXNAMLEN        8      /* length of the index name in id */
#define ID_LEN          31      /* length of the handle id + \0   */

#define SQL_NO_DATA_FOUND 100   /* return value for non implemented func.s */

/* datatype definitions */

/* structure to hold document handle and rank value */
struct docrank {
  char                  handle[ID_LEN]; /* document handle              */
  double                rankval;        /* ranking value               */
};

/* structure to hold result handle list */
struct result {
  unsigned long         result_count;   /* number of returned handles  */
  void                  *handle_list;   /* pointer to handle list      */
};
```

Abbildung A.2: IRapityp.h

der Datenbank anmelden. In Abbildung A.1 ist ein Beispiel für ein Datenbankskript dargestellt. Das Skript muß zum Ausführen noch an die jeweilige Syntax der verwendeten Datenbank angepaßt werden, zeigt aber die von dieser Architektur vorgegebenen Signaturen der einzelnen Funktionen.

Die Signaturen lehnen sich an den SQL/MM-Standard an ([SQLMM]), wurden aber um den Parameter `opt_arg` erweitert. Dieser Parameter dient der Steuerung der Suche, z.B. Auswahl eines Retrieval-Modells, Setzen verschiedener Parameter wie `maxdocs`, und darf von der Datenbankerweiterung nicht verändert werden. Die Funktion `count()` wurde bisher von keiner Textretrieval-Erweiterung unterstützt und ist auch nicht im SQL/MM-Standard definiert. Ihre Funktion ist in Abschnitt 5.3.1.2 erläutert.

Eine komplette Datenbankerweiterung umfaßt noch weitere Skripte zum Erstellen von Verwaltungstabellen und Anmelden von Triggerfunktionen. Diese sind aber stark von der Erweiterung abhängig und unterliegen keinen Vorgaben durch die Kopplungsarchitektur.

## A.2 Der Wrapper

Der Wrapper stellt die zentrale Schnittstelle dieser Architektur dar. Die hier definierten Funktionen bieten der Datenbank einen einheitlichen Zugriff auf die Suchfunktionalität von externen Systemen, egal ob es sich dabei um Indexmaschinen, Suchmaschinen oder gar Volltextdatenbanken handelt.

Zu dem Wrapper-Template gehören drei Dateien. Zwei davon sind Header-Dateien zur Definition der verwendeten Datentypen und Funktionen. Die dritte

```
/*****************************************************************************\
 * Filename : IRapi.h
 * Author   : Raiko Nitzsche (raiko@informatik.uni-rostock.de)
 * Date     : 05/10/2000
 * Version  : 1.2
 *
 * Purpose  :
 *      DB-IR API functions to access an external Search Server
 *
 * History  :
 *       | Who Ver Date      | Comment
 *       |-------------------|-----------------------------------------------
 *       | rns 1.0 02/16/00  | initial version containing only IR_search()
 *       | rns 1.1 03/02/00  | added IR_setoptions(), IR_count(), IR_rank()
 *       | rns 1.2 05/10/00  | removed IR_setoptions(), renamed all functions
 *       |                   | added control parameter to the search func.s
 *
\*****************************************************************************/

/* Datatype definitions */
#include "IRapityp.h"

/* Function Headers */

/* function to search a specified external index
 * returns a list containing the handles of matching documents */
RETCODE IR_Search(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           struct result *result_list   /* return list          */
           );

/* function to search a specified external index
 * returns a list containing the handles of matching documents
 * with their ranking values */
RETCODE IR_RankSearch(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           struct result *result_list   /* return list          */
           );

/* function to search a specified external index
 * returns the number of matching documents */
RETCODE IR_SearchCount(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           long count                   /* return value         */
           );
```

Abbildung A.3: IRapi.h

ist der Quelltext für den Wrapper selbst.

In der in Abbildung A.2 dargestellten Header-Datei `IRapityp.h` werden die Datentypen für die Ergebnisliste definiert. Die Struktur `result` beschreibt das eigentliche Suchergebnis. Dieses setzt sich aus der Anzahl der gefundenen Dokumente und einer Liste der entsprechenden Handlewerte zusammen. Im Falle eines Aufrufs von `contains()` enthält diese Liste nur die Handlewerte selbst, während bei einem Aufruf von `rank()` Paare von Handle- und Ranking-Wert zurückgeliefert werden. Diese Wertepaare werden durch die Struktur `docrank` beschrieben. Bei einem Aufruf von `count()` ist die Ergebnisliste leer, da nur die Anzahl der Ergebnisse benötigt wird.

Die Header-Datei `IRapi.h` in Abbildung A.3 definiert schließlich alle bisher festgelegten Aufrufe für die Kopplungsarchitektur (siehe Abschnitt 5.3.1.2 für die Erläuterung der Funktionen). Sie ermöglichen die konfigurierbare Suche durch eine externen Suchmaschine und bieten verschieden Ergebnislisten zur weiteren Nutzung in der Datenbankerweiterung (siehe oben). Der Rückgabewert aller Funktionen ist vom Typ `RETCODE`, welcher durch ODBC/CLI definiert ist. Es handelt sich hierbei

```
/*****************************************************************************\
 * Filename : wrapper.c
 * Author   : Raiko Nitzsche (raiko@informatik.uni-rostock.de)
 * Date     : 03/02/2000
 * Version  : 1.1
 *
 * Purpose  :
 *
 * Contents :
 *      IR_Search()     - call external search engines search function
 *      IR_RankSearch() - call external search engines rank function
 *      IR_SearchCount()- call external search engines count function
 *
 * History  :
 *      | Who Ver Date      | Comment
 *      |-------------------|-------------------------------------------------
 *      | rns 1.0 02/16/00  | initial version containing just IR_search()
 *      | rns 1.1 03/02/00  | added function bodies for new DB-IR API
 *
\*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "IRapityp.h"           /* data type definitions            */
#include "mgss.h"               /* search engine specific functions */


/* IR_search :
 *      DB-IR API function to call external SearchServer to search for a
 *      given argument in a specific index
 *
 * Parameters :
 *      see below ...
 *
 *      RETURN  SQL error code
 */
RETCODE IR_search(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           struct result *result_list   /* return list          */
           )
{
  /* local variables */
  char  *c;                             /* dummy for quote convertion   */

  /* MG does not allow double quotes around the search string */
  /* delete double quotes */
  while ( (c = strchr( searchstr, (int)'"')) ) *c = ' ';


  return( MG_query( indexname, searchstr, options, result_list));
}

   .
   .
   .
```

Abbildung A.4a: wrapper.c

um einen Fehlercode der Datenbankfunktion. Die Rückgabe der Ergebnisliste erfolgt über den Parameter `result_list`, dessen Typ in `IRapityp.h` festgelegt wurde. Die Struktur der Listenelemente ist abhängig von der Funktion und wurde weiter oben bereits erläutert.

Die letzte Datei ist das Wrapper-Modul selbst. Hier werden die eben definierten Suchfunktionen implementiert. Die Aufgabe dieses Moduls besteht darin, die durch die Architektur definierten Aufrufe auf die entsprechenden Funktionen des Suchmaschinentreibers umzusetzen. Dabei werden gleichzeitig einige Anpassungen der Parameter vorgenommen.

Die Abbildung A.4a zeigt den Wrapper für die Testimplementation der MG-Suchmaschine. Für sie ist eine Anpassung der Schreibweise der Suchargumente

```
   .
   .
   .

/* IR_rank :
 *      DB-IR API function to call external SearchServer to search a
 *      given argument in a specific external index and return ranking        .
 *      informations
 *
 * Parameters :
 *      see below ...
 *
 *      RETURN  SQL error code
 */
RETCODE IR_rank(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           struct result *result_list   /* return list          */
           )
{
  return( SQL_NO_DATA_FOUND);
}

/* IR_count :
 *      DB-IR API function to call external SearchServer to search a
 *      given argument in a specific external index and return the number
 *      of matches
 *
 * Parameters :
 *      see below ...
 *
 *      RETURN  SQL error code
 */
RETCODE IR_count(
           char *indexname,             /* name of the index    */
           char *searchstr,             /* search argument      */
           char *options,               /* options parameter    */
           long count                   /* return value         */
           )
{
  return( SQL_NO_DATA_FOUND);
}
```

Abbildung A.4b: wrapper.c

notwendig, da MG keine Anführungszeichen um die Suchargumente erlaubt. Der Grund für die Umsetzung in diesem Modul und nicht im Suchmaschinentreiber ist, daß der Aufbau der Suchargumente in den Datenbanksystemen unterschiedlich ist. Der Text-Extender von DB2 UDB nutzt z.B. folgendes Format: `'"relational" AND "optimizer"'`, während das Excalibur Text-DataBlade von Informix schon die gleiche Schreibweise wie MG nutzt: `'relational optimizer'`. Um den Suchmaschinentreiber nicht an das verwendete Datenbanksystem anpassen zu müssen, nutzt dieser die von MG geforderte Schreibweise und die jeweiligen Wrapper müssen das Suchargument in der richtigen Form liefern.

Die entsprechende Suchfunktion des MG-Treibers heißt `MG_query()` und übernimmt alle Parameter von `IR_search()`. Die Ergebnisliste und der Fehlercode werden in gleicher Weise wie bei `IR_search()` zurückgeliefert.

In Abbildung A.4b sind die leeren Programmimplementationen des Wrapper-Templates zu sehen. Das Template kann zu Kompilier- und Testzwecken von der Datenbankerweiterung alleinstehend eingebunden werden. Alle Funktionen liefern in diesem Fall eine leere Ergebnisliste und täuschen so eine erfolglose Suche vor.

## A.3 Das Steuerprogramm

Das Steuerprogramm in der Testimplementation ist ein Shellskript, das alle zur Indizierung einer Textspalte notwendigen Befehle aufruft. Die einzelnen Befehle wer-

```
#! /bin/sh
#
# Name    : crtindex
# Author  : Raiko Nitzsche
# Date    : 05/21/00
# Version : 1.1
#
# Purpose : This script creates an index on a database text column using
#           an external search engine
#
# Usage   : crtindex dbname schema.tblname colname handle
#
# dbname  : name of the database
# schema  : name of the table schema
# tblname : name of the table
# colname : name of the column
# handle  : name of the handle column
#
# Changes : 1.0 initial version - fixed data source
#           1.1 outsourcing of special tasks to create a generic interface   .
#               for the index creation
#

if [ $# -eq 0 -o $# -gt 5 ]
then
  echo "Usage: $0 dbname schema tblname colname handle"
  exit
fi

# check for necessary environment
if [ ! (`./chk_env_db` -a `./chk_env_se`) ]
then
  # either database or search engine environment is missing
  exit 1
fi

# calculate the new index number
idxname=`./get_idx_name`

echo "create handle column..."
# alter table - add new handle column
./alter_db ${1} ${2} ${3} ${4} ${5}

# update handle values
./gen_hd ${idxname} ${1} ${2} ${3} ${5}

echo "start indexing..."
# execute indexing program
./create_idx ${1} ${2} ${3} ${4} ${5} ${idxname}

exit 0
```

Abbildung A.5: Das Steuerprogramm `crtindex`

den von der Datenbankerweiterung oder dem Suchmaschinentreiber bereitgestellt, um die Austauschbarkeit der Module zu gewährleisten. In Abbildung A.5 ist das Programm zu sehen.

Die Parameter des Steuerprogramm `crtindex` sind die vollständige Qualifizierung der zu indizierenden Textspalte und der Name für die neu anzulegende Handlespalte. Vollständige Qualifizierung bedeutet, daß nicht nur die Tabelle und Spalte angegeben werden müssen, sondern auch die Datenbank und das Schema der Tabelle.

Als erstes werden mit Hilfe der Programme `chk_env_db` und `chk_env_se` die Environment-Variablen überprüft. In einer fertigen Endanwenderinstallation sollte dieser Schritt nicht mehr notwendig sein, da entsprechende Variablen bei der Installation des System gesetzt werden. In einer Testumgebung, in der bereits installierte Systeme miteinandergekoppelt werden, ist diese Überprüfung allerdings sinnvoll.

Der nächste Schritt ist das Ermitteln eines eindeutigen Indexnamens mit Hilfe des Suchmaschinenprogramms `get_idx_name`. Unter diesem Namen werden die Do-

```
#! /bin/sh
#
# Name    : create_idx
# Author  : Raiko Nitzsche
# Date    : 05/21/00
# Version : 1.0
#
# Usage   : create_idx dbname schema.tblname colname handle idxname
#
# Purpose : This script creates a FULCRUM index on a database text column     .
#
# dbname  : name of the database
# schema  : name of the table schema
# tblname : name of the table
# colname : name of the column
# handle  : name of the handle column
# idxname : name or the index
#
# Changes : 1.0 initial version
#

if [ $# -eq 0 -o $# -gt 6 ]
then
  echo "Usage: $0 dbname schema tblname colname handle idxname"
  exit 1
fi

idxname=${6}

# delete FULCRUM execution file
rm -f crtfulindex.fte

echo "create FULCRUM db did file..."
cat crtfulindex.did | sed 's/dbname/'${1}'/; s/tblname/'${3}'/g;
                          s/idcol/'${5}'/; s/txtcol/'${4}'/'> ${idxname}.did

echo "create FULCRUM execution file..."
cat crtfulindex.tpl | sed 's/IXNAME/'${idxname}'/g' > crtfulindex.fte

# insert the seed row into the index table
echo "INSERT INTO $idxname ( FT_SFNAME, FT_FLIST)
         VALUES( '${idxname}.did', 'rnodbc!rnodbc/,@');" >> crtfulindex.fte

# add the VALIDATE INDEX clause
echo "" >> crtfulindex.fte
cat crtfulindex.val | sed 's/IXNAME/'${idxname}'/g' >> crtfulindex.fte

echo "execute FULCRUM execution file..."
execsql -0 crtfulindex.fte > crtfulindex.log

exit 0
```

Abbildung A.6: Das `create_idx` Programm für Fulcrum

kumente der hier angegebenen Textspalte indiziert. Er wird dabei in die Dokument-Handles einkodiert, um bei einer Suchanfrage das Dokument seiner richtigen Indexdomäne zuordnen zu können.

Nachdem der Indexname bekannt ist, kann die neue Handlespalte hinzugefügt und die entsprechenden Einträge generiert werden. Dies geschieht durch den Aufruf der Programme `alter_db` und `gen_hd`, welche von der Datenbankerweiterung bereitgestellt werden.

Als letztes wird die Indizierung gestartet. Dazu wird das Programm `create_idx` aufgerufen, das wiederum zur Suchmaschine gehört und alle für diese Maschine notwendigen Schritte zur Indexerzeugung ausführt. Der nächste Abschnitt beschreibt diesen Vorgang am Beispiel von Fulcrum.

### A.3.1 Indexerzeugung mit Fulcrum

Die Indexerzeugung von Texten, die in einer Datenbank gespeichert sind, unterteilt sich in mehrere Stufen. Abbildung A.6 zeigt das Programm `create_idx` für den

```
-- crtfulindex.did - DID file to describe the datasource within the dbms     --
SET LOGDETAILS ALL

DATASOURCE dbname
USER db2admin
PASSWORD admpwd

COLUMN tblname.idcol KEY STORE 200
COLUMN tblname.txtcol LONGCHAR

-- end of crtfulindex.did
```

Abbildung A.7: Template für die DID-Datei

```
-- crtfulindex.tpl - Template for creating a FULCRUM index table             --

CREATE SCHEMA IXNAME
CREATE TABLE IXNAME ( id        VARCHAR(30)     200)

-- -----------------
-- Table Parameters
-- -----------------
STOPFILE 'support.stp' -- Minimal stop word set
-- BASEPATH '../idxdocs'  -- External files are in this subdirectory

-- end of crtfulindex.tpl
;
```

Abbildung A.8: Template für die ExecSQL-Datei

Fulcrum SearchServer, daß diese Schritte nacheinander abarbeitet.

Als erstes muß eine sogenannte *DID*-Datei erstellt werden, welche dem Datenbank-Textreader mitteilt, woher die Dokumente und zusätzliche Informationen zu beziehen sind. Die hier erzeugten Indexe enthalten jeweils zwei Spalten, eine für die Dokumente selbst und ein für den zugehörigen Handlewert. Die *DID*-Datei sieht dementsprechend immer ähnlich aus und unterscheidet sich nur durch den Namen der zugehörigen Text- und Handlespalte. In Abbildung A.7 ist das Template für die *DID*-Datei zu sehen, bei dem die Parameter `tblname` durch den Namen der Tabelle, `idcol` durch den Namen der Handlespalte und `txtcol` durch den Namen der Textspalte zu ersetzen sind. Die anderen Parameter für die Identifizierung der Datenquelle und der Nutzerauthorisierung sind durch Fulcrum und die Anbindung der Datenquelle über ODBC bedingt. Sie müssen bei der Installation der Suchmaschinentreiber durch den Datenbank-Administrator entsprechend gesetzt werden und sollten vor unauthorisierten Lesezugriffen geschützt sein. Die *DID*-Datei muß im entsprechenden Datenverzeichnis von Fulcrum gespeichert werden, da sie bei jedem Zugriff auf den Index benötigt wird.

Der nächste Schritt ist die Erzeugung eines *ExecSQL*-Skripts, in dem alle weiteren Kommandos zur Indizierung ausgeführt werden. Auch dafür gibt es wieder ein Template, da es sich bei allen Aufrufen gleicht. Das Template ist in Abbildung A.8 zu sehen. Hier wird zuerst eine Tabelle und ein Schema mit dem Namen des Indexes angelegt. Das Programm `crt_index` ersetzt hierzu den Variablennamen `IXNAME` durch den vom Steuerprogramm erhaltenen neuen Indexnamen. Eine Textspalte ist bei Fulcrum-Tabellen immer vorhanden, also muß nur noch die Handlespalte neu definiert werden. Die Tabellenparameter geben an, wo die Stopwortliste und die Dokumente zu finden sind. Der Pfad für die Dokumente wird bei der Lösung per Datenbank-Textreader nicht benötigt. Auf Grund der Probleme mit dieser Lösung wurde aber zu Testzwecken eine Variante entwickelt, bei der die Dokumente vorher in das Filesystem ausgelagert und durch einen normalen Textreader eingelesen werden. In diesem Fall wird auch der Basispfad für die Dokumente benötigt.

```
-- crtfulindex.tpl - Template for creating a FULCRUM index table             --

CREATE SCHEMA IX000001
CREATE TABLE IX000001 ( id      VARCHAR(30)     200)

-- -----------------
-- Table Parameters
-- -----------------
STOPFILE 'support.stp' -- Minimal stop word set
-- BASEPATH '../idxdocs'  -- External files are in this subdirectory

-- end of crtfulindex.tpl
;

INSERT INTO IX000001 ( FT_SFNAME, FT_FLIST)
                 VALUES( 'IX000001.did', 'rnodbc!rnodbc/,@');

VALIDATE INDEX IX000001 VALIDATE TABLE;
```

Abbildung A.9: Beispiel einer generierten ExecSQL-Datei

Damit die Dokumente in die Tabelle eingelesen werden können, muß eine sogenannte *Seed Row* (Saat-Spalte) in die Tabelle eingefügt werden, in der der zu verwendende Textreader (hier der ODBC-Datenbank-Reader) und die zugehörige Steuerdatei (die vorher erzeugte *DID*-Datei) eingetragen werden. Danach wird mit dem `VALIDATE INDEX`-Kommando die Indizierung gestartet. Alles Weitere wird jetzt automatisch durch den SearchServer erledigt, eingeschlossen Indexupdates. Nähere Erläuterungen zu diesem Vorgang findet man in [SSD98].

In Abbildung A.9 ist ein fertig generiertes *ExecSQL*-Skript zu sehen, das einen Index namens `IX000001` anlegt. Nachdem das Skript erstellt wurde, wird es mit dem Befehl `execsql` ausgeführt. Mit Beendigung dieses letzten Befehls ist auch das Steuerprogramm beendet. Das Skript wird nicht mehr benötigt bzw. bei jedem Aufruf des Steuerprogramms wieder neu erstellt. Deshalb wird es nicht in einem Fulcrum Systempfad gespeichert, sondern wieder gelöscht.

# Anhang B

# Tabellarischer Vergleich

| | **FULCRUM** | **MG System** |
|---|---|---|
| DB-Anbindung | ODBC-DB-Reader | nein<br>(aber möglich) |
| Anfrage-<br>schnittstelle | ODBC mit<br>SQL-Dialekt | Programmaufruf<br>mit Retrieval-<br>Kommando |
| Anzahl verwalt-<br>barer Indexe | mehrere | mehrere |
| Unterstützte<br>Dokumenttypen | nur Text | Text<br>Graphiken |
| Speicherung<br>der Dokumente | nur<br>Referenzen | komprimiert<br>in eigener DB |
| Metadaten<br>speicherbar | ja | nein |
| Verfügbarkeit | kommerziell<br>Quellen für<br>DB-Reader<br>verfügbar | öffentlich<br>freier Quellcode<br>mit Dokumentation |

Tabelle B.1: Vergleich der IR-Werkzeuge

| | **DB2 UDB** | **Informix** |
|---|---|---|
| Erweiterbarkeit | Datentypen<br>Funktionen<br>Index (ab V7.1) | Datentypen<br>Funktionen<br>Anfragesprache<br>Zugriffspfade<br>Speicherstrukturen<br>Optimierer |
| Programmiersprachen<br>für<br>- UDFs | C/C++<br>JAVA<br>COBOL<br>FORTRAN<br>REXX | C/C++<br>JAVA<br>SPL |
| - UDTs | nicht möglich | |
| Zugriff auf die<br>Datenbank | ODBC/JDBC<br>Embedded SQL | ODBC/JDBC<br>Embedded SQL |

Tabelle B.2: Vergleich der ORDBS

| | **Suchmaschinen mit DB-Anbindung** | **ORDBS mit Suchmaschine** | **Middleware-Ansatz** |
|---|---|---|---|
| Zugriff auf<br>die Dokumente | über die<br>Suchmaschine | über die<br>Datenbank | über die<br>Middleware |
| DB-Funktionalität | keine | komplett | komplett |
| IR-Funktionalität<br>- Ranking<br>- Rel.Feedback<br>- inkr.Anfragen | komplett<br>ja<br>ja<br>ja | Suchanfragen<br>ja<br>nein<br>nein | komplett<br>ja<br>ja<br>ja |
| Nutzung der<br>Standard-API | ja | ja | ja |
| Erweiterungsfähig | kaum | ja | ja |
| Austauschbarkeit<br>der Komponenten | sehr<br>gut | möglich | möglich |
| Kommunikations-<br>overhead | gering | gering | hoch |
| Implementations-<br>aufwand | gering | mittel | hoch |

Tabelle B.3: Vergleich der Kopplungsarchitekturen

# Abkürzungsverzeichnis

| | |
|---|---|
| **API** | Application Programming Interface |
| **BLOB** | Binary Large Object |
| **CLOB** | Character Large Object |
| **DB** | Datenbank |
| **DBMS** | Datenbankmanagementsystem |
| **DID** | Database Interface Definition |
| **HTML** | Hypertext Markup Language |
| **IR** | Information Retrieval |
| **LOB** | Large Object |
| **OID** | Object Identifier |
| **OODB** | Objektorientierte DB |
| **OODBMS** | Objektorientiertes DBMS |
| **ORDB** | Objektrelationale DB |
| **ORDBMS** | Objektrelationales DBMS |
| **RDB** | Relationale DB |
| **RDBMS** | Relationales DBMS |
| **RTF** | Rich Text Format |
| **SPL** | Stored Procedure Language |
| **SQL** | Structured Query Language |
| **UDF** | User Defined Function |
| **UDR** | User Defined Routine |
| **UDT** | User Defined Type |
| **XML** | Extended Markup Language |

# Abbildungsverzeichnis


2.1 Architektur des Fulcrum SearchServers . . . . . 8
2.2 Funktionsweise des MG Systems . . . . . 10
2.3 Beispielkonfiguration der Textsuchmaschine . . . . . 12
2.4 Funktionsweise einer Websuchmaschine . . . . . 13

3.1 Architektur des Text-Extenders . . . . . 21
3.2 Schema der DataBlade-Technologie . . . . . 23

4.1 Textretrieval-System mit Suchmaschine . . . . . 30
4.2 Textretrieval-System mit ORDBMS . . . . . 32
4.3 Textretrieval-System mit Komponenten-DBMS . . . . . 33
4.4 Textretrieval-System mit Middleware . . . . . 34

5.1 Suchmaschine als externer Index eines ORDBMS . . . . . 42
5.2 Suchmaschine als DB Anwendung bei der Indexerzeugung . . . . . 53

6.1 Textretrieval-Erweiterung mit Fulcrum Suchmaschine . . . . . 66
6.2 Textretrieval-Erweiterung mit Fulcrum Suchmaschine reale Umsetzung mittels ODBC-Treibermanager . . . . . 67

A.1 extension.ddl . . . . . 79
A.2 IRapityp.h . . . . . 80
A.3 IRapi.h . . . . . 81
A.4a wrapper.c . . . . . 82
A.4b . . . . . 83
A.5 Das Steuerprogramm `crtindex` . . . . . 84
A.6 Das `create_idx` Programm für Fulcrum . . . . . 85
A.7 Template für die DID-Datei . . . . . 86
A.8 Template für die ExecSQL-Datei . . . . . 86
A.9 Beispiel einer generierten ExecSQL-Datei . . . . . 87

# Tabellenverzeichnis

B.1 Vergleich der IR-Werkzeuge . . . . . 88
B.2 Vergleich der ORDBS . . . . . 89
B.3 Vergleich der Kopplungsarchitekturen . . . . . 89

# Literaturverzeichnis


[Aok91] Aoki, P. M.: *Implementation of Extended Indexes in POSTGRES.* SIGIR Forum, Vol. 25, S. 2-9, 1991.

[Car86] Carey, M.: *The Architectur of the EXODUS Extensible DBMS.* In Proc. 1st Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September 1986.

[CG99] Chaudhuri, S.; Gravano, L.: *Evaluating Top-k Selection Queries.* In Proc. VLDB, S. 399-410, September 1999.

[Cha98] Chamberlin, D.: *A complete Guide to DB2 Universal Database.* Morgan Kaufmann Publishers, San Francisco, 1. Auflage, 1998.

[CK97] Carey, M.; Kossmann, D.: *On saying "Enough Already!" in SQL.* In Proc. SIGMOD, Mai 1997.

[CK98] Carey, M.; Kossmann, D.: *Reducing the braking distance of an SQL query engine.* In Proc. VLDB, August 1998.

[Dav96] Davis, J.: *Informix Universal Server, Extending the Relational Management System to Manage Complex Data.* White Paper, 1996.

[DDS95] De Fazio, S.; Daoud, A.; Smith, L. A.; Srinivasan, J.: *Integrating IR and RDBMS Using Cooperative Indexing.* In Proc. SIGR, Seattle, WA, S. 84-92, 1995.

[DM97] Deßloch, St.; Mattos, N.M.: *Integrating SQL Databases with Content-specific Search Engines.* In Proc. VLDB, S. 528-537, 1997.

[DR99] Donjerkovic, D.; Ramakrishnan, R.: *Probabilistic Optimization of Top N Queries.* In Proc. VLDB, S. 411-422, September 1999.

[Fuh92] Fuhr, N.: *Konzepte zur Gestaltung zukünftiger Information-Retrieval-Systeme.* In: R. Kuhlen, Hrsg., Experimentelles und praktisches Information Retrieval, S. 59-75, Universtitätsverlag Konstanz, 1992.

[Fuh93] Fuhr, N.: *A Probabilistic Relational Model for the Integration of IR and Databases.* Universität Dortmund, Fachbereich Informatik, 1993.

[Fuh94] Fuhr, N.: *Logical and Conceptual Models for the Integration of Information Retrieval and Database Systems.* Universität Dortmund, Fachbereich Informatik, 1994.

[Fuh96] Fuhr, N.: *Models for Integrated Information Retrieval and Database Systems.* Universität Dortmund, Fachbereich Informatik, 1996.

[GFH97] Grossman, D. A.; Frieder, O.; Holmes, D. O.; Roberts, D. C.: *Integrating Structured Data and Text: a retrieval approach.* Journal of the American Society of Information Science, Vol. 48, Nr. 2, Februar 1997.

[Gut84] Guttman, A.: *R-Trees: A Dynamic Index Structure for Spatial Searching.* Proc. ACM SIGMOD Conf., S.47-57, Juni 1984.

[Heu97] Heuer, A.: *Objektorientierte Datenbanken - Konzepte, Modelle, Standards und Systeme.* Addison-Wesley, 2. Auflage, Oktober 1997

[HP00] Heuer, A.; Priebe, D.: *Integrating a Query Language for Structured and Semi-Structured Data and IR Techniques.* TechReport, Universität Rostock, Fachbereich Informatik, Mai 2000.

[IBM99] IBM Corp.: *Text Extender: Administration and Programming*, 1999.

[INF97] Informix Corp.: *Excalibur TextSearch DataBlade Users Guide*, July 1997.

[INF00] Informix Corp.: *Developing DataBlade Modules for Infomix Internet Foundation.*, 2000. (`http://www.informix.com`)

[Jag90] Jagadish, H.V.: *Spatial Search with Polyhedra.* Proc. 6th IEEE Intl. Conf. on Data Engineering, 1990.

[LH00] Lindner, W.; Heuer, A.: *Das MacWrap Project: Nutzung eines Objekt-Relationalen DBMS für ein offenes Multimedia-Datenbanksystem.* Tagungsband 12. Workshop Grundlagen von Datenbanken, Kiel, Juni 2000.

[Loh91] Lohman, G., et. al.: *Extensions to Starbust: Objects, Types, Functions and Rules.* CACM 34(10), 1991.

[LS88] Lynch, C.; Stonebraker, M.: *Extended User Defined Indexing with Applications to Textual Databases.* Proc. 14th VLDB Conf., Los Angelas, CA, S. 306-3017, 1988.

[Mey96] Meyer, J.: *Entscheidungsgrundlage für die Auswahl von Volltextdatenbanksystemen für den MeDoc-Dienst.* MeDoc Bericht TP2 und Pilotanwender, Juli 1996.

[Nit99] Nitzsche, R.: *User Defined Index Types and User Defined Search for Text Extender.* Studienarbeit, Universität Rostock, Fachbereich Informatik, 1999.

[Por99] Porst, B.: *Untersuchung zu Datentyperweiterungen für XML-Dokumente und ihre Anfragemethoden am Beispiel von DB2 und Informix.* Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1999.

[Rah98] Rahm, E.: *Evaluation of object-relational Database Systems for Fulltext Retrival.* Report Nr. 6 (1998), ISSN 1430-370, Universität Leipzig, Institut für Informatik, Mai 1998.

[Sch86] Schwarz, P., et. al.: *Extensibility to the Starbust Database System.* Proc. 1st Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September 1986.

[Sch99] Schäuble, P.: *Information Retrieval und Datenbanken.* Vortrag auf der ADI 99, Rostock, September 1999.

[SQLMM] ISO/IEC 13249-2:2000: *SQL Multimedia and Application Packages.* Part 2: Full-Text, March 2000.

[SSD98] Fulcrum Technologies Inc. *Fulcrum SearchServer : Database Intergration*, July 1998.

[SSI98] Fulcrum Technologies Inc. *Fulcrum SearchServer : Introduction to SearchServer*, July 1998.

[SSS98] Fulcrum Technologies Inc. *Fulcrum SearchServer : SearchSQL Reference*, July 1998.

[Sto96] Stonebraker, M.: *Object-relational DBMSs - The Next Great Wave.* Morgan Kaufmann Publishers, 1996.

[Tit98] Titzler, P.: *Realisierungsvorschlag für die Implementierung einer verteilten Suchmaschine im WWW.* Studienarbeit, Universität Rostock, Fachbereich Informatik, 1998.

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

[WFC98] Wang, Y.; Fuh, G.; Chow, J.-H.; Grandbois, J.; Mattos, N.M.; Tran, B.: *An extensible architecture for supporting spatial data in RDBMS.* In Proc. of Workshop on Software Engineering and Database Systems, 1998.

[WMB99] Witten, I. H.; Moffat, A.; Bell, T. C.: *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishing, San Francisco, 2. Auflage, May 1999.