Diplomarbeit

Kombinierte Anfrageverarbeitung auf Struktur und Inhalt von multimedialen Dokumenten

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von	:	Heiko Jahnkuhn
Matrikelnummer	:	000200028
geboren am	:	18.07.1980 in Rostock
Erstgutachter	:	Prof. Dr. Andreas Heuer
Zweitgutachter	:	Prof. Dr. Thomas Kirste
Betreuer	:	Ilvio Bruder
		Temenushka Ignatova
Abgabedatum	:	15.12.2005

Zusammenfassung

Retrieval-Konzepte existieren bislang nur für die einzelnen Medien. Ein medienübergreifendes Retrieval über das Gesamtobjekt eines multimedialen Dokuments existiert bisher nur in rudimentärer Form, oder wurde ausschließlich anwendungsabhängig entworfen. Ziel dieser Arbeit ist es, eine kombinierte Anfrageverarbeitung auf Strukturen und Inhalte eines multimedialen Dokuments zu untersuchen, wobei der Fokus auf den Einsatz existierender Standards gesetzt wird. Es wird somit eine Backend-Lösung vorgestellt, die unabhängig vom verwendeten Szenario gültig ist, und als Grundlage allgemein eingesetzt werden kann.

Abstract

Existing retrieval concepts just work on seperate media types. A media spanning retrieval for whole multimedia documents have only rudimentary functionalities, or have been designed application defined, respectively. The aim of this work consists of analizing a combined query processing for structure and content of multimedia documents, whereby using existing standards describes the main focus of these retrieval concepts. Consequently, this paper presents a backend solution, which is application independent, furthermore it is a general foundation for media spanning retrieval.

CR-Klassifikation

E.1 DATA STRUCTURES

H.2.1 Database Management - Logical Design

H.2.3 Database Management - Languages

H.2.4 Database Management - Systems

H.2.8 Database Applications

H.3.3 Information Search and Retrieval

H.5.1 Multimedia Information Systems

Key Words

SQL, database, query processing, distributed databases, data integration, multimedia documents, multimedia database, eNoteHistory

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Motivation . . . 7
1.2 Das Projekt eNoteHistory . . . 8
1.3 Aufbau der Arbeit . . . 11

2 SQL-Standards **12**
2.1 SQL:1999/2003 . . . 12
2.1.1 SQL/Foundation . . . 13
2.1.2 SQL/MED . . . 16
2.1.3 SQL/XML . . . 19
2.2 SQL/MM . . . 20
2.2.1 Full-Text . . . 21
2.2.2 Still Image . . . 24
2.2.3 Spatial . . . 27

3 Konzeption der Anfrageverarbeitung **30**
3.1 Das Dokumentenmodell . . . 30
3.1.1 Allgemeiner Aufbau . . . 30
3.1.2 Package Image . . . 32
3.1.3 Package Text . . . 33
3.1.4 Komponentenbeziehungen . . . 34
3.2 Aufbau des Multimedia-Datenbanksystems . . . 35
3.2.1 Speicherung der Dokumente . . . 36
3.2.2 Speicherung der Metadaten . . . 38
3.2.3 Verwaltung von Component-Beziehungen . . . 39
3.2.4 Aufbau der Fremdtabellen . . . 41
3.3 Anfragemöglichkeiten . . . 44
3.3.1 Anfragen auf Struktur . . . 44
3.3.1.1 Geometrische Anfragen . . . 45
3.3.1.2 Unscharfe Anfragen . . . 46

3.3.1.3 Logische modellbezogene Anfragen . . . 47
3.3.1.4 Logische selbstdefinierte Anfragen . . . 48
3.3.2 Anfragen auf Inhalt . . . 49
3.3.2.1 Anfragen auf Text . . . 49
3.3.2.2 Anfragen auf Bild . . . 50
3.3.2.3 Anfragen auf Metadaten . . . 52
3.3.3 Kombinierte Anfragen . . . 53
3.4 Anfragebearbeitung . . . 58
3.4.1 Anfragevorverarbeitung . . . 58
3.4.2 Optimierung . . . 61
3.4.3 Anfrageausführung . . . 62
3.5 Probleme . . . 63

4 Praktische Relevanz 65
4.1 IBM DB2 . . . 65
4.1.1 Integration externer Daten . . . 67
4.1.2 Unterstützung von XML . . . 68
4.1.3 Verwaltung von Bildobjekten . . . 68
4.1.4 Verwaltung von Volltextobjekten . . . 69
4.1.5 Verwaltung geometrischer Objekte . . . 70
4.2 Oracle . . . 71
4.2.1 Integration externer Daten . . . 72
4.2.2 Unterstützung von XML . . . 72
4.2.3 Verwaltung von Bildobjekten . . . 73
4.2.4 Verwaltung von Volltextobjekten . . . 73
4.2.5 Verwaltung geometrischer Objekte . . . 74
4.3 Fazit . . . 75

5 Prototypische Implementierung 76
5.1 Vorbetrachtungen . . . 76
5.2 Aufbau des Prototyps . . . 78
5.2.1 Document_T . . . 78
5.2.2 Component_T . . . 79
5.2.3 ImageComponent_T . . . 80
5.2.4 Images . . . 80
5.2.5 Relations_T . . . 80
5.3 Import der Daten . . . 81
5.4 Anfragemöglichkeiten . . . 81
5.4.1 Anfragen auf Struktur . . . 82

5.4.2 Anfragen auf Inhalt 84
5.4.3 Kombinierte Anfragen 86
5.5 Probleme 91

6 Schlussbetrachtungen **93**
6.1 Zusammenfassung 93
6.2 Ausblick 94

Literaturverzeichnis **96**

Abbildungsverzeichnis **99**

Tabellenverzeichnis **100**

A SQL-Befehle der Konzeption **101**
A.1 DDL-Anweisungen 101
A.2 Routinen 102
A.3 Hilfsfunktionen 107

B SQL-Befehle des Prototyps **110**
B.1 DDL-Anweisungen 110
B.2 Routinen 112

C Schemata der eNoteHistory Datenbank **119**
C.1 Dict 119
C.2 Features 120
C.3 Metadata 121
C.4 IPFV 122

Kapitel 1

Einleitung

1.1 Motivation

Es existiert derzeit eine Menge von Informationssystemen, durch welche Informationen aus Daten ermittelt werden können. Retrieval-Konzepte bezüglich dieser Daten existieren in den meisten Fällen allerdings nur für einzelne Medien wie Bilder oder Texte. Ein Retrieval über das Gesamtobjekt eines multimedialen Dokuments wird bisher nur in wenigen Systemen unterstützt. Systeme, wie beispielsweise Multimap [Spe98], sind jedoch meistens anwendungsspezifisch entworfen, und somit auch nur in einem abgegrenzten Kontext verwendbar. Weiterhin bauen derartige Systeme nicht auf existierende Standards auf, sondern arbeiten mit individuellen Lösungen, die für den entsprechenden Einsatz geeignet sind.

Ein konkretes Informationssystem, welches im kommenden Abschnitt noch genauer beschrieben wird, wird durch das Projekt eNoteHistory[1] bereitgestellt. Ein Teilziel des Projekts bildet dabei die Digitalisierung von historischen Notenhandschriften. Die Datenbank dieses Projekts stellt somit eine Grundmenge an multimedialen Dokumenten zur Verfügung, auf welchen ein Retrieval bezüglich Inhalt und Struktur durchgeführt werden kann.

Ziel dieser Arbeit ist es nun, eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt von multimedialen Dokumenten zu entwerfen, die sich komplett auf bestehende und kommende Standards stützt, und somit von der Grundlage her anwendungsunabhängig ist. Das Hauptaugenmerk der theoretischen Konzeption liegt dabei auf den Einsatzmöglichkeiten der Standards SQL:1999, SQL:2003 und SQL/MM. Eine mögliche praktische Umsetzung in bestehenden, objektrelationalen Systemen wird ebenfalls untersucht.

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

1.2 Das Projekt eNoteHistory

Das Projekt eNoteHistory ist ein Gemeinschaftsprojekt des Instituts für Musikwissenschaft der Universität Rostock, des Datenbanklehrstuhls der Informatik und des Fraunhofer Institut für Graphische Datenverarbeitung (IGD). Ziel dieses Projektes ist es, die Schreibererkennung von Notenhandschriften mit Hilfe von Computeranalysen zu vereinfachen [eNo05]. Weiterhin wird durch dieses Projekt eine Grundmenge von multimedialen Dokumenten bereitgestellt, um auf diesen ein Retriel zu ermöglichen.

Vom späten 17. Jahrhundert bis zum Beginn des 19. Jahrhunderts wurden Kompositionen überwiegend handschriftlich vervielfältigt und verbreitet. Dabei ist es für die Musikwissenschaft von großer Bedeutung, Notenhandschriften ihren Schreibern zuordnen zu können. Somit kann man eingrenzen, wo, wann, warum und für wen eine Komposition geschrieben beziehungsweise abgeschrieben wurde. Auch über die Verbreitung von Kompositionen können somit Aussagen getroffen werden. Um nun den Schreiber einer Komposition identifizieren zu können, können sowohl individuelle Merkmale der Handschrift, aber auch das verwendete Papier, Tinte, Wasserzeichen und weitere Einzelheiten verwendet werden.

In der Universität Rostock lagern derzeit über 5000 Notenhandschriften des 17. und 18. Jahrhunderts, welche im Rahmen des Projekts eNoteHistory digitalisiert und analysiert werden sollen. Weiterhin sollen Mechanismen entwickelt werden, mit Hilfe derer eine Schreibererkennung auf Grund der oben genannten Merkmale durchgeführt werden kann. Dazu wurden mit Hilfe von Musikexperten 80 Features aus 13 Featuregruppen bestimmt, mit welchen eine Handschrift beschrieben werden kann. Diese Gruppen lauten *Schlüssel, Taktvorzeichen, Viertelpausen, Schriftneigung, Kaudierung, Form der Fähnchen, Schreibgewohnheiten, Laufweite, Bebalkung, Vorzeichen, Form der Notenköpfe, Schlusszeichen* und *Rastral.* Auf Grundlage dieser Features und Featuregruppen wurde dann ein hierarchisches Ordnungssystem (Feature Base) entwickelt, welches eine Zerlegung komplexer Zeichen in ihre Grundelemente gestattet. Diese Feature Base bildet daraufhin die Grundlage der manuellen Merkmalsbestimmung. Wird nun eine Handschrift analysiert, entsteht durch Zuweisung von Werten zu einzelnen Features ein hochdimensionaler Feature-Vektor, der die Schreibercharakteristik repräsentiert, und in der Datenbank des Projekts gespeichert wird. Dieser Prozess ist in Abbildung 1.1 dargestellt.

Neben der manuellen Analyse unter Nutzung der Feature Base wird zusätzlich auch eine automatische Handschriftenanalyse vom Fraunhofer Institut verfolgt. Diese basiert auf der Merkmalsextraktion mittels rechnergestützter Bildverarbeitungsfunktionen. In diesem mehrstufigen Prozess wird zunächst das Bildrauschen verringert, daraufhin der Vordergrund vom Hintergrund getrennt, und anschließend eine Objekterkennung für Primitive wie Notenköpfe und -Hälse durchgeführt. Hierauf wird allerdings an dieser Stelle nicht näher eingegangen.

Der logische Aufbau der zum Projekt gehörenden Datenbank, welcher ausführlich in [eNo05] beschrieben ist, basiert auf der Einteilung des Projekts in vier Projektteile, die in separaten Schemata organisiert sind. Diese Schemata, *Dict*, *Features*, *Metadata* und *IPFV*, deren Aufbau in UML-Darstellung auch in Anhang C dargestellt ist, dienen zur logischen Abgrenzung der Daten, und werden im Folgenden grundlegend vorgestellt. Auf

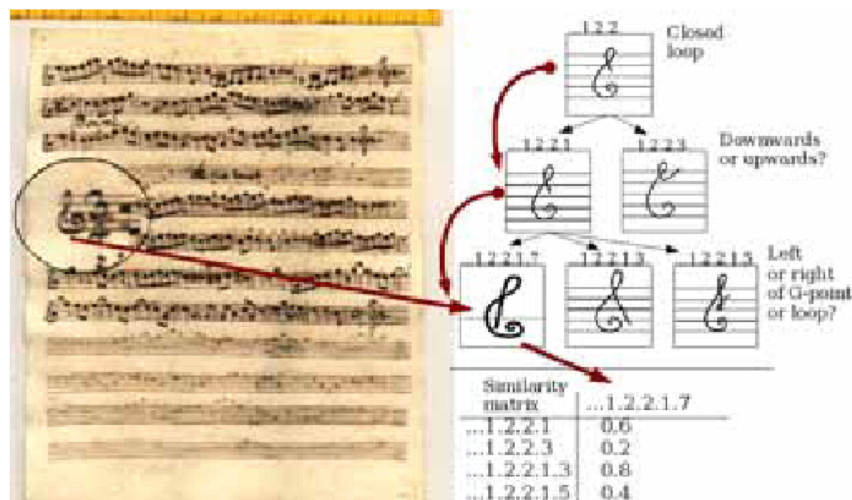


Abbildung 1.1: Ablauf der manuellen Handschriftenanalyse [GVK+03]

sämtliche Attribute sowie deren genaue Bedeutung wird an dieser Stelle allerdings nicht näher eingegangen. Hierzu wird auf die entsprechende Fachliteratur [eNo05, GVK+03] verwiesen.

DICT Das Schema Dict dient zur Definition von Tabellen, in welchen Handschriftenmerkmale gespeichert werden. Durch diese Tabellen wird die Baumstruktur der Feature Base, welche durch die Charakteristik der Notenhandschriften induziert wird, repräsentiert. Zentrum dieses Schemas bildet die Relation *Nodes*. Hierin werden werden einerseits Attribute wie beispielsweise Nummer, Code oder Beschreibung gespeichert, andererseits wird eine Referenz auf den zugehörigen Vaterknoten abgelegt, wodurch die Baumstruktur entsteht. Zusätzlich wird in der Tabelle *Node_Type* der Typ jedes einzelnen Knotens, wie beispielsweise *feature prefix* oder *value*, gespeichert. Um nun auch Feature Vektoren vergleichen zu können, werden in der Tabelle *Distances* die Distanzmatrizen gespeichert, die zur Berechnung der Abstände zwischen zwei Wertepaaren verwendet werden.

Features Dieses Schema der eNoteHistory-Datenbank dient zur Speicherung der Ergebnisse der Handschriftenanalyse. Das Ergebnis einer solchen Analyse ist stets ein Feature-Vektor, der eine ausgewählte Menge der Handschriftenmerkmale vereint. Ein Feature-Vektor kann nun sowohl zu einem oder mehreren Musik-Manuskripten, als auch zu einem oder mehreren Schreibern zugeordnet werden (vergl. Schema *Metadata*). Zur Speicherung der Ergebnisse werden nun drei Relationen verwendet. Die Tabelle *Featurevectors* speichert dabei die Feature-Vektoren, die Tabelle *FVValues* speichert die ermittelten Werte eines Features und ordnet sie den korrespondierenden Vektoren zu. Durch die Tabelle *MMS_VF* wird abschließend die Zuordnung von Feature-Vektoren zu Musikmanuskripten erfasst.

Metadata Das Schema Metadata bildet den Kernpunkt der Datenbank, da hierin sämtliche, erfassten Manuskripte, Arbeiten, Komponisten und Kopisten abgespeichert werden. Da dieses Schema auch dementsprechend groß ist, wird an dieser Stelle nur Stichpunktartig der Aufbau wiedergegeben. Einstiegspunkte in das Schema bilden die Klassen Musikmanuskript, Musikstück, Komponist, Textautor und Schreiber. Aus diesen und weiteren Klassen wird dann der folgende Aufbau bestimmt:

- einem Musikstück werden Textautor und Komponist zugeordnet
- Musikstücke werden Manuskripten zugeordnet
- einem Manuskript werden zugehörige Bibliothek und Kollektion zugeordnet
- ein Manuskript besteht aus Einleitung und Sektionen
- einer Einleitung wird ein Typ zugeordnet
- einer Sektion wird ein Typ zugeordnet
- eine Sektion besteht aus Seiten
- Seiten werden Schreibern zugeordnet
- einer Seite wird ein Digitalisat zugeordnet[2]

IPFV In diesem Schema werden nun die relevanten Bildteile eines Digitalisats abgespeichert, welche stets durch das umschließende Rechteck beschrieben werden[3]. Die zentrale Relation dieses Schemas beschreibt dabei die *Region Of Interest* (kurz ROI) des Bildes. Dabei wird das zu analysierende Notensystem umrahmt, und somit der Rand aus der Analyse ausgegrenzt. Innerhalb dieser ROI werden dann weiterhin die Primitive Notenkopf, Notenhals, Notenlinien und Taktstriche durch umschließende Rechtecke beschrieben, und den jeweiligen Tabellen abgespeichert.

Das Ziel dieses Projekts liegt nun einerseits darin, real existierende Notenhandschriften zu digitalisieren, und mit Metadaten zu annotieren, so dass aus diesen „normalen“ Dokumenten multimediale Dokumente werden, andererseits wird ein Retrieval zur Berechnung von Ähnlichkeiten der Schriftmerkmale bereitgestellt, wodurch der zugehörige Schreiber ermittelt werden kann. Hinsichtlich der prototypischen Implementierung in Kapitel 5 ist hauptsächlich der erste Teil von Interesse. So wird der bereitgestellte Bestand multimedialer Dokumente dazu verwendet, um auf ihnen eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt zu realisieren. Aber auch die Integration bereits vorhandener Retrieval-Mechanismen in die kombinierte Anfrageverarbeitung ist ein Aspekt dieser Arbeit, auf welchen beispielsweise in Kapitel 6 eingegangen wird.

[2] in der existierenden eNoteHistory-Datenbank ist diese Tabelle im Separaten Schema *Images* abgelegt

[3] ein Rechteck wird durch einen Punkt, Höhe, Breite und Drehwinkel definiert

1.3 Aufbau der Arbeit

Die theoretischen Grundlagen, die für das Verständnis dieser Arbeit notwendig sind, werden in **Kapitel 2** erläutert. Dieses beschäftigt sich mit den aktuellen Standards der *Structured Query Language*, kurz SQL. Dabei werden zum einen die allgemeinen Datenbankstandards SQL:1999 und SQL:2003 grundlegend vorgestellt, zum anderen wird die multimediale Erweiterung dieser Sprache, der Standard SQL/MM in geeignetem Umfang beschrieben.

Kapitel 3 beschreibt daraufhin ausführlich die Konzeption der in dieser Arbeit untersuchten kombinierten Anfrageverarbeitung auf Struktur und Inhalt multimedialer Dokumente. Dieses Kapitel bildet somit den Kernpunkt dieser Arbeit. Da allerdings auch existierende Standards nicht immer vollständig in etablierten Datenbankmanagementsystemen umgesetzt werden, untersucht **Kapitel 4** die praktische Relevanz der Konzeption, wie also die standardgestützte, kombinierte Anfrageverarbeitung in bestehenden Datenbankmanagementsystemen umgesetzt werden kann. Ausgehend von dieser Untersuchung erfolgt daraufhin eine prototypische Implementierung im Rahmen des eNoteHistory-Dokumentenbestandes, welche in **Kapitel 5** vorgestellt wird.

Abschließend enthält **Kapitel 6** zum einen eine kurze Zusammenfassung, in der die vorgestellte Arbeit rekapituliert wird, zum anderen wird ein Ausblick über mögliche zukünftige Entwicklungen gegeben, die im Rahmen der Weiterentwicklung dieser Arbeit von Interesse sind.

Kapitel 2

SQL-Standards

Die *Structured Query Language*, kurz SQL, ist seit den späten achtziger Jahren eine der wichtigsten Grundlagennormen für Datenbankanwendungen weltweit. Seit ihrem ersten Erscheinen im Jahr 1987 im Arbeitsausschuss JTC 1/SC 32 „Datenmanagement und Datenaustausch“ des internationalen Kommitees ISO/IEC JTC 1 „Informationstechnik“ wurde sie kontinuierlich weiterentwickelt [OP04].

In diesem Kapitel wird nun das für diese Arbeit notwendige Grundlagenwissen vorgestellt. Dabei unterteilt sich dieses Kapitel in zwei Teile. Im ersten Teil werden die seit SQL:1999 und SQL:2003 neu eingeführten Möglichkeiten vorgestellt und erläutert. Der zweite Teil befasst sich mit dem Standard SQL/MM, durch welchen die multimedialen Erweiterungen von SQL definiert werden. Dieser frührere Part 6 des SQL-Standards wurde unter dem Namen *Multimedia and Application Packages* ausgegliedert und als eingenständiger Standard verabschiedet. Auf eine Einführung in sämtliche Möglichkeiten von SQL:1999/2003 und SQL/MM wird an dieser Stelle verzichtet. Vielmehr werden die Datentypen, Methoden und Konzepte vorgestellt, die im weiteren Verlauf dieser Arbeit von Bedeutung sind.

2.1 SQL:1999/2003

Der 1999 in Kraft getretene Standard SQL:1999 verfolgte die bereits in SQL:1992 nachträglich eingeführte Idee imkrementeller Teilnormen. Es existiert somit kein kompletter monolithischer Standard, der regelmäßig komplett überarbeitet und veröffentlicht werden muss. Stattdessen wird die Norm in mehrere Teile aufgeteilt, so dass unter anderem vernünftige Zeitpläne für deren Weiterentwicklung erstellt werden konnten. Zunächst bestand die Norm aus fünf Teilen, wurde jedoch später um weitere Teile ergänzt. Die aktuelle Version von SQL, SQL:2003, behält diese Struktur grundlegend bei, erweitert sie jedoch um das für diese Arbeit sehr wichtige XML-Package. Da SQL:1999 sehr viel neues Material enthielt, dessen Spezifikation sich später als schwierig oder gar falsch erwies, wurde in der aktuellen Revision sehr viel Wert auf Fehlerkorrektur gelegt, weniger auf die Erweiterung

des Funktionsumfangs. Daher bezieht sich ein Großteil der kommenden Abschnitte bereits auf SQL:1999, auf relevante Änderungen zu SQL:2003 wird gegebenfalls eingegangen. Der Aufbau des aktuellen Standards wird in *SQL/Framework* [ISO03b] definiert, und besitzt folgende Form:

- *Part 1: Framework* (SQL/Framework) beschreibt den generellen Aufbau des Standards und Beziehungen der Teile, und enthält Begriffe und Definitionen
- *Part 2: Foundation* (SQL/Foundation) definiert Syntax und Semantik von SQL
- *Part 3: Call-Level Interface* (SQL/CLI) spezifiziert Strukturen und Prozeduren zum Aufrufen von SQL-Befehlen aus Anwendungsprogrammen heraus
- *Part 4: Persistent Stored Modules* (SQL/PSM) definiert Syntax und Semantik gespeicherter Prozeduren
- *Part 9: Management of External Data* (SQL/MED) spezifiziert die Integration und Verwendung extern gespeicherter Daten
- *Part 10: Object Language Bindings* (SQL/OLB) definiert Erweiterungen für zur Unterstützung der Integration von SQL-Statements in Java
- *Part 11: Information and Definition Schemas* (SQL/Schemata) spezifiziert das Definitions- und das Informationsschema
- *Part 13: SQL Routines and Types Using the Java^{TM}Programming Language* (SQL/JRT) beschreibt die Integration von Java-Methoden und Klassen
- *Part 14: XML-Related Specifications* (SQL/XML) beschreibt, wie SQL in Verbindung mit XML verwenet wird

Hinsichtlich der hier vorgestellten Arbeit sind lediglich die Teile 2, 9 und 14 von Belang. Nachfolgend werden somit zum einen neue und wichtige Konstrukte aus SQL/Foundation (auch Kern-SQL genannt) vorgestellt, zum anderen wird auf die Teile SQL/MED und SQL/XML genauer eingegangen.

2.1.1 SQL/Foundation

Ein grundlegender Schwerpunkt bei der Entwicklung von SQL:1999 war die Erweiterung der Sprache um objektrelationale Elemente. Somit wurden benutzerdefinierte Typen (**UDTs**) eingeführt, die in die Klassen "Ausgeprägter Typ" (*distinct type*) und "Strukturierter Typ" unterteilt werden. Ein **ausgeprägter Typ** entspricht dabei einer Kopie eines Basisdatentyps, der unter einem neuen Namen abgelegt wird[1]. Hierdurch kann beispielsweise ein Datentyp *Euro* durch folgenden Befehl erzeugt, und als Datentyp eines Attributs verwendet werden:

```
CREATE TYPE Euro AS DEZIMAL(12,2) FINAL
```

[1] Ausgeprägt deshalb, weil ein Vergleich oder eine Zuweisung zwischen ausgeprägtem und Quelldatentyp nicht erlaubt ist

Wichtiger im Hinblick auf die Ojektrelationalität ist jedoch die Definition eines **strukturierten Typs**. Dieser entspricht im Wesentlichen einem Objekttyp mit Attributen und Methoden. Weiterhin ist ein strukturierter Typ vollständig vergkapselt, so dass außerhalb seiner Typdefinition sein Verhalten, nicht aber die Implementierung der Attribute und Methoden sichtbar ist. Dies unterscheidet einen strukturierten Typ von einem Zeilentyp. Die Definition eines strukturierten Typs soll nun durch folgendes Beispiel verdeutlicht werden:

```
CREATE TYPE Person_T AS (
   Name VARCHAR(30),
   Anschrift AdressTyp,
   Gehalt Euro,
   Telefone VARCHAR(20) ARRAY[3]
)
NOT FINAL
REF IS SYSTEM GENERATED
METHOD Wohnort() RETURNS AdressTyp
```

Wie an diesem Beispiel gezeigt wird, können ausgeprägte und strukturierte Typen überall dort verwendet werden, wo ein Basisdatentyp erwartet wird. Weiterhin lassen sich nun strukturierte Typen auch in einer Subtyp-Supertyp-Beziehungen, also einer Typhierarchie anlegen. Dies geschieht durch ein UNDER-Prädikat hinter dem Typnamen des CREATE TYPE-Befehls. Allerdings ist in SQL keine Mehrfachvererbung erlaubt, so dass nur ein Typ in der UNDER-Klausel referenziert werden kann. Zusätzlich zu der Möglichkeit strukturierte Typen als Datentyp zu verwenden, kann dieser auch mit einer Tabelle assoziiert werden, die dann einer typisierten Tabelle entspricht. Hierbei wird zum einen jedes Attribut des Typs auf eine Spalte der Tabelle abgebildet, zum anderen wird implizit eine ID-Spalte erzeugt, durch die ein Identifizierer für jede Zeile gespeichert wird. Analog zu strukturierten Typen können auch diese strukturierten Tabellen in einer Tabellenhierarchie angeordnet werden, wie folgendes Beispiel verdeutlicht:

```
CREATE TABLE Student OF Student_T
  UNDER Person ( REF IS oid SYSTEM GENERATED )
```

Hier wird eine Tabelle Student vom Typ Student_T, der Subtyp von Person_T sein soll, als Subtabelle der Tabelle Person angelegt. Integritätsbedingungen an Attribute, auch Constraints genannt, werden dabei immer in der Tabellendefinition, nicht in der Typdefinition angegeben. Werden nun Anfragen an die Tabelle Person gestellt, werden automatisch sämtliche Elemente der Student-Relation mituntersucht. Soll dies verhindert werden, kann durch Verwendung des Prädikats ONLY in der From-Klausel eine Relation auf die flache Extension eingeschränkt werden. Eine Anfrage der Form `FROM ONLY Person` untersucht somit nur Personen, keine Studenten. Wird nun eine solche typisierte Tabelle in der From-Klausel referenziert, kann auf das Ergebnis auch eine definierte Methode mittels Punktnotation gebunden werden. Hierfür muss jedoch das Ergebnis erst mittels DEREF derefenziert werden, um aus den einzelnen Objektwertattributen einen Objektwert (Instanz des strukturierten Typs) zu erzeugen, wie folgendes Beispiel zeigt:

```
SELECT DEREF(oid).Wohnort()
FROM Person
```

Zusätzlich zu den strukturierten und ausgeprägten Datentypen wurden nun auch zu den bereits existierenden die neuen Typen BOOLEAN, BLOB, CLOB, BIGINT und XML, sowie die Typkonstruktoren REF, ROW, ARRAY und MULTISET hinzugefügt, wobei BIGINT, XML und MULTISET erst in SQL:2003 definiert wurden. Tabelle 2.1 zeigt die nun in SQL verfügbaren Datentypen auf. Attribute können nun zum einen durch Basisdatentypen, ausgeprägte oder strukturierte Typen definiert werden, zum anderen durch die Typkonstruktoren anonym strukturiert (ROW), als Referenz (REF), Liste (ARRAY) oder Multimenge (MULTISET) spezifiziert werden[2]. Durch den Konstruktor **REF**

Typ	Instant	Beispielwert
BOOLEAN	Wahrheitswert	`TRUE`
SMALLINT	ganze Zahl	`1704`
INTEGER		
BIGINT		
DECIMAL(p,q)	Festkommazahl	`1003.44`
NUMERIC(p,q)		
FLOAT(p)	Fließkommazahl	`1.5E-4`
REAL		
DOUBLE PRECISION		
CHAR(q)	alphanumerische	`'SQL:1999 + Delta = SQL:2003'`
VARCHAR(q)	Zeichenkette	
CLOB		
BIT(q)	binäre	`B'11011011'`
BIT VARYING(q)	Zeichenkette	
BLOB		
DATE	Datum	`DATE'1997-06-19'`
TIME	Zeit	`TIME'11:30:49'`
TIMESTAMP	Zeitstempel	`TIMESTAMP'2002-08-23 14:15:00'`
INTERVAL	Zeitintervall	`INTERVAL'48'HOUR`
XML	XML-Wert	`<Titel>SQL:2003\<Titel>`

Tabelle 2.1: Basisdatentypen von SQL:1999 bzw. SQL:2003 [Tür03]

wird nun ein Referenztyp erzeugt, der auf einen strukturierten Typ verweist. Beispielsweise kann das Attribut *Vater* des Typs *Person* als Referenz auf eine weitere Person durch den Befehl `Vater REF(Person_t) [SCOPE(Person)]` definiert werden. Das Prädikat **SCOPE** verweist hierbei auf die Tabelle, die dem referenzierten Typ zugeordnet wurde, und wird nur in der Tabellendefinition, nicht in der Typdefinition verwendet. Seit SQL:1999 können Attribute und Referenzen auch listenwertig, seit SQL:2003 sogar multimengenwertig sein. Der Konstruktor **ARRAY** gibt hierbei an, dass das Attribut aus mehreren, geordneten Einträgen besteht, beispielsweise kann durch die Definition

[2]Ein reiner, mengenwertiger Konstruktor, wie beispielsweise SET, wird durch SQL nicht definiert

`Kinder REF(Person_T) ARRAY[5]` ein Attribut *Kinder* angelegt werden, dass aus maximal fünf Referenzen auf weitere Personen besteht. Zur Verwendung in Anfragen kann dann dieses Attribut entweder durch den UNNEST-Operator in der From-Klausel entschachtelt werden, oder das gesuchte Element wird durch Postitionsangabe ermittelt. Einfacher zu verwenden ist allerdings der Mengenkonstruktor **MULITSET**, da hierfür, im Gegensatz zu ARRAY, keine initiale Größe für die Anzahl angegeben werden muss. Allerdings ist ein Attribut, dass mit MULTISET deklariert wurde, multimengenwertig, so dass Einträge auch mehrfach auftreten können. Die Referenzierung in der From Klausel erfolgt analog zu ARRAY durch den UNNEST-Operator, wie die folgende Anfrage verdeutlicht.

```
SELECT Name, t.Telefon
FROM Person, UNNEST(Telefone)t(Telefon)
```

Eine weitere Möglichkeit der Attributdefinition bietet der Konstruktor **ROW**. Hierdurch wird in SQL die Möglichkeit bereitgestellt, Attribute ohne Verwendung strukturierter Typen zu strukturieren. Beispielsweise kann ein Attribut Anschrift, das aus Strasse, Ort und PLZ besteht, einerseits als UDT definiert, andererseits direkt in der Deklaration des UDT Person_T mittels ROW integriert werden. Der Zugriff auf die Teilattribute des anonym strukturierten Typs erfolgt dabei durch Punktnotation.

Eine weitere wichtige Neuerung hinsichtlich der Anfragemöglichkeiten bilden **rekursive Anfragen**, die seit SQL:1999 existieren. In [Tür03] erfolgt diesbezüglich eine hinreichende Untersuchung in Bezug auf Syntax, Sicherheit, Zyklenbehandlung und Suchstrategien. Daher wird an dieser Stelle nicht näher darauf eingegangen. Dennoch soll eine beispielhafte Anfrage, ebenfalls aus [Tür03], die Verwendung dieser mächtigen Funktionalität verdeutlichen, und diese Untersuchung von SQL/Foundation abschließen.

```
WITH RECURSIVE Erreichbar(Abflug, Ziel) AS (
   SELECT Abflug, Ziel
   FROM Flug
   WHERE Abflug = 'London'
  UNION ALL
   SELECT e.Abflug, f.Ziel
   FROM Erreichbar e, Flug f
   WHERE e.Ziel = f.Abflug
)
SELECT DISTINCT *
FROM Erreichbar
```

2.1.2 SQL/MED

Teil 9 des Standards, SQL/MED [ISO03e], beschreibt nun die Verwendung extern gespeicherter Daten durch SQL, und existiert seit SQL:1999, wurde jedoch in SQL:2003 stark

erweitert. So waren in SQL:1999 lediglich SELECT-FROM-Anfragen an extern gespeicherte Daten möglich. Jetzt besteht allerdings die Möglichkeit, weiterhin eine komplexe Where-Klausel in die Anfrageformulierung mit einzubeziehen. Die Integration von extern gespeicherten Daten kann dabei in zwei Klassen zerlegt werden, DATALINKS und FOREIGN-Integration.

Durch die Definition eines **Datalink** wird nun eine Dateireferenz in Form einer URL dauerhaft in der Datenbank gespeichert, wobei die Datei selbst auf dem ursprünglichen System verbleibt. Ein Datalink-Wert kann dabei nicht nur als Wert in einer Spalte auftreten, sondern auch als Attributwert eines UDT. In der Deklaration eines Attributs vom Typ Datalink wird dabei durch eine Vielzahl von möglichen Optionen spezifiziert, wie streng die Datenbank die entsprechende Datei kontrolliert. Die Möglichkeiten reichen dabei von keiner Kontrolle, wobei die Datei nicht einmal existieren muss, bis hin zu völliger Kontrolle, wodurch spezifiziert wird, wer was lesen und schreiben darf, und ein Löschen des Datalink-Wertes in der Datenbank zum physikalischen Löschen der Datei führt. Diese Zugriffskontrolle wird durch einen Datalinker realisiert, der sich über dem Dateisystem befindet und jeden Zugriff auf die entsprechende Datei abfängt, und auf erforderliche Zugriffsrechte untersucht. Die nachfolgende Liste der Verbindungsoptionen stammt aus [Lin02]:

- *link control* (`NO LINK CONTROL` oder `FILE LINK CONTROL`)
- *integrity control option* (`ALL`, `SELECTIVE` oder `NONE`)
- *read permission option* (File-Server oder Datenbanksystem)
- *write permission option* (File-Server oder `BLOCKED`)
- *recovery option* (`NO` oder `YES`)
- *unlink option* (`RESTORE`, `DELETE` oder `NONE`)

Eine beispielhafte Attributdefinition ist in Kapitel 3 gegeben. Ein Nachteil des Datalink-Konzepts ist allerdings, dass nur die Datei, nicht aber dessen Inhalt in der Datenbank verwendet werden kann. Diesen Schwachpunkt hebt die zweite Variante der Datenintegration auf.

Mächtiger als die Definition eines Datalinks ist die Integration der Daten durch das Foreign-Prinzip. Dieses besteht grundlegend aus einem Foreign Server, der die externen Daten enthält, einem Foreign Table, der eine objektrelationale Sichtweise auf diese Daten in der Datenbank darstellt, und dem Foreign-Data Wrapper, der die Abbildung der externen Daten auf die objektrelationale Sichtweise realisiert. Der durch diese drei Komponenten definierte Aufbau von SQL/MED ist in Abbildung 2.1 dargestellt.

Ein **Foreign Table** realisiert nun eine transparente, objektrelationale Sicht auf die extern gespeicherten Daten, die auch in nichtrelationaler Form vorliegen können. Dadurch können Daten als Relation in das Datenbanksystem eingebunden, und in SQL-Anfragen referenziert werden. Für einen Anwender ist es daher bezüglich der Anfrageformulierung

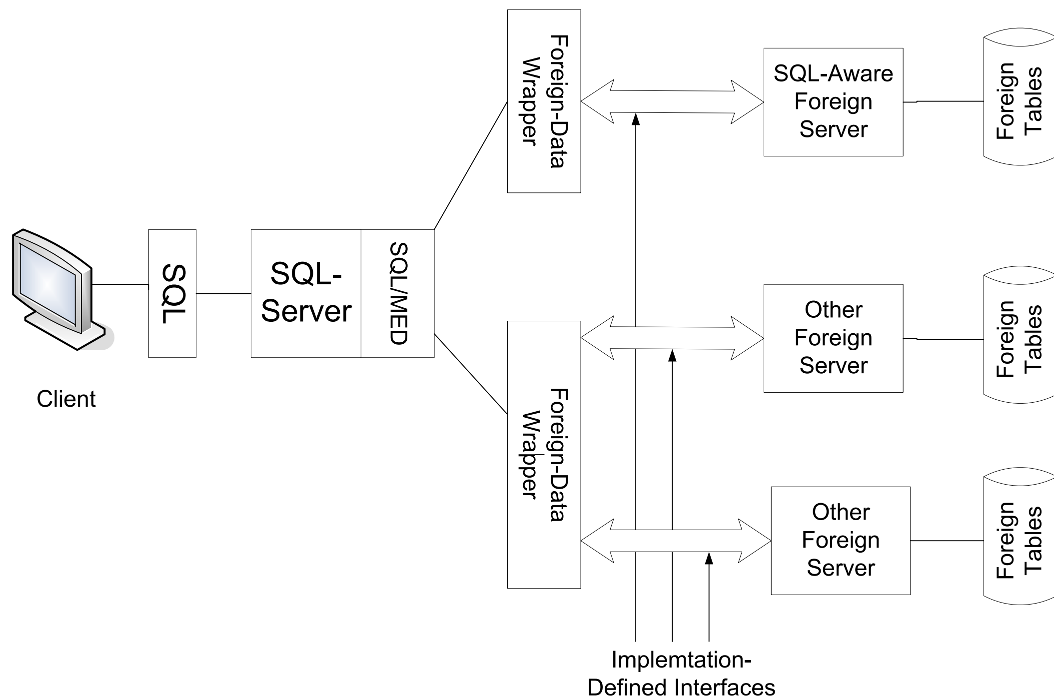


Abbildung 2.1: Aufbau einer SQL/MED-Umgebung

irrelevant, ob die in der Anfrage referenzierten Daten im Datenbanksystem, oder auf einem entfernten Rechner liegen. Dieses Prinzip wird als Verteilungstransparenz bezeichnet [HS99].

Der **Foreign Server** ist nun ein bekannter Server, der außerhalb der SQL-Umgebung externe Daten verwaltet. Hierbei kann unterschieden werden zwischen einem *SQL-aware foreign server*, der SQL-Teilanfragen verarbeiten kann, und einem *non-SQL-aware foreign server*, der keinerlei SQL-Fähigkeiten besitzt. Wird nun ein SQL-Server integriert, können die Schemainformationen über ein Import-Statement übernommen werden, andernfalls muss die vom Fremdserver angebotene Tabelle explizit durch ein CREATE FOREIGN TABLE-Statement definiert werden.

Das Verbindungsstück zwischen dem Foreign Server und dem Foreign Table bildet nun der **Foreign-Data Wrapper**, welcher es dem SQL-Server ermöglicht, auf die Daten des Fremdservers zuzugreifen. Hierbei wird jeder Fremdserver durch genau einen Wrapper angesprochen, ein Wrapper hingegen kann jedoch für mehrere Fremdserver genutzt werden. Wird nun ein Wrapper durch ein CREATE FOREIGN DATA WRAPPER-Statement angelegt, so kann die Funktionalität des Wrappers einerseits aus Methoden des SQL-Servers bestehen, andererseits kann durch eine LIBRARY-Klausel auch eine Bibliothek angegeben werden, in der die Abbildungs- und Transformationsvorschriften spezifiziert sind. Diese Vorschriften sind allerdings implementierungsabhängig.

Auf die Anfragebearbeitung in SQL/MED-Umgebungen wird ausführlich in [MMJ+01] eingegangen, daher wird hier nur ein grundlegender Überblick gegeben. Die Anfragebearbeitung wird nun aufgeteilt in Anfrageplanung und Anfrageausführung. Die **Anfrageplanung**, in welcher der Wrapper und der SQL-Server einen Ausführungsplan für die Teilanfrage erzeugen, basiert auf einem Request/Reply-Paradigma. Zunächst erzeugt der SQL-Server ein Request, das die entsprechende Anfrage repräsentiert. Dieses analysiert der Wrapper und schickt an den SQL-Server den Teil des Requests zurück, der vom Fremdserver nicht durchgeführt werden kann. Diese Teile müssen dann vom SQL-Server kompensiert werden. Welche Anteile nun vom Fremdserver ausgeführt werden können, ermittelt der Wrapper durch eine ausgeprägte Kommunikation mit dem Fremdserver, deren genauer Ablauf in [MMJ+01] beschrieben ist. In der anschließenden **Anfrageausführung** wird dann der Anteil der Anfrage, den der Fremdserver bearbeiten kann, an diesen weitergeleitet. Das resultierende Ergebnis wird dann unter Verwendung eines Iterators durchlaufen und an den SQL-Server gesendet.

2.1.3 SQL/XML

Die extensible Markup Language, kurz XML [W3C03], hat sich in kurzer Zeit als Dokumentenformat für den Austausch von Daten etabliert. Somit musste auch das SQL-Normierungsgremium auf diese neue Anforderung reagieren und brachte mit SQL/XML [ISO03c] eine Teilnorm, die eine integrierte Verwendung von SQL und XML vorsieht. Diese Teilnorm brachte zum einen den neuen Basisdatentyp XML, zum anderen eine Reihe von Methoden zum Generieren von XML-Dokumenten sowohl aus Werten herkömmlicher Datentypen, als auch aus Werten des neuen Datentyps XML [Tür03]. Auf das Mapping-Verfahren, dass einen Großteil der Teilnorm ausmacht, soll an dieser Stelle jedoch nicht eingegangen werden. Stattdessen werden, basierend auf [Tür03], einige Methoden zur Verwendung von XML in Anfragen vorgestellt.

Die Einführung des Basisdatentyps XML ermöglicht nun das Speichern und Verarbeiten von XML-Werten in einer SQL-Datenbank. Der definierte Wertebereich besteht dabei aus den folgenden drei Möglichkeiten:

- NULL
- XML-Inhalt (XML-Element oder Wald von XML-Elementen)
- XML-Dokument (XML-Inahlt mit Vorspann)

Weiterhin wird zur Verarbeitung XML-wertiger Attribute eine Reihe von Methoden bereitgestellt, die in [Tür03] vorgestellt werden. Kernfunktion ist dabei die Funktion **XMLGEN**, die allerdings nicht in [ISO03c] zu finden ist[3]. Dennoch wird sie auch hier als wichtige Funktion verwendet. Die folgende Liste zeigt nun die verwendbaren Funktionen auf:

[3] [ISO03c] ist allerdings auch nur ein Working Draft

- **XMLGEN** generiert einen XML-Wert basierend auf einer Anfrage, die in der XML-Anfragesprache XQuery 1.0 [W3C04b] formuliert wird.
- **XMLELEMENT** erzeugt ein XML-Element aus einer Werteliste.
- **XMLFOREST** generiert aus einer Sequenz von beliebigen Werten einen Wald von XML-Elementen.
- **XMLCONCAT** konkateniert mehrere XML-Elementen.
- **XMLAGG** aggregiert die XML-Elemente einer Gruppe von Zeilen.
- **XMLSERIALIZE** überführt ein XML-Wert in einen spezifizierten Datentyp.

Wie bereits erwähnt erzeugt die Methode XMLGEN einen XML-Wert mit Hilfe eines XQuery-Ausdrucks, der die Anfrage enthält. Innerhalb der XQuery-Anfrage können nun Attribute, der in der From-Klausel referenzierten Tabelle, durch ein vorangestelltes $-Zeichen integriert werden. Soll nun ein XML-wertiges Attribut, ob erzeugt oder bereits in der Relation vorhanden, auf bestimmte Selektionsbedingungen hin untersucht werden, wird von SQL die Funktion **XMLSERIALIZE** angeboten, welche einen XML-Wert in ein angegebenen Datentyp überführt. Nachfolgend wird nun eine Beispielanfrage vorgestellt, wodurch die Verwendung von XML-wertigen Attributen illustriert werden soll.

```
SELECT XMLGEN('<Gehalt Name ="{$Bewerbung/Vorname/text()}
                            {$Bewerbung/Nachname/text()}">
                  <Euro>{12*$Gehalt}</Euro>
               </Gehalt> ') AS Mitarbeitergehalt
FROM Angestellte
WHERE XMLSERIALIZE(
         CONTENT XMLGEN($Mitarbeitergehalt/@Name)
         AS VARCHAR(100)) = 'John Doe'
```

In dieser Anfrage wird zunächst der Inhalt des XML-wertigen Attributs *Bewerbung* extrahiert und in einer Variablen *Mitarbeitergehalt* zusammen mit dem "klassischen" Attribut *Gehalt* abgelegt. Diese wird dann durch die Funktion XMLSERIALIZE in ein Varchar überführt und kann daraufhin auf eine Selektionsbedingung hin untersucht werden. Diese grundlegende Untersuchung von SQL/XML soll an dieser Stelle ausreichend sein, da weiteren Konstrukten, Methoden und Mapping-Vorschriften im weiteren Verlauf dieser Arbeit keine Bedeutung zukommt.

2.2 SQL/MM

In diesem Abschnitt wird nun der Standard *Multimedia and Application Packages*, kurz SQL/MM, vorgestellt, welcher analog zum SQL-Standard ebenfalls aus mehreren Teilen besteht. Durch diesen Standard wird es ermöglicht, die in Kapitel 3 beschriebenen Inhalte multimedialer Dokumente zu verarbeiten. Der genaue Aufbau von SQL/MM besitzt hierbei folgende Form:

- *Part 1: Framework* beschreibt, wie auch in Abschnitt 2.1 den generellen Aufbau des Multimedia-Standards [ISO02]
- *Part 2: Full Text* beschreibt und definiert die Verarbeitung von Volltextobjekten
- *Part 3: Spatial* spezifiziert ein Anwendungspaket für die Verarbeitung von geometrischen Objekten
- *Part 5: Still Image* definiert die Verarbeitung von Bildobjekten
- *Part 6: Data Mining* stellt ein Anwendungspaket für Data Mining bereit

Da Part 1 lediglich den Aufbau des Standards definiert und Part 6 im Rahmen dieser Arbeit keine Rolle spielt, wird nun nachfolgend auf den Aufbau und die Verarbeitung von räumlichen, Bild- und Volltext-Objekten genauer eingegangen.

2.2.1 Full-Text

Die früheren Möglichkeiten von SQL, Textobjekte zu vergleichen, lagen in exakten Operatoren wie der Test auf Gleichheit, die einzigen unscharfen Operator waren LIKE und der in SQL:1999 neu hinzugekommene SIMILAR-Operator. Diese Möglichkeiten genügten jedoch nicht mehr den Anforderungen einer effektiven und effizienten Suche nach Textdokumenten in Datenbanken [Tür03]. Daher wurde der strukturierte Typ FullText[ISO01a] definiert, welcher höhere Operationen und strukturelle Einheiten beinhaltet. Der Aufbau des Typs FullText wird dabei durch folgenden DDL-Befehl beschrieben:

```
CREATE TYPE FullText AS {
   Contents CHARACTER VARYING(FT_MaxTextLength),
   Language CHARACTER VARYING(FT_MaxLanguageLength)
      DEFAULT FT_DefaultLanguage
}
INSTANTIABLE
NOT FINAL
--weitere Definitionen
```

Ein Objekt dieses Typs besteht somit aus einem textuellen Inhalt, sowie einer assoziierten Sprache. Das Textmodell dieses Datentyps, also die oben erwähnten strukturellen Einheiten, besteht aus Worten, Sätzen und Absätzen, wobei ein Satz aus einem oder mehreren Worten und ein Absatz aus einem oder mehreren Sätzen besteht. Das Erkennen dieser strukturellen Einheiten ist von sprachspezifischen Regeln, Konventionen und Heuristiken abhängig, die allesamt implementierungsabhängig sind [Tür03]. Um nun ein Objekt dieses Typs zu instanziieren, bietet SQL zwei Konstrukor-Methoden *FullText* an. Die erste Variante besitzt als Parameter zum einen den textuellen Inhalt, zum anderen die Sprache, mit der dieser Text assoziiert wird. Der zweite Konstrukor verwendet keinen Sprach-Parameter, statt dessen wird die standardmäßig eingestellte Sprache verwendet.

Um nun eine Suche über ein Volltextobjekt zu ermöglichen, stellt SQL/MM zwei Methoden zur Verfügung, die Methoden *Contains* und *Score*, die beide einen Parameter p vom Typ Pattern_Type[4] als Suchmuster übergeben bekommen. Der Unterschied dieser Methoden liegt nun darin, dass *Contains* einer boolschen Suche entspricht, das Ergebnis ist entweder 1 oder 0, *Score* hingegen einer linearen Suche, das Ergebnis ist ein Wert zwischen 0 und 1, je nachdem wie stark die Bedingung erfüllt wurde. Um nun die Score-Methode zusätzlich als Ranking-Funktion zu verwenden, beispielweise ausgehend von der Termhäufigkeit, kann diese Methode in der Select-Klausel einer SFW-Anfrage verwendet, benannt und anschließend in der Order By-Klausel referenziert werden. In einem Beispiel in [Mel03] wird dies durch folgende Anfrage illustriert:

```
SELECT stock_number, liner_notes.Score('"carpet"') as rank
FROM DVD_information
WHERE rank > 0
ORDER BY rank DESCENDING
```

Die Methoden Contains und Score werden nun mittels Punktnotation an ein Volltextobjekt gebunden, wobei als Parameter ein Suchmuster übergeben wird. Die Möglichkeiten dieser Suchmusterdefinition bilden den Kernpunkt dieses Packages, und werden in folgende Klassen eingeteilt, die weiterhin miteinander kombininiert werden können:

- Begriffs- und Phrasensuche
- Kontextsuche
- linguistische Suche
- Proximity-Suche
- Suchmusterexpansion

Begriffs- und Phrasensuche Durch dieses Suchmuster wird getestet, ob ein bestimmter Begriff, beziehungsweise eine bestimmte Phrase, im Text vorkommt. Eine Selektionsbedingung könnte zum Beispiel folgende Form besitzen:

```
Text.Contains('"Sein oder nicht Sein, das ist hier die Phrase"') = 1
```

Kontextsuche Anhand dieser Suchmethode kann ermittelt werden, ob bestimmte Worte in einem angegebenen Kontext bezüglich des Textmodells liegen. Die beiden Vertreter dieses Musters bilden die Konstrukte 'IN SAME SENTENCE AS' und 'IN SAME PARAGRAPH AS', wodurch ermittelt wird, ob zwei angegebene Suchmuster im selben Satz, beziehungsweise Absatz vorhanden sind.

[4] ein von Varchar abgeleiteter Distinct-Typ

Linguistische Suche Die linguistische Suche verwendet das vorhandene Wissen über die Stammwortreduktion. So werden beispielweise durch das Prädikat 'STEMMED FORM OF "Häuser"' zusätzlich zum Wort "Häuser" auch Wörter wie "Haus" oder "Häusern" in die Ergebnismenge mit aufgenommen. Somit kann diese Suchmusterklasse auch in die Suchmusterexpansion eingeordnet werden.

Proximity-Suche Die Proximity- oder Abstandssuche testet, ob bestimmte Begriffe innerhalb einer vordefinierten Enfernung existieren. Weiterhin kann spezifiziert werden, ob die Reihenfolge im Suchmuster relevant ist, wie das folgende Beispiel zeigt:

```
Text.Contains('"Anfrage" NEAR "Verarbeitung" WITHIN 2 WORDS IN ORDER')=1
```

Hierbei dürfen nun zwischen dem Wort "Anfrage" und dem Wort "Verarbeitung" maximal zwei weitere Wörter liegen, wobei die Reihenfolge der Suchwörter relevant ist. Statt der Einheit WORDS können auch die Einheiten CHARACTERS, SENTENCES und PARAGRAPHS spezifiziert werden. Ist die Suchreihenfolge irrelevant, so kann statt IN ORDER das Prädikat ANY ORDER verwendet werden.

Suchmusterexpansion Durch eine Expansion des Suchmusters wird eine Möglichkeit bereitgestellt, die Suche implizit durch verwandte oder ähnliche Wörter zu erweitern. Durch das Prädikat 'SOUNDS LIKE' wird eine phonetische Suche spezifiziert, es werden also ähnlich klingende Wörter gesucht. Das Prädikat 'FUZZY FORM OF' erlaubt es, ähnlich geschriebene Wörter zu suchen, so dass auch einzelne Rechtschreibfehler ignoriert werden können. Eine weitere Möglichkeit bildet das Prädikat 'IS ABOUT', wodurch eine thematische Suche spezifiziert wird. Somit können Volltexte ermittelt werden, die einer bestimmten Thematik zuzuordnen sind, beziehungsweise mit dieser in Verbindung stehen. Die vierte und mächtigste Möglichkeit bildet die Verwendung eines Thesaurus. Eine beispielhafte Anfrage mittels eines Thesaurus kann dabei folgende Form haben:

```
Text.Contains('THESAURUS "Informatik"
                EXPAND Which TERM OF "DB2"') = 1
```

Hierbei wird nun das Wort "DB2" unter Verwendung des Thesaurus für Informatik expandiert. Welche Expansionsart verwendet wird, muss durch den hier verwendeten Platzhalter "Which" spezifiziert werden. Hierbei können nun folgende Expansionen angegeben werden:

- BROADER bezieht Oberbegriffe in die Suche mit ein ("DB2" → "Datenbank", "IBM")
- NARROWER expandiert das Suchmuster um Unterbegriffe ("SQL" → "SQL/MM")
- SYNONYM erweitert um Synonyme ("SQL/MM" → "SQL Multimedia and Application Packages")

- PREFERED spezifiziert einen Diskriptor aus einer Menge von mehreren äquivlanten Ausdrücken (“Structured Query Language“ → “SQL“)
- RELATED ermittelt thematisch verwandte Wörter (“SQL“ → “DB2“)
- TOP ermittelt den Oberbegriff der Hierarchie (“SQL“ → “Computer Language“)

Diese hier vorgestellten Suchmuster können nun in Contains- und Score-Anfragen verwendet und kombininiert werden. Eine Kombination von Suchmustern erfolgt dabei durch die drei boolschen Operatoren Und ('&'), Oder (' | ') und Nicht ('NOT'). Weiterhin kann zu einem Suchmuster auch die Sprache angegeben werden, in der das Suchmuster vorliegt. Eine komplexe Selektionsbedingung könnte nun folgende Form haben:

```
Text.Score('STEMMED FORM OF GERMAN "Internationale Standards"
              IN SAME SENTENCE AS SOUNDS LIKE "Verarbeitung"
          | THESAURUS "Informatik"
                EXPAND RELATED TERM OF "SQL" NEAR
            THESAURUS "Biologie"
                EXPAND NARROWER TERM OF "Bioinformatik"
            WITHIN 4 WORDS ANY ORDER') > 0.7
```

2.2.2 Still Image

Durch das Erweiterungspaket SQL/MM-StillImage [ISO03a] werden nun Funktionalitäten zum Speichern und Bearbeiten von Bildern in Datenbanken bereitgestellt. Hierfür wird ein strukturierter Datentyp SI_StillImage definiert, welcher die Basis dieser Erweiterung darstellt. Der Aufbau dieses Typs wird hierbei durch folgenden DDL-Befehl beschrieben:

```
CREATE TYPE SI_StillImage AS (
   SI_content BINARY LARGE OBJECT(SI_MaxContentLength),
   SI_contentLength INTEGER,
   SI_reference DATALINK SI_DLLinkControl SI_DLIntegrityOption
      SI_DLReadPermission SI_DLWritePermission
      SI_DLRecoveryOption SI_DLUnlinkOption,
   SI_format CHARACTER VARYING(SI_MaxFormatLength),
   SI_height INTEGER,
   SI_width INTEGER,
   SI_retainFeatures SMALLINT
)
INSTANTIABLE
NOT FINAL
--weitere Definitionen
```

Das ursprüngliche Bild wird nun entweder in Binärform im Attribut *SI_Content* gespeichert oder es wird ein Datalink spezifiziert, welcher auf das Orginaldokument verweist, und dieses unter Kontrolle der Datenbank hält. Durch das Attribut *SI_retainFeatures* wird spezifiziert, ob die Features des Bildes berechnet und erhalten werden sollen. Wird das Attribut auf 1 gesetzt, werden die Features, die im weiteren Verlauf erläutert werden, berechnet. Weiterhin werden die Attribute Höhe und Breite, sowie das Formats des Bildes abgespeichert. Nach [Mel03] existiert eine Unmenge an Standards für Speicherung und Management von Bildern, allerdings nur einige wenige Standards - sowohl *de jure* als auch *de facto* - die eine Kodierung in das Format Still Image anbieten. Einige ausgewählte Formate sind in Tabelle 2.2 angegeben.

Format	Bedeutung
BMP	Bitmap
CGM	Copmupter Graphics Metafile
GIF	Graphics Interchange Format
JPEG (oder JPG)	Joint Photographics Experts Group
PCD	Photo Compact Disc
PNG	Portable Network Graphics
TIFF (oder TIF)	Tagged Image File Format
WMF	Windows Metafile Format

Tabelle 2.2: Formate für Still Images aus [Mel03]

Um nun Objekte vom Typ SI_StillImage instanziieren zu können, stellt SQL/MM eine Reihe von Konstrukor-Methoden zur Verfügung, die sich darin unterscheiden, ob ein BLOB oder eine Referenz zur Instanziierung verwendet wird, ob das Format explizit angegeben oder der Default-Wert verwendet wird und ob die Features zur weiteren Verwendung berechnet werden. Um nun die Attribute instanziierter Objekte zu ändern, besitzen diese Objekte folgende, teilweise überladene Methoden:

- *SI_setContent* ändert den Inahlt
- *SI_changeFormat* spezifiziert ein neues Format
- *SI_Resize* ändert Höhe und Breite des Bildes
- *SI_Scale* ändert ebenfalls Höhe und Breite des Bildes
- *SI_Rotate* rotiert das Bild um einen angegebenen Winkel
- *SI_InitFeatures* berechnet die Feature-Werte eines Bildes und setzt das Attribut *SI_retainFeatures* auf 1
- *SI_ClearFeatures* löscht die Feature-Werte und setzt das Attribt *SI_retainFeatures* auf 0

Weiterhin besitzt ein Objekt die Methoden *SI_Thumbnail* und *SI_Score*. Während die Methode *SI_Thumbnail* lediglich ein Vorschaubild mit geringer Auflösung erzeugt[5], dient die Score-Methode dazu, StillImage-Objekte auf Ähnlichkeit zu untersuchen. Die Ähnlichkeit bezieht sich hierbei auf folgende Features:

1. SI_AverageColor - Durchschnittsfarbwert
2. SI_ColorHistogram - Liste der Frequenzen aller Farbwerte
3. SI_PositionalColor - Ein Array der Durchschnittsfarbwerte für jedes Rechteck innerhalb des Bildes
4. SI_Texture - besteht aus coarseness (Größe von Bereichen mit gleichem Pixelinhalt), contrast (Helligkeitsunterschiede) und directionality (dominierende Richtung)
5. SI_FeatureList - gewichtete Kombination der vier vorangegangenen Features

Die Features 1. bis 4. entsprechen hierbei einfachen low-level-Features. Um nun diese Features zu berechnen, so dass sie anschließend als Parameter verwendet werden können, existieren korrespondierende Konstrukor-Methoden, die als Übergabeparameter ein Objekt vom Typ SI_StillImage besitzen. Um beispielweise ein Histogram des Bildes *example* zu erzeugen, wird der Befehl *SI_ColorHistogram(example)* verwendet. Soll nun kein einzelner Feature, sondern eine gewichtete Kombination mehrerer Features zum Vergleich herangezogen werden, so wird die Konstrukor-Methode *SI_FeatureList* verwendet. Diese Methode besitzt acht Parameter, die vier Feature-Werte, sowie eine Gewichtung pro Feature. Die Reihenfolge der Features entspricht dabei der oben angeführten Nummerierung.

Um nun ein Bild mit einem Feature-Wert bzw. einem Feature-Vektor zu vergleichen, bietet SQL mehrere Möglichkeiten. In der ersten Variante wird die Score-Methode des SI_StillImgage-Objekts verwendet und ein berechneter Feature(-Vektor) als Parameter übergeben. Die zweite Möglichkeit basiert auf der SI_Score-Methode, welche Features besitzen. Somit wird ein Feature erzeugt und an dessen SI_Score-Methode ein Objekt vom Typ SI_StillImage übergeben. Die dritte Variante basiert auf der hier nicht weiter vorgestellten Verwendung von Funktionen, die ausschließlich anhand der Übergabeparameter ein entsprechendes Ergebnis erzeugen. Die Verwendung dieser drei Varianten zeigen folgende drei Befehle, wobei *example* einem eingescannten Bild und *SIAttribut* einem Attribut einer Relation der Datenbank, beide vom Typ SI_StillImage, entsprechen.

```
example.SI_Score(SI_ColorHistgram(SIAttribut)) < 0.3
FeatureList(SIAttribut).SI_Score(example) < 0.3
SI_ScoreByTexture(SI_Texture(example), SIAttribut) < 0.3
```

Abschließend soll die Verwendung des Packages Still Image an einem Beispiel verdeutlicht werden. Nachfolgend existiere nun eine Relation Pictures mit einem Attribut *SIAttribut* vom Typ SI_StillImage, einem Attribut *Title* und eventuell weiteren Attributen, sowie

[5]Inwieweit die Implementierung dieser Methode von der der Scale-Methode abweicht, ist nicht bekannt [ISO03a]

ein eingescanntes Bild mit dem Namen „example“. Aus dem eingescannten Bild sollen nun die Features SI_PositionalColor und SI_Texture ermittelt werden, wobei die Textur doppelt so stark wie die positionalColor in die Berechnung eingehen soll. Aus der Relation Pictures sollen die Titel der Bilder ausgegeben werden, deren Ähnlichkeit bezüglich dieser Feature-Liste einen Wert kleiner als 0.7 ergibt[6]. Die zugehörige Anfrage könnte dann folgende Form besitzen:

```
SELECT Title
FROM Pictures
WHERE 0.7 > pic.SI_Score(
                   SI_FeatureList( SI_AverageColor(example),0.0,
                                   SI_ColorHistogram(example),0.0,
                                   SI_PositionalColor(example),0.33,
                                   SI_Texture(example),0.67)
```

2.2.3 Spatial

Das dritte hier vorgestellte Package des Standards SQL/MM stellt das Package Spatial [ISO01b] dar. Hiermit werden Datentypen, Methoden und Funktionen zum Speichern, Lesen und Verarbeiten von geometrischen Objekten definiert. Analog zu den beiden vorrangegangenen Multimedia-Packages steht ein neu eingeführter, strukturierter Datentyp im Mittelpunkt, auf welchen hin ausgerichtet neue Methoden definiert werden. Im Gegensatz zu den Packages Still Image und Full Text wird für geometrische Objekte allerdings eine Hierarchie von Typen aufgebaut. Der Supertyp für geometrische Objekte ist der abstrakte Typ ST_Geometry. Von diesem werden nun weitere, teilweise instanziierbare Typen abgeleitet. Die dadurch definierte Hierarchie dieser Datentypen ist in Abbildung 2.2 angegeben, wobei grau markierte Typen ebenfalls als nicht instanziierbar deklariert sind. Durch die in der Abbildung dargestellten Typen können nun nulldimensionale (ST_Point), eindimensionale (ST_Curve) und zweidimensionale Objekte (ST_Surface), als auch Mengen eines Typs (ST_GeomCollection) instanziiert, und als Attributwert verwendet werden. Instanzen eines Typs beziehen sich dabei immer auf ein Referenzsystem, wobei nur Objekte des selben Referenzsystems miteinander vergleichbar sind. Ein Referenzsystem gibt an, in welchem Abstand bezüglich eines Äquators eine Koordinate gesetzt wird. Die entsprechenden Einheiten sind dabei implementierungsabhängig. Hinsichtlich der Aufteilung eines multimedialen Dokuments in Kapitel 3 sind beispielsweise Pixel oder Zentimeter ein geeignetes Maß und der linke sowie der untere Rand ein geeigneter Äquator. Ein Rechteck könnte nach [Sto03] somit durch folgenden Befehl erzeugt werden, wobei '1' einem Verweis auf das Referenzsystem entspricht:

```
ST_Polygon('polygon((10 10, 10 20, 20 10, 10 10 )),1)
```

[6]Im Gegensatz zur Score-Methode aus SQL/MM Full Text bedeutet hier ein kleinerer Wert stärkere Ähnlichkeit

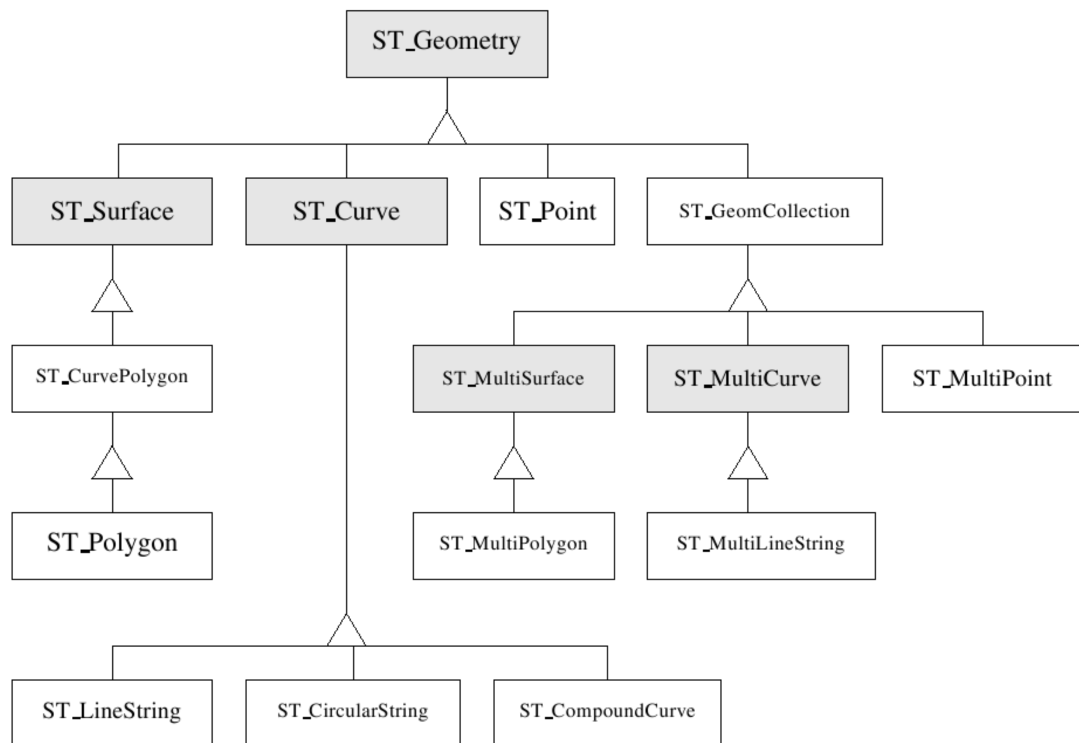


Abbildung 2.2: ST_Geometry-Typhierarchie [Sto03]

Zur Instanziierung wird nun von SQL für jeden instanziierbaren Typ eine Reihe von Konstrukor-Methoden bereitgestellt, welche grundlegend in drei Klassen eingeteilt werden können. Die erste Klasse beschreibt Konstrukor-Methoden, die ein geometrisches Objekt aus einem BLOB mittels der Methode *ST_WKBToSQL* erzeugen. Hierfür müssen sämtliche Informationen des Objekts in binärer Form kodiert vorliegen. Die zweite Klasse verwendet zur Instanziierung die Methode *ST_WKTToSQL*, wodurch ein geometrisches Objekt aus einem CLOB erzeugt wird. Die textuelle Repräsentation des Objekts wird dabei durch eine BNF beschrieben, die den Aufbau des CLOBs definiert. Die dritte Klasse enthält Konstrukor-Methoden, die ein geometrisches Objekt aus weiteren geometrischen Objekten erzeugen. Beispielsweise kann ein Objekt vom Typ ST_Polygon durch einen Konstrukor erzeugt werden, der ein Objekt vom Typ ST_LineString übergeben bekommt, ein Objekt vom Typ ST_LineString wiederum kann durch ein Array von Objekten des Typ ST_Point erzeugt werden, u.s.w. Zusammenfassend kann man sagen, dass die Instanziierung eines geometrischen Objekts darauf basiert, ein Objekt aus einer Menge weiterer Objekte zusammenzusetzen. Die Menge dieser Objekte kann dabei binär, textuell oder typisiert vorliegen.

Um nun erzeugte geometrische Objekte auch zu bearbeiten, abzufragen oder zu vergleichen, bietet SQL eine umfangreiche Menge von Methoden an. Diese können hier jedoch nicht alle vorgestellt werden, da allein der Supertyp ST_Geometry bereits 48 unterschiedliche besitzt, obwohl dieser nicht einmal instanziierbar ist. Diese Methoden werden nun an die jeweiligen Untertypen vererbt, so dass diese über eine noch größere Anzahl an Me-

thoden verfügen. Daher wird an dieser Stelle lediglich eine Liste ausgwählter Methoden gegeben, die von jedem Objekt mittels Punktnotation verwendet werden können.

- *ST_GeometryType* ermittelt den genauen Typ eines Objekts
- *ST_IsEmpty* untersucht, ob das das aufrufende geometrische Objekt leer ist
- *ST_Envelope* erzeugt das umschließende Rechteck
- *ST_Boundary* erzeugt den geometrischen Abschluss des Objekts
- *ST_Buffer* erzeugt eine Pufferzone um ein Objekt
- *ST_Intersection* erzeugt ein Objekt, dass dem Durchschnitt zweier Objekte entspricht
- *ST_Union* erzeugt ein Objekt, dass der Vereinigung zweier Objekte entspricht
- *SI_Difference* erzeugt ein Objekt, dass der der Differenz zweier Objekte entspricht
- *SI_SymmDifference* erzeugt ein Objekt, dass der der symmetrischen Differenz zweier Objekte entspricht
- *ST_Distance* ermittelt die miniale Entfernung zwischen zwei Objekten
- *ST_Equals* überprüft, ob zwei Objekte die selbe Punktmenge enthalten, also die symmetrische Differenz leer ist
- *ST_Disjoint* überprüft, ob zwei Objekte disjunkt sind
- *ST_Intersects* überprüft, ob zwei Objekte eine gemeinsame Schnittmenge besitzen
- *ST_Touches* überprüft, ob zwei Objekte einen Schnittpunkt besitzen
- *ST_Contains* testet, ob das aufrufende Objekt das Übergebene enthält

Diese grundlegende Einführung in das Spatial-Package soll an dieser Stelle als Grundlage zum Verständnis des Kapitel 3 ausreichend sein. Für weitere Informationen wird auf die Fachliteratur verwiesen. Beispielsweise erfolgt in [Sto03] eine ausführliche Einführung in das Multimedia-Package von SQL.

Kapitel 3

Konzeption der Anfrageverarbeitung

In diesem Kapitel wird eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt multimedialer Dokumente untersucht. Da jedoch eine Anfrageverarbeitung nicht losgelöst von der zugrunde liegenden Struktur der Dokumente und des Datenbanksystems betrachtet werden kann, sind im Vorfeld zuerst diese Themen zu untersuchen. Dieses Kapitel unterteilt sich nun nachfolgend in vier Aufgabenbereiche. Zu Beginn wird ein konzeptionelles Modell eines multimedialen Dokuments erstellt. Ausgehend von diesem konzeptionellen Modell wird im zweiten Teil der logische Aufbau eines Multimedia-Datenbanksystems beschrieben, in welchem die multimedialen Dokumente gespeichert werden. Im dritten Teil dieses Kapitels werden dann bezüglich der eingeführten Systemarchitektur die Anfragemöglichkeiten vorgestellt, auf deren Bearbeitung dann im vierten Teil genauer eingegangen wird.

3.1 Das Dokumentenmodell

Das hier vorgestellte Dokumentenmodell beschreibt den allgemeinen konzeptionellen Aufbau multimedialer Dokumente. Zur Beschreibung des Modells wird die Unified Modeling Language [Oes04], kurz UML, verwendet. Das diese zur Modellierung von multimedialen Dokumenten geeignet ist, wird durch eine diesbezügliche Untersuchung in [Czy05] bestätigt. Der Entwurf dieses Modells wurde bereits in Hinblick auf die Möglichkeiten von SQL:1999/2003 und SQL/MM durchgeführt und wird nun im Folgenden vorgestellt.

3.1.1 Allgemeiner Aufbau

Der allgemeine Aufbau multimedialer Dokumente ist in Abbildung 3.1 dargestellt. Grundlage hierfür ist das in [IB05] vorgestellte Framework für Applikationen von Bilddatenbanken, da es auf Grund des Abstraktionsniveaus allgemein verwendbar ist. Einstiegspunkt in das Modell bildet hierbei die Klasse **Collection**, die als Wurzelelement die Gesamtheit

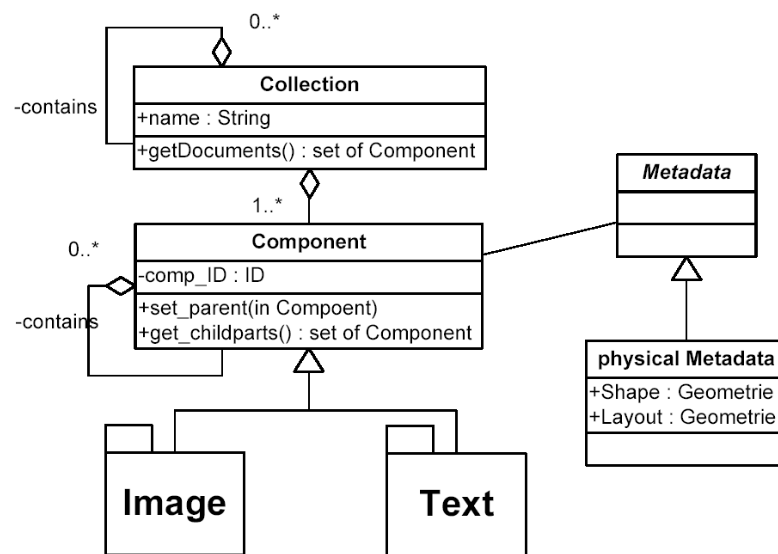


Abbildung 3.1: Konzeptioneller Aufbau multimedialer Dokumente

der der zugrunde liegenden Dokumente enthält. Da es im Hinblick auf konkrete Anwendungen unter Umständen sinnvoll ist, eine Collection in Teilkollektionen zu unterteilen, kann eine Collection wiederum aus Collections bestehen.

Die wichtigste Klasse zur Beschreibung eines Dokuments bildet die Klasse **Component**. Ein komplettes Dokument wird als Component-Objekt spezifiziert. Da ein Dokument aus Teildokumenten aufgebaut ist, die ebenfalls wieder als Dokumente betrachtet werden können, können analog Component-Objekte aus weiteren Component-Objekten bestehen. Ein wichtiges Mittel zur semantischen Beschreibung dieser Component-Objekte bildet die abstrakte Klasse **Metadata**, durch die spezielle Informationen bezüglich der jeweiligen Component-Objekte beschrieben werden. Diese Klasse wurde als abstrakt, also nicht direkt verwendbar spezifiziert, da die Wahl der jeweiligen Metadaten anwendungsabhängig ist. So ist es im Bereich der digitalen Bibliotheken von Vorteil die Standards METS[1] oder Dublin Core[2] zu verwenden. In anderen Szenarien sind andere Metadatenstandards wie Beispielsweise MPEG-7[3] von Bedeutung. Ein Spezialfall bildet allerdings die von Metadata abgeleitete Klasse **physical Metadata**. Hierdurch wird einerseits die visuelle Repräsentation, beziehungsweise die geometrische Form eines Component-Objekts durch das Attribut *Shape* beschrieben, andererseits wird durch das Attribut *Layout* definiert, wie die Subcomponents eines Component-Objekts angeordnet sind.

Component-Objekte, die nun nicht aus weiteren Component-Objekten bestehen, also als atomarer Dokumententeil spezifiziert wurden, sind demzufolge entweder Text- oder Bild-

[1]http://www.loc.gov
[2]http://dublincore.org
[3]http://www.chiariglione.org/mpeg/

Objekte. Dies wird durch die Spezialisierung der Klasse Component in die Packages **Text** und **Image** realisiert, die nun im Folgenden genauer beschrieben werden.

3.1.2 Package Image

Der konzeptionelle Aufbau eines Bild-Objekts wird in der Abbildung 3.2 dargestellt. Den Einstiegspunkt dieses Packages stellt die Klasse **Image** dar. Analog zu Dokumenten

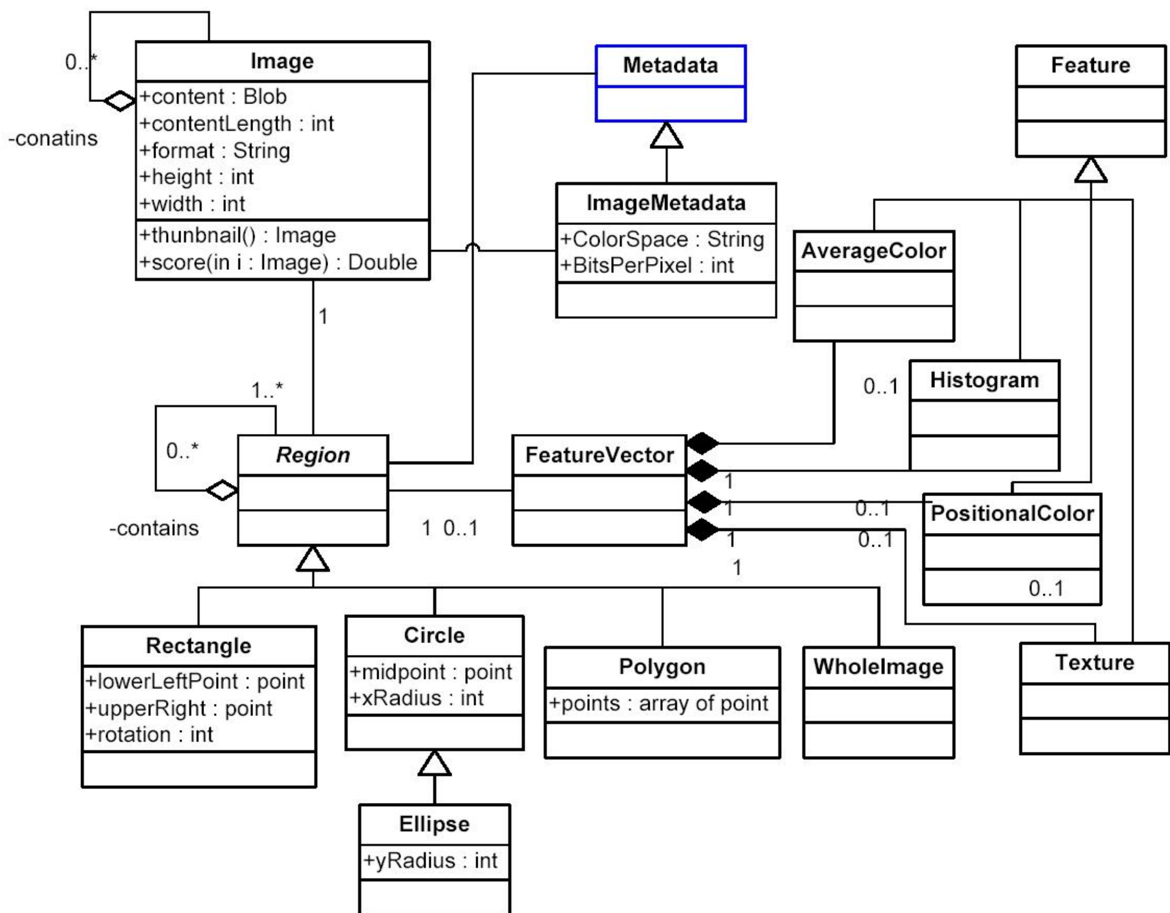


Abbildung 3.2: Konzeptioneller Aufbau eines Image-Objekts

können auch Bilder aus Teilbildern bestehen, die selbst wieder als Bild aufgefasst werden können. Diesen Bildern können nun zusätzlich zu den ererbten Metadaten und den angegebenen Attributen[4] weiterhin Informationen bezüglich Farbraum und Bits pro Pixel zugeordnet werden. Des weiteren stellt die Klasse Image zwei Methoden zur Verfügung, *score* und *thumbnail*. Hierbei liefert die Methode *thumbnail* eine Vorschau des Ausgangsbilds. *Score* gibt an, inwieweit zwei Bilder bezüglich der Features übereinstimmen.

[4]Die fünf Attribute ergeben sich direkt aus dem SQL/MM-Package SI_Still Image

Weiterhin wird einem Bild eine Menge von Regionen zugeordnet, die durch die abstrakte Klasse **Region** beschrieben werden. Die konkrete geometrische Form einer Region wird durch Spezialisierung zu einer der angegebenen Unterklassen bestimmt. Analog zu Bildern können auch Regionen Metadaten zugeordnet werden, beispielsweise der Name einer Person, die in der Region zu sehen ist. Des weiteren kann einer Region eine Kombination von vier low-level-features[5] zugeordnet werden. Die Kombination der vier Features wird durch die Klasse **FeatureVector** beschrieben, die eine Komposition der Features **histogram, average color, positional color** und **texture** darstellt.

3.1.3 Package Text

Der relativ einfache konzeptionelle Aufbau eines Text-Objekts wird in der Abbildung 3.3 dargestellt. Der Aufbau dieses Packages ergibt sich hierbei direkt aus dem SQL/MM-

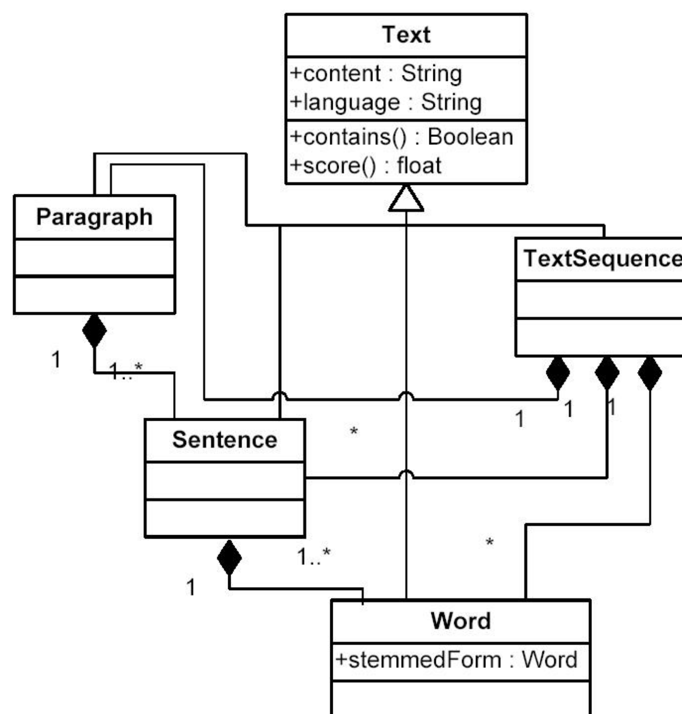


Abbildung 3.3: Konzeptioneller Aufbau eines Text-Objekts

Package Full Text. Ein Text-Objekt wird demnach spezialisiert zu einem Absatz (**Paragraph**), einem Satz (**Sentence**), einem Wort (**Word**) oder einer **Textsequenz**. Weiterhin besteht ein Absatz aus einem oder mehreren Sätzen, und ein Satz aus einem oder mehreren Worten. Im Gegensatz zu Satz und Absatz stellt die Klasse Textsequenz eine logisch nicht abgeschlossene Wortfolge dar. Wenn beispielsweise ein Satz in der Mitte getrennt wird, um eine Graphik einzufügen, ist keiner der beiden Teile einer der anderen Klassen zuzuordnen. Ein weiteres Beispiel wäre das Label eines Bildes, was ebenfalls kein

[5]Diese ergeben sich ebenfalls direkt aus dem SQL/MM-Package SI_Still Image

kompletter Satz ist. Weiterhin bietet ein Text-Objekt neben den Attributen *content* und *language* zwei Funktionen zum Vergleich von Text-Objekten an. Die Methode *contains* ermittelt hierbei, ob eine Bedingung erfüllt ist, die Methode *score* hingegen gibt zusätzlich an, wie stark eine Bedingung erfüllt ist. Die Klasse Word besitzt zusätzlich das Attribut *stemmedForm*, welches einer Stammwortreduktion des jeweiligen Wortes entspricht.

3.1.4 Komponentenbeziehungen

Um zusätzlich zum Inhalt der Dokumente auch Beziehungen zwischen einzelnen Component-Objekten in der Dokumentbeschreibung und dem späteren Retrieval verwenden zu können, werden fünf attributierte Assoziationen zur Unterteilung von Component-Beziehungen verwendet, die in Abbildung 3.4 dargestellt sind, und in dieser Form in [IB05] eingeführt wurden.

Abbildung 3.4: Components und ihre Beziehungen [IB05]

Mit Hilfe der Klasse **Spatial** können geometrische Beziehungen von Component-Objekten dargestellt werden. Dadurch, dass Component-Objekte das Attribut *Shape* besitzen, welches die geometrische Form des Objekts beschreibt, können sie mit anderen Component-Objekten in geometrische Beziehung gesetzt werden, was für eine strukturbezogene Anfrage von Bedeutung ist. Beispielsweise können somit visuelle Anfragen der Form „Das Bild links neben dem Text“ realisiert werden.

Ein weiterer wichtiger Bestandteil zur Beschreibung eines Dokuments bildet eine attributierte Assoziation mit der Klasse **Logical**. Durch diese Klasse ist es möglich, beispielsweise das eingeschränkte Textmodell aus Abschnitt 3.1.3 um anwendungsrelevante Teile zu erweitern. Im Bereich digitaler Bibliotheken ist es unter anderem notwendig, ein Buch nicht

nur in Worte, Sätze und Absätze zu unterteilen, sondern auch logische Komponenten wie Kapitel zu beschreiben. Dies wird durch diese Klasse ermöglicht.

Um nun weiterhin auch Hypertextstrukturen, wie sie beispielsweise in HTML verwendet werden, darzustellen, dient die Klasse **Interactional**. Hierbei werden Component-Objekte durch Links in Beziehung gestellt. Somit kann ein Coponent-Objekt auf ein anderes Component-Objekt verweisen, aber auch von mehreren anderen referenziert werden.

Die Klasse **Temporal** beschreibt hierbei temporale Beziehungen, wie beispielsweise Präsentationen oder Videodatein. Hierdurch kann man darstellen, dass beispielsweise ein Component-Objekt zeitlich vor einem anderen definiert wurde. Da sich diese Arbeit jedoch auf die diskreten Datentypen Bild und Text beschränkt, dient diese Klasse lediglich der Vollständigkeit des Modells, wird aber nicht weiter verwendet.

Durch die Klasse **Interpreted** können abgeleitete high-level-Beziehungen dargestellt werden, die durch eine Kombination der vier low-level-Beziehungen ermittelt werden können. Diese Klasse wurde in [IB05] jedoch als abstrakt deklariert, da die konkreten Analysefunktionen zur Interpretation anwendungsabhängig definiert werden müssen.

3.2 Aufbau des Multimedia-Datenbanksystems

Aufbauend auf dem in Abschnitt 3.1 beschriebenen, konzeptuellen Modell eines Multimediadokuments wird in diesem Abschnitt beschrieben, wie dieses Modell unter Verwendung von SQL:1999/2003 und SQL/MM auf einen logischen Datenbankentwurf abgebildet werden kann. Hierfür sind zwei Architekturen denkbar, auf welche das konzeptuelle Modell abgebildet werden kann. Die erste Variante besteht aus einem einfachen zentralen Datenbanksystem. Dieses hat zum einen den Vorteil, dass die Architektur komplett SQL:1999/2003-konform ist, und zum anderen die multimedialen Dokumente zentral verwaltet werden können. Die einzelnen Medien können dabei zentral unter Nutzung der SQL/MM-Datentypen SI_StillImage und FullText verarbeitet werden. Nachteilig ist jedoch, dass Funktionen, wie beispielsweise spezialisierte Verarbeitungsfunktionalitäten, effiziente Verarbeitung hoher Datenaufkommen oder Verwendung von Spezialhardware [Lin02] im Allgemeinen nur in speziellen Medienservern zu finden sind. Somit ist diese Variante im Allgemeinen weniger attraktiv.

Diesen Nachteil hebt die zweite Architekturvariante, die prinzipiell eine Erweiterung der ersten ist, auf. Die zweite Architektur, die in dieser Arbeit auch weiter verfolgt wird, basiert auf der Daten- und Funktionsintegration in ein objektrelationales Datenbanksystem. Die Daten- und Funktionsintegration erfolgt hierbei durch die Anbindung von spezialisierten Medienservern für Text und Bild durch die in Kapitel 2 beschriebenen **Foreign Data Server**. Die Daten dieser Fremdserver werden dabei im SQL-Server als **Foreign Tables** repräsentiert. Die Interaktion zwischen SQL- und Fremdserver erfolgt dabei durch die **Foreign-Data Wrapper**. Somit werden die oben erwähnten Nachteile des zentralen Ansatzes aufgehoben. Der genaue Aufbau dieser Architektur wird nun nachfolgend genauer beschrieben, wobei auf die Definition von zugehörigen Integritätsbedingungen,

wie beispielsweise Primärschlüssel und referenzielle Integrität, verzichtet wird. Eine diesbezügliche Betrachtung würde auf Grund des zusätzlichen Umfangs den Rahmen dieser Arbeit sprengen.

3.2.1 Speicherung der Dokumente

Zur Speicherung der Dokumente, sowie zur Rekonstruktion von kompletten als auch von Teildokumenten werden drei Klassen von typisierten Tabellen verwendet. Den ersten Teil, und auch Ausgangspunkt der Speicherung, bildet die Relation **Document**, deren Elemente vom Typ **Document_T** sind. Der CREATE-Befehl dieses Typs hat folgende Form:

```
CREATE TYPE Document_T AS (
   Document DATALINK FILE LINK CONTROL
             INTEGRETY ALL
             READ PERMISSION DB
             WRITE PERMISSION BLOCKED
             RECOVERY YES
             ON UNLINK RESTORE,
   DocTree XML,
   Metadata [XML | REF() | ROW()],
   Subcomponents REF(Component_T) MULTISET,
   Layout ST_GeomCollection,
   Content Integer)
```

Das Attribut *Document* stellt hierbei eine Referenz auf das Orginaldokument dar. Dieses Orginaldokument kann nun außerhalb der Datenbank gespeichert werden, unterliegt aber dennoch der Zugriffskontrolle des Systems, so dass Änderungen am Dokument überwacht werden und die Konsistenz der Daten gewährleistet werden kann. Das XML-Attribut *DocTree* stellt einen Dokumentenbaum dar, der durch die Hierarchie der Component-Objekte bestimmt wird. Die Motivation für dieses Attribut liegt zum einen darin, dass durch die mengenorientierte Speicherung der Component-Subcomponent-Beziehungen die relative Reihenfolge von Geschwistern verloren geht, zum anderen in der Verwendung bei unscharfen Anfragen (siehe Abschnitt 3.3.1.2). Anhand dieser Abstraktion eines multimedialen Dokuments ist es effizient möglich, relative Strukturen eines Dokuments zu untersuchen, ohne dieses teilweise oder komplett zu rekonstruieren. Der Inhalt des Attributs ist hierbei gültig bezüglich folgender DTD:

```
<!ELEMENT DocTree (Component*)>
<!ELEMENT Component (Component* | Mediatype)>
<!ATTLIST Component ComponentID ID #REQUIRED>
<!ELEMENT Mediatype (#PCDATA)>
<!ATTLIST Mediatype MediaID ID #REQUIRED>
```

Das Attribut *Metadata* enthält weitere Informationen über das Dokument. Auf die verschiedenen Möglichkeiten der Speicherung von Metadaten wird im folgenden Abschnitt noch genauer eingegangen. Des weiteren enthält ein Document-Tupel eine Menge der Component-Objekte, deren Vater-Component das jeweilige Dokument ist. Durch das Attribut *Layout* wird die visuelle Struktur des Dokuments bestimmt. Dieses Attribut ist vom Typ ST_GeomCollection, ein von ST_Geometry abgeleiteter Typ. Der Inhalt dieses Attributs ergibt sich ausschließlich aus dem Shape-Attribut der nachfolgend eingeführten Component-Relation. Dieses Attribut kann somit analog zum Attribut DocTree a posteriori durch sämtliche im Dokument enthaltenen Component-Objekte berechnet werden. Das Attribut *Content* simuliert hierbei eine Referenz auf den kompletten textuellen Inhalt des Dokuments, der auf dem Textserver gespeichert wird. Die Referenz zeigt hierbei auf einen Eintrag der Fremdtabelle Text.

Der zweite Teil zur Speicherung der Dokumente bildet der Typ **Component_T**, sowie die zugehörige **Component**-Relation. Der Aufbau des Typs ist analog zum konzeptionellen Modell. Der folgende CREATE-Befehl zeigt den Aufbau des Typs **Component_T**.

```
CREATE TYPE Component_T AS (
   Document REF(Document_T),
   Shape ST_Surface,
   Metadata [XML | REF() | ROW()],
   Subcomponents REF(Component_T) MULTISET)
```

Somit enthält ein solches Tupel zum einen eine Referenz auf das zugehörige Dokument. Zum anderen einmal das Attribut *Shape*, welches die geometrische Form des Component-Objekts spezifiziert, dann das Attribut *Metadata*, wodurch Component-spezifische Informationen angegeben werden, sowie eine Menge von weiteren Component-Objekten, aus denen sich das Aktuelle zusammensetzt. Stellt man ein multimediales Dokument durch einen Dokumentenbaum dar, entsprechen diese definierten Component-Objekte den inneren Knoten des Baums. Sie sind in der Form abstrakt, als dass sie nicht direkt als Text- oder Bild-Objekt spezifiziert wurden. Um nun die Blätter eines Dokumentenbaums, entweder Text- oder Bild-Objekte, zu speichern, werden vom Typ **Component_T** die Subtypen **ImageComponent_T** und **TextComponent_T** abgeleitet, und entsprechende Relationen erzeugt. Die Definitionen erweitern den Supertyp **Component_T** dabei lediglich um das Attribut *Content* vom Typ Integer. Dadurch wird eine Referenz auf den zugehörigen Eintrag in der korrespondierenden Fremdtabelle simuliert. Bezüglich des Attributs *Shape* muss erwähnt werden, dass dieses nur die geometrische Darstellung innerhalb einer Seite repräsentiert. Um nun ein Dokument auch dreidimensional zu modellieren, beispielsweise ein Buch, das aus mehreren Seiten besteht, kann nun einerseits diese dritte Dimension in der Relation *SpatialRelations* abgelegt werden, andererseits kann hierfür die z-Achse eines ST_Geometry-Objekts verwendet werden. Nachfolgend wird jedoch vereinfachend davon ausgegangen, dass Referenzen auf dieses Attribut sich stets auf dieselbe Seite beziehen.

Den dritten Teil der Speicherung bildet das Äquivalent der Collection-Klasse des konzeptionellen Modells. Zur Realisierung der Hierarchie einzelner Collections wird nun durch

typisierte Tabellen eine Tabellenhierarchie aufgebaut. Hierfür wird folgender einfacher Supertyp erstellt, von welchem die weiteren Subtypen direkt oder indirekt abgeleitet werden.

```
CREATE TYPE Collection (
   Documents REF(Document_T))
```

Um nun diese Collection in beispielsweise zwei weitere Collections zu unterteilen, können die folgenden beiden CREATE-Befehle verwendet werden:

```
CREATE TYPE Coll1 UNDER Collection
CREATE TYPE Coll2 UNDER Collection
```

Um nun derartige Subtypen ebenfalls in weitere Collections zu strukturieren, kann jeder Subtyp wiederum als Supertyp von weiteren Typen im UNDER-Teil referenziert werden. Allerdings darf in der UNDER-Klausel nur genau ein Supertyp referenziert werden, eine Mehrfachvererbung wird von SQL nicht angeboten.

3.2.2 Speicherung der Metadaten

Zur Speicherung der Metadaten für Dokumente und Component-Objekte sind nun durch SQL:2003 prinzipiell zwei Möglichkeiten denkbar. Die erste Variante basiert auf der allgemeinen Technik, eine Metadata-Relation zu verwenden, so dass für jedes Metadatum ein Attribut in die Relation übernommen wird. Hierbei kann man weiterhin unterscheiden, ob entweder eine Referenz (**REF**) auf eine weitere Metadatentabelle verwendet wird oder die Metadatenrelation mittels **ROW**-Konstruktor als Tupeltyp integriert wird. Bei Verwendung des Metadatenstandards Dublin Core beispielsweise wird somit eine Relation mit 15 Attributnamen verwendet. Der Vorteil dieser Variante liegt hauptsächlich in der Indizierung einzelner Attribute, beispielsweise ein Index über das Attribut *title*. Wenn jedoch ein Dokument in viele Component-Objekte zerlegt wird, welche alle mit Metadaten annotiert werden, kann dies dazu führen, dass die Relation überaus viele NULL-Werte enthält. Dieser Nachteil wird in der zweiten Variante behoben.

Die zweite Möglichkeit der Zuweisung von Metadaten an Dokumente und Component-Objekte basiert auf Part 14 von SQL:2003, SQL/XML. Somit ist es möglich, Dokumenten und Component-Objekten ein Attribut Metadata zuzuweisen, das vom Datentyp XML ist. Dies hat den Vorteil, dass auch nur diejenigen Metadaten in das Attribut übernommen werden, die auch verwendet werden. Eine Verwendung von NULL-Werten für nicht verwendete Attribute ist somit nicht mehr notwendig. Allerdings können zur Indizierung des Inhalts der XML-Attribute nicht die klassischen Index-Verfahren, wie beispielsweise Hashing oder B-Bäume [HS99] verwendet werden, da diese nur über dem kompletten Attributinhalt angelegt werden. Da aber auf XML-Attributen auch einzelne Pfadausdrücke indiziert werden, müssen hierfür spezielle Indizes, wie beispielsweise Pfad- und Strukturindizes [KM03] eingesetzt werden.

In vielen Anwendungen ist vor allem eine Kombination der beiden Varianten sinnvoll. Beispielsweise können Metadaten bezüglich der Dokumente in relationaler Form integriert werden, während für die einzelnen Component-Objekte anwendungsspezifische Annotationen in Form von XML verwendet werden.

3.2.3 Verwaltung von Component-Beziehungen

Um die in Abschnitt 3.1.4 beschriebenen Beziehungen darzustellen, und im späteren Verlauf der Anfrageverarbeitung nutzen zu können, können diese Beziehungen entweder direkt gespeichert werden oder durch eine vordefinierte Menge an Operatoren zur Laufzeit berechnet werden. Die Variante der vordefinierten Operatoren ist vor allem für spatiale Beziehungen von Interesse, die relationale Speicherung hingegen bei logischen, interaktionalen und interpretierten Beziehungen. Nachfolgend wird nun die Verwaltung dieser vier Komponentenbeziehungen beschrieben.

Spatiale Beziehungen Spatiale Beziehungen können nach [PK94] in folgende Klassen eingeteilt werden:

- topologische Beziehungen (Nachbarschaft, Inzidenz)
- gerichtete Beziehungen (nördlich, nordöstlich)
- Abstandsbeziehungen
- vergleichende oder ordinale Beziehungen (Inklusion, Präpositionen)
- unscharfe (fuzzy) Beziehungen (weit entfernt, dicht)

Durch die Vielzahl der möglichen spatialen Beziehungen zwischen Component-Objekten ist es schon aus Gründen des Speicherungsaufwands[6] nicht empfehlenswert, diese in relationaler Form zu speichern. Effizienter ist es daher, ausgehend von den gegebenen Informationen (shape, DocTree), diese on-the-fly, durch vorgegebene Routinen, berechnen zu lassen. Auf diese Funktionen wird, durch den starken Bezug zur Anfragefunktionalität, genauer in Abschnitt 3.3.1 eingegangen. Eine Liste der vordefinierten Routinen, sowie deren Implementierung, ist auch in Anhang A.2 gegeben.

[6]Bei 100 Media-Component-Objekten eines Dokuments sind bereits für die Speicherung der Before-Beziehungen (siehe Abschnitt 3.3.1.2) mehr als 4450 Eintäge der Form *Before(Ci,Cj)* notwendig

Logische Beziehungen Da logische Beziehungen anwendungsabhängig definiert werden, ist hierfür eine relationale Speicherung zu wählen. Vordefinierte Routinen sind nicht sinnvoll, da beispielsweise eine Funktion *getChapter()* in einer digitalen Bibliothek zwar äußerst wünschenswert ist, im eNoteHistory Projekt jedoch nicht benötigt wird. Daher wird zunächst eine typisierte Relation **Relations** eingeführt, von welcher daraufhin die Relation **LogicalRelations** abgeleitet wird. Der Typ **Relations_T** hat hierbei folgende Form:

```
CREATE TYPE Relations_T AS(
   Description varchar,
   Root REF(Component_T),
   Comps REF(Component_T) MULTISET)
```

Durch diese Relation können nun zweierlei logische Beziehungen dargestellt werden. Zum einen kann durch das Tripel ('Chapter', ID001, NULL) definiert werden, dass das Component-Objekt mit der OID 'ID001' ein Kapitel repräsentiert. Zum anderen kann durch ('Content',ID002,(ID105, ID106, ID107)) definiert werden, dass der eigentliche Inhalt eines Dokuments sich nur aus den drei Component-Objekten der Comps-Liste ergibt, die alle direkt oder indirekt Kindelement von Root sind[7]. Das bedeutet, man kann einerseits Component-Objekte attributieren, andererseits logische 1:1- und 1:n-Beziehungen darstellen.

Interaktionale Beziehungen Da durch Links referenzierte Objekte ebenfalls in einer 1:n-Beziehung stehen, wird hierfür ebenfalls eine Relation vom Supertyp **Relations_T** abgeleitet, die Relation **InteractionalRelations**. Hierbei stellt das Attribut Root das Ziel der Referenz dar, das Attribut Comps entspricht hierbei einer Liste der Component-Objekte, die das Objekt Root durch einen Link referenzieren. Diese Speicherungstechnik entspricht also der einer invertierten Liste. Angemerkt werden muss an dieser Stelle, dass die Linkverfolgung in dieser Arbeit nicht in die Anfragebearbeitung eingeht. Somit liegt der Hauptverwendungszweck dieser Relation im Ranking-Verfahren. Es kann beispielsweise die Relevanz eines Dokuments durch Anzahl der Component-Objekte bestimmt werden, die auf dieses Dokument verweisen. Dieses Prinzip wird zum Beispiel unter dem Namen PageRank™ von Google[8] verwendet.

Interpretierte Beziehungen Da auch interpretierte Beziehungen, analog zu logischen, a priori nicht bestimmbar sind, da sie anwendungsabhängig durch bestimmte Funktionen ermittelt werden, müssen diese ebenfalls relational gespeichert werden. Daher wird von der Tabelle **Relations** die Tabelle **InterpretedRelations** abgeleitet. So kann beispielsweise das Tripel ('describes',ID102,ID237) bedeuten, dass das Component-Objekt ID102 jenes

[7]Root beschreibt in diesem Fall das hierarchisch kleinste Component-Objekt, dass die restlichen Component-Objekte enthält

[8]http://www.google.com

mit der OID ID237 näher beschreibt[9]. Die Interpretation dieser Information kann jedoch wie auch die Erstellung anwendungsspezifisch sein.

3.2.4 Aufbau der Fremdtabellen

Dieser Teilabschnitt beschreibt nun den Aufbau der Fremdtabellen, durch welche die Daten, die auf den spezialisierten Medienservern gespeichert sind, in der Datenbank repräsentiert werden. Hierfür wird zum einen auf den Aufbau der Foreign Tables, zum anderen aber auch auf die Definition der Foreign Data Wrapper und Foreign Server eingegangen. Diese Konzepte wurden bereits in Kapitel 2 vorgestellt.

Foreign Table Text Die Fremdtabelle **Text** repräsentiert nun die textuellen Inhalte der TextComponent-Objekte. Der Aufbau dieser Fremdtabelle ist dabei relativ simpel, wie der folgende Create-Befehl zeigt.

```
CREATE FOREIGN TABLE Text (
   ID Integer,
   Content CLOB,
   Language Varchar)
SERVER TextServer
```

Durch diese Fremdtabelle werden somit Textfragmente lediglich durch das Attribut *Content*, sowie der Sprache, in der das Textobjekt vorliegt, repräsentiert. Durch das Attribut *ID* kann nun das jeweilige Textobjekt von einer Relation referenziert werden. Das Attribut *Content* ist hierbei allerdings nur ein CLOB, da Attribute in Fremdtabellen ausschließlich Basisdatentypen sein dürfen, FullText allerdings ein strukturierter Typ ist. Daten, die nun vom Medienserver an das Datenbanksystem gesendet werden, werden zunächst als CLOB interpretiert. Werden die Ergebnisse nicht weiter ausgewertet, werden also keine Volltextoperatoren vom Datenbanksystem selbst auf diesen Textobjekten angewandt, ist der Datentyp CLOB zur Repräsentation ausreichend. Andernfalls bietet SQL mit der Konstruktormethode *FullText(String VARCHAR)*, bzw. *FullText(String VARCHAR, Language VARCHAR)* die Möglichkeit, aus dem Textobjekt ein Objekt vom Typ FullText zu erzeugen.

Diese Tabelle bietet nun eine objektrelationale Sicht auf die Daten, die einem speziellen Textserver gespeichert werden. Dieser Server hat nunmehr die Aufgabe, die Textobjekte eines multimedialen Dokuments zu speichern, sowie ein Retrieval über die jeweiligen Objekte bereitzustellen. Die Möglichkeiten des Retrieval reichen hierbei von der einfachen Bereitstellung der Objekte, bis hin zu *Information Retrieval*-Funktionalitäten. Ausgehend von den bereitgestellten Retrieval-Möglichkeiten wird daraufhin ein Foreign Data Wrapper erstellt. Der exakte Aufbau eines solchen Wrappers, also die Library- und Options-Klausel, ist allerdings abhängig vom Speicherungs- und Retrieval-Konzept des Text Servers. Je

[9] z.B. das Label eines Bildes

nachdem, welche Konzepte der Textserver bereitstellt, müssen fehlende Funktionalitäten vom Wrapper kompensiert werden. Nachdem nun ein entsprechender Wrapper modelliert wurde, kann der eigentliche Medienserver für Textobjekte durch ein CREATE FOREIGN SERVER-Statement erzeugt werden. Der Name des Foreign Server wird daraufhin in der SERVER-Klausel des CREATE FOREIGN TABLE-Statements referenziert.

In einem Beispielszenario könnte nun der Textserver für jedes in der Datenbank gespeicherte Dokument eine Datei besitzen, die dem kompletten textuellen Inhalt des Dokuments entspricht. Die Aufteilung des Textes gemäß der Aufteilung des Dokuments in Component-Objekte soll durch `'\t'` erfolgen. Dann könnte ein zugehöriger Wrapper durch den nachfolgenden Befehl erzeugt werden[10]:

```
CREATE FOREIGN DATA WRAPPER TextWrapper
  LIBRARY '/usr/bin/sharelibs/filmgr.shr'
  LANGUAGE C
  OPTIONS (recordsep '\t')
```

Die Implementierung des Wrappers liegt hierbei in der Datei 'filmgr.shr' und liegt in C-Code vor. Bietet der Textserver beispielsweise keine phonetische Suche, so existiert in dieser Datei eine Regel, wie diese Funktionalität durch den Wrapper kompensiert werden kann. Erfolgt nun auf Datenbankseite der Aufruf *Text(Document)*, so kann die zugehörige Datei beispielsweise durch eine Hashtabelle ermittelt werden. Erfolgt jedoch der Aufruf *Text(TextComponent)*, so erfolgt eine Aufteilung der Datei in durch `'\t'` getrennte Textteile, die entsprechend weiterbearbeitet werden können. Diese Textobjekte werden dann abschließend an das Multimedia-Datenbanksystem geschickt. Dieses einfache Beispiel soll lediglich die Funktionsweise verdeutlichen. Im Allgemeinen werden Medienserver verwendet, die eine komplexe und effiziente Speicherung der Daten anbieten, beispielsweise IR-Systeme wie Fulcrum oder Stairs, oder Erweiterungen von Systemen wie der TextExtender für DB2. Da nun solche Systeme unter Umständen Anfragefunktionalitäten anbieten, die über die hier vorgestellten Möglichkeiten von SQL/MM hinausgehen, können diese Funktionen auch integriert, und für das Multimedia-Datenbanksystem zur Verfügung gestellt werden. Auf eine solche Erweiterung wird ausführlich in [Lin02] eingegangen.

Foreign Table Images Der Aufbau der Fremdtabelle **Images** erfolgt analog zu dem in Abschnitt 3.1.2 beschriebenen konzeptuellen Modell. Der folgende Create-Befehl beschreibt den Aufbau der Tabelle, wobei sich der Basisdatentyp BLOB zum strukturierten Typ SI_StillImage analog verhält, wie der Datentyp CLOB zum Typ FullText.

```
CREATE FOREIGN TABLE Images (
   ID Integer,
   Picture BLOB,
   isRoot Boolean,
   Subpictures Integer MULTISET,
```

[10]Dieses Beispiel samt der Generic Option *recordsep* stammt aus [Mel03]

```
   Regions ROW (RegionID Integer,
                Shape CLOB,
                Metadata XML) MULTISET,
   ImageMetadata ROW (colorSpace Varchar,
                      BitsPerPixel SmallInt))
SERVER ImageServer
```

Analog zur Texttabelle besitzt nun ein Bild eine *ID*, um von anderen Relationen referenziert zu werden. Das Attribut *Picture* repräsentiert das eigentliche Bild. Durch die Konstruktormethode *SI_StillImage(pic BLOB)* kann aus dem Large Object ein Objekt vom Typ SI_StillImage erzeugt und weiterbearbeitet werden. Wird nun ein Bild, beispielsweise durch Segmentierung, in weitere Bilder unterteilt, werden diese Teilbilder ebenfalls als Eintrag in der Fremdtabelle erzeugt. Somit entspricht die Liste von Integerwerten des Attributs *Subpictures* einer Liste von Referenzen auf weitere Bilder. Um nun in der Anfragebearbeitung unterscheiden zu können, ob es sich um ein Teil- oder komplettes Bild handelt, dient das Attribut *isRoot*, welches auf True gesetzt wird, wenn es sich um ein komplettes Bild handelt. Eine grundlegende Erweiterung des SQL/MM-Packages Still Image bildet der Tupeltyp *Regions*. Hiermit können gemäß Abschnitt 3.1.2 bestimmten Teilen eines Bildes weitere Informationen zugeordnet werden. Die Attributliste des Tupeltyps besteht hierbei aus der ID der Region, den der Region zugeordneten Metadaten in Form von XML[11], sowie dem Attribut *Shape*, welches die Form der Region spezifiziert. Analog zu Component-Objekten werden Formen durch Elemente vom Typ Spatial repräsentiert. Da allerdings der ST_Geometry ein strukturierter Typ ist, wird an dieser Stelle der Datentyp CLOB als Transporttyp verwendet. Somit kann auf Datenbankseite die Methode *ST_WKTToSQL* verwendet werden, um aus einem CLOB ein ST_Geometry-Objekt zu erzeugen. Durch diese planare Beschreibung der Regionen wird explizit keine Hierarchie von Regionen angegeben. Jedoch kann unter der Annahme, dass eine Region, in der weitere Regionen komplett enthalten sind, auch diese Regionen enthält, a posteriori eine Hierarchie der Regionen bestimmt werden. Das letzte Attribut der Fremdtabelle bildet der Tupeltyp *ImageMetadata*. Hierdurch werden die bildspezifischen Information des Farbraums und der Farbtiefe eines Bilds gespeichert.

Analog zur Fremdtabelle Text muss auch für diese ein Foreign Data Wrapper konzipiert werden, durch welchen ein Foreign Server definiert werden kann. Wichtig ist an dieser Stelle, dass dieser Server eine Funktion *2BLOB* bereitstellt, falls ein nicht SQL/MM-konformes System verwendet wird. Durch diese Funktion wird es erst ermöglicht, Bilder dieses Servers in das Multimedia-Datenbanksystem zu integrieren. Die Aufgabe dieser Funktion liegt nun darin, ein beliebiges Bild derart in ein Binary Large Object zu überführen, dass auf Datenbankseite die Methode *SI_StillImage(b BLOB)* angewendet werden kann, so dass ein korrektes Objekt vom Typ StillImage erzeugt wird. Der Datentyp BLOB entspricht somit einem Transportdatentyp zwischen Image Server und Multimedia-Datenbanksystem.

[11] Hierfür sind sowohl low-level-Features wie histogram möglich, aber auch high-level-Features wie beispielsweise der Name der Person, die in der Region zu sehen ist

Da die Funktionalitäten existierender Bilddatenbanken weit über die von SQL/MM Still Image bereitgestellten Möglichkeiten hinausgehen, ist hierbei eine Funktionsintegration besonders wichtig. Wird beispielsweise ein MPEG-7-basiertes System wie „Caliph & Emir“[12] verwendet, können selbst Anfragen auf semantische Beziehungen zwischen Regionen von Teilbildern formuliert werden. Durch diese Funktionalitäten werden allerdings die von SQL/MM Still Image gebotenen Methoden überflüssig. Da nun die Funktionsintegration abhängig vom verwendeten System ist, wird an dieser Stelle auch nicht weiter darauf eingegangen und eventuell mögliche Funktionen nicht berücksichtigt, sodass nachfolgend nur auf die von SQL angebotenen Methoden eingegangen wird.

3.3 Anfragemöglichkeiten

Um eine Anfrageverarbeitung untersuchen zu können, ist im Vorfeld zu klären, welche Anfragemöglichkeiten existieren. Daher wird an dieser Stelle zunächst untersucht, welche Möglichkeiten SQL bietet, um Anfragen an das Multimedia-Datenbanksystem bezüglich der multimedialen Dokumente zu stellen. Dabei wird zunächst einmal unterschieden zwischen Anfragen auf Struktur, und Anfragen auf Inhalt. Daraufhin werden entsprechende Kombinationsmöglichkeiten untersucht.

3.3.1 Anfragen auf Struktur

Um Anfragen auf Strukturen zu stellen, muss zunächst einmal unterschieden werden, ob es sich hierbei um geometrische bzw. visuelle oder logische Strukturen handelt. Eine Anfrage auf geometrische oder visuelle Strukturen wäre beispielsweise „der Text links neben dem Bild“. Hierbei spielen die spatialen Beziehungen und der Dokumentenbaum eine entscheidende Rolle. Allerdings sind derartige spatiale Beziehungen nicht immer absolut gegeben, sondern können sich beispielsweise erst durch die Darstellung des Dokuments, wie bei HTML, ergeben. In diesen Fällen sind keine exakten geometrischen Beziehungen vorhanden, vielmehr unscharfe Relationen, wie „in der Nähe von“.

Anfragen auf logische Strukturen kann man in zwei Klassen unterteilen, zum einen bezogen auf das konzeptionelle Modell, zum anderen auf logische Beziehungen, die anwendungsspezifisch definiert werden. Beispiele hierfür wären „Gib mir den Absatz[13]“ im Gegensatz zu „Gib mir das Kapitel[14]“.

[12] http://sourceforge.net/projects/caliph-emir

[13] Paragraph ist Teil der Text-Packages

[14] wenn Kapitel als logische Relation definiert wurde

3.3.1.1 Geometrische Anfragen

Die hierdurch klassifizierten Anfragen zeichnen sich dadurch aus, dass spatiale Beziehungen zwischen zwei Component-Objekten unter Verwendung des Attributs *Shape* verglichen werden. Hierzu können zum einen die exakten Formen wie ST_CurvePolygon, aber auch approximierte Formen wie ST_Envelope(), ein umschließendes Rechteck, welches für Vergleiche einfacher zu handhaben ist, für einen Vergleich herangezogen werden. Voraussetzung für diese Klasse struktureller Anfragen ist, dass für beide zu vergleichende Component-Objekte Werte für das Attribut *Shape* spezifiziert wurden.

Die erste Möglichkeit, basierend auf ST_Geometry-Formen Anfragen zu formulieren, besteht darin, dass der Anwender spatiale Beziehungen selbständig definiert. Das bedeutet, dass er eigenständig definiert, welche besonderen Punkte einer Region über welche Beziehung mit welchem speziellen Punkt einer anderen Region in Beziehung steht. Diese möglichen Vergleiche entsprechen den Möglichkeiten der allgemeinen Geometrie, die Syntax wird durch das SQL/MM-Package Spatial definiert. Auf Grund der kombinatorischen Vielfalt wird dieser Ansatz jedoch nicht weiter untersucht. Ein entsprechendes Beispiel hierfür könnte jedoch wie folgt aussehen:

```
SELECT C1, C2
FROM Component C1, Component C2
WHERE C1.Document = C2.Document
   AND C1.Shape.ExteriorRing().ST_NumPoints() = 3
   AND C2.Shape.ExteriorRing().ST_NumPoints() = 5
   AND C1.Shape.ExteriorRing().ST_Point_N(3).ST_Within(
             C2.Shape.ST_Buffer(50,'Pixel'))
```

Hierbei sollen diejenigen Component-Objekte ermittelt werden, bei denen die erste Form einem Dreieck und die zweite einem Fünfeck entspricht, und weiterhin der dritte Punkt der ersten Form in der gegebenen Pufferzone der zweiten liegen soll. Durch diese Anfrageart kann zwar jede mögliche Beziehung dargestellt werden, jedoch ist es aus Anwendersicht, aber auch im Hinblick auf Optimierungen, vorteilhaft, sich auf eine abgeschlossene Grundmenge von Methoden zu beschränken. Hierfür bietet SQL bereits eine Reihe interessanter Methoden wie beispielsweise *ST_Distance* oder *ST_Touches* an. Diese wurden bereits in Kapitel 2 vorgestellt. Ein Beispiel bezüglich der durch SQL gegebenen Funktionen könnte folgendermaßen aussehen:

```
SELECT C1, C2
FROM Component C1, Component C2
WHERE C1.Document = C2.Document
   AND C1.Shape.ST_Touches(C2.Shape)
```

Diese Beispielanfrage beschreibt zwei Component-Objekte, die eine gemeinsame Schnittstelle besitzen. Zusätzlich zu diesen SQL-Methoden werden nun für die Anfrageformulierung auf Strukturen noch weitere Methoden bereitgestellt,

- Left: ST_Geometry × ST_Geometry → Boolean
- Right: ST_Geometry × ST_Geometry → Boolean
- Above: ST_Geometry × ST_Geometry → Boolean
- Under: ST_Geometry × ST_Geometry → Boolean
- Between: ST_Geometry × ST_Geometry × ST_Geometry → Boolean

Diese neu definierten Funktionen können nun in Anfrageformulierungen verwendet werden. Sucht man beispielsweise ein Text-Objekt, das links von einem Bild liegen soll, so kann dies mit folgender Beispielanfrage formuliert werden:

```
SELECT C1, C2
FROM TextComponent C1, ImageComponent C2
WHERE C1.Document = C2.Document
   AND Left(C1.Shape,C2.Shape)
```

3.3.1.2 Unscharfe Anfragen

Diese Klasse von Anfragen zeichnet sich dadurch aus, dass, im Gegensatz zu den vorangegangenen Anfragen, keine konkreten geometrischen Formen untersucht werden, sondern relative Strukturen, die sich unter Umständen sogar erst zur Darstellungszeit ergeben. Hierbei können auch Component-Objekte verglichen werden, deren geometrische Form gar nicht gegeben ist. Um nun diese Beziehungen auswerten zu können, wird das XML-Attribut *DocTree* der Relation Document verwendet, eine XML-Darstellung des entsprechenden Dokuments. Wie auch bei Anfragen auf geometrische Strukturen kann ein Anwender nun selbstständig beliebige Vergleiche definieren, es werden jedoch auch für diese Anfrageklasse eine Reihe von Methoden bereitgestellt, um eine allgemeine, grundlegende Anfrageformulierung zu gewährleisten. Diese angebotenen Funktionen lauten wie folgt:

- Near: Component × Component → Float
- Before: Component × Component → Boolean
- After: Component × Component → Boolean
- Between: Component × Component × Component → Boolean

Näher zu erläutern ist an dieser Stelle lediglich die Methode *Near*, da sie auf einer Metrik basiert, die jedoch anwendungsabhängig definiert werden muss. Diese Funktion verwendet zwei Component-Objekte als Parameter, und erzeugt abhängig von der Anzahl dazwischenliegender Component-Objekte eine Fließkommazahl, welche angibt, wie nah die Elemente liegen. Je höher die Ausgabe, desto dichter liegen sie beieinander. Um diese Methode verwenden zu können, wird an dieser Stelle folgende einfache Metrik M verwendet:

$$M := \left(\frac{C_{total}}{C_{total} - C_{between}}\right)^2$$

Hierbei steht C_{total} für die Gesamtanzahl der Component-Objekte und $C_{between}$ für die Anzahl der Component-Objekte, die zwischen den als Parametern übergebenen Component-Objekten liegen. Für die Berechnung dieser Werte wird die abstrahierte Form des Dokuments, das Attribut *DocTree* verwendet, um ohne Rekonstruktionsmaßnahmen den Aufbau des Dokuments zu untersuchen. Folgende Beispielanfrage zeigt nun die Verwendung der Near-Methode.

```
SELECT C1, C2
FROM TextComponent C1, ImageComponent C2
WHERE C1.Document = C2.Document
   AND Near(C1,C2) > 0.7
```

3.3.1.3 Logische modellbezogene Anfragen

Logische Anfragen bezüglich des konzeptionellen Modells unterscheiden sich darin, ob die Komponenten zum Vergleich in der Where-Klausel referenziert werden oder ob sie als Ausgabegranularität spezifiziert werden. Werden sie in der Where-Klausel verwendet, beispielsweise in abstandsbasierten Textanfragen wie „IN SAME SENTENCE AS“, gilt dies als inhaltsbezogen und wird in Abschnitt 3.3.2 vorgestellt. Dieser Teilabschnitt bezieht sich demnach auf Anfragen, deren Ausgabegranularität textspezifisch (Absatz, Satz oder Wort) oder bildspezifisch (Bild, Teilbild oder Region) beträgt. Hierfür werden folgende Methoden bereitgestellt:

- Text: Document → Fulltext
- Paragraph: Document → Fulltext Array
- Sentence: Document → Fulltext Array
- Word: Document → Fulltext Array
- Text: Component → Fulltext
- Paragraph: Component → Fulltext Array
- Sentence: Component → Fulltext Array
- Word: Component → Fulltext Array

- Pictures: Document → ImageComponent Array
- Pictures: Component → ImageComponent Array
- Picture: ImageComponent → Image
- Subpicture: Image → StillImage Array
- Region: Image → Integer Array
- MBC: Component Array → Component

Die Methode *Text* extrahiert hierbei den textuellen Inhalt eines Dokuments/Component-Objekts. Die Methoden *Paragraph*, *Sentence* und *Word* basieren auf der in SQL/MM Full Text definierten Funktion *Segmentize*, wodurch ein Volltextobjekt in entsprechende logische Einheiten segmentiert wird. Somit wird ein Dokument/Component-Objekt in die jeweiligen Fragmente aufgeteilt. Durch die Methode *Pictures* werden sämtliche Image-Components ermittelt, die in einem Document/Component-Objekt enthalten sind. Aus diesen ImageComponents kann daraufhin anhand der Methode *Picture* der korrespondierende Eintrag der Fremdtabelle Images ermittelt werden. Um nun analog zu Textobjekten auch ein Bild zu segmentieren, werden die Methoden *Subpicture* und *Region* bereitgestellt. Sie ermitteln die zu einem Bild gehörigen Teilbilder und Regionen.

Die vorangegangenen Methoden dienten nun dazu, ein Objekt in weitere Objekte zu fragmentieren. Im Gegensatz dazu steht die Methode *MBC*[15]. Sie besitzt nun als Eingabe ein Array von Component-Objekten, aus denen der hierarchisch kleinste Knoten des Dokumentenbaums ermittelt wird, der alle übergebenen Component-Objekte enthält. Die Motivation hierfür liegt darin begründet, das in Anfragen eine Menge von Component-Objekten als Ausgabe erzeugt werden kann. Um aus dieser Ausgabe nun einen logisch abgeschlossenen Dokumententeil zu erzeugen, wird diese Methode verwendet.

3.3.1.4 Logische selbstdefinierte Anfragen

Im Gegensatz zu den vorangegangenen Anfragemöglichkeiten ist es bei dieser Anfrageklasse nicht möglich, vordefinierte Methoden bereitzustellen, da a priori nicht bestimmbar ist, welche logischen Beziehungen anwendungsabhängig spezifiziert werden. Statt dessen wird für diese Art von Anfragen die Relation LogicalRelation verwendet. Folgende Beispielanfrage soll die Verwendung verdeutlichen, wobei der Text eines Kapitels ermittelt werden soll, in dem „Anfrage“ und „Verarbeitung“ vorkommen, und maximal ein Wort dazwischen liegen darf:

```
SELECT Text(C)
FROM Component C
WHERE (C IN (
          SELECT Comp
          FROM LogicalRelations
```

[15]**M**inimal **B**ounding **C**omponent

```
        WHERE Description = 'Chapter'))
    AND Text(C).contains(
        '"Anfrage" NEAR "Verarbeitung" WITHIN 1 WORD ANY ORDER') = 1
```

3.3.2 Anfragen auf Inhalt

Inhaltsbezogene Anfragen können sich auf den Inhalt von Text, Bildern und Metadaten beziehen. Da in Abschnitt 3.2 nicht festgelegt wurde, in welcher Form Metadaten gespeichert werden, müssen an dieser Stelle alle Möglichkeiten genau untersucht werden. Da Anfragen auf Inhalt stets als Selektionskriterium aufgefasst werden können, ist es an dieser Stelle ausreichend, lediglich die Where-Klausel zu untersuchen, auf eine Verwendung der Group By- und Having-Klausel wird an dieser Stelle nicht eingegangen.

3.3.2.1 Anfragen auf Text

Auf inhaltsbezogene Anfragen auf Textobjekte wurde bereits in Kapitel 2 eingegangen. Die dort vorgestellten Funktionalitäten, die das SQL/MM-Package Full Text bereitstellt, können zur Anfrageformulierung komplett verwendet und kombiniert werden. Daher wird an dieser Stelle lediglich eine grundlegende Einführung über die Anfrageformulierung gegeben. Die beiden wesentlichen Methoden zum Vergleichen von Text-Objekten bilden *Contains* und *Score*. Die Verwendung dieser beiden Methoden erfolgt fast identisch, allerdings ist es bei der Score-Methode notwendig, das Ergebnis mit einem Schwellwert zu vergleichen. Die Suchfunktionalität, also die Mächtigkeit der Anfrageformulierung, ist jedoch bei beiden dieselbe. Anfragen werden nun derart formuliert, dass zunächst ein zu vergleichender Text durch die Methode *Text* aus einem Component-Objekt extrahiert wird. Das Ergebnis dieser Extraktion kann sodann in der Where-Klausel referenziert und selektiert werden, wie folgendes Beispiel zeigt.

```
SELECT Text(C) as T
FROM Component C
WHERE T.SCORE('"Anfrage" AND ("Bearbeitung" OR "Verarbeitung")') > 0.7
```

Kombinationsmöglichkeiten, auf die weiterführend in Abschnitt 3.3.3 eingegangen wird, können nun unter anderem darin bestehen, dass die From-Klausel nicht die Component-Relation, sondern eine weitere verschachtelte Anfrage enthält, die als Ergebnis eine Menge von Component-Objekten referenziert. Eine andere Möglichkeit besteht darin, das Text-Objekt T in logische Abschnitte aufzuspalten. Folgendes Beispiel verdeutlicht dies, wobei nur noch Sätze untersucht werden.

```
SELECT Sen
FROM UNNEST(Sentence(Component)) Sen
WHERE Sen.SCORE('"Anfrage" AND ("Bearbeitung" OR "Verarbeitung")') > 0.7
```

3.3.2.2 Anfragen auf Bild

Bevor nun die Anfragemöglichkeiten untersucht werden, muss zunächst auf die Diskrepanz zwischen dem konzeptionellen Modell und dem SQL/MM-Package Still Image eingegangen werden. Die Anfragemöglichkeiten, die nun SQL bereitstellt, sind sehr gering. Es können lediglich die Attribute und eine gewichtete Linearkombination der Features histogram, texture, average und positional color in die Anfrage miteinbezogen werden, deren Berechnung nur bezüglich des kompletten Bilds erfolgt. Eine Feature-Berechnung über ein ganzes Bild ist nach dem Modell jedoch nur auf einer Region wholeImage möglich. Eine Möglichkeit bestünde nun darin, jede SQL-konforme Anfrage vor der Bearbeitung zu transformieren, so dass sie sich stets auf eine Region wholeImage bezieht. Um nun keine derartigen unnötigen Restriktionen bzw. zusätzliche Bearbeitungsschritte einzuführen, werden die Anfragemöglichkeiten, die SQL/MM bietet, auch weiterhin unterstützt, obwohl sie nicht modellkonform sind. Folgende Anfrage ist somit nicht konform bezüglich des in Abschnitt 3.1.2 eingeführten Modells, da die Features auf Bild- und nicht auf Regionsebene ermittelt werden, sie kann aber dennoch verwendet werden, da sie SQL/MM-konform ist.

```
SELECT SI_StillImage(Picture) AS Pic
FROM Images
WHERE SI_FeatureList( SI_AverageColor(example),0.0,
                      SI_ColorHistogram(example),0.0,
                      SI_PositionalColor(example),0.33,
                      SI_Texture(example),0.67).SI_Score(Pic) < 0.5
```

Um nun modellkonforme Anfragen zu formulieren, müssen die Feature-Berechnungen und -Vergleiche auf Regionen durchgeführt werden. Weiterhin können auch zur Region gehörige Metadaten in die Anfrageformulierung mit einbezogen werden. Folgendes Beispiel verdeutlicht dies, wobei aus einem Bild, das kein Teilbild ist, mit einem Farbraum RGB eine Region, auf der John Doe zu sehen ist, mit einem Beispielbild "example" verglichen werden soll.

```
SELECT SI_StillImage(Picture) AS Pic
FROM Images I, I.Regions R
WHERE I.isRoot = TRUE AND
      XMLSERIALIZE(
           CONTENT XMLGEN($Metadata/content/person/text())
           AS VARCHAR(100)) = 'John Doe')  AND
      I.ImageMetadata.ColorSpace = "RGB" AND
      SI_FeatureList( SI_AverageColor(example),0.0,
                      SI_ColorHistogram(example),0.0,
                      SI_PositionalColor(example),0.33,
                      SI_Texture(example),0.67).
         SI_Score(Region2Image(I.ID, R.Shape)) < 0.5
```

Hierbei stellt die neu eingeführte Methode *Region2Image* eine notwendige Funktionalität zur Verfügung. Sie erzeugt aus einem markierten Bildausschnitt, welches kein Teilbild ist, ein Objekt vom Typ SI_StillImage, so dass die von SQL gegebene Feature-Extraktion durchgeführt werden kann. Diese Methode stellt allerdings auch ein Problem dar, und zwar wenn Regionen nicht als Rechtecke spezifiziert werden, sondern als Kreise oder Polygone. Die von SQL bereitgestellten Methoden zur Feature-Extraktion arbeiten auf Still Image-Objekten, die stets einem Rechteck entsprechen. Wird nun die Methode *Region2Image* auf eine Region angewendet, muss diese in ein Rechteck transformiert werden. Eine einfache Möglichkeit bestünde darin, die Methode *ST_Envelope*[16] auf die Form der Region anzuwenden, allerdings würde dadurch die Fläche der Region erweitert werden. Dies hätte bei einer anschließenden Feature-Extraktion jedoch eine Ergebnisverfälschungen zur Folge, da auch Bildteile in die Berechnung eingehen, die außerhalb der spezifizierten Region liegen. Da nun für die Verwaltung der Bilder ein spezieller Medienserver verwendet wird, wird an dieser Stelle vorausgesetzt, dass dieser eine Methode bereitstellt, welche eine beliebige Region ohne Informationsverfälschung in ein Rechteck transformiert, andernfalls muss diese Verfälschung toleriert werden. Eine zweite Möglichkeit könnte darin liegen, die low-level-Features einer Region bereits im Vorfeld zu berechnen und im Attribut *Metadata* einer Region abzulegen. Daraufhin müsste allerdings entweder die Methode SI_Score überladen werden, oder der Medienserver müsste aus den gegebenen Features ein Dummy-Bild erzeugen, dass genau diese Feature-Werte besitzt, so dass die SI_Score-Methode von SQL/MM weiterhin verwendet werden kann.

Wie an diesen Anfragen bereits zu erkennen ist, können inhaltliche Anfragen auf Bilder in zwei Klassen zerlegt werden, dokumentgebundene und kontextfreie Anfragen. Kontextfreie Anfragen, wie die Vorangegangenen, zeichnen sich dadurch aus, dass das Multimedia-Datenbanksystem quasi als Bild-Datenbank verwendet wird. Anfragen werden somit ausschließlich auf die Fremdtabelle Images gestellt, ohne einen Bezug zum zugehörigen Dokument zu bilden. Um nun dokumentenbezogene Anfragen stellen zu können, wurden in Abschnitt 3.3.1 die Methoden *Pictures* und *Picture* eingeführt. Die Methode *Pictures* ermittelt hierbei aus einem angegebenem Component-Objekt alle enthaltenen ImageComponents, die darauf aufbauende Methode *Picture* gibt daraufhin den zu einem ImageComponent gehörigen Eintrag der Fremdtabelle Images zurück. Die Methode *Pictures* ist somit dann sinnvoll, wenn aus einer inneren Anfrage alle enthaltenen ImageComponents ermittelt werden sollen. Folgende Anfrage illustriert nun die dokumentengebundene Suche auf Bildobjekten.

```
SELECT SI_StillImage(Pic.Picture)
FROM (SELECT Picture(IC)
      FROM ImageComponent IC
      WHERE XMLSERIALIZE(
               CONTENT XMLGEN($Metadata/content/person/text())
               AS VARCHAR(100)) = 'John Doe') Pic,
     Pic.Regions R
WHERE  SI_FeatureList( SI_AverageColor(example),0.0,
```

[16] berechnet das umschließende Rechteck

```
                    SI_ColorHistogram(example),0.0,
                    SI_PositionalColor(example),0.33,
                    SI_Texture(example),0.67).
         SI_Score(Region2Image(Pic.ID, R.Shape)) < 0.5
```

Hierbei wird in der inneren Anfrage eine Ergebnismenge aus vorselektierten ImageComponents erzeugt, aus diesen die zugehörigen Einträge der Image-Tabelle ermittelt, und nach einer regionsspezifischen Selektion das entsprechende Bild ausgegeben. Statt nun in der From-Klausel die Relation ImageComponent zu referenzieren, wäre auch der Aufruf `UNNEST(Pictures(SFW-Block))` möglich. Eine weitere wichtige Möglichkeit besteht darin, nicht Bilder sondern Component-Objekte auszugeben, die ein bestimmtes Bild enthalten. Das Ergebnis einer solchen Anfrage kann somit, im Gegensatz zur vorangegangenen Anfrage, orthogonal in einer weiteren From-Klausel, analog zur Component-Relation, referenziert werden. Eine solche Anfrage könnte beispielsweise folgende Form haben:

```
SELECT IC
FROM ImageComponent IC
WHERE Content IN (SELECT ID
                  FROM Images I, I.Regions R
                  WHERE SI_ColorHistogram(example).
                        SI_Score(Region2Image(I.ID, R.Shape)) < 0.5
```

3.3.2.3 Anfragen auf Metadaten

Ausgehend von der Beschreibung zur Speicherung von Metadaten aus Abschnitt 3.2.2 können Anfragen auf Metadaten einerseits objektrelational, andererseits XML-basiert gestellt werden. Nachfolgend wird nun angenommen, dass o.B.d.A. Metadaten der Document-Relation durch Referenzen auf Einträge einer Metadata-Relation realisiert werden, Metadaten der Component-Relation hingegen durch Attribute vom Typ XML. Tupelwertige Attribute für Metadaten sind, wie in Abschnitt 3.2.4 beschrieben, in der Fremdtabelle Image vorhanden. Durch diese verschiedenen Metadaten werden nachfolgend die Anfragemöglichkeiten vorgestellt.

Anfragen durch Verwendung von Fremdschlüsseleigenschaften bilden die klassische Variante, die seit SQL-89 Level 2 mit IEF möglich ist [HS00]. Hierbei werden die partizipierenden Relationen in der From-Klausel referenziert, und anschließend durch die Where-Klausel über ein Schlüsselattribut verknüpft. Das folgende Beispiel soll diese Variante verdeutlichen, wobei das in der Metadata-Relation spezifizierte Attribut Title mit dem Wort „Anfrageverarbeitung“ beginnen soll.

```
SELECT Doc
FROM Document Doc, Metadata Meta
WHERE Doc.MetadataID = Meta.ID
   AND Meta.Title LIKE 'Anfrageverarbeitung%'
```

Die zweite Möglichkeit der Anfrageformulierung existiert seit SQL:1999 durch Einführung tupelwertiger Attribute, wie das Attribut ImageMetadata zeigt. Die einzelnen Teile des Attributs können nun mittels Punktnotation referenziert werden[17]. Das folgende Beispiel zeigt, wie auf das Attribut Colorspace des tupelwertigen Attributs ImageMetadata zugegriffen werden kann.

```
SELECT Picture
FROM Images
WHERE ImageMetadata.Colorspace = 'CMYK'
```

Der dritte und auch mächtigste Teil der Anfragemöglichkeiten auf Metadaten besteht in der seit SQL:2003 bestehenden Möglichkeit, XML-Attribute zu verwenden, auf welche mittels XQuery-Ausdrücken zugegriffen werden kann. Die Funktionsweise von SQL/XML wurde bereits in Kapitel 2 vorgestellt. Die starke Mächtigkeit dieser Anfrageklasse liegt hauptsächlich in der enormen Ausdruckskraft von XML begründet. Durch die somit erhaltene Möglichkeit, ein Attribut beliebig tief zu strukturieren und auch auf jedes einzelne Fragment zugreifen zu können, kann nun jede beliebige Information dargestellt und zur Anfrageformulierung verwendet werden. Es ist sogar möglich Metadaten in MPEG-7-Format[MSS02] abzulegen und abzufragen, sofern keine MPEG-7 spezifischen Erweiterungen wie beispielsweise der Datentyp *Matrix* verwendet werden. In der folgenden Beispielanfrage sollen nun Component-Objekte ermittelt werden, in denen John Doe eine Aktion durchführt.

```
SELECT C
FROM Component C
WHERE XMLSERIALIZE(
         CONTENT XMLGEN($Metadata/action/people/person/text())
         AS VARCHAR(100)) = 'John Doe'
```

Interessant hinsichtlich der Anfrageformulierung auf Metadaten ist auch der Konstruktor *FullText*, der aus einem Textobjekt ein Volltextobjekt erzeugt. Somit können die extrahierten Metadaten in den Typ FullText überführt werden, so dass daraufhin die in Kapitel 2 beschriebenen Information Retrieval-Techniken, wie phonetische oder abstandsbasierte Suche, verwendet werden können.

3.3.3 Kombinierte Anfragen

In den beiden vorangegangenen Teilabschnitten wurde untersucht, welche Anfragemöglichkeiten auf Strukturen, und welche auf Inhalt existieren. Darauf aufbauend wird nun untersucht, wie diese Anfragen verknüpft werden können, um eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt von multimedialen Dokumenten zu realisieren. Dabei

[17] Dies ist im ersten Beispiel auch möglich, sofern das Attribut *Metadata* mittels **REF()** definiert wurde

wird an dieser Stelle vereinfachend davon ausgegangen, dass in der Select-Klausel einer inneren Anfrage keine Methodenaufrufe, ausgenommen den Methoden *Pictures* und *MBC*, verwendet werden[18], sondern stets Component-Objekte als Ausgabe spezifiziert werden. Somit wird gewährleistet, dass Anfragen orthogonal verwendbar sind. Das Ergebnis einer Anfrage auf Inhalt kann damit als Eingabe einer Anfrage auf Strukturen verwendet werden und vice versa. Anfragen mit Methodenaufrufen in der Select-Klausel sind hier somit stets die äußerste Anfrage, die weitere geschachtelte Anfragen enthalten kann.

Praktisch können Anfragen, die Methoden in der Select-Klausel verwenden, natürlich auch innerhalb einer Anfrage verwendet werden. Zum Beispiel in einer Bedingung der Where-Klausel, wie `...WHERE Text(C1) IN (SELECT Text(C2) FROM...))`. Um diese Anfragen nun auch zu behandeln, sind Anfrageuntersuchungen bezüglich der Typisierung für jedes Anfragefragment sowie Kontext des Anfragefragments durchzuführen. Um dies zu vermeiden, wird die oben beschriebene Restriktion eingeführt.

Die einfachste Möglichkeit, Anfragen auf Struktur und Inhalt zu kombinieren, besteht in der Kombination von Selektionsbedingungen. Im folgenden Beispiel wird eine Anfrage beschrieben, in der ein TextComponent links von einem Bild liegt und der Text ein Wort enthält, das wie „Verarbeitung“ klingt.

```
SELECT C1, C2
FROM TextComponent C1, ImageComponent C2
WHERE C1.Document = C2.Document
   AND Left(C1.Shape,C2.Shape)
   AND Text(C1).Contains('SOUNDS LIKE "Verarbeitung"')=1
```

Die zweite und auch mächtigere Möglichkeit besteht in der Schachtelung von Anfragen. Wie bereits oben beschrieben, werden dabei Ergebnisse von Anfragen in weitere Anfragen eingebettet. Wichtig zu erwähnen ist an dieser Stelle, dass Anfragen, die aus kombinierten Selektionsbedingungen bestehen, sich in eine verschachtelte Anfrage umformen lassen, so dass jede Teilanfrage entweder text- oder inhaltsbezogen ist. Nachfolgend werden nun die Kombinationsmöglichkeiten von Anfragen auf Struktur und Inhalt untersucht. Grundsätzlich muss dabei unterschieden werden, ob die innere Anfrage eine ein- oder mehrelementige Ergebnismenge erzeugt.

Inhalt auf Inhalt Die erste Möglichkeit besteht nun darin, dass Anfragen auf Inhalte weitere Anfragen auf Inhalte enthalten. Besitzt die innere Anfrage hierbei eine einelementige Ergebnismenge, entspricht dies einer einfachen Selektion der Ausgangsmenge. Es entstehen somit keine Konflikte, da die Ergebnisstruktur identisch zur Component-Relation ist, wie folgende einfache Anfrage zeigt:

[18]beispielsweise *Pictures(Component C)*

```
SELECT C1, C2
FROM Component C1,
     (SELECT C
      FROM Component C
      WHERE Text(C).Contains('"Anfrage"')=1) C2
WHERE C1.Document=C2.Document
   AND XMLSERIALIZE(
         CONTENT XMLGEN($C1/Metadata//title/text())
         AS VARCHAR(100)) LIKE 'Verarbeitung%'
```

Angemerkt sei an dieser Stelle, dass die Selektionsbedingung bezüglich der Metadaten in einen SFW-Block innerhalb der From-Klausel, analog zur inneren Anfrage, verschoben werden kann. Somit entstehen wiederum zwei elementare Anfragen auf Inhalt, die sich auf ein einiges Component-Objekt beziehen und durch die äußere Klausel verknüpft werden.

Komplizierter wird die Kombination, wenn bereits die innere Anfrage mehrere Elemente in der Select-Klausel referenziert. Beispielsweise kann die eben beschriebene Anfrage in einer weiteren Anfrage verwendet werden. In diesem Fall kann das Ergebnis der inneren Anfrage nicht als Objektmenge in der äußeren Anfrage verwendet werden. Es müssen vielmehr die Component-Objekte jeweils einzeln verwendet werden, wie folgendes Beispiel, in dem Q1 der vorangegangenen Anfrage entspricht, verdeutlicht:

```
SELECT C1, C2, C3
FROM Q1, ImageComponent C3
WHERE C1.Document=C3.Document
   AND Picture(C3).SI_SCORE(SI_FeatureList...)
```

Struktur auf Inhalt In der zweiten Kombinationsvariante werden nun Anfragen auf Inhalt in Anfragen auf Strukturen eingebettet. Geben die inhaltsbezogenen Anfragen wiederum nur ein objektwertiges Element aus, ist dies analog zur Verwendung der Component-Relation. So können beispielsweise zwei einfache Inhaltsanfragen in der From-Klausel referenziert werden, und die beiden Ergebnismengen auf spatiale Beziehungen untersucht werden, zum Beispiel wie in der folgenden Anfrage, wobei Q1 und Q2 jeweils einfachen Anfragen auf Inhalte entsprechen:

```
SELECT C1, C2
FROM (Q1) C1, (Q2) C2
WHERE C1.Document=C2.Document
   AND Before(C1,C2))
```

Werden nun Anfragen geschachtelt, die eine Relation von Objekten als Ausgabe besitzen, kann nicht die Ausgabe als Component-Objekt behandelt werden. Statt dessen müssen die objektwertigen Attribute der Relation referenziert werden. Wäre Q2 im vorangegangenen Beispiel eine Anfrage, die eine Relation von zwei Component-Objekten erzeugt, könnte die Anfrage folgende Form besitzen:

```
SELECT C1, C2 ,C3
FROM (Q1) C1, (Q2)
WHERE C1.Document=C2.Document
   AND Before(C1,C2))
   AND Near(C1,C3)<0.9
```

Zu erwähnen ist in diesem Zusammenhang auch die Methode *MBC*. Dadurch, dass sie aus einer Liste von Component-Objekten ein neues erzeugt, und zwar das minimale Umschließende, kann die Ausgabe einer Anfrage, die einzig diese Methode in der Select-Klausel verwendet, wiederum analog zur Component-Relation verwendet werden. Beispielsweise könnte die nachfolgende Anfrage wieder in eine einfache inhaltsbezogene Anfrage geschachtelt werden, in der dann der Inhalt von C4 untersucht werden kann.

```
SELECT MBC(ARRAY[C1,C2,C3]) as C4
FROM Q2
```

Inhalt auf Struktur Kombinationen dieser Art zeichnen sich dadurch aus, dass die inneren Anfragen im allgemeinen eine Relation von objektwertigen Attributen zurückgeben, außer wenn die Methode *MBC* verwendet wird. In diesem Fall kann die strukturbezogene Anfrage analog zur Component-Relation verwendet werden. Andernfalls muss die Anfrage umgeformt werden, da strukturbezogene Anfragen im Allgemeinen eine weitaus größere Ergebnismenge generieren als inhaltsbezogene Anfragen[19]. Dies soll folgendes Beispiel verdeutlichen:

```
SELECT C1,C2
FROM (SELECT C1,C2
      FROM Component C1, Component C2
      WHERE C1.Document = C2.Document
            AND Near(C1,C2) > 0.6)
WHERE Text(C1).Contains(Text(C2))=1
```

Würde die Anfrage in dieser Form bestehen bleiben, würde durch den inneren SFW-Block ein exponentiell großes Ergebnis erzeugt werden, das dann durch den zweiten SFW-Block wieder stark verkleinert werden würde. Daher ist es notwendig, Anfragen dieser Art entsprechend umzuformen, so dass zuerst die inhaltsbezogene Anfrage ausgeführt wird. Hierauf wird in Abschnitt 3.4 noch genauer eingegangen. Die neue Anfrage entspricht somit wieder dem Fall **Struktur auf Inhalt**, und besitzt folgende Form:

```
SELECT C1,C2
FROM (SELECT C1,C2
      FROM Component C1, Component C2
      WHERE C1.Document = C2.Document
            AND Text(C1).Contains(Text(C2))=1)
WHERE Near(C1,C2) > 0.6)
```

[19]Dies ist natürlich nicht immer der Fall. Es ist auch die entgegengesetzte Richtung möglich

Struktur auf Struktur Die vierte Kombinationsmöglichkeit beschreibt den Fall, dass strukturbezogene Anfragen innerhalb einer anderen strukturbezogenen Anfrage verwendet werden. Da bereits die innere Anfrage im Allgemeinen eine Relation mit objektwertigen Attributen erzeugt, ist lediglich dieser Fall zu untersuchen. Allerdings kann dieser Fall mit dem Szenario verglichen werden, in dem eine strukturbezogene Anfrage auf eine inhaltsbezogene Anfrage Bezug nimmt, welche mehrere Component-Objekte in der Select-Klausel referenziert. Daher wird diese Möglichkeit nicht näher betrachtet. Weiterhin muss untersucht werden, ob es möglich ist, diese verschachtelten Anfragen zu entschachteln und somit eine planare, strukturbezogene Anfrage zu erzeugen, die sich lediglich aus einer komplexen Where-Klausel ergibt. Hierauf wird noch in Abschnitt 3.4 genauer eingegangen.

Umschließender SFW-Block Abschließend muss nun noch untersucht werden, wie nun in der äußeren, umschließenden Anfrage Methodenaufrufe in die Select-Klausel integriert werden. Dadurch wird die Möglichkeit bereitgestellt nicht nur Component-Objekte auszugeben, sondern weiterhin die vordefinierten logischen Strukturen. Die hierfür notwendigen Funktionen wurden in Abschnitt 3.3.1 eingeführt. Das Ergebnis der inneren Anfrage, die in der From-Klausel der Äußeren referenziert wird, kann nun abschließend durch Methodenaufrufe weiter bearbeitet werden. Ist beispielsweise das Ergebnis der inneren Anfrage eine Liste C von Component-Objekten, könnte eine darauf zugreifende Anfrage folgende Form haben:

```
SELECT Par
FROM UNNEST(Paragraph(innerSFWBlock)) Par
WHERE Par.Score('SOUNDS LIKE "Anfrageverarbeitung"') > 0.8
```

Statt der Methode *Paragraph* können nun weitere Methoden, beispielsweise die Methode *Pictures*, eingesetzt werden. Sollen nun jedoch statt der Bilder lediglich Vorschaubilder ausgegeben werden, ist eine Anfrage folgender Form zu verwenden:

```
SELECT SI_StillImage(Picture(Pic).Picture)
FROM ImageComponent Pic
WHERE Pic IN (innerSFWBlock)
```

Das Ergebnis wird somit nicht in der From- sondern in der Where-Klausel referenziert. Um nun keine Dokumententeile, sondern das Orginaldokument auszugeben, wird in der Select-Klausel der Befehl `DISTINCT DLVALUE(DEREF(C.Document))` eingesetzt, welcher eine URL auf das Orginaldokument ausgibt. Allerdings ist dabei keine weitere Selektion durch die Where-Klausel möglich. Ein abschließend zu untersuchender Punkt ist die Zugehörigkeit zu Collections, so dass in der Anfragebearbeitung nur die Dokumente bearbeitet werden, die den angegebenen Collections angehören. Dies erfolgt durch ein zusätzliches Prädikat in der Where-Klausel, wie das nachfolgende Beispiel verdeutlicht.

```
SELECT SelectList
FROM innerSFWBlock C
WHERE Condition AND
   C.Document IN(
      SELECT Doc
      FROM Coll1)
```

Um nun mehrere unterschiedliche Collections in die Anfrageverarbeitung zu integrieren, können nun mehrere SELECT-FROM-Blöcke innerhalb des IN-Prädikats aufgelistet und durch Mengenoperatoren wie beispielsweise UNION oder EXCEPT kombiniert werden. Das Problem, dass zunächst alle Dokumente bearbeitet werden und erst abschließend eine Restriktion auf die entsprechenden Collections geschieht, muss durch den Optimierer gelöst werden. Im einfachsten Fall geschieht dies dadurch, dass in der Where-Klausel des innersten SFW-Blocks durch die Bedingung `AND C.Document IN (...)` die Zugehörigkeit getestet wird.

Abschließend kann festgehalten werden, dass Anfragen auf Struktur und Anfragen auf Inhalt beliebig kombiniert werden können. Hierbei muss lediglich sichergestellt werden, dass auf das Ergebnis von Anfragen, die eine Relation mit mehreren objektwertigen Attributen erzeugen, attributweise zugegriffen wird. Ergebnisse von Anfragen, die hingegen eine Menge von Objekten erzeugen, werden weiterhin objektwertig, identisch zur Component-Relation behandelt.

3.4 Anfragebearbeitung

In diesem Abschnitt wird nun der Ablauf der Anfragebearbeitung, jedoch nur bezüglich der kombinierten Anfrageformulierung, untersucht. Der Ablauf gliedert sich hierbei in die Anfrageverarbeitung und die Anfrageausführung. Das Ziel der Anfrageverarbeitung liegt dabei darin, einen Ausführungsplan zu erzeugen, der mit minimalen Kosten ausgeführt werden kann. Sie umfasst die Vorverarbeitung (Übersetzung) und Optimierung von Anfragen. Das Ergebnis der Anfrageverarbeitung ist dann ein Zugriffsplan, der als Eingabe für die Anfrageausführung dient [Lin02]. Werden die Schritte der Anfrageverarbeitung und Anfrageausführung zum Begriff der Recherche zusammengefasst, so ist abschließend eine eventuell notwendige Bewertung der Anfrageergebnisse, das Ranking, zu untersuchen. Hierauf wird allerdings erst in Abschnitt 3.5 eingegangen.

3.4.1 Anfragevorverarbeitung

Zu Beginn dieser Phase prüft ein Parser zunächst, ob die Anfrage syntaktisch korrekt, und damit eine gültige Anfrage ist. Im Gegensatz zu [Lin02] ist es an dieser Stelle nicht notwendig, anhand von Metadaten die Korrektheit bezüglich der Subsysteme zu überprüfen, da die dort gespeicherten Daten über die Fremdtabellen referenziert und hierüber auf Korrektheit geprüft wird.

Ist nun die Anfrage syntaktisch korrekt, folgt der nächste Analyseschritt, die Zerlegung der geschachtelten Anfragen. Wie bereits in Abschnitt 3.3.3 beschrieben, besteht eine kombinierte Anfrage entweder aus mehreren verschachtelten Anfragen, oder es erfolgt eine Kombination der Selektionsmöglichkeiten innerhalb der Where-Klausel. Im zweiten Fall muss die Anfrage dahingehend transformiert werden, dass sie aus einer verschachtelten Anfrage besteht, sodass eine Teilanfrage, die keine weitere Teilanfrage enthält entweder struktur- oder inhaltsbezogen ist. Das Ergebnis dieses Teilschrittes ist sodann ein Anfragebaum, der aus Teilanfragen TQ_i besteht. Die Blätter dieses Anfragebaums sind dabei entweder struktur- oder inhaltsbezogen. Eine Teilanfrage hat im weiteren Verlauf folgende abstrahierte Form:

$$TQ_i^{Inhalt_{[T|I|M|C]}|Struktur}([Eingabe](Bedingung_i(Components)) \rightarrow [Ausgabe])$$

Eine Anfrage wird somit entweder auf Struktur oder Inhalt, speziell **T**(ext), **I**(mage), **M**(etadata) oder **C**(ombined), gestellt[20]. Die Ein- und Ausgabe entspricht einer durch Komma getrennten Liste von Component-Objekten. Diese können entweder aus der Component-Relation referenziert oder durch eine innere Anfrage erzeugt worden sein. Die Components innerhalb der Bedingung beschreiben hierbei die Component-Objekte, auf die sich die Selektionsbedingung bezieht, ausgenommen den Vergleichen `Ci.Document` = `Cj.Document`, da diese bei dokumentbezogenen Anfragen obligatorisch sind.

Im nächsten Teilschritt wird dann dieser Anfragebaum derart transformiert, dass Anfragen auf Inhalt immer vor Anfragen auf Strukturen ausgeführt werden, sofern dies möglich ist. Existierende Abhängigkeiten müssen dabei berücksichtigt und mittransformiert werden. Folgendes kurzes Beispiel soll dies verdeutlichen:

$$TQ_1^{Inhalt}([(TQ_2^{Struktur}[C1, C2](Cond_1(C1, C2)) \rightarrow [C1, C2])](Cond_2(C1)) \rightarrow [C1, C2])$$

Hierbei untersucht die innere Anfrage TQ_2 eine Struktur $Cond_1$ zwischen C1 und C2 und gibt entsprechende Component-Paare C1 und C2 aus. Diese Paare nimmt TQ_1 daraufhin als Eingabe, untersucht den Inhalt von C1 bezüglich $Cond_2$, und gibt diejenigen Paare von Objekten aus, die der Selektionsbedingung genügen. Diese Verschachtelung wird nun dergestalt umgeformt, dass zuerst die Anfrage TQ_1 ausgeführt wird, dann erst die Anfrage TQ_2. Das Ergebnis dieser Umformung hat dann folgende Form:

$$TQ_2^{Struktur}([(TQ_1^{Inhalt}[C1](Cond_2(C1)) \rightarrow [C1]), C2](Cond_1(C1, C2)) \rightarrow [C1, C2])$$

Das Ziel dieses Teilschritts liegt darin, dass Blätter im Anfragebaum ausschließlich inhaltsbezogene Anfragen darstellen. Auf Fälle, in denen dies nicht möglich ist, wird gesondert in Abschnitt 3.5 eingegangen. Die Motivation dieses Teilschritts basiert auf der Annahme, dass rein spatiale Anfragen im Allgemeinen eine sehr geringe Selektivität besitzen, teilweise sogar eine größere Ausgabe- als Eingabemenge erzeugen. Eine Selektionsbedingung der

[20]Diese Differenzierung der Inhaltsanfragen ist für den Abarbeitungsort der Teilanfrage entscheidend

Form `WHERE Before(C1,C2)` liefert bei n Media-Component-Objekten eines Dokuments stets eine Tupelmenge von mindestens $\binom{n}{2}$ Elementen.

Dieser voroptimierte Anfragebaum des vorangegangenen Teilschrittes wird nun im nächsten Teilschritt weiter umgeformt. Das Ziel dieser Umformung liegt nun darin, dass die Blätter des Anfragebaums jeweils von einem Medienserver, oder dem Datenbanksystem selbst, vollständig bearbeitet werden können, sich also entweder auf Basis- oder Fremdtabellen beziehen. Untersucht werden müssen demnach nur Teilanfragen der Form $TQ_i^{Inhalt_C}$, da diese nicht komplett von einem der Systeme bearbeitet werden können. Sie müssen somit gemäß der folgenden Untersuchung in verschachtelte oder verknüpfte Anfragen umgeformt werden. Ausgangspunkt ist dabei folgende abstrakte Anfrage:

$$TQ_1^{Inhalt_C}([C1](Cond_1(C1)) \rightarrow [C1])$$

Weiterhin beschreibt das Attribut W_{Meta} den Teil der Where-Klausel, der sich ausschließlich auf Metadaten bezieht, und W_{Media} den Teil, der sich ausschließlich auf Text- oder Bild-Objekte bezieht. Untersucht werden müssen nun die Kombinationsmöglichkeiten dieser Attribute.

Liegt nun die Where-Klausel in der Form W_{Meta} *OR* W_{Media} vor, so werden zwei separate Anfragen, eine auf Metadaten, eine auf Mediadaten, erzeugt, die anschließend mittels UNION vereinigt werden. Besitzt die Where-Klausel hingegen die Form W_{Meta} *AND* W_{Media}, so könnte zwar analog zum vorangegangenen Beispiel zwei Anfragen erzeugt, und mittels INTERSECT verknüpft werden. In Hinsicht auf die Anfrageausführung ist es jedoch sinnvoller, diese Anfragen zu verschachteln, so dass die äußere Anfrage lediglich das Ergebnis der inneren untersuchen muss, nicht die komplette Component-Relation. Daher wird die Anfrage TQ_1 in folgende Form transformiert:

$$TQ_2^{Inhalt_M}([(TQ_3^{Inhalt_{T|I}}([C1](Cond_3(C1)) \rightarrow C1))](Cond_2(C1)) \rightarrow [C1])$$

Hat die Where-Klausel nun die Form W_{Media} *AND* (W_{Meta} *OR* W_{Media}), ist das Ergebnis der Umformung fast identisch zum vorangegangenen Fall, bis auf dass die äußere Anfrage TQ_2 weiterhin eine kombinierte Anfrage bleibt. Diese muss nun jedoch nicht weiter untersucht werden, da sie in der From-Klausel eine weitere Anfrage referenziert, somit im Anfragebaum keinem Blatt, sondern einem inneren Knoten entspricht. Wird nun hingegen zuerst die Konjunktion, und dann die Disjunktion untersucht, die Bedingung hat somit die Form (W_{Media} *AND* W_{Meta}) *OR* W_{Media}, ist dies eine Kombination der ersten beiden Fälle. Das Ergebnis ist demnach eine verschachtelte Anfrage, die der Konjunktion entspricht, welche mittels UNION mit der zweiten Anfrage verknüpft wird. Ein Optimierungsansatz wäre an dieser Stelle, dass in der verschachtelten Anfrage zuerst die Selektion anhand von Metadaten durchgeführt wird. Somit können die innere der verschachtelten Anfrage und die nicht verschachtelte Anfrage parallel ausgeführt werden, da sie sich auf zwei verschiedene Systeme beziehen. Weitere Selektionsbedingungen bilden eine Kombination der hier vorgestellten Fälle, bedürfen somit keiner weiteren Untersuchung.

An diesem Punkt der Anfragevorverarbeitung wird nun die kombinierte Anfrage auf Struktur und Inhalt durch einen Baum repräsentiert, der aus verschachtelten und verknüpften SFW-Blöcken besteht. Die Blätter dieses teilweise optimierten Anfragebaums sind hierbei gemäß [Lin02] von einem der Medienserver oder dem Datenbanksystem selbst, vollständig bearbeitbar. Ein Beispiel hierfür ist in Abbildung 3.5 dargestellt. Abschließend wird dieser Anfragebaum in eine interne Darstellung überführt und in der anschließenden Optimierungsphase optimiert.

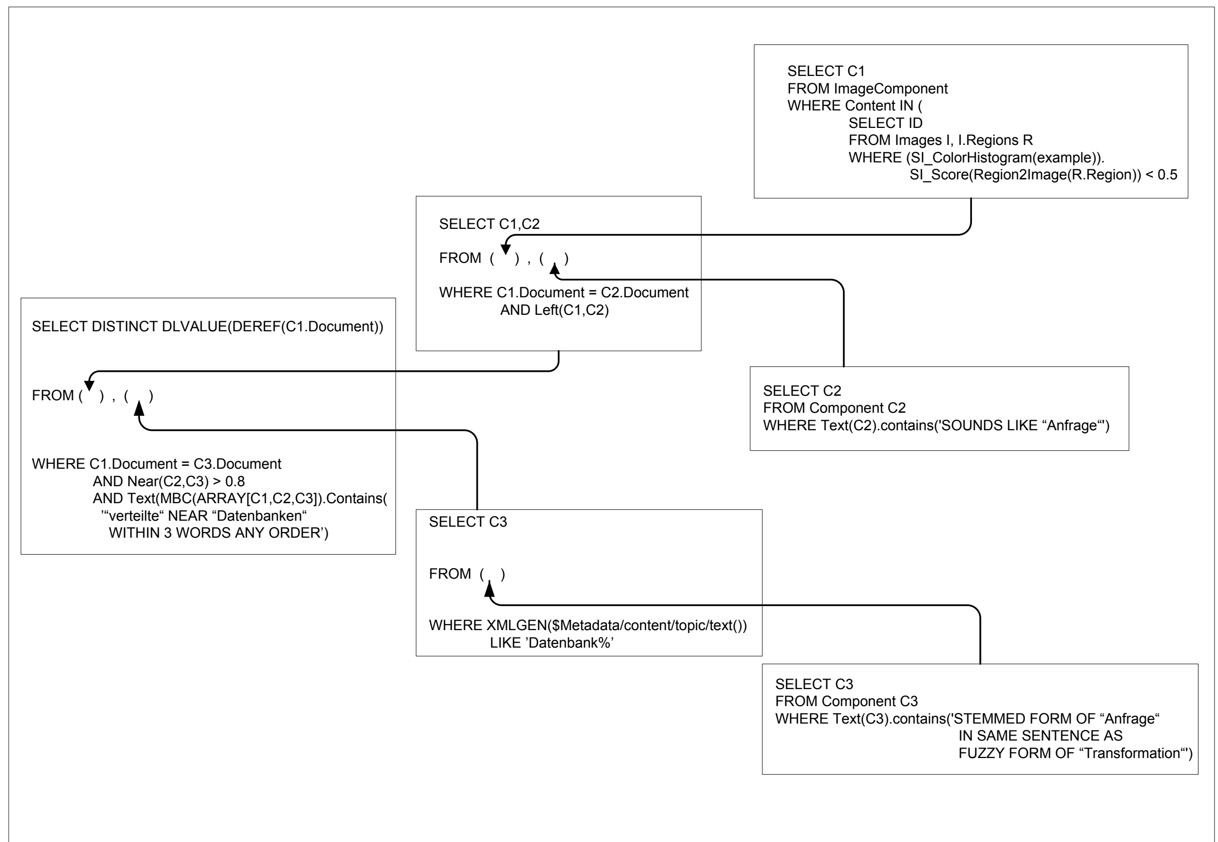


Abbildung 3.5: Beispielbaum einer verschachtelten Anfrage

3.4.2 Optimierung

Die Optimierung von Anfragen bildet einen wichtigen Teil der Anfragebearbeitung. Sie besteht nach [HS99] aus der logischen Optimierung, der physischen Optimierung und der kostenbasierten Auswahl. Wie nun eine Optimierung in Systemen mit Daten- und Funktionsintegration durchgeführt werden kann, wurde beispielsweise in [Lin02] bezüglich der dort eingeführten Compensator-Architektur ausführlich untersucht. Einige Ansatzpunkte der Optimierung wurden bereits in der Vorverarbeitung verwendet, können aber auch in die Optimierungsphase verschoben werden. Einen weiteren Ansatzpunkt bietet das Attribut *Layout* der Document-Relation bei geometrischen Anfragen. Wurde, beispielsweise während der syntaktischen Analyse erkannt, dass geometrische Anfragen verwendet

wurden, kann durch eine Erweiterung der Where-Klausel der innersten SFW-Blöcke um Prädikat `C.Document.Layout.ST_IsEmpty() = 0` sichergestellt werden, dass nur Components der Dokumente untersucht werden, deren visuelle Struktur auch definiert ist. Auf eine weiterführende Untersuchung der Optimierung muss an dieser Stelle allerdings verzichtet werden, da sie den Rahmen dieser Arbeit sprengen würde. Statt dessen wird auf entsprechende Fachliteratur verwiesen. Die Ausgabe dieses Schrittes ist somit ein optimaler Zugriffsplan, der an die Ausführungsphase übergeben wird.

3.4.3 Anfrageausführung

Die Anfrageausführung kann prinzipiell in zwei Klassen zerlegt werden, die in der Abarbeitung verzahnt ablaufen. Die eine Klasse beschreibt die Anfrageausführung auf Basistabellen, wie beispielsweise der Component-Relation. Die zweite Klasse beschreibt Anfragen, die auf den Fremdtabellen für Text- und Bild-Objekte arbeiten. Die Anfrageverarbeitung auf Fremdtabellen wird nach [MMJ+01] in die Phasen *Query Planning Phase* und *Query Execution Phase* aufgeteilt[21]. Diese wurden bereits in Kapitel 2 vorgestellt.

Die Anfrageausführung beginnt nun bei den Blättern des optimalen Zugriffsplans. Zugriffe auf Basisrelationen werden vom Datenbanksystem durchgeführt, Zugriffe auf Fremdtabellen werden an den zuständigen Wrapper weitergeleitet. In der Query Planning Phase des Wrappers wird nun durch Kommunikation zwischen foreign data wrapper und foreign server ausgehandelt, welche Befehle vom Fremdserver durchgeführt und welche vom Wrapper kompensiert werden müssen. Das resultierende Ergebnis wird daraufhin an das Datenbanksystem übermittelt. Die Ergebnisse dieser Zugriffe werden dann gemäß der Anfrage vereinigt und durch weitere Zugriffe weiter bearbeitet. Hierbei wurde bereits durch die Optimierungsphase bestimmt, ob noch weitere Interaktionen mit den Medienservern notwendig sind oder ob sämtliche übrigen medienspezifischen Funktionalitäten durch das Datenbanksystem durchgeführt werden. Ist dies der Fall, kann man die Anfrageausführung in Hol- und Verarbeitungsphase unterteilen. In der Holphase werden dann sämtliche für die Bearbeitung notwendigen Objekte wie Bilder, Teilbilder, Regionen und Textobjekte an das Datenbanksystem übermittelt. Daraufhin kann in der Bearbeitungsphase die restliche Anfrageausführung zentral erbracht werden. Diese Variante hat allerdings den Nachteil, dass der Datentransfer zunächst explosionsartig ansteigt, nach der Übergabe dann komplett abbricht. Kann hingegen die Ausführung nicht in Hol- und Verarbeitungsphase getrennt werden, erfolgen die Zugriffe auf die Fremdserver und somit der Datentransfer, verteilt über die komplette Anfrageausführung. Der Datentransfer wird somit nicht zum „Flaschenhals“ der Anfrageausführung, nachteilig wirkt sich allerdings der höhere Kommunikationsaufwand zwischen Wrapper und Fremdserver aus. Welche Variante nun konkret verwendet wird, muss im Vorfeld in der Optimierungsphase ermittelt werden.

[21]Hierfür werden die englischen Begriffe verwendet, um einen Nanenskonflikt zwischen der Anfrageausführung und der Ausführungsphase zu vermeiden

3.5 Probleme

Ein überaus wichtiges Problem bildet die in Abschnitt 3.4 erwähnte Bewertung der Anfrageergebnisse. Ein Ranking-Verfahren bezüglich der hier eingeführten Architektur muss gleich mehrere Thematiken verbinden. Da in dieser Arbeit multimediale Dokumente, wie beispielsweise HTML-Seiten, in einer objektrelationalen Datenbank gespeichert werden, müssen sowohl klassische Ranking-Verfahren, die den Inhalt der Dokumente auswerten, Verwendung finden, aber auch Techniken aus dem Web Information Retrieval, wie linktopologische Verfahren. Auf die Möglichkeiten und Unterschiede dieser Techniken wird unter anderem in [Lew05] eingegangen. Aus dieser Unterteilung wird bereits ein weiteres Problem deutlich. Klassische Ranking-Verfahren arbeiten entweder auf Texten oder auf Bildern. Über Ranking-Verfahren, die in kommerziellen Systemen verwendet werden, um beispielsweise die Qualität von Dokumenten bezüglich einer Anfrage zu ermitteln, wird in [Lew05] geschrieben, dass Unternehmen diese Verfahren nicht veröffentlichen, um sich einen Vorteil gegenüber der Konkurrenz zu verschaffen. Somit wird eine Untersuchung dieser Problematik weiter erschwert. Ein weiteres, mit dem medienübergreifenden Ranking-Verfahren eng verwandtes Problem stellt die verteilte Datenhaltung dar. Da Anfragen unter Umständen auf Daten von drei verschiedenen Servern zugreifen, müssen weiterhin verteilte Ranking-Verfahren betrachtet werden. Im Gegensatz zu den Techniken des Web Information Retrieval liegen hierbei nicht die Dokumente auf verschiedenen Rechnern, vielmehr kann bereits ein einzelnes Dokument auf verschiedenen Rechnern partitioniert vorliegen. Ein auf dieser Arbeit aufbauendes Ranking-Verfahren sollte nun all diese Themen vereinen. Projekte, die sich nun mit dieser Problematik befassen, sind unter anderem Garlic [RAH+96] und Heron [KEUB+98]. Auf Grund der Komplexität dieser Thematik liegt dies jedoch ausserhalb des Fokus dieser Arbeit.

Ein für die Performance der Anfragebearbeitung gravierendes Problem stellen Teilanfragen auf Strukturen dar, die im Anfragebaum als Blatt auftreten und nicht durch Transformation zu einem inneren Knoten umgeformt werden können. Dies tritt im Allgemeinen bei Verwendung der Methode *MBC* auf. Beispielsweise kann folgende Anfrage nicht problemlos derart umgeformt werden, sodass sich die innere Anfrage auf den Inhalt und die äußere auf die Struktur bezieht.

```
SELECT C3
FROM (SELECT MBC(ARRAY[C1,C2]) AS C3
      FROM TextComponent C1, ImageComponent C2
      WHERE C1.Document = C2.Document
         AND Left(C1.Shape,C2.Shape)
WHERE Text(C3).CONTAINS('Anfrageverarbeitung')
```

Eine optimierte Lösung dieser Anfrage muss zunächst mittels folgender Anfrage alle Component-Objekte ermitteln, welche zum einen aus weiteren Components bestehen und zum anderen das Wort „Anfrageverarbeitung“ enthalten:

```
SELECT C3
FROM ONLY Component C3
WHERE Text(C3).CONTAINS('Anfrageverarbeitung')
```

Auf Datenbankseite muss daraufhin ermittelt werden, ob dieses Component-Objekt sowohl ein Text- als auch ein ImageComponent enthält, wobei das Text- links vom ImageComponent liegt. Ist dies der Fall muss überprüft werden, ob das Ergebnis der Methode *MBC* angewandt auf diese Component-Objekte genau dem Component C3 entspricht. Um dies zu ermöglichen, müsste zusätzlich zu der in Abschnitt 3.3.1 beschriebenen Methode *Pictures* eine Methode *Texts* eingeführt werden, welche eine Liste aller enthaltenen TextComponents zurückgibt. Somit könnten diese beiden Methoden auf das Component-Objekt C3 angewendet und die Ergebnisse entsprechend weiterverarbeitet werden.

Kapitel 4

Praktische Relevanz

Dieses Kapitel behandelt nun die praktische Relevanz des in Kapitel 3 vorgestellten Konzepts. Die Relevanz bezieht sich hierbei auf die Unterstützung, die ausgereifte, bestehende, SQL-basierte Systeme bezüglich der Konzeption zur Verfügung stellen. Zwar ist der aktuelle Standard SQL:2003 noch sehr neu, allerdings werden dort auch viele Konzepte integriert, die in kommerziellen Systemen bereits seit längerem verwendet werden, wie beispielsweise die Integration von XML. Die hier durchgeführte Untersuchung bezieht sich allerdings ausschließlich auf die beiden marktführenden Systeme von IBM und Oracle und kann nur sehr oberflächlich durchgeführt werden, da eine tiefergehende Betrachtung den Rahmen dieser Arbeit sprengen würde und auch nicht im Fokus dieser Arbeit liegt. Für nähere Informationen wird jeweils entsprechende Literatur angegeben.

4.1 IBM DB2

Das erste hier untersuchte System ist die Universal Database von IBM[1], kurz **IBM DB2 UDB** in der aktuellen Version 8.2 [IBM04c, IBM04d]. Das Prinzip, das IBM mit dieser Datenbank verfolgt, basiert auf der Extender-Politik. Es wird dabei durch die Datenbank DB2 nur der Datenbankkern mit allgemeinen Funktionalitäten ausgeliefert. Zusätzliche Features wie die Verarbeitung von geometrischen oder Bildobjekten bedürfen sogenannten Extendern, die nach dem Einbinden die Funktionalität der Datenbank entsprechend erweitern. Dies hat den Vorteil, dass die entsprechenden Komponenten jeweils separat weiterentwickelt werden können, ohne auf die Entwicklung anderer Komponenten Auswirkungen auszuüben. Nachfolgend wird nun die Unterstützung allgemein notwendiger Konstrukte untersucht, auf die Möglichkeiten der speziellen Extender wird anschließend gesondert eingegangen.

Eine Grundvoraussetzung für die Umsetzung der in Kapitel 3 vorgestellten Konzeption bildet die Möglichkeit der Definition von strukturierten Datentypen, kurz **UDT**, und korrespondierenden typisierten Tabellen, wie beispielsweise *Document_T* und *Document*.

[1] http://www.ibm.com

Diese Funktionalität wird nun von DB2 vollständig umgesetzt, so dass die Grundvoraussetzung für eine Umsetzung der Konzeption bereits gegeben sind. Die exemplarische Umsetzung der typisierten Collection-Relation hätte dann folgende Form, wobei der Zusatz `MODE DB2SQL` obligatorisch ist:

```
CREATE TYPE Collection_T AS (
   Documents REF(Document_T))
MODE DB2SQL

CREATE TABLE Collection OF Collection_T
{
   REF IS OID USER GENERATED,
   Documents WITH OPTIONS SCOPE Document
}
```

Ein zweiter wesentlicher Punkt der Konzeption war die Deklaration von Funktionen, die in Abschnitt 3.3.1 eingeführt wurden. Hierbei konnte unterschieden werden zwischen einfachen SQL-Routinen, wie beispielsweise der Segmentierungsfunktion *PARAGRAPH*, und externen Routinen, deren Methodendeklaration in einer externen Host-Programmiersprache vorliegt, wie die Funktion *Region2Image*. Beide Varianten werden nun von DB2 unterstützt, so dass diesbezüglich keinerlei Restriktionen existieren und die Konzeption hinsichtlich der Funktionen komplett umgesetzt werden kann.

Für die Umsetzung birgt das Typsystem von DB2 allerdings einige Probleme. Ein wesentlichen Mangel stellt dabei die fehlende Möglichkeit eines anonym strukturierten Typs, in Standard-SQL der **ROW**-Konstruktor, dar. Aus diesem Grund müssen die in Kapitel 3 verwendeten, mittels ROW strukturierten Attribute in abgeänderter Form realisiert werden. Da DB2 die Definition von UDTs unterstützt, müssen nun sämtliche anonym strukturierte Attribute durch UDTs umgesetzt werden. Beispielsweise muss die Deklaration des Regions-Attribut der Image-Tabelle durch eine Referenz auf eine typisierte Tabelle des strukturierten Typs Regions_T ersetzt werden, der durch die oben beschriebene Syntax erzeugt wird. Dies hat in der Anfrageformulierung zur Folge, dass eine Pfeilstatt einer Punktnotation verwendet werden muss. Ein weiteres Problem liegt in den fehlenden Konstruktoren **ARRAY** und **MULTISET**, durch die ein Attribut aus einer Liste beziehungsweise Multimenge von Werten bestehen kann. Dieser Nachteil muss in der Umsetzung durch die klassische Realisierung von 1:n- und n:n-Beziehungen kompensiert werden. So enthält beispielsweise die Relation Component kein multimengenwertiges Attribut *Subcomponents*, das die Kindknoten referenziert, sondern ein Attribut *Supercomponent*, welches einer Referenz auf das übergeordnete Component-Objekt entspricht. Dies muss dann in der Anfrageformulierung und der Funktionsdefinition berücksichtigt werden.

4.1.1 Integration externer Daten

Zur Integration externer Daten wurden in der Konzeption die Konstrukte **Datalink** und **Foreign Table** verwendet. Ein Datalink, welcher einer Referenz auf eine extern gespeicherten Datei entspricht, wird auch von DB2 angeboten. Hierfür wird bei der Instantiierung ein Pfadausdruck angegeben, durch welchen die Datei lokalisiert wird. Analog zu SQL kann hierbei der Grad der Kontrolle der Datenbank über die Datei angegeben werden. Dieser Grad reicht von keiner bis zur vollständigen Kontrolle, bei welcher durch Löschen des Attributs auch die Datei gelöscht wird. Die Syntax hierfür ist der Referenz [IBM04d] zu entnehmen. Wie auch bei SQL kann bei DB2 keine Anfrage auf den Inhalt der Datei gestellt werden, jedoch auf ein optional definierbares Comment-Attribut. Dieses kann durch den Befehl DLCOMMENT extrahiert und verglichen werden. Der nachfolgende Befehl beschreibt die Attributzuweisung des Attributs *Document* der Document-Relation:

```
DLVALUE('/home/user/DA.pdf','URL','kombinierte Anfrageverarbeitung')
```

Die zweite Möglichkeit der Datenintegration, in der Konzeption die Verwendung von Foreign Tables, wird in DB2 in leicht abgewandelter Form ebenfalls unterstützt. Die Integration basiert hierbei auf der Definition einer tabellenwertigen Funktion. Die zu integrierenden Daten werden dabei beispielsweise von einem OLE DB-Provider zur Verfügung gestellt[2]. Um diesen in der Datenbank zu registrieren, wird ein CREATE SERVER-Statement verwendet, in welchem auch der Wrapper spezifiziert wird. Daraufhin kann die entsprechende tabellenwertige Funktion definiert werden. Der folgende DDL-Aufruf verdeutlicht dies, wobei das Äquivalent der Fremdtabelle Text definiert wird:

```
CREATE FUNCTION Text()
  RETURNS TABLE (ID Integer,
                 Content CLOB,
                 Language Varchar)
  LANGUAGE OLEDB
  EXTERNAL NAME textserver!db.relation;
```

Diese Funktion kann sodann an die Funktion `TABLE` als Parameter übergeben werden und daraufhin in der From-Klausel referenziert werden. Zusätzlich können für die Funktion auch Parameter definiert werden. Durch diese Parameter ist es möglich, Anfragen und Selektionsbedingungen zu formulieren und an den OLE DB-Provider mitzusenden, welcher diese dann verarbeiten kann. Ein entsprechendes Beispiel hierfür ist in [IBM04d] zu finden. Somit kann festgehalten werden, dass die für die Konzeption notwendige Integration externer Daten von DB2 in korrespondierender Form bereitgestellt wird.

[2] Es ist natürlich auch die Verwendung andersartiger Server möglich, allerdings wird die Integration von OLEDB-Providern durch vordefinierte Funktionalitäten, wie beispielsweise der OLEDB-Wrapper, stark vereinfacht

4.1.2 Unterstützung von XML

Zur Integration von XML in DB2 stellt IBM den *XML Extender*[IBM03b] bereit, welcher es ermöglicht, XML-Dokumente zu speichern und zu bearbeiten. Im Mittelpunkt stehen dabei die folgenden neuen Datentypen:

- **XMLVARCHAR** für Attribute kleiner als 3 Kilobyte
- **XMLCLOB** für Attribute größer als 3 Kilobyte
- **XMLFILE** für außerhalb von DB2 gespeicherte Dokumente

Um nun auch den Inhalt von XML-wertigen Attributen ausgeben zu lassen, um diese entweder in die Projektionsliste zu übernehmen oder als Selektionsbedingung auszuwerten, stellt der XML Extender nun eine Reihe von *extractDATATYPE*-Methoden zur Verfügung. Die Zeichenkette DATATYPE repräsentiert hierbei die von DB2 bereitgestellten Basisdatentypen. Soll beispielsweise ein Component ermittelt werden, in welchem die Person John Doe vorkommt, geschieht dies durch folgenden Aufruf:

```
SELECT oid
FROM Component
WHERE db2xml.extractVarchar(Metadata,'/content/person') = 'John Doe'
```

Wie an diesem Beispiel zu erkennen ist, verwendet die extract-Funktion zur Lokalisierung eines XML-Fragments XPath-Ausdrücke[W3C04a]. Allerdings ist eine Verwendung von XQuery-Ausdrücken, wie sie SQL/XML definiert sind, im XML Extender nicht möglich. Diese Restriktion muss in der Umsetzung der Konzeption berücksichtigt werden.

4.1.3 Verwaltung von Bildobjekten

Das Speichern von Text- und Bildobjekten erfolgt laut Konzeption zwar auf einem speziellen Medienserver, dennoch sollte die Datenbank die Möglichkeit bieten, so viele Funktionalitäten wie möglich zu unterstützen, um Anfragen kompensieren zu können und nicht immer an den entsprechenden Medienserver auslagern zu müssen, so dass die Kommunikationskosten gering gehalten werden.

Zum Speichern und Verwalten von Bildobjekten bietet DB2 den Extender für Audio, Image und Video, kurz *AIV Extender* [IBM03a], dessen Entwicklung allerdings eingestellt wurde. In diesem Extender wird der für die Verarbeitung von Bildern erforderliche Datentyp **DB2IMAGE** definiert. Für diesen Datentyp existiert nun ein Konstruktor, durch den eine Instanz erzeugt wird. Hierbei kann unterschieden werden, ob die Instanz durch ein BLOB oder eine Referenz auf die entsprechende Datei erzeugt werden soll. Hinsichtlich der Konzeption ist hauptsächlich der BLOB interessant, da die Daten des Medienservers für Bildobjekte nach Kapitel 3 in Form eines BLOB an das Datenbanksystem übermittelt werden. In [IBM03a] wird diesbezüglich folgendes Beispiel in embedded SQL gegeben:

```
DB2IMAGE(              /*Image Extender UDF*/
   CURRENT SERVER,     /*database server name in*/
                       /CURRENT SERVER register*/
   :himage,            /*image as BLOB from host variable*/
   'ASIS',             /*keep the image format*/
   :hvInt_Stor,        /*store image in DB as BLOB*/
   'Anita''s picture'), /*comment*/
```

Um nun Anfragen auf Bilder zu stellen, bietet DB2 eine *Query By Image Content*-Funktionalität, kurz QBIC, an. Die Ähnlichkeitsberechnung erfolgt dabei analog zu SQL/MM Still Image über die Features *histogram*, *texture* sowie *average* und *positional color*. Hierzu muss allerdings ein QBIC-Katalog erstellt werden, der die Bilder der Datenbank zum Suchen indiziert. Somit ist es hinsichtlich der Konzeption nicht möglich, ein vom Image Server erhaltenes Bild on-the-fly bezüglich der Features zu untersuchen. Durch diese Restriktion müssen nun inhaltliche Anfragen auf Bilder stets an den entsprechenden Medienserver für Bildobjekte übermittelt werden. Arbeitet dieser ebenfalls mit einer DB2-Datenbank hätte eine Beispielanfrage folgende Form, ebenfalls entnommen aus [IBM03a]:

```
SELECT name, description
FROM fabric
WHERE CAST (swatch_img as varchar(250))IN
  SELECT CAST (image_id as varchar(25))
  FROM TABLE (QbScoreTBFromStr(
               QbTextureFeatureClass file=<server,"patterns/ptrn07.gif">'
               'fabric',
               'swatch_img', 10))
       AS T1));
```

4.1.4 Verwaltung von Volltextobjekten

Zur Bereitstellung von Information Retrieval-Funktionalitäten stellt IBM gleich zwei Extender zur Verfügung, den Text Extender[IBM04b] und den Net Search Extender[IBM04a]. Allerdings bieten beide keinen eigenen Datentyp mit entsprechenden IR-Methoden an. Stattdessen werden Features wie Stammwortreduktion und die Verwendung eines Thesaurus ausschließlich durch einen Index realisiert. Es besteht somit nicht die Möglichkeit, das textuelle Ergebnis einer Anfrage in ein Volltextobjekt zu casten, um darauf Methoden anzuwenden. Dies stellt ein wesentliches Problem dar, wenn Textobjekte auf einem Medienserver gespeichert werden und auf das Ergebnis einer inneren Anfrage eine IR-Anfrage gestellt werden soll, da auf dem Ergebnis kein Index existiert. Die Suchfunktionalitäten, der nun Text Extender zur Verfügung stellt, werden dabei in folgende Klassen eingeteilt:

- Unscharfe Suche (FUZZY FORM)
- Eine umschließende Suche für die koreanische Sprache (BOUND)
- Phonetische Suche (SOUNDS LIKE)
- Suche mit Hilfe eines Thesaurus (EXPAND)
- Freitext- und Hybridsuche (IS ABOUT)

Da diese Funktionalitäten aber nur nutzbar sind, wenn ein Index angelegt wurde, kann der Text-Extender nur auf dem Medienserver für Textobjekte oder, im Gegensatz zur Konzeption, in einer zentralisierten Architektur eingesetzt werden. In dem Fall könnte eine beispielhafte Anfrage folgende Form besitzen:

```
SELECT oid
FROM TABLE(Text())
WHERE CONTAINS
        (Content,
         'THESAURUS "myterms"
         EXPAND "SYN"
         TERM OF "document management system"') = 1
```

4.1.5 Verwaltung geometrischer Objekte

Der Extender, der das Speichern und Verarbeiten von geometrischen Objekten ermöglicht, ist der *Spatial Extender* [IBM01]. Analog zu SQL/MM wird durch diesen Extender der abstrakte Datentyp **ST_Geometry** eingeführt. Durch Ableitung einzelner Klassen wird nun ebenfalls eine Typhierarchie eingeführt, die einer Untermenge von SQL/MM Spatial entspricht. Somit ist es möglich, null-, ein- und zweidimensionale Objekte, als auch Mengen eines Typs als Attribut zu deklarieren. Die Instantiierung eines Attributs soll nun an einem Beispiel illustriert werden, im dem ein Rechteck als Attributwert definiert wird:

```
db2gse.ST_PolyFromText(
   'polygon((10.00 20.00,
            20.00 20.00,
            20.00 10.00,
            10.00 10.10,
            10.00 20.00))',
            db2gse.coordref()..srid(0)))
```

Um nun einzelne ST_Geometry-Objekte in Beziehung zu setzen, bietet der Spatial Extender ebenfalls eine Reihe von Methoden an, welche mit denen von SQL/MM identisch sind. Auf eine komplette Vorstellung sämtlicher Typen und Methoden wird an dieser Stelle verzichtet, stattdessen wird auf die Referenzliteratur [IBM01] verwiesen. Zusammenfassend

kann festgehalten werden, dass die Funktionalitäten, die für eine Umsetzung der Konzeption notwendig sind, durch den Spatial Extender bereitgestellt werden. Die Funktionsweise dieses Extenders soll nun abschließend durch eine Beispielanfrage illustriert werden.

```
SELECT C1.oid, C2.oid
FROM Component C1, Component C2
WHERE C1.Document = C2.Document
   AND db2gse.ST_Distance(C1.Shape, C2.Shape) < 10
```

4.2 Oracle

Im Gegensatz zu DB2 verfolgt Oracle[3] mit der hier vorgestellten Version **Oracle 10g** [Ora03a] das Prinzip einer ganzheitlichen Datenbank, die alle verwendbaren Funktionalitäten bereits von vornherein anbietet. Diese müssen lediglich aktiviert werden. Es ist nicht notwendig, spezielle Erweiterungen einzubinden, um die in Kapitel 3 vorgestellte Konzeption umzusetzen. Nachfolgend wird nun zunächst das Typkonzept von Oracle 10g vorgestellt, bevor daraufhin auf die Verwendung spezieller Funktionalitäten, wie XML- und geometrische Objekte, genauer eingegangen wird.

Das Typsystem von Oracle ist in einigen Bereichen besser zur Umsetzung der Konzeption geeignet als die oben beschriebene Datenbank DB2. Auch Oracle bietet die Möglichkeit, benutzerdefinierte Datentypen und Funktionen zu definieren, jedoch ist das Typsystem von Oracle umfangreicher. So existiert einerseits mit dem Konstruktor **VARRAY** die Möglichkeit, ein listenwertiges Attribut anzulegen, andererseits bietet Oracle mit dem Befehl `AS TABLE OF` die Möglichkeit, eine Multimenge, analog dem Konstruktor **MULTISET** aus SQL:2003, zu erzeugen, wie beispielsweise das Attribut *Subcomponents* der Component-Relation. Dies wird dann durch folgenden Befehl realisiert:

```
CREATE TYPE Subcomponents AS TABLE OF Component_T
```

Dieser Typ kann daraufhin als Nestet Table mittels Alter-Statement in die Typdefinition von Component_T aufgenommen werden. Somit ist es nicht notwendig, wie in DB2 die Zeigerstruktur zu invertieren, um eine 1:n-Beziehung darzustellen.

Nachteilig ist allerdings, dass, wie auch bei DB2, kein Analogon zum ROW-Konstruktor existiert, sodass dies ebenfalls durch einen UDT umgesetzt werden muss, der dann als Attribut verwendet wird.

---

[3]http://www.oracle.com/

4.2.1 Integration externer Daten

Ähnlich zu DB2 bietet auch Oracle Äquivalente zu Datalink und Foreign Table. Das Gegenstück zum Datentyp DATALINK bildet in Oracle der Typ **BFILE**. Hierdurch kann eine externe Datei referenziert und als Attributwert verwendet werden. Allerdings wird im Gegensatz zum SQL- und DB2-Datentyp DATALINK keine Konsistenz der Datei gewährleistet. Es ist somit Aufgabe des Administrators sicherzustellen, dass die Datei existiert und der Datenbestand diesbezüglich konsistent bleibt.

Um nun extern gespeicherte Daten nicht nur zu referenzieren, sondern auch den Inhalt zu untersuchen, bietet Oracle die Möglichkeit der Definition eines **External Table** an. Dabei wird an die normale Deklaration einer Tabelle die ORGANIZATION EXTERNAL-Klausel angehängt. Diese Klausel spezifiziert dabei, wo die Daten gespeichert sind und wie auf diese Daten zugegriffen wird. Allerdings sind in [Ora03a] weder Informationen über die Verwendung von Wrappern, noch über die Integration von Daten anderer Datenbanken vorhanden, sodass diesbezüglich an dieser Stelle keine Aussage getroffen werden kann. Es wird lediglich beschrieben, wie auf den Inhalt von extern gespeicherten Dateien zugegriffen wird. Dies erfolgt in der Deklaration der ORGANIZATION EXTERNAL-Klausel, wodurch diese stark anwächst. Auf ein Beispiel wird daher verzichtet.

4.2.2 Unterstützung von XML

Zur Verwendung von XML in Typ- und Tabellendefinitionen stellt Oracle den Datentyp **XMLType** zur Verfügung. Die Verwendung dieses Datentyps erfolgt analog zu den "normalen" Datentypen, sodass die Konzeption diesbezüglich problemlos umgesetzt werden kann. Um auf den Inhalt eines Attributs vom Typ XMLType zuzugreifen, besitzt dieser Typ die nachfolgend aufgelisteteten Methoden:

- *extract* extrahiert den Inhalt eines XML-wertigen Attributs
- *extractValue* extrahiert zunächst den Inhalt mittels extract(), und wendet darauf die Methode getStringVal() an
- *existsNode* testet, ob ein angegebener Knoten existiert

Innerhalb dieser Funktionen kann dabei mittels XPath [W3C04a], einer Anfragesprache des W3C[4], die auf Pfadausdrücken basiert, innerhalb des XML-wertigen Attributs navigiert werden, wie folgende Beispielanfrage verdeutlicht:

```
SELECT C.oid
FROM Component C
WHERE C.Metadata.extract('/title/text()').getStringVal() LIKE '%Anfrage%'
```

[4]World Wide Web Consortium

Eine weitere Möglichkeit der Anfrageformulierung basiert nach [Ora05a] auf der Verwendung von XQuery-Ausdrücken, analog zu SQL/XML. Hierfür wird die Konstruktormethode *XMLQuery* verwendet, in welcher dann ein XQuery-Ausdruck formuliert werden kann. Allerdings wird in [Ora05a] lediglich beschrieben, wie diese Funktion in der Select-Klausel verwendet werden kann, um eine Ausgabemenge zu erzeugen. Wie diese Funktionalität nun auch in der Where-Klausel verwendet werden kann, wurde bereits in Kapitel 2 im Kontext der Funktion *XMLGEN* erläutert.

4.2.3 Verwaltung von Bildobjekten

Zum Speichern, Bearbeiten und Anfragen von Bildobjekten stellt Oracle das Feature *interMedia* [Ora03b] bereit, welches diese Funktionalitäten in der Datenbank aktiviert. Durch dieses Feature wird laut [Ora03b] eine vollständige Unterstützung des SQL-Standards *ISO/IEC 13249-5:2001 SQL MM Part5 : StillImage*, eine Vorgängerversion des in Kapitel 2 vorgestellten Standards, realisiert. Eine Untersuchung der Verwaltung von Bildobjekten erübrigt sich somit, da die Liste der Anforderungen gemäß der Konzeption identisch ist zu der Liste der Möglichkeiten, die Oracle bereitstellt. Selbst die Syntax ist identisch zum Standard, sodass die Konzeption diesbezüglich komplett umgesetzt werden kann. Zusätzlich zum SQL/MM-Datentyp **SI_StillImage** führt Oracle noch den Datentyp **ORDImage** ein, welcher die Möglichkeiten der Verwaltung von Bildobjekten noch zusätzlich erweitert.

4.2.4 Verwaltung von Volltextobjekten

Zur Verwendung von Information Retrieval-Funktionalitäten bietet Oracle analog zu DB2 leider keinen eigenen Datentyp mit entsprechenden IR-Methoden an. Stattdessen werden ebenfalls Suchfunktionalitäten wie unscharfe Suche und die Verwendung eines Thesaurus ausschließlich durch einen Index realisiert. Vorteilhaft ist allerdings, dass auch extern gespeicherte Daten indiziert werden können, so dass auch Daten, die von einem speziellen Volltextserver verwaltet werden, von Oracle indiziert werden können. Dieses Feature, Oracle Text bezeichnet, ist dabei vollständig in Oracle integriert. Eine Klassifizierung der Suchoperatoren erfolgt in [Ora05b] auf folgende Weise:

- Stichwortsuche
- Kontextsuche
- Boolesche Operatoren zur Suchkombination
- Linguistische Suche
- Mustersuche

Diese Suchfunktionalitäten sind isomorph zu denen, die in SQL/MM Full Text spezifiziert werden, gehen sogar darüber hinaus, sodass die Konzeption bezüglich der Anfrageformulierung korrekt umgesetzt werden kann. Eine abschließende Beispielanfrage soll die Syntax zur Volltextsuche von Oracle illustrieren.

```
SELECT id
FROM Text
WHERE CONTAINS(Content,
               '"Anfrage" IN SAME SENTENCE AS "Verarbeitung"',1) > 0
```

4.2.5 Verwaltung geometrischer Objekte

Zur Verwaltung geometrischer Objekte bietet Oracle das Package *Oracle Spatial* [Ora03c] an. Damit wird ein neuer Datentyp **SDO_Geometry** definiert, welcher das Äquivalent zum Package SQL/MM Spatial bildet. Analog zu SQL/MM existiert eine Reihe von speziellen Ausprägungen eines SDO_Geometry-Objekts, und zwar null-, ein- und zweidimensionale Objekte sowie Mengen eines Typs. Folgender Ausdruck repräsentiert die Definition eines Rechtecks in Oracle, entnommen aus [Ora03c]:

```
SDO_GEOMETRY(
   2003, -- two-dimensional polygon
   NULL,
   NULL,
   SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
   SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
   -- define rectangle (lower left and upper right) with
   -- Cartesian-coordinate data
```

Um nun einzelne SDO_Geometry-Objekte in Beziehung zu setzen, bietet Oracle ebenfalls eine Reihe von Methoden an, deren Mächtigkeit im Wesentlichen mit der von SQL/MM isomorph ist. Auf eine komplette Vorstellung sämtlicher Typen und Methoden wird an dieser Stelle verzichtet, stattdessen wird auf die Referenzliteratur [Ora03c] verwiesen. Zusammenfassend kann festgehalten werden, dass die Funktionalitäten, die für eine Umsetzung der Konzeption notwendig sind, von Oracle bereitgestellt werden. Die Funktionsweise von Oracle Spatial soll nun abschließend durch eine Beispielanfrage illustriert werden.

```
SELECT C1.oid, C2.oid
FROM Component C1, Component C2
WHERE C1.Document = C2.Document
   AND sdo_within_distance (
      C1.Shape, C2.Shape,
      'distance=15 unit=pixel') = 'TRUE';
```

4.3 Fazit

Aus der vorangegangenen Untersuchung der Möglichkeiten, die IBM DB2 und Oracle hinsichtlich der Konzeption zur Verfügung stellen, wurde deutlich, dass praktisch alle relevanten Konzepte zumindest theoretisch umgesetzt sind[5]. Somit bieten sich beide Datenbanken zur Realisierung der Konzeption an. In Tabelle 4.1 ist nochmals eine Kurzzusammenfassung der vorherigen Abschnitte wiedergegeben. Wie auch an dieser Gegenüber-

	DB2 V8.2	Oracle 10g
Allgemeine Funktionalitäten	↓	↘
Datenintegration	→	↓
XML-Verwaltung	↓	→
Bildverwaltung	↗	↑
Volltextverwaltung	→	→
Verwaltung geometrischer Objekte	→	→

↑: reicht über die Möglichkeiten von SQL hinaus
→: ist zu den Möglichkeiten von SQL isomorph
↓: reicht nicht an die Möglichkeiten von SQL heran
↘; ↗: von der Mächtigkeit her zwischen SQL und Kontrahent

Tabelle 4.1: Vergleich zwischen DB2 und Oracle bezüglich der Konzeption

stellung zu erkennen ist, ist es sehr schwierig, eine Datenbank zu favorisieren, da beide sowohl Vor- als auch Nachteile gegenüber der jeweils anderen haben. So ist DB2 in Bezug auf die Datenintegration auf jeden Fall Oracle vorzuziehen, bleibt jedoch hinsichtlich der XML-Verarbeitungsfunktionalität auf der Strecke, da keine XQuery-Ausdrücke integriert werden können[6]. Oracle kommt unter anderem zu Gute, dass einerseits das Typkonzept mächtiger ist und andererseits der Standard SQL/MM StillImage vollständig umgesetzt wurde und somit das Konzept in dieser Hinsicht eins zu eins umgesetzt werden kann. Es hängt somit von den Anforderungen der Umsetzung, und nicht zuletzt von den Präferenzen des Umsetzenden ab, für welche Datenbank sich im Einzelfall entschieden wird. Da jedoch der Datenintegration in Kapitel 3 eine starke Gewichtung zukommt, wird an dieser Stelle die Datenbank **IBM DB2 UDB V8.2** favorisiert.

[5]Dies liegt auch daran, dass Oracle und IBM im Normierungsgremium vertreten sind und bestrebt sind, eigene Entwicklungen in die Normierung einfließen zu lassen

[6]Allerdings wird laut [NvdL05] in der kommenden Version von DB2 der Datentyp XML als Basisdatentyp eingeführt, auf welchen dann mit XQuery zugegriffen werden kann

Kapitel 5

Prototypische Implementierung

Nachdem in Kapitel 3 die Konzeption einer kombinierten Anfrage auf Struktur und Inhalt multimedialer Dokumente ausgiebig erläutert wurde, wird sich dieses Kapitel mit einer prototypischen Implementierung befassen. Ziel dieser Implementierung ist es nicht, das entwickelte Konzept vollständig umzusetzen. Vielmehr soll es die grundsätzliche Machbarkeit des Konzepts untermauern. Die Umsetzung erfolgt dabei auf einem Teil des Datenbestandes des eNoteHistory Projekts. Weiterhin dient dieser Prototyp nicht dazu, das Projekt eNoteHistory um relevante Funktionalitäten zu erweitern, es wird lediglich der existierende Datenbestand des Projekts verwendet, um eine prototypische Umsetzung der Konzeption zu realisieren. Der Aufbau dieses Kapitels kann nunmehr in vier Teile eingeteilt werden. Zu Beginn wird vorgestellt, welcher Teil des eNoteHistory Datenmodells in das Dokumentenmodell der Konzeption übernommen wird. Ausgehend von dieser Abbildung wird daraufhin der logische Aufbau des Prototyps, sowie der notwendige Import der Daten beschrieben, auf welchen abschließend die Anfragemöglichkeiten untersucht werden.

5.1 Vorbetrachtungen

In diesem Abschnitt wird nun beschrieben, wie das Datenmodell des Projekts eNoteHistory teilweise auf den in Abschnitt 3.2 vorgestellten logischen Aufbau eines Multimedia-Datenbanksystems abgebildet werden kann. Hierzu erfolgt zunächst eine Reduktion des eNoteHistory-Projekts auf einen für die prototypische Implementierung hinreichenden Ausschnitt. So werden die Schemata *Dict* und *Features* komplett sowie das Schema *Metadata* teilweise aus dem Prototyp ausgegrenzt. Der Prototyp bezieht sich demnach auf ausgewählte Daten des Schemas *IPFV* sowie auf einen Teil der Daten des Schemas *Metadata*. Um diese Daten zu ermitteln, muss zunächst eine Menge der Klassen des eNoteHistory-Projekts bestimmt werden, die injektiv auf das Multimedia-Datenbanksystem aus Abschnitt 3.2 abgebildet werden können. Hierzu bietet sich folgende Hierarchie an:

- *Collection* enthält *Music_Manuscripts*
- *Music_Manuscript* besteht aus *Incipit* und *Music_Manuscript_Section*
- *Music_Manuscript_Section* besteht aus *Music_Manuscript_Pages*
- *Music_Manuscript_Pages* wird ein *Page_Image* und ein *Page_Image_ROI* zugeordnet
- *Page_Image_ROI* enthält die Regionen *Staff_Lines*, *Bar_Lines*, *Note_Head* und *Note_Stem*

Diese Klassen können nun auf die typisierten Tabellen der Konzeption injektiv abgebildet werden. In Abbildung 5.1 sind nun sowohl die Klassen des eNoteHistory-Projekts dargestellt, die in der prototypischen Implementierung verwendet werden, als auch die injektive Abbildung dieser Klassen auf die typisierten Tabellen des Prototyps. Aus Gründen der Übersichtlichkeit werden jedoch nur die Attribute aufgezeigt, die für die Abbildung und die spätere Anfrageverarbeitung von Bedeutung sind.

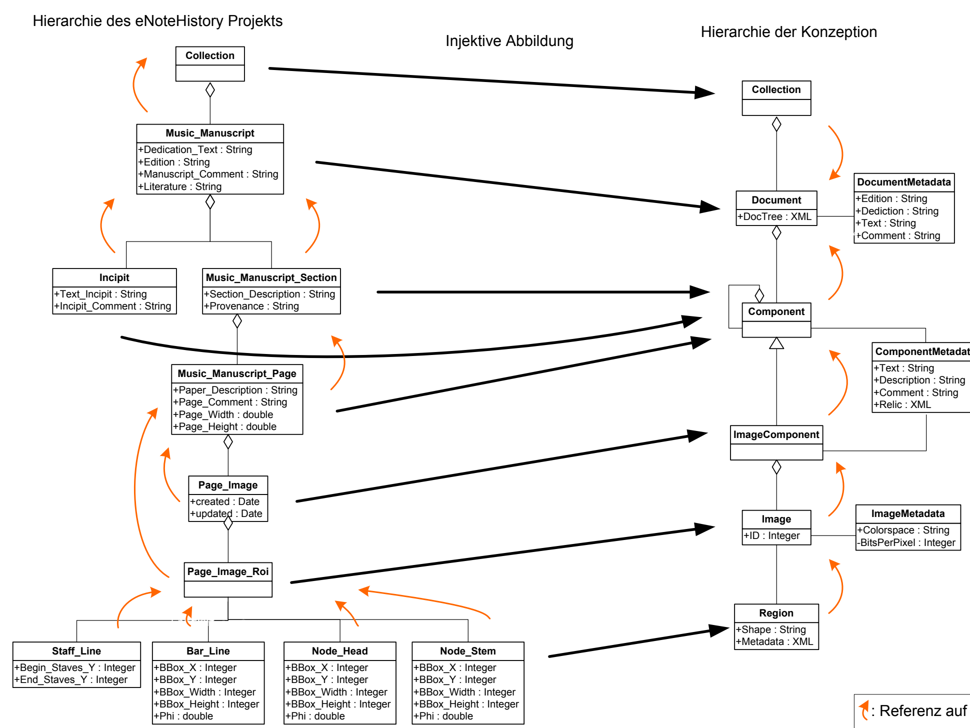


Abbildung 5.1: Abbildung des eNoteHistory Projekts auf die Konzeption

5.2 Aufbau des Prototyps

Der Aufbau des Prototyps entspricht im Wesentlichen dem in Abschnitt 3.2 beschriebenen logischen Aufbau eines Multimedia-Datenbanksystems, auf Grund der Architektur der eNoteHistory Datenbank jedoch mit einigen Änderungen. Nachfolgend werden nun diese Änderungen separat für jede typisierte Tabelle vorgestellt. Die kompletten DDL-Befehle sind dem Anhang B.1 zu entnehmen.

5.2.1 Document_T

Ein Dokument des eNoteHistory-Projekts wird durch die *Klasse Music_Manuscript* dargestellt. Da jedoch ein Manuskript aus mehreren Musikwerken (*Music_Works*) besteht, die auch eigenständig und in weiteren Manuskripten existieren können, kann für ein Dokument kein Datalink spezifiziert werden, der einem Verweis auf das Originaldokument entspricht. Der Typ *Document_T* besitzt dennoch der Vollständigkeit halber ein Attribut *Document* vom Typ Datalink, allerdings wird dieser mit der Option `NO LINK CONTROL` versehen, sodass ein korrespondierendes Dokument nicht existieren muss. Allerdings wird durch den Comment-Teil des Datalinks der Wert des Attributs *Shelf_Mark* des korrespondierenden Manuskripts abgelegt. Dadurch kann a posteriori weiterhin die Beziehung zwischen Dokument und zugehörigem Manuskript wieder hergestellt werden.

Ein weiteres Umsetzungsproblem bildet die Granularität eines eNoteHistory-Dokuments. Eine Seite eines Dokuments (*Manuscript_Page*) besteht ausschließlich aus einem zugehörigen Bild mit Attributen (*Page_Image*). Daraus resultiert, dass Component-Objekte nicht geometrisch auf einer Seite angeordnet werden können, da sie aus mindestens einer Seite bestehen. Die Konsequenz hieraus ist, dass Component-Objekte keinen Wert für das Attribut *Shape* besitzen, welches die geometrische Form angibt, woraus sich für den Typ *Document_T* ergibt, dass das Attribut *Layout* zwar in der Typdefinition existiert, jedoch keine Verwendung findet.

Im Gegensatz zu Dokumenten der Konzeption besitzen Dokumente des eNoteHistory-Projekts keinen textuellen Inhalt, sondern bestehen ausschließlich aus Bildern und Metadaten. Somit wird dass Attribut *Content* ebenfalls keinen Inhalt besitzen.

Weiterhin existiert kein multimengenwertiges Attribut Subcomponents, welches die direkten Kindknoten im Dokumentenbaum referenziert. Da DB2 kein multimengenwertiges Attribut unterstützt, muss ,wie in Kapitel 4 beschrieben, die Zeigerstruktur invertiert werden.

Der Aufbau und Inhalt der Metadaten eines Dokuments, in der Konzeption wurden mehrere Möglichkeiten vorgestellt, wird im Prototyp durch eine typisierte Tabelle *DocMetadata* realisiert, welche das Attribut *Metadata* referenziert. Die zu einem Dokument gehörigen Metadaten ergeben sich aus der Klasse *Music_Manuscript* und lauten wiefolgt:

- *Edition*
- *Comment*
- *Dedication*
- *Literature*

Diese ausgewählten Attribute werden in der Datenbank als Volltextobjekte deklariert, wodurch in der Anfrageformulierung auch Anfragen auf Inhalte formuliert werden können, welche spezielle Information Retrieval-Funktionalitäten enthalten. Allerdings können durch den *Net Search Extender* ausschließlich Tabellen indiziert werden, die einen Primärschlüssel verwenden. Wird nun die OID der typisierten Tabelle *DocMetadata* als Primärschlüssel verwendet, können die Einträge der Tabelle nicht mehr von der Tabelle *Document* mittels Pfeilnotation referenziert werden. Anfragen der Form `SELECT Metadata->Comment FROM Document` wären demnach nicht mehr möglich. Um diesen Effekt zu umgehen, wird ein Attribut *Dummy* in der typisierten Tabelle eingeführt, welches daraufhin als Primärschlüssel verwendet wird. Der Inhalt dieses Attributs ergibt sich während des Imports der Daten aus der Anzahl der bereits eingetragenen Datensätze, sodass die Primärschlüsseleigenschaft gewährleistet wird.

5.2.2 Component_T

Ein Component-Objekt ist nun als Teil eines Manuskripts entweder die Einleitung (*Incipit*), eine Sektion (*Music_Manuscript_Section*) oder eine Seite des Manuskripts (*Music_Manuscript_Page*). Analog zu *Document_T* werden auch Metadaten der Component-Objekte in einer speziellen, typisierten Tabelle *CompMetadata* gespeichert, welche folgende Attribute besitzt:

- *Text* - Text_Incipit
- *Comment* - Incipit_Comment, Page_Comment
- *Description* - Section_Description, Paper_Description
- *Relic*

Die Attribute *Text*, *Comment* und *Description*, deren Inhalt sich aus den angegebenen Attributen des eNoteHistory-Projekts ergeben, entsprechen ebenfalls textuellen Attributen, auf denen eine Volltextsuche möglich sein wird[1]. Das zusätzliche Attribut *Relic*, welches im eNoteHistory-Projekt nicht vorkommt, wird als XML-wertiges Attribut definiert und kann somit Inhalte von Attributen repräsentieren, die zusätzlich zu den drei vorangegangenen existieren. Ein Beispiel hierfür wäre das Attribut *Provenance* der Klasse *Music_Manuscript_Section*.

[1] Analog zu den Metadaten von Dokumenten muss zur Indizierung das zusätzliche Attribut *Dummy* eingeführt werden

Wie auch der Typ *Document_T* enthält der Typ *Component_T* kein Attribut *Subcomponents*. Stattdessen wird das Attribut *Supercomponent* definiert, welches den jeweiligen Vater-Component referenziert. Die Notwendigkeit hierfür liegt, wie bereits erwähnt, darin, dass die Zeigerstruktur invertiert werden muss, um die Hierarchie der Component-Objekte abzubilden.

5.2.3 ImageComponent_T

Da es auf Grund der Einfüge-Reihenfolge nicht möglich ist, die Zeigerstruktur nur teilweise zu invertieren, dies würde zu einem Abhängigkeitszyklus führen, enthält der Typ *ImageComponent_T* kein Attribut *Content*. Dieser Typ ist somit identisch zum Typ Component_T.

5.2.4 Images

Die Tabelle *Images*, in der Konzeption als Fremdtabelle spezifiziert, wird im Prototyp als Basistabelle definiert und bildet hinsichtlich des eNoteHistory-Projekts das Äquivalent zur Klasse *Page_Image_ROI*. Um den Aufbau der Konzeption umzusetzen, wird einerseits das multimengenwertige Attribut *Subpictures* durch das Attribut *SuperPicture* ersetzt, andererseits wird das anonym strukturierte Attribut ImageMetadata durch eine Referenz auf die typisierte Tabelle ImageMetadata_T ersetzt. Wie auch bei der Referenzstruktur der Component-Objekte wird zusätzlich ein Attribut *ImageComp* eingeführt, welches auf das korrespondierende ImageComponent verweist.

Um nun einem Bild zusätzlich eine Menge von Regionen zuzuordnen, wird hierfür ebenfalls die Zeigerstruktur invertiert. Es wird dabei eine Tabelle *Regions* angelegt, die zusätzlich zu den Attributen der Konzeption einen Verweis auf das zugehörige Bild besitzt.

5.2.5 Relations_T

Der Aufbau der typisierten Tabellen zur Speicherung von Component-Beziehungen sowie die abgeleiteten Tabellen werden ebenfalls in den Prototyp übernommen. Jedoch wird nur die Tabelle *LogicalRelations* verwendet. Sie dient dazu, die logischen selbstdefinierten Einheiten *Incipit*, *Section* und *Page* zu speichern. Da hierzu das multimengenwertige Attribut *Comps* nicht benötigt wird, ist es ausreichend, dieses als VARCHAR zu deklarieren.

5.3 Import der Daten

Zum Import ausgewählter Daten des eNoteHistory-Projekts wird das Java-Programm **Insert.java** zur Verfügung gestellt, welches auf Grund des Umfangs lediglich in der eventuell beigefügten CD zu finden ist. Der Ablauf dieses Programms ergibt dabei aus der Referenzstruktur der typisierten Tabellen, die im vorangegangenen Abschnitt beschrieben wurden. So werden zunächst 30 Einträge der Tabelle Metadata.Music_Manuscript ermittelt, denen Einträge der Tabelle IPFV.Page_Image_ROI zugeordnet sind. Dadurch wird gewährleistet, dass jedes Dokument auch ImageComponents mit Bildern und Regionen enthält. Daraufhin werden für jedes Dokument iterativ Einleitung, Sektionen, Seiten, Seitenbilder und Regionen ermittelt und in die korrespondierenden Tabellen des Prototyps eingefügt. Die Abbildung der Attribute der Tabellen des eNoteHistory-Projekts auf die des Prototyps wurden bereits in Abschnitt 5.1 beschrieben. Wichtig zu erwähnen ist an dieser Stelle, dass die zu generierende OID jeder typisierten Tabelle sich aus der Anzahl der Elemente ergibt, die sich bereits in der Tabelle befinden. Auf Grund dieser Generierung und der Einfügereihenfolgen wird eine implizite Durchnummerierung der Component-Objekte realisiert, die der einer Tiefensuche entspricht. Somit kann bereits durch die OID ermittelt werden, ob ein Component vor oder nach einem anderen Component im Dokumentenbaum liegt, was für unscharfe Anfragen unerlässlig ist.

Während diese Tabellen durchlaufen werden, wird parallel der Dokumentenbaum zusammengestellt. Hierzu werden zwei String-Objekte initialisiert, Anfang und Ende des Baums. Wird nun ein Component in den Prototyp eingefügt, wird an den Anfang ein öffnendes Component-Tag inklusive OID als Attribut und an ein schließenden Component-Tag das Ende des Baums angefügt. Nachdem alle Einträge eines Dokuments, Components, Images und Regions, eingefügt wurden, der Dokumentenbaum ist also vollständig, wird dieser mittels UPDATE-Statement als Attributwert des Dokuments gesetzt.

Um nun auf die gespeicherten Metadaten der Dokumente und Component-Objekte Volltextanfragen zu stellen, werden abschließend die angelegten Indizes aktualisiert, die in Abschnitt 5.2 beschrieben wurden. Die Datenbank besitzt nach Abschluss des Imports folgende Parameter:

- 30 Dokumente
- 716 Components
- 511 ImageComponents
- 427 Einträge der Tabelle Images
- 142.389 gespeicherte Regionen

5.4 Anfragemöglichkeiten

Dieser Abschnitt befasst sich nun mit den möglichen Anfragen, die innerhalb der prototypischen Implementierung auf die gespeicherten multimedialen Dokumente durchgeführt

werden können. Auf Grund der Heterogenität zwischen dem logischen Aufbau der Konzeption (Abschnitt 3.2) und dem des Prototyps müssen auch bei den Anfragemöglichkeiten einige Restriktionen getroffen werden. Welche Anfragemöglichkeiten nicht und welche weiterhin in welcher Form möglich sind, wird nun in den kommenden Unterabschnitten untersucht. Da DB2 jedoch nicht die Möglichkeit bietet, ganze Objekttypen in der Select-Klausel zu referenzieren, wird nachfolgend statt der typisierten Tabelle stets die identifizierende OID als Projektionsattribut verwendet. Wird die Ausgabe einer Anfrage daraufhin in der From-Klausel einer weiteren Anfrage verwendet, kann aus der OID mittels `DEREF`-Operator wieder das entsprechende Component-Objekt erzeugt werden. Eine genauere Beschreibung dieser Anfrageformulierung erfolgt in Abschnitt 5.4.3 im Zusammenhang mit kombinierten Anfragen. Wird allerdings mittels `IN`-Prädikat überprüft, ob eine OID in der Ergebnismenge einer anderen Anfragen existiert, so müssen sowohl die OID als auch das Ergebnis einer Anfrage mittels `CAST`-Funktion in eine Zeichenkette transformiert werden. Ein Beispiel hierfür ist in Abschnitt 5.4.1 gegeben.

5.4.1 Anfragen auf Struktur

Die vier in Abschnitt 3.3.1 vorgestellen Anfrageklassen auf Strukturen bezogen sich ausschließlich auf Component-Objekte. Da jedoch Component-Objekte des Prototyps nicht geometrisch auf einer Seite angeordnet sein können, da sie aus mindestens einer Seite bestehen, ist die Anfrageklasse der geometrischen Anfragen in der Form, wie sie in der Konzeption beschrieben wurde, nicht realisierbar. Nachfolgend werden nun diese vier Anfrageklassen untersucht. Anzumerken ist an dieser Stelle, dass die Funktionen *Between()* in *ExactBetween()* und *FuzzyBetween()* umbenannt werden, da das Wort Between von DB2 reserviert ist.

Geometrische Anfragen Wie bereits erwähnt, können geometrische Anfragen nicht auf Component-Objekte gestellt werden. Um dennoch geometrische Anfragen prototypisch umzusetzen, wird die Tabelle *Regions* verwendet. Da jedes beliebige Teildokument ebenfalls als vollständiges multimediales Dokument aufgefasst werden kann, kann analog eine *Region of Interest* inklusive zugehöriger Regionen als multimediales Dokument interpretiert werden. Einträge dieser Tabelle besitzen eine geometrische Form, welche im Attribut *Shape* abgespeichert ist. Die gespeicherten Attributwerte liegen dabei in der Form vor, dass der *Spatial Extender* daraus Werte vom Typ *ST_Geometry* erzeugen kann[2], wie in Abschnitt 4.1.5 beschrieben. Die Einheit des Referenzsystems besteht analog zum eNoteHistory-Projekt aus Pixeln. Um nun den Suchraum weiter eingrenzen zu können, besitzt eine Region, bis auf *Staff_Lines*, stets das XML-wertige Attribut *Metadata*. Der Inhalt dieses Attributs ergibt sich dabei aus dem Attribut *Phi* der Tabellen des eNoteHistory-Projekts, welches den Drehwinkel der umschließenden Ellipse beschreibt. Ausgehend von diesen gegebenen Attributen können nun geometrische Anfragen derart formuliert werden, dass geometrische Beziehungen zwischen Regionen untersucht werden können. So

[2] Aus dem unteren linken Punkt sowie Höhe und Breite des umschließenden Rechtecks wurden beim Import die vier korrespondierenden Eckpunkte sowie der Typ abgespeichert

kann beispielsweise ermittelt werden, ob ein Notenkopf rechts und oberhalb von einem Notenhals liegt, was einer linken Abwärtskaudierung entspricht. Die folgende Anfrage verdeutlicht nun, wie geometrische Anfragen auf Regionen formuliert werden können.

```
SELECT R1.Image
FROM Regions R1, Regions R2
WHERE R1.Image = R2.Image
   AND DB2XML.ExtractDouble(R1.Metadata,'/Metainfo/Rotation') > 5.1
   AND Above(R1.ID,R2.ID) = 1
```

Unscharfe Anfragen Unscharfe Anfragen beziehen sich gemäß der Konzeption auf relative Strukturen der Component-Objekte. Eine Anfrage ermittelt beispielsweise, ob ein gegebenes Component-Objekt in der Dokumentenordnung vor einem anderen liegt oder ob sie „in der Nähe" liegen. Hierzu werden die Funktionen, die in Abschnitt 3.3.1.2 angegeben sind, in analoger Weise umgesetzt. Auch die dort aufgezeigte Metrik zur Berechnung der Nähe wird zur Berechnung verwendet. Die Implementierung dieser Methoden ist in Anhang B.2 wiedergegeben. Folgende Beispielanfrage soll nochmals die Anfrageformulierung für unscharfe Anfragen verdeutlichen:

```
SELECT C1.oid, C2.oid
FROM Component C1, Component C2
WHERE C1.Document = C2.Document
   AND Before(CAST(C1.oid as VARCHAR(10)),CAST(C2.oid AS VARCHAR(10)) = 1
   AND Near(CAST(C1.oid as VARCHAR(10)),CAST(C2.oid AS VARCHAR(10)) > 0.7
```

Logische modellbezogene Anfragen In Bezug auf logische modellbezogene Anfragen müssen die größten Einschränkungen getroffen werden, zum einen hinsichtlich des logischen Aufbaus des Prototyps, zum anderen auf Grund fehlender Funktionalitäten seitens DB2. Da in der prototypischen Implementierung keine TextComponents verwendet werden, fallen sowohl die Textextraktionsmethoden *Text()*, als auch die darauf aufbauenden Segmentierungsfunktionen *Paragraph()*, *Sentence()* und *Word()* weg. Auch die Funktion MBC(), welche das minimale, umschließende Component-Objekt aus einer Liste gegebener Components bestimmt, kann nicht umgesetzt werden, da DB2 keinen Konstruktor ARRAY oder ein vergleichbares Äquivalent zur Verfügung stellt. Die zur Verfügung gestellten Methoden hinsichtlich logischer modellbezogener Anfragen beschränken sich demnach auf die Bestimmung von enthaltenen ImageComponents eines Dokuments respektive Components und die Ausgabe aller Regionen eines Image-Eintrags. Da bei den Methoden *Pictures()* allerdings als Parameter stets die als VARCHAR gecastete OID eines Dokuments beziehungsweise Components übergeben wird, müssen diese umbenannt werden, da sie dieselbe Signatur besitzen. Nachfolgend wird nun die Liste der bereitgestellten Funktionen angegeben. Anschließend wird zur Illustration eine beispielhafte Anfrage aufgezeigt, in welcher zunächst ein Component-Objekt ermittelt wird und anschließend sämtliche ImageComponents des zugehörigen Dokuments ausgegeben werden.

- *PicturesFromDoc* ermittelt ImageComponents eines Dokuments
- *PicturesFromComp* ermittelt ImageComponents eines Components
- *Picture* ermittelt den zu einem ImageComponent zugehörigen Eintrag der Tabelle *Images*
- *Regions* ermittelt sämtliche Regionen eines Eintrags der Tabelle *Images*

```
SELECT IC
FROM TABLE(hj016.PicturesFromDoc((
            SELECT CAST(document as varchar(10))
            FROM Component
            WHERE Metadata->Comment Like '%Del Sig. Balbi%'))) Q
```

Logische selbstdefiniert Anfragen Diese Klasse der strukturellen Anfragen bestimmt Strukturen, die sich aus dem Projekt eNoteHistory ergeben. So besteht ein Dokument aus Einleitung und Sektionen und Sektionen wiederum aus Seiten. Diese selbstdefinierten logischen Strukturen wurden beim Import der Daten in der Relation *LogicalRelations* abgespeichert, sodass diese nun problemlos wiedergewonnen werden können, wie folgende Anfrage verdeutlicht, in welcher eine Einleitung gesucht ist:

```
SELECT oid
FROM Component
WHERE CAST(oid as VARCHAR(10)) IN
         CAST( SELECT root
               FROM LogicalRelations
               WHERE Description = 'Incipit')
           AS VARCHAR(10))
```

5.4.2 Anfragen auf Inhalt

Anfragen auf Inhalte multimedialer Dokumente wurden in Kapitel 3 in die drei Kategorien bildbezogen, textbezogen und metadatenbezogen eingeteilt. Diese starre Trennung kann im Prototyp nicht weiter verfolgt werden. Da die hier verwendeten Dokumente keinen Textanteil im ursprünglichen Sinne besitzen, werden an dieser Stelle Volltextanfragen mit Anfragen auf Metadaten teilweise kombiniert. Die Verwendung der Information Retrieval-Funktionalitäten, die der Net Search Extender von DB2 bereitstellt, erfolgt somit auf dem Teil der Metadaten, die als CLOB definiert und mit einem Textindex versehen wurden. Reine Anfragen auf Metadaten erfolgen beispielsweise auf dem XML-wertigen Attribut *Relic*. Bildbezogene Anfragen basierend auf Ähnlichkeitsberechnungen bezüglich der in Kapitel 3 beschriebenen Features werden in der prototypischen Implementierung komplett ausgegrenzt. Dies liegt daran, dass sämtliche Digitalisate eine sehr hohe Ähnlichkeit bezüglich der Features besitzen, die in Kapitel 3 beschrieben wurden. Ein Vergleich

bezüglich dieser Features ist somit nicht aussagekräftig. Daher werden Anfragen auf Bilder ausschließlich auf die textuellen Attribute der Tabellen *ImageComponent* und *Images* gestellt.

Anfragen auf Text Da in der prototypischen Implementierung keine TextComponents, wie sie in der Konzeption eingeführt wurden, existieren, können demzufolge auch keine Volltextanfragen auf komplette Component-Objekte gestellt werden. Um aber dennoch Volltextanfragen im Prototyp verwenden zu können, wurden die oben beschriebenen Attribute der Metadaten inklusive Volltextindex definiert. Anfragen unter Verwendung von Information Retrieval-Techniken können somit auf die Attribute der Tabellen *DocMetadata* und *CompMetadata* gestellt werden. Im Allgemeinen wird dafür die Methode Contains verwendet, da ein Nutzer lediglich ermitteln möchte, ob eine gegebene Volltextbedingung erfüllt ist. Die Methode Score gibt jedoch einen Wert zwischen 0 und 1 zurück, je nachdem wie stark die Bedingung erfüllt ist. Diese Funktionalität ist besonders bei Ranking-Verfahren von Interesse. Da ein Ranking jedoch stets in der umschließenden Anfrage spezifiziert wird, wird hierauf erst in Abschnitt 5.4.3 hinsichtlich des umschließenden SFW-Blocks eingegangen. Folgende Anfrage soll nun eine Anfrage auf Text unter Verwendung von Information Retrieval-Techniken verdeutlichen, wobei sowohl Attribute der Dokumente als auch der ComponentObjektee in der Anfrage referenziert werden:

```
SELECT oid
FROM Component
WHERE contains(Metadata->Description,
      '"doppelrandiger Schild" IN SAME SENTENCE AS "aufrechter Loewe"')=1
  AND contains(Document->Metadata->Comment,'STEMMED FORM OF "Hamburg"')=1
```

Anfragen auf Bild Wie bereits erwähnt, werden Anfragen auf Bilder nicht durch die in Kapitel 3 beschriebenen Feature-Vergleiche realisiert, da sämtliche gespeicherte Bilder eine sehr hohe Ähnlichkeit bezüglich dieser Features besitzen. Anfragen auf Bilder werden in der prototypischen Umsetzung daher stets auf Attribute der Bilder und Regionen angewandt. Zu vergleichende Attribute sind dabei zum einen die Metadaten der ImageComponents (*height* und *width* innerhalb des XML-wertigen Attributs *Relic*), zum anderen die Metadaten eines Bildes (*Colorspace* und *BitsPerPixel*) und die einer Region (*Rotation* innerhalb des XML-wertigen Attributs *Metadata*). Die Hierarchie der Verschachtelung wird dabei durch die Referenzstruktur vorgegeben. So wird zunächst eine Selektion bezüglich der Regionen durchgeführt und als Ergebnis die zugehörigen ImageIDs zurückgegeben. Das Ergebnis dieser Ausgabe wird nun von einer Anfrage bezüglich der Tabelle Images derart referenziert, dass nur die ImageComponents der Images zurückgegeben werden, deren ID in der Ergebnismenge der inneren Anfrage liegt und die eventuell eine weitere Selektionsbedinung bezüglich der Metadaten erfüllen. Die nachfolgende Anfrage verdeutlicht nun, in welcher Form Anfragen auf Bilder formuliert werden können:

```
SELECT ImageComp
FROM Images
```

```
WHERE Metadata->ColorSpace = 'RGB'
  AND ID IN (SELECT Image
             FROM Regions
             WHERE DB2XML.ExtractDouble(
                        R1.Metadata,'/Metainfo/Rotation') > 5.1)
  AND db2xml.extrachVarchar(
          ImageComp->Metadata->Relic,'/Metainfo/width') < 33
```

Anfragen auf Metadaten Anfragen auf Metadaten wurden bereits im Zusammenhang mit Anfragen auf Text und Bild eingeführt, da eine strikte Trennung diese drei Klassen nicht sinnvoll möglich ist. Daher kann an dieser Stelle auf eine tiefergehende Untersuchung von Anfragen auf Metadaten verzichtet werden. Selektionsbedingungen können nun sowohl auf den Volltextattributen *Text, Comment, Dedication, Description, Edition* und *Literature*, als auch auf dem XML-wertigen Attribut *Relic*, welches die Pfade *Provenance*, *Created* und *Updated* enthält, formuliert werden. Nachfolgende Beispielanfrage soll nun eine einfache Anfrage auf Metadaten verdeutlichen:

```
SELECT oid
FROM Component
WHERE db2xml.extractVarchar(Metadata->Text,'/Metainfo/Provenance')
         LIKE '%Hamburg%' AND
      CAST(Document->Metadata->Dedication AS VARCHAR(1024))
         LIKE '%for my son%'
```

5.4.3 Kombinierte Anfragen

Nachdem in den beiden vorangegangenen Abschnitten untersucht wurde, in welcher Form Anfragen auf Strukturen und Inhalte formuliert werden können, befasst sich dieser Abschnitt mit einer sinnvollen Kombination dieser Möglichkeiten, um kombinierte Anfragen auf Strukturen und Inhalte zu untersuchen. Analog zur Konzeption kann hierbei unterschieden werden, ob die Kombination durch eine kombinierte Selektionsbedingung oder durch verschachtelte Anfragen realisiert wird. Da jedoch im Gegensatz zur Konzeption keine spezialisierten Medienserver in das Datenbanksystem integriert werden, sondern stattdessen ein zentralisierter Ansatz verfolgt wird, ist die Formulierung der Anfrage nicht denselben Restriktionen unterworfen, wie in der Konzeption. So müssen Anfragen auf Bilder nicht zwingend als Blätter des Anfragebaums formuliert werden, sondern können auch als innerer Knoten verwendet werden. Bevor im Folgenden auf verschachtelte Anfragen eingegangen wird, verdeutlicht die nachstehende Beispielanfrage eine kombinierte Selektionsbedingung.

```
SELECT C1.oid, C2.oid
FROM Component C1, Component C2
WHERE CAST(C1.Document AS VARCHAR(10)) =
```

```
      CAST(C2.Document AS VARCHAR(10))
   AND Left(CAST(C1.oid AS VARCHAR(10)),CAST(C1.oid AS VARCHAR(10))) = 1
   AND Contains(C1.Metadata->Comment, 'SOUNDS LIKE "Verarbeitung"') = 1
```

Prinzipiell kann zwar jede der nachfolgenden Verschachtelungen als planare Anfrage formuliert werden, jedoch soll durch diese prototypische Implementierung die allgemeine Machbarkeit der Konzeption demonstriert werden. Daher müssen diese Verschachtelungen ebenfalls untersucht werden.

Inhalt auf Inhalt Diese erste Klasse kombinierter Anfragen besteht aus Anfragen auf Inhalt, welche auf das Ergebnis einer weiteren inhaltsbezogenen Anfrage zugreift. Diesbezüglich sind prinzipiell zwei Szenarien denkbar. Im ersten Fall wird in der inneren Anfrage eine Objektmenge erzeugt, von welcher ausgehend in der äußeren Anfrage weiternavigiert wird. Ein Beispiel hierfür ist die Selektion eines Component-Objekts, und eine daran anschließende Selektion des zugehörigen Vaterelements. Da jedoch stets die OID des Component-Objekts, das selektiert werden soll, ausgegeben wird, wird an dieser Stelle eine Zwischenstufe eingeführt, welche zunächst anhand der ausgegebenen OID der inneren Anfrage die OID des zugehörigen Vaterelements als Referenz auf die typisierte Tabelle castet. Dies ist zum einen notwendig, um in der äußeren Anfrage wiederum mittels Pfeilnavigation auf beispielsweise die Metadaten zuzugreifen, zum anderen kann das Vaterelement durch einen Alias-Namen referenziert werden, was die Anfrage intuitiver macht. Folgende Anfrage verdeutlicht dies:

```
SELECT DISTINCT P
FROM (SELECT CAST(DEREF(O1)..parentcomponent
              AS REF(hj016.Component_t) SCOPE Component) as P
      FROM (SELECT oid as O1
            FROM Component
            WHERE contains(Comment,'FUZZY FORM OF "Haus"')=1) Q1
      WHERE DEREF(O1)..ParentComponent is not null) Q2
WHERE db2xml.extractvarchar(
               P->metadata->relic,'/Metainfo/Provenance') LIKE 'Italien%'
```

Das zweite denkbare Szenario, welches auch als Beispiel in der Konzeption verwendet wird, erzeugt aus dem Ergebnis zweier innerer Anfragen[3] eine Tupelmenge, die bestimmten Selektionsbedingungen genügt. So werden zunächst zwei Anfragen auf Inhalt formuliert, welche daraufhin in der umschließenden Anfrage verknüpft werden und eine entsprechende Ergebnismenge erzeugen. Dies soll durch folgende Anfrage illustriert werden:

```
SELECT C1.oid as O1, O2
FROM Component C1,
     (SELECT oid as O2
```

[3]beziehungsweise einer Anfrage auf Inhalt und einer Referenzierung der Component-Relation

```
      FROM Component
      WHERE Contains(metadata->Text,'STEMMED FORM OF "Klavier"')=1) C2
WHERE CAST(C1.Document AS VARCHAR(10)) =
      CAST(DEREF(O2)..Document AS VARCHAR(10))
    AND db2xml.extractvarchar(
      C1.Metadata->Relic,'/Metainfo/Provenance') LIKE 'England%'
```

Inhalt auf Struktur Inhaltsbezogene Anfragen, die auf das Ergebnis von strukturellen Anfragen zugreifen, wurden in ähnlicher Form bereits im Zusammenhang mit Anfragen auf Bilder eingeführt. So werden zunächst geometrische Anfragen auf Regionen gestellt und anschließend inhaltliche Anfragen auf die zugehörigen Images oder ImageComponents formuliert. Es ist ebenfalls möglich, diese Anfrageklasse ausschließlich auf Component-Objekten zu realisieren. So könnten zunächst unscharfe Anfragen auf Component-Objekte gestellt und anschließend die erzeugte Ausgabe auf inhaltliche Bedingungen getestet werden. Da jedoch bereits in der Konzeption aufgezeigt wurde, dass diese Art der Anfrageformulierung überaus ineffizient ist, müssen diese derart transformiert werden, dass zuerst die inhaltlichen Bedingungen überprüft werden und daraufhin erst die strukturellen. Diese Anfragen sind dann aber der folgenden Kombinationsklasse zuzuordnen. Die folgende Beispielanfrage soll nun nochmals illustrieren, in welcher Form inhaltsbezogene Anfragen auf strukturelle Anfragen angewendet werden können.

```
SELECT ImageComp
FROM Images
WHERE ID IN (SELECT R1.Image
             FROM Regions R1, Regions R2, Regions R3
             WHERE R1.Image = R2.Image
               AND R1.Image = R3.Image
               AND Above(R1.ID,R2.ID) = 1
               AND Left(R1.ID,R3.ID) = 1)
      AND db2xml.extrachVarchar(
             ImageComp->Metadata->Relic,'/Metainfo/width') < 33
```

Struktur auf Inhalt Diese Klasse der kombinierten Anfragen bildet praktisch die relevanteste Kombinationsmöglichkeit. Anfragen dieser Art können sowohl ausschließlich auf Regionen, ausschließlich auf Component-Objekten, als auch auf Regionen und Component-Objekten beruhen. Kombinierte Anfragen, die sich ausschließlich auf Regionen beziehen, wurden bereits ebenfalls im Zusammenhang mit bildbezogenen Anfragen beschrieben. So können zuerst alle Regionen bestimmt werden, die einen vorgegebenen Drehwinkel besitzen und diese dann auf geometrische Beziehungen untersucht werden. Praktisch relevant sind jedoch vor allem Anfragen, die sich ausschließlich auf Component-Objekte beziehen. So können einerseits separate Anfragen auf Inhalte in der From-Klausel referenziert und bezüglich struktureller Bedingungen verknüpft werden, andererseits können diese inhaltsbezogenen Anfragen auch in einer Selektionsbedingung formuliert werden, so dass nur eine Anfrage in der From-Klausel referenziert wird, die

entsprechend viele Projektionsattribute besitzt. Die beiden folgenden Anfragen verdeutlichen diese Möglichkeiten, wobei beide semantisch äquivalent sind. Dennoch ist die erste Variante deutlich effizienter.

```
SELECT  O1, O2
FROM (SELECT c1.oid as O1, c2.oid as O2
      FROM Component c1, Component c2
      WHERE c1.Metadata->Comment Like '%Del Sig. Balbi%'
               AND Contains(C1.Metadata->Description,
                      '"doppelrandiger Schild" IN SAME
                               SENTENCE AS "aufrechter Loewe"')=1
               AND CAST(c1.Document as varchar(10)) =
                   CAST(c2.Document as varchar(10))) Q
WHERE Before(cast(o1 as VARCHAR(10)),cast(o2 as VARCHAR(10))) = 1
```

```
SELECT  O1, O2
FROM (SELECT oid as O1
      FROM Component
      WHERE CAST(Metadata->Comment AS VARCHAR(1024))
               Like '%Del Sig. Balbi%') Q1
     (SELECT oid as O2
      FROM Component
      WHERE contains(Metadata->Description,'"doppelrandiger Schild"
                 IN SAME SENTENCE AS "aufrechter Loewe"')=1) Q2
WHERE CAST(DEREF(O1)..Document AS VARCHAR(10)) =
      CAST(DEREF(O2)..Document AS VARCHAR(10))
   AND Before(cast(O1 as VARCHAR(10)),cast(O2 as VARCHAR(10))) = 1
```

Struktur auf Struktur Anfragen dieser Kombinationsklasse sind prinzipiell nur dann sinnvoll, wenn strukturelle Anfragen auf Component-Objekte auf strukturelle Anfragen bezüglich Regionen gestellt werden. Wie auch in der Konzeption erwähnt, können verschachtelte, strukturelle Anfragen, die sich ausschließlich auf Regionen oder Component-Objekte beziehen, entschachtelt und als planare Anfrage formuliert werden, so dass diese Fälle nicht untersucht werden müssen. Demnach liegen Anfragen dieser Kombinationsklasse stets in der Form vor, dass zunächst geometrische Anfragen auf Regionen formuliert werden und auf den korrespondierenden ImageComponents, beziehungsweise deren Vaterelemente, strukturelle Beziehungen untersucht werden. Die nachfolgende Beispielanfrage verdeutlicht dies.

```
SELECT O1, DEREF(O2)..ParentComponent AS O3
FROM (SELECT ImageComp as O1
      FROM Images
      WHERE ID IN (SELECT R1.Image
```

```
                 FROM Regions R1, Regions R2
                 WHERE R1.Image = R2.Image
                    AND DB2XML.ExtractDouble(
                          R1.Metadata,'/Metainfo/Rotation') > 5.1
                    AND Above(R1.ID,R2.ID) = 1)) IC1
     (SELECT ImageComp as O2
      FROM Images
      WHERE ID IN (SELECT R1.Image
                 FROM Regions R1, Regions R2
                 WHERE R1.Image = R2.Image
                    AND DB2XML.ExtractDouble(
                          R1.Metadata,'/Metainfo/Rotation') < 0.1
                    AND Left(R1.ID,R2.ID) = 1 )) IC2
WHERE CAST(DEREF(O1)..Document AS VARCHAR(10)) =
      CAST(DEREF(O2)..Document AS VARCHAR(10))
    AND After(CAST(O1 AS VARCHAR(10)),CAST(O2 AS VARCHAR(10))) = 1
    AND Near(CAST(O1 AS VARCHAR(10)),CAST(O2 AS VARCHAR(10))) > 0.8
```

Umschließender SFW-Block Nachdem in den vorangegangenen Teilabschnitten beschrieben wurde, in welcher Weise Anfragen auf Struktur und Inhalt kombiniert werden können, wird abschließend untersucht, welche Ausgabemöglichkeiten bezüglich solcher kombinierten Anfragen existieren. Am Naheliegensten ist sicherlich eine Sortierung der Ausgabemenge bezüglich einer bestimmten Bedingung, im Information Retrieval-Bereich als Ranking bezeichnet. Da an dieser Stelle kein komplexes Ranking-Verfahren vorgestellt werden soll, soll diesbezüglich die Methode *Score* ausreichend sein. Diese Methode erzeugt, wie bereits beschrieben, ausgehend vom übergebenen Parameter eine Dezimalzahl, die angibt, wie stark die angegebene Selektionsbedingung erfüllt ist. Wird diese Funktion nun in der Select-Klausel verwendet, kann sie weiterhin in der Order By-Klausel referenziert werden, um beispielsweise anhand der Termhäufigkeit ein Ranking durchzuführen. Die nachfolgende Anfrage soll dies verdeutlichen.

```
SELECT DISTINCT DEREF(O1)..Document,
                Score(CAST(DEREF(O2)..Metadata
                      AS REF(Compmetadata_t)
                      SCOPE Compmetadata)->Comment,'"Sonata"') AS Rank
FROM (innerSFW) Q
ORDER BY Rank ASC
```

Soll jedoch im Gegensatz zur vorangegangenen Anfrage nicht die OID des Dokuments, sondern der korrespondierende Eintrag der Tabelle *Document* ermittelt werden, kann dies nicht durch den `DEREF`-Operator realisiert werden, da DB2 keinen strukturierten Typ mit Attributen vom Typ DATALINK mittels `DEREF` konstruieren kann. Somit muss ein Dokument, beziehungsweise gewünschte Attribute, durch die folgende Anfrage bestimmt werden, wobei `SFW` einer Anfrage entspricht, die die OID von Dokumenten projiziert.

```
SELECT *
FROM Document
WHERE CAST(oid AS VARCHAR(10)) IN (SFW)
```

Wichtig in diesem Zusammenhang ist der Befehl `DLCOMMENT(Document)`. Wird dieser Befehl in der Select-Klausel verwendet, wird der Wert des Attributs *Shelf_Mark* des korrespondierenden Musikmanuskripts des eNoteHistory-Projekts ausgegeben. Dadurch wird es möglich, das Ergebnis einer kombinierten Anfrage in der Datenbank des eNoteHistory-Projekts weiterzuverarbeiten. Beispielsweise können die zugehörigen Manuskripte ermittelt und in der Webpräsenz[4] des Projekts graphisch dargestellt werden.

Hinsichtlich einer externen Anwendung, die auf die prototypische Implementierung zugreift, ist es sinnvoll, auch die zugehörigen Bilder, die sich bezüglich einer Anfrage qualifiziert haben, ausgeben zu lassen, welche in der Tabelle *Images* gespeichert sind. Um nun einen Bezug von der Ausgabe einer inneren Anfrage, die Component-Objekte projiziert, zu den korrespondierenden Bildern herzustellen, können die Funktionen *PicturesFromComp()* und *PicturesFromDoc()* verwendet werden, um zunächst die gewünschten Image-Components zu ermitteln. Daraufhin werden die Einträge der Tabelle *Images* ermittelt, deren Attribut *ImageComp* in der Ergebnismenge der jeweiligen Funktion liegt und das entsprechende Bild ausgegeben. Folgende Anfrage verdeutlicht dies.

```
SELECT Picture
FROM Images
WHERE CAST(ImageComp AS VARCHAR(10)) IN
  (SELECT CAST(IC AS VARCHAR(10))
   FROM TABLE(PicturesFromDoc((
                SELECT CAST(Document as VARCHAR(10))
                FROM Component
                WHERE Metadata->Comment Like '%Del Sig. Balbi%'))) Q )
```

5.5 Probleme

Ein schwerwiegendes Problem, welches auf einer Schwachstelle von DB2, respektive dem Net Search Extender basiert, liegt in der Bearbeitung von Volltextanfragen. So sind zwar sämtliche, oben verwendeten Contains- und Score-Prädikate syntaktisch korrekt, jedoch werden sie nicht bearbeitet. Statt dessen erfolgt eine Fehlermeldung, dass kein passender Index gefunden , obwohl dieser ordnungsgemäß erstellt und aktualisiert wurde. Der Fehler liegt darin, dass der Net Search Extender nicht auf Spalten typisierter Tabellen anwendbar ist. Der Index wird zwar korrekt auf der Tabelle *CompMetadata* angelegt, die Methoden Score und Contains greifen jedoch auf die Tabelle *CompMetadata_Hierarchy* zu, auf welcher kein Index existiert. Diesbezüglich ist jedoch keinerlei Information auffindbar. Dieses Problem wurde allerdings von einem IBM-Mitarbeiter als Design-Lücke bestätigt.

[4] http://www.enotehistory.de

Wie der Net Search Extender bildet auch der XML-Extender eine Schwachstelle der prototypischen Implementierung. Dieser bietet zwar laut [IBM03b] eine Unterstützung für XPath-Pfadausdrücke, diese werden jedoch nicht korrekt umgesetzt, wie das folgende Beispiel verdeutlicht. Der Ausdruck `'/DocTree//Component[CID="240"]//@CID'` beispielweise selektiert laut W3C sämtliche Attribute *CID*, die in der self-or-descendent-Achse des Component-Elements liegen, dessen Wert des Attributs *CID* gleich "240" ist. Der XML-Extender erzeugt allerdings nur den Wert 240 als Ausgabe, die Attribute der descendent-Achse werden nicht ausgegeben. Ein ähnliches Problem entsteht, wenn beispielweise der Ausdruck `'/Metainfo'` als Pfadausdruck spezifiziert wird. Dieser sollte als Ausgabe sämtliche Elemente, Attribute und Textstellen ausgeben, die in der descendent-Achse des Attributs *Metainfo* liegen. Stattdessen wird die leere Menge ausgegeben, gleich ob das Attribut weitere Elemente enthält. Daraus lässt sich schließen, dass der XML-Extender keine descendent-Achse der Elemente in die Anfragebearbeitung mit einbezieht. Ein weiteres Problem liegt in den fehlenden XPath-Operatoren. Navigationsaufrufe wie beispielsweise `child::node()` oder `text()` werden vom Extender in keiner Weise unterstützt. Ebenfalls problematisch ist die Umsetzung des Typkonzepts von XML. So wird laut XML-Spezifikation bei Pfadausdrücken, die auf einen nicht existierenden Pfad verweisen, lediglich die leere Menge als Ausgabe erzeugt. Der XML-Extender generiert allerdings eine Fehlermeldung, dass der Pfad nicht gefunden wurde. Dies bereitet vor allem Probleme bei der Weiterverarbeitung von Anfragen, da ebenfalls keine Funktion *existsNode()*, wie sie Oracle bereitstellt, existiert, mit welcher im Vorfeld getestet werden kann, ob ein spezifizierter Pfad im Element vorhanden ist. Wird beispielsweise die OID zusammen mit einem Pfadausdruck projiziert, und der Pfadausdruck existiert im entsprechenden Element nicht, so wird nur die Fehlermeldung, nicht aber die zugehörige OID ausgegeben. Diese Umsetzung der Integration der Standards XML und XPath erschweren eine funktionsfähige, prototypische Implementierung enorm.

Kapitel 6

Schlussbetrachtungen

6.1 Zusammenfassung

In dieser Arbeit wurden die Möglichkeiten einer kombinierten Anfrageverarbeitung auf Struktur und Inhalt multimedialer Dokumente untersucht. Da bisherige Systeme, welche eine derartige Anfrage bereits unterstützen, stets anwendungsabhängig entworfen und somit lediglich in einem abgegrenzten Kontext verwendbar sind, wurde in dieser Arbeit versucht, eine standardisierte Backend-Lösung zu entwerfen. Da diese ausschließlich auf den existierenden und etablierten Standards SQL:1999/2003 und SQL/MM basiert, ist sie als anwendungsunabhängig zu bezeichnen. Dabei kann die vorliegende Arbeit in drei separate Teile unterteilt werden.

Im ersten Teil der Arbeit, welcher den Kernpunkt darstellt, wird ausgehend von den oben genannten Standards eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt untersucht. Dieser Teil kann wiederum in drei Teilabschnitte unterteilt werden. Dabei wurde zunächst ein allgemeines Dokumentenmodell vorgestellt, mit dessen Hilfe ein multimediales Dokument beschrieben werden kann. Ausgehend von diesem Modell wurde ein vollständig SQL-konformes Multimedia-Datenbanksystem entworfen, auf welches im dritten Abschnitt die entsprechende kombinierte Anfrageverarbeitung untersucht wurde. Dabei wurden zunächst rein strukturbezogene Anfragen untersucht, welche in die Klassen geometrische, unscharfe, logische modellbezogene und logische selbstdefinierte Anfragen unterteilt werden können. Anschließend wurde auf rein inhaltsbezogene Anfragen eingegangen, welche in bildbezogen, textbezogen und metadatenbezogen klassifiziert werden können. Abschließend wurden Kombinationsmöglichkeiten dieser Klassen untersucht, um kombinierte Anfragen auf Struktur und Inhalt zu realisieren. Diese Untersuchung ergab, dass derartige Kombinationen durch verschachtelte Anfragen realisiert werden können, wobei bestimmte Kombinationen aus Performance-Gründen transformiert werden müssen.

Der zweite Teil der Arbeit untersucht, ob und in welcher Form die markführenden, objektrelationalen Datenbankmanagementsysteme *DB2 V8.2* und *Oracle 10g* geeignet sind, um das vorgestellte Multimedia-Datenbanksystem praktisch umzusetzen und daraufhin eine kombinierte Anfrage zu gewährleisten. Das Ergebnis dieser Untersuchung zeigte,

dass fast sämliche notwendigen Voraussetzung zumindest grundlegend bereitgestellt werden. Da diese Untersuchung ausschließlich anhand der entsprechenden Dokumentationen durchgeführt wurde, kann sie als theoretische Untersuchung einer praktischen Umsetzung bezeichnet werden.

Ausgehend von der theoretischen Untersuchung wurde im dritten Teil eine praktische Umsetzung realisiert, die die Möglichkeiten der Konzeption anwendungsnah demonstrieren sollte. Obwohl nach der theoretischen Untersuchung fast sämtliche notwendigen Funktionalitäten zur Verfügung standen, war eine problemlose Umsetzung dennoch nicht möglich, da Probleme auftraten, die a priori nicht bekannt waren. Dennoch wurde eine prototypische Implementierung realisiert, welche die grundsätzliche Machbarkeit der Konzeption untermauert. Hierfür wurde zunächst ein Ausschnitt des eNoteHistory-Datenmodells bestimmt, welcher auf das Modell der Konzeption abgebildet werden kann. Daraufhin wurde das theoretische Multimedia-Datenbanksystem der Konzeption im objektrelationalen Datenbanksystem DB2 implementiert, und der ermittelte Ausschnitt des eNoteHistory-Datenbestandes in das System eingefügt. Abschließend wurden dann auf dem eingeführten System Anfragen auf Strukturen, auf Inhalte und mögliche Kombinationen untersucht. Dabei konnte zwar die Konzeption auf Grund der eNoteHistory-Datenstruktur nicht vollständig umgesetzt werden, jedoch wurde deutlich, dass fast sämtliche theoretisch untersuchten Anfragen auch praktisch sinnvoll umgesetzt werden können.

6.2 Ausblick

Diese Arbeit basiert auf zwei Faktoren, zum einen dem theoretischen Standard SQL, zum anderen der praktischen Umsetzung dieses Standards in existierenden Datenbankmanagementsystemen. Diese zwei Faktoren sind es ebenfalls, die Ansatzpunkte für eventuelle Erweiterungen liefern.

Erweiterungsansätze bezüglich des Standards, können nun in kommende und fehlende Funktionalitäten unterteilt werden. Eine für diese Arbeit interessante Erweiterung der kommenden SQL-Version liegt in der XML-Unterstützung, die gegenüber SQL:2003 wesentlich ausgebaut werden soll. Hierauf wird unter anderem in [Mül05] eingegangen. So sollen einerseits Spalten des Datentyps XML anstelle einfacher XML-Dokumente auch komplette XQuery-Sequenzen enthalten, zum anderen wird die SQL-Funktion *XMLQuery()* eingeführt, mit welcher es möglich sein wird, XQuery-Anfragen innerhalb von SQL zu formulieren und auszuwerten. Inwieweit diese Funktionalitäten zur Speicherung, Verwaltung und Retrieval von Metadaten integriert werden können, bedarf einer weiteren Untersuchung.

Ein Aspekt, der aus dieser Arbeit komplett ausgegrenzt wurde und auf fehlenden SQL-Funktionalitäten basiert, ist die Integration kontinuierlicher Daten, wie Audio- und Video-Objekten. Dies liegt zum einen daran, dass diesbezüglich keine Äquivalente in der SQL-Multimedia-Spezifikation existieren, zum anderen werden von SQL keine hierfür notwendi-

gen temporalen Funktionalitäten mehr angeboten[1]. Sollen nun auch zeitliche Beziehungen verwaltet und in Anfragen verwendbar gemacht werden, kann somit keine standardisierte Lösung angeboten werden. Um nun auch eine raumzeitliche Entwicklung eines Components darzustellen, besteht ein Lösungsansatz darin, dass statt des Attributs *Shape* ein mengenwertiges Attribut *TemporalShape* definiert wird, welches aus dem Tupel (Shape,Time) besteht. Auf diese Weise ist es möglich, nicht nur eine Form zu einem gegebenen Zeitpunkt zu definieren, wie dies bisher durch die Relation *TemporalRelations* geschah, vielmehr kann nun die zeitliche Entwicklung in Tupelschreibweise dargestellt werden.

Der zweite Aspekt dieser Arbeit, die Unterstützung des SQL-Standards in existierenden Datenbankmanagementsystemen, bietet ebenfalls Spielraum für eventuelle Erweiterungen. Zwar wurde in Kapitel 4 deutlich, dass fast sämtliche geforderten Voraussetzungen zumindest grundlegend angeboten werden, jedoch existiert weiterhin eine gewisse Deskrepanz zwischen Standard und Unterstützung. Eine diesbezüglich interessante Erweiterung bietet die kommende Version der Datenbank DB2. Diese wird laut [NvdL05] eine vollständig integrierte Unterstützung für XML und XQuery bereitstellen. Diese Erweiterung kann ebenfalls in der prototypischen Implementierung Verwendung finden, da voraussichtlich die in Abschnitt 5.5 aufgezeigten Schwachstellen behoben werden.

Ein weiterer Ansatzpunkt liegt in der prototypischen Implementierung. Da Anfragen auf Bilder bezüglich der in Kapitel 3 beschriebenen Features nicht sinnvoll sind, besteht eine plausible Erweiterung im Vergleich von Bildern unter Verwendung der Features des eNoteHistory-Projekts. So kann statt der StillImage-Feature-basierten Score-Funktion eine Score-Funktion definiert werden, die auf der bereitgestellten Distance-Methode des Projekts basiert. Somit werden statt der Features histogram, texture, average und positional color entweder die 80 Features der 13 Featuregruppen oder die automatische Handschriftenanalyse des eNoteHistory-Projekts zur Ähnlichkeitsbestimmung verwendet. Inwieweit diese prototypische Implementierung dann verwendet werden kann, um das Projekt eNoteHistory zu erweitern, ist ein weiterer Ansatzpunkt.

[1]Zunächst als Teil des Multimedia-Packages Spatial geplant, daher der Präfix ST, wurde der Teil, der temporale Beziehungen definiert, als Part 7 in den SQL:1999-Standard integriert. Auf Grund starker Kontroversen wurde dieser Teil jedoch wieder entfernt, und die Entwicklung eingestellt.

Literaturverzeichnis

[Czy05] Sebastian W.H. Czymaj. *Konzeptuelle Datenmodellierung für die inhaltsbasierte Suche in Kollektionen von digitalen Bildern.* Universität Rostock, Institut für Informatik, Diplomarbeit, 2005.

[eNo05] *Beschreibung des eNoteHistory Projekts.* 2005.

[GVK+03] Roland Göcke, Jörg Voskamp, Ekkehard Krüger, Tobias Schwinger, Ilvio Bruder, Andreas Finger, Andreas Heuer, and Temenushka Ingnatova. Ein system zur identifikation der schreiber von historischen musikhandschriften. In *4. IuK-Tage M-V*, 2003.

[HS99] Andreas Heuer and Gunter Saake. *Datenbanken: Implementierungstechniken.* mitp, 1999.

[HS00] Andreas Heuer and Gunter Saake. *Datenbanken: Konzepte und Sprachen.* mitp, 2000.

[IB05] Temenushka Ignatova and Ilvio Bruder. *Utilizing a Multimedia UML Framework for an Image Database Application.* 2005.

[IBM01] *IBM DB2 Spatial Extender Version 7, User's Guide and Reference.* 2001.

[IBM03a] *IBM DB2 Universal Dataabse Version 8, Image, Audio and Video Extenders, Administration and Programming.* 2003.

[IBM03b] *IBM DB2 Universal Dataabse Version 8.2, XML Extender Administration and Programming.* 2003.

[IBM04a] *IBM DB2 Net Search Extender V8.1.0, A Performance Overview.* 2004. White Paper.

[IBM04b] *IBM DB2 Universal Dataabse Version 8, Text Extender Administration and Programming.* 2004.

[IBM04c] *IBM DB2 Universal Dataabse Volume 8.2, SQL Reference Volume 1.* 2004.

[IBM04d] *IBM DB2 Universal Dataabse Volume 8.2, SQL Reference Volume 2.* 2004.

[ISO01a] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Multimedia and Application Packages - Part 2: Full-Text.* ISO/IEC CD 13249-2:2002 (E), 2001.

[ISO01b] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial.* ISO/IEC CD 13249-3:2002 (E), 2001.

[ISO02] *ISO/IEC Int. Standard (IS) Information Technology - SQL Multimedia and Application Packages - Part 1: Framework.* ISO/IEC FCD 13249-1:2002 (E), 2002.

[ISO03a] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Multimedia and Application Packages - Part 5: Still Image.* ISO/IEC CD 13249-5:2003 (E), 2003.

[ISO03b] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Part 1: Framework (SQL/Framework).* ISO/IEC 9075-1:2003 (E), 2003.

[ISO03c] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Part 14: XML-Related Specifications (SQL/XML).* ISO/IEC 9075-14:2003 (E), 2003.

[ISO03d] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Part 2: Foundation (SQL/Foundation).* ISO/IEC 9075-2:2003 (E), 2003.

[ISO03e] *ISO/IEC Int. Standard (IS) Information technology - Database languages - SQL Part 9: Management of External Data (SQL/MED).* ISO/IEC 9075-9:2003 (E), 2003.

[KEUB+98] Werner Kießling, Katharina Erber-Urch, Wolf-Tilo Balke, Thomas Birke, and Matthias Wagner. The heron project - multimedia database support for history and human sciences. In *GI Jahrestagung*, pages 309–318, 1998.

[KM03] Meike Klettke and Holger Meyer. *XML und Datenbanken.* dpunkt, 2003.

[Lew05] Dirk Lewandowski. *Web Information Retrieval.* Juli 2005.

[Lin02] Wolfgang Lindner. *Konzepte für ein Offenes, Objektrelationales Multimedia-Datenbanksystem.* Universität Rostock, Institut für Informatik, Dissertation, 2002.

[Mel03] Jim Melton. *ADVANCED SQL:1999.* Morgan Kaufmann, 2003.

[MMJ+01] Jim Melton, Jan-Eike Michels, Vanja Josifovski, Krishna Kulkarni, Peter Schwarz, and Kathy Zeidenstein. Sql and management of external data. In *ACM SIGMOD Record*, volume 30, 2001.

[MSS02] B. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG7.* Wiley, 2002.

[Mül05] Thomas Müller. Sql-basierte datenbankzugriffe und xml: Verarbeitung von anfrageergebnissen in anwendungsprogrammen. In *17. Workshop: Grundlagen von Datenbanken*, pages 94–98, 2005.

[NvdL05] Matthias Nicola and Bert van der Linden. Native xml support in db2 universal database. In *31st VLDB Conference*, pages 1164–1174, 2005.

[Oes04] Bernd Oestereuch. *Die UML 2.0 Kurzreferenz für die Praxis.* Oldenburg, 2004.

[OP04] Andreas Osterhold and Peter Pistor. *Die SQL-Normen*, Normen der Reihe IOS/IEC 9075.* 2004.

[Ora03a] *Oracle Database SQL Reference 10g Release 1.* 2003.

[Ora03b] *Oracle interMedia User's Guide.* 2003.

[Ora03c] *Oracle Spatial 10g Release 1, User's Guide and Reference.* 2003.

[Ora05a] *Mastering XML DB Queries in Oracle Database 10g Release 2.* 2005. White Paper.

[Ora05b] *Oracle Text, An Oracle Technical White Paper.* 2005.

[PK94] Dimitris Papadias and Marinos Kavouras. Acquiring, representing and processing spatial relations. In *6th International Symposium on Spatial Data Handling*, 1994.

[RAH+96] Mary Tork Roth, Manish Arya, Laura M. Haas, Michael J. Carey, William F. Cody, Ronald Fagin, Peter M. Schwarz, Joachim Thomas II, and Edward L. Wimmers. The garlic project. In *SIGMOD Conference*, page 557, 1996.

[Spe98] Günter Specht. Architekturen von multimedia-datenbanksystemen zur speicherung von bildern und videos. In *Inhaltsbezogene Suche von Bildern und Videosequenzen in digitalen multimedialen Archiven, Beiträge eines Workshops der 22. Jahrestagung für Künstliche Intelligenz*, pages 7–25, 1998.

[Sto03] Knut Stolze. Sql/mm spatial - the standard to manage spatial data in a relational database system. In *BTW*, pages 247–264, 2003.

[Tür03] Can Türker. *SQL:1999 & SQL:2003.* dpunkt, 2003.

[W3C03] *Extensible Markup Language (XML) 1.0 (Third Edition).* W3C Recommendation, February 2003. http://www.w3.org/TR/2004/REC-xml-20040204/.

[W3C04a] *XML Path Language (XPath) 2.0.* W3C Working Draft, October 2004. http://www.w3.org/TR/xpath20/.

[W3C04b] *XQuery 1.0: An XML Query Language.* W3C Working Draft, July 2004. http://www.w3.org/TR/2004/WD-xquery-20040723/.

Abbildungsverzeichnis

1.1 Ablauf der manuellen Handschriftenanalyse [GVK+03] 9

2.1 Aufbau einer SQL/MED-Umgebung 18

2.2 ST_Geometry-Typhierarchie [Sto03] 28

3.1 Konzeptioneller Aufbau multimedialer Dokumente 31

3.2 Konzeptioneller Aufbau eines Image-Objekts 32

3.3 Konzeptioneller Aufbau eines Text-Objekts 33

3.4 Components und ihre Beziehungen [IB05] 34

3.5 Beispielbaum einer verschachtelten Anfrage 61

5.1 Abbildung des eNoteHistory Projekts auf die Konzeption 77

Tabellenverzeichnis

2.1 Basisdatentypen von SQL:1999 bzw. SQL:2003 [Tür03] 15

2.2 Formate für Still Images aus [Mel03] 25

4.1 Vergleich zwischen DB2 und Oracle bezüglich der Konzeption 75

Anhang A

SQL-Befehle der Konzeption

A.1 DDL-Anweisungen

```
CREATE TYPE Document_T AS (
   Document DATALINK FILE LINK CONTROL
            INTEGRETY ALL
            READ PERMISSION DB
            WRITE PERMISSION BLOCKED
            RECOVERY YES
            ON UNLINK RESTORE,
   DocTree XML,
   Metadata XML,
   Subcomponents REF(Component_T) MULTISET,
   Layout ST_GeomCollection,
   Content INTEGER)
   REF IS SYSTEM GENERATED
CREATE TABLE Document OF TYPE Document_T (
   REF IS oid SYSTEM GENERATED,
   Subcomponents WITH OPTIONS SCOPE Component )

CREATE TYPE Component_T AS (
   Document REF(Document_T),
   Shape ST_Surface,
   Metadata XML,
   Subcomponents REF(Component_T) MULTISET)
   REF IS SYSTEM GENERATED
CREATE TABLE Component OF TYPE Component_T (
   REF IS oid SYSTEM GENERATED,
   Subcomponents WITH OPTIONS SCOPE Component)
```

```
CREATE TYPE TextComponent_T UNDER Component_T AS (
   Content INTEGER)
   REF IS SYSTEM GENERATED
CREATE TABLE TextComponent OF TYPE TextComponent_T (
   REF IS oid SYSTEM GENERATED)

CREATE TYPE ImageComponent_T UNDER Component_T AS (
   Content INTEGER)
   REF IS SYSTEM GENERATED
CREATE TABLE ImageComponent OF TYPE ImageComponent_T (
   REF IS oid SYSTEM GENERATED)

CREATE TYPE Collection_T (
   Documents REF(Document_T))
   REF IS SYSTEM GENERATED
CREATE TABLE Collection OF TYPE Collection_T (
   REF IS oid SYSTEM GENERATED,
   Documents WITH OPTIONS SCOPE Document_T)

CREATE FOREIGN TABLE Text (
   ID INTEGER,
   Content CLOB,
   Language VARCHAR)
SERVER TextServer

CREATE FOREIGN TABLE Images (
   ID INTEGER,
   Picture BLOB,
   isRoot BOOLEAN,
   Subpictures INTEGER MULTISET,
   Regions ROW (RegionID INTEGER,
                Shape CLOB,
                Metadata XML) MULTISET,
   ImageMetadata ROW (colorSpace VARCHAR,
                      BitsPerPixel SMALLINT))
SERVER ImageServer
```

A.2 Routinen

```
CREATE FUNCTION Above (S1 ST_Geometry, S2 ST_Geometry)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
   (IF (HighestY(S1) > HighestY(S2)
```

```
         AND (Between(LowestX(S1),LowestX(S2),HighestX(S2))
              OR Between(HighestX(S1),LowestX(S2),HighestX(S2))
              OR Between(LowestX(S2),LowestX(S1),HighestX(S1))
              OR Between(HighestX(S2),LowestX(S1),HighestX(S1))))
    THEN TRUE
    ELSE FALSE)

CREATE FUNCTION After (C1 Component_T, C2 Component_T)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN (H_Number(C1) > H_Number(C2))

CREATE FUNCTION Before (C1 Component_T, C2 Component_T)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN (After(C2,C1))

CREATE FUNCTION Between (C1 Component_T, C2 Component_T, C3 Component_T)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN (H_Number(C1) > H_Number(C2) AND H_Number(C1) < H_Number(C3))

CREATE FUNCTION Between (S1 ST_Geometry, S2 ST_Geometry, S3 ST_Geometry)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
   (IF (HighestY(S1) < HighestY(S2)
         AND HighestY(S1) > HighestY(S3)
         AND HighestX(S1) < HighestX(S2)
         AND HighestX(S1) > HighestX(S3))
    THEN TRUE
    ELSE FALSE)

CREATE FUNCTION Left (S1 ST_Geometry, S2 ST_Geometry)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
   (IF (HighestX(S1) < HighestX(S2)
         AND (Between(LowestY(S1),LowestY(S2),HighestY(S2))
              OR Between(HighestY(S1),LowestY(S2),HighestY(S2))
              OR Between(LowestY(S2),LowestY(S1),HighestY(S1))
              OR Between(HighestY(S2),LowestY(S1),HighestY(S1))))
    THEN TRUE
    ELSE FALSE)
```

```
CREATE FUNCTION MBC (CArr Component_T ARRAY[MaxLength])
RETURNS Component_T
LANGUAGE SQL
BEGIN
   DECLARE Tree XML;
   SET Tree = CArr[1].Document.DocTree;
   DECLARE Query VARCHAR;
   DECLARE Condition VARCHAR;
   SET Condition = '$x//@ComponentID =CArr[1].oid
        and fn:not($x/child::node()//@ComponentID = '+CArr[1].oid+')';
   FOR i = 2 TO CARDINALITY(CArr)
     BEGIN
       SET Condition = Condition +
          'and $x//@ComponentID ='CArr[i].oid
          and fn:not($x/child::node()//@ComponentID ='+CArr[i].oid+')';
     END
   SET Query = 'let $x := $Tree//Component
                where ' + Condition +
                return $x/@ComponentID';
   RETURN
      (SELECT C
       FROM Component C
       WHERE C.OID = XMLSERIALIZE(XMLGEN(Query) AS VARCHAR));
END

CREATE FUNCTION Near (C1 Component_T, C2 Component_T)
RETURNS DOUBLE PRECISION
LANGUAGE JAVA
EXTERNAL NAME Near_java --anwendungsabhaengige Methodendefinition

CREATE FUNCTION Paragraph(C Component_T)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(C).Segmentize('PARAGRAPH')

CREATE FUNCTION Paragraph(Doc Document_T)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(Doc).Segmentize('PARAGRAPH')

CREATE FUNCTION Picture (IC ImageComponent_T)
RETURNS INTEGER
LANGUAGE SQL
RETURN
   (SELECT ID
    FROM Images
```

```
    WHERE ID = IC.Content)

CREATE FUNCTION Pictures (C Component_T)
RETURNS ImageComponent_T ARRAY[MaxLength]
LANGUAGE SQL
BEGIN
   DECLARE Ret ARRAY[MaxLength];
   DECLARE Query VARCHAR;
   SET Query = 'let $x := $C/DocTree//Component[@ComponentID = '+C.oid+']
                return $x//@ComponentID';
   (SELECT IC INTO Ret
    FROM ONLY ImageComponent IC
    WHERE C.Document = IC.Document
       AND C.oid IN XMLSERIALIZE(XMLGEN(Query) AS VARCHAR));
    RETURN  Ret;
END

CREATE FUNCTION Pictures (Doc Document_T)
RETURNS ImageComponent_T ARRAY[MaxLength]
LANGUAGE SQL
BEGIN
   DECLARE Ret ARRAY[MaxLength];
   (SELECT IC INTO Ret
    FROM ONLY ImageComponent IC
    WHERE C.Document = Doc)
   RETURN Ret
END

CREATE FUNCTION Region (ImageID INTEGER)
RETURNS SI_StillImage ARRAY[MaxLength]
LANGUAGE SQL
BEGIN
   DECLARE Ret ARRAY[MaxLength]
   (SELECT Region.Region INTO Ret
    FROM Images
    WHERE ID = ImageID)
   RETURN Ret
END

CREATE FUNCTION Right (S1 ST_Geometry, S2 ST_Geometry)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
   (IF (HighestX(S1) > HighestX(S2)
         AND (Between(LowestY(S1),LowestY(S2),HighestY(S2))
              OR Between(HighestY(S1),LowestY(S2),HighestY(S2))
```

```
              OR Between(LowestY(S2),LowestY(S1),HighestY(S1))
              OR Between(HighestY(S2),LowestY(S1),HighestY(S1))))
    THEN TRUE
    ELSE FALSE)

CREATE FUNCTION Sentence(C Component_T)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(C).Segmentize('SENTENCE')

CREATE FUNCTION Paragraph(Doc Document_T)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(Doc).Segmentize('SENTENCE')

CREATE FUNCTION Subpicture (ImageID INTEGER)
RETURNS SI_StillImage ARRAY[MaxLength]
LANGUAGE SQL
BEGIN
   DECLARE Ret ARRAY[MaxLength];
   (SELECT DISTINCT Picture INTO Ret
    FROM GetSubpix --rekursive Funktion)
   RETURN Ret
END

CREATE FUNCTION Text (C Component_T)
RETURNS FT_FullText
LANGUAGE SQL
BEGIN
   DECLARE Arr ARRAY[MaxLength];
   DECLARE Sub ARRAY[MaxLength];
   DECLARE Query VARCHAR;
   SET Query = 'let $x := $C/DocTree//Component[@ComponentID = '+C.oid+']
                return $x//@ComponentID';
   (SELECT Content INTO Arr
    FROM ONLY TextComponent TC
    WHERE C.Document = TC.Document
       AND C.oid IN XMLSERIALIZE(XMLGEN(Query) AS VARCHAR));
    ORDER BY oid);
   DECLARE Ret VARCHAR;
   FOR i IN 1 TO CARDINALITY(Sub)
       Ret = Ret + Sub[i];
   RETURN FT_FullText(Ret)
END
```

```
CREATE FUNCTION Text (Doc Document_T)
RETURNS FT_FullText
LANGUAGE SQL
RETURN
   (SELECT FT_FullText(Content)
    FROM Text
    WHERE ID = Doc.Content)

CREATE FUNCTION Under (S1 ST_Geometry, S2 ST_Geometry)
RETURNS BOOLEAN
LANGUAGE SQL
RETURN
   (IF Above(S2,S1))
    THEN TRUE
    ELSE FALSE)

CREATE FUNCTION Word(C Component_T)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(C).Segmentize('WORD')

CREATE FUNCTION Word(Doc Document)
RETURNS FullText ARRAY[FT_MaxLength]
LANGUAGE SQL
RETURN Text(Doc).Segmentize('WORD')
```

A.3 Hilfsfunktionen

```
WITH RECURSIVE GetSubpix(Picture,Subpictures) AS (
   SELECT Picture, Subpictures
   FROM Images
   WHERE ID = ParameterID
  UNION ALL
   SELECT Picture, Subpictures
   FROM GetSubpix
)

CREATE FUNCTION H_Number(C1 Component_T)
RETURNS INTEGER
LANGUAGE SQL
BEGIN
   DECLARE Query VARCHAR,
   -- mittels Tiefensuche die Knotennummer ermitteln
   -- im Prototyp erfolgt dies bereits bei Generierung der Daten
```

```
   -- Position wird dabei durch OID repraesentiert
   SET Query = '...'
   RETURN
      (SELECT XMLSERIALIZE(CONTENT XMLGEN(Query) AS INTEGER)
       FROM Component C
       WHERE C1 = C)
END

CREATE FUNCTION HighestX (Geo ST_Geometry)
RETURNS DOUBLE
LANGUAGE SQL
BEGIN
   DECLARE Points ST_Point ARRAY[4],
   DECLARE Ret DOUBLE,
   SET Points = Geo.ST_Envelope().ST_Points(),
   SET Ret = Points.ST_PointN(1).ST_X(),
   IF (Points.ST_PointN(2).ST_X() > Ret)
     THEN SET Ret = Points.ST_PointN(2).ST_X(),
   IF (Points.ST_PointN(3).ST_X() > Ret)
     THEN SET Ret = Points.ST_PointN(3).ST_X(),
   RETURN Ret
END

CREATE FUNCTION HighestY (Geo ST_Geometry)
RETURNS DOUBLE
LANGUAGE SQL
BEGIN
   DECLARE Points ST_Point ARRAY[4],
   DECLARE Ret DOUBLE,
   SET Points = Geo.ST_Envelope().ST_Points(),
   SET Ret = Points.ST_PointN(1).ST_Y(),
   IF (Points.ST_PointN(2).ST_Y() > Ret)
     THEN SET Ret = Points.ST_PointN(2).ST_Y(),
   IF (Points.ST_PointN(3).ST_Y() > Ret)
     THEN SET Ret = Points.ST_PointN(3).ST_Y(),
   RETURN Ret
END

CREATE FUNCTION LowestX (Geo ST_Geometry)
RETURNS DOUBLE
LANGUAGE SQL
BEGIN
   DECLARE Points ST_Point ARRAY[4],
   DECLARE Ret DOUBLE,
   SET Points = Geo.ST_Envelope().ST_Points(),
   SET Ret = Points.ST_PointN(1).ST_X(),
```

```
   IF (Points.ST_PointN(2).ST_X() < Ret)
     THEN SET Ret = Points.ST_PointN(2).ST_X(),
   IF (Points.ST_PointN(3).ST_X() < Ret)
     THEN SET Ret = Points.ST_PointN(3).ST_X(),
   RETURN Ret
END

CREATE FUNCTION LowestY (Geo ST_Geometry)
RETURNS DOUBLE
LANGUAGE SQL
BEGIN
   DECLARE Points ST_Point ARRAY[4],
   DECLARE Ret DOUBLE,
   SET Points = Geo.ST_Envelope().ST_Points(),
   SET Ret = Points.ST_PointN(1).ST_Y(),
   IF (Points.ST_PointN(2).ST_Y() < Ret)
     THEN SET Ret = Points.ST_PointN(2).ST_Y(),
   IF (Points.ST_PointN(3).ST_Y() < Ret)
     THEN SET Ret = Points.ST_PointN(3).ST_Y(),
   RETURN Ret
END
```

Anhang B

SQL-Befehle des Prototyps

B.1 DDL-Anweisungen

```
CREATE TABLE hj016.Regions (
   ID INTEGER NOT NULL,
   Image INTEGER,
   Shape VARCHAR(256),
   Metadata DB2XML.XMLVARCHAR,
   PRIMARY KEY(ID))

CREATE TYPE hj016.ImageMetadata_T AS (
   Colorspace VARCHAR(10),
   BitsPerPixel INTEGER)
MODE DB2SQL
CREATE TABLE hj016.ImageMetadata OF hj016.ImageMetadata_T (
   REF IS OID USER GENERATED)

CREATE TYPE hj016.CompMetadata_T AS (
   dummy INTEGER,
   Text CLOB(100K),
   Comment CLOB(100K),
   Description CLOB(100K),
   Relic DB2XML.XMLVARCHAR )
MODE DB2SQL
CREATE TABLE hj016.CompMetadata OF hj016.CompMetadata_T (
   REF IS OID USER GENERATED,
   dummy WITH OPTIONS NOT NULL,
   PRIMARY KEY(dummy))


CREATE TYPE hj016.Docmetadata_t as (
```

```
   dummy INTEGER,
   Edition CLOB(100k),
   Comment CLOB(100k),
   Dedication CLOB(100k),
   Literature CLOB(100k))
MODE DB2SQL
CREATE TABLE hj016.docmetadata of hj016.docmetadata_t (
   REF IS oid USER GENERATED,
   dummy WITH OPTIONS NOT NULL,
   PRIMARY KEY(dummy))

CREATE TYPE hj016.Document_T AS (
   Document DATALINK
            NO LINK CONTROL,
   Metadata REF(hj016.DocMetadata_T),
   Layout VARCHAR(256),
   Content INTEGER,
   DocTree DB2XML.XMLCLOB NOT LOGGED)
MODE DB2SQL
CREATE TABLE hj016.Document OF hj016.Document_T (
   REF IS oid USER GENERATED,
   Metadata WITH OPTIONS SCOPE hj016.DocMetadata)

CREATE TYPE hj016.Collection_T AS (
   Document REF(hj016.Document_T))
MODE DB2SQL
CREATE TABLE hj016.Collection OF hj016.Collection_T (
   REF IS oid USER GENERATED,
   Document WITH OPTIONS SCOPE hj016.Document)

CREATE TYPE hj016.Component_T AS (
   Document REF(hj016.Document_T),
   Shape VARCHAR(256),
   Metadata REF(hj016.CompMetadata_T),
   ParentComponent REF(hj016.Component_T))
MODE DB2SQL
CREATE TABLE hj016.Component OF hj016.Component_T (
   REF IS oid USER GENERATED,
   Document WITH OPTIONS SCOPE hj016.Document,
   Metadata WITH OPTIONS SCOPE hj016.CompMetadata,
   ParentComponent WITH OPTIONS SCOPE hj016.Component)

CREATE TYPE hj016.ImageComponent_T UNDER hj016.Component_T MODE DB2SQL
CREATE TABLE hj016.ImageComponent OF hj016.ImageComponent_T (
   REF IS oid USER GENERATED,
   Document WITH OPTIONS SCOPE hj016.Document,
```

```
   Metadata WITH OPTIONS SCOPE hj016.CompMetadata,
   ParentComponent WITH OPTIONS SCOPE hj016.Component)

CREATE TABLE hj016.Images (
   ID INTEGER,
   ImageComp REF(hj016.ImageComponent_T) SCOPE hj016.ImageComponent,
   Picture BLOB(5M),
   isRoot SMALLINT,
   ParentPicture INTEGER,
   ImageMetadata REF(hj016.ImageMetadata_T) SCOPE hj016.ImageMetadata)

CREATE TYPE hj016.Relations_T AS (
   Description VARCHAR(200),
   Root REF(hj016.Component_T),
   Comps varchar(256))
MODE DB2SQL
CREATE TABLE hj016.Relations OF hj016.Relations_T (
   REF IS OID USER GENERATED,
   Root WITH OPTIONS SCOPE hj016.Component)

CREATE TYPE hj016.LogicalRelations_T UNDER hj016.Relations_T MODE DB2SQL
CREATE TABLE hj016.LogicalRelations OF hj016.LogicalRelations_T (
   REF IS OID USER GENERATED,
   Root WITH OPTIONS SCOPE hj016.Component)

CREATE TYPE hj016.InterprRelations_T UNDER hj016.Relations_T MODE DB2SQL
CREATE TABLE hj016.InterprRelations OF hj016.InterprRelations_T (
   REF IS OID USER GENERATED,
   Root WITH OPTIONS SCOPE hj016.Component)

CREATE TYPE hj016.InteracRelations_T UNDER hj016.Relations_T MODE DB2SQL
CREATE TABLE hj016.InteracRelations OF hj016.InteracRelations_T (
   REF IS OID USER GENERATED,
   Root WITH OPTIONS SCOPE hj016.Component)
```

B.2 Routinen

```
CREATE FUNCTION Above (Region1 Int, Region2 Int)
RETURNS SMALLINT
LANGUAGE SQL
BEGIN ATOMIC
   DECLARE LX1 Int;
   DECLARE HX1 Int;
   DECLARE LY1 Int;
```

```
   DECLARE HY1 Int;
   DECLARE LX2 Int;
   DECLARE HX2 Int;
   DECLARE LY2 Int;
   DECLARE HY2 Int;
   SET LX1 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/LX')
              FROM Regions
              WHERE ID = Region1);
   SET HX1 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/HX')
              FROM Regions
              WHERE ID = Region1);
   SET LY1 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/LY')
              FROM Regions
              WHERE ID = Region1);
   SET HY1 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/HY')
              FROM Regions
              WHERE ID = Region1);
   SET LX2 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/LX')
              FROM Regions
              WHERE ID = Region2);
   SET HX2 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/HX')
              FROM Regions
              WHERE ID = Region2);
   SET LY2 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/LY')
              FROM Regions
              WHERE ID = Region2);
   SET HY2 = (SELECT db2xml.extractInteger(
                                Metadata,'/Metainfo/Rectangle/HY')
              FROM Regions
              WHERE ID = Region2);
   IF (HY1 > HY2
         AND  (LX1 Between LX2 and HX2
              OR HX1 Between LX2 and HX2
              OR LX2 Between LX1 and HX1
              OR HX2 Between LX1 and HX1))
      THEN RETURN 1;
      ELSE RETURN 0;
   END IF;
END
```

```
CREATE FUNCTION After (C1 VARCHAR(10), C2 VARCHAR(10))
RETURNS SMALLINT
LANGUAGE SQL
BEGIN ATOMIC
   DECLARE Num1 Int;
   DECLARE Num2 Int;
   SET Num1 = (SELECT Metadata->Dummy
               FROM Component
               WHERE CAST(oid AS VARCHAR(10)) = C1);
   SET Num2 = (SELECT Metadata->Dummy
               FROM Component
               WHERE CAST(oid AS VARCHAR(10)) = C2);
   IF (Num1 > Num2)
      THEN RETURN 1;
      ELSE RETURN 0;
   END IF;
END

CREATE FUNCTION Before (C1 VARCHAR(10), C2 VARCHAR(10))
RETURNS SMALLINT
LANGUAGE SQL
RETURN After(C2,C1)

CREATE FUNCTION FuzzyBetween(
                    C1 VARCHAR(10), C2 VARCHAR(10), C3 VARCHAR(10))
RETURNS SMALLINT
LANGUAGE SQL
BEGIN ATOMIC
   IF((After(C1,C2)=1) AND (Before(C1,C3)=1))
      THEN RETURN 1;
      ELSE RETURN 0;
   END IF;
END

CREATE FUNCTION ExactBetween (Region1 Int, Region2 Int, Region3 Int)
RETURNS SMALLINT
BEGIN ATOMIC
   DECLARE LX1 Int;
   DECLARE HX1 Int;
   DECLARE LY1 Int;
   DECLARE HY1 Int;
   DECLARE LX2 Int;
   DECLARE HX2 Int;
   DECLARE LY2 Int;
   DECLARE HY2 Int;
```

```
DECLARE LX3 Int;
DECLARE HX3 Int;
DECLARE LY3 Int;
DECLARE HY3 Int;
SET LX1 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LX')
           FROM Regions
           WHERE ID = Region1);
SET HX1 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/HX')
           FROM Regions
           WHERE ID = Region1);
SET LY1 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LY')
           FROM Regions
           WHERE ID = Region1);
SET HY1 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/HY')
           FROM Regions
           WHERE ID = Region1);
SET LX2 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LX')
           FROM Regions
           WHERE ID = Region2);
SET HX2 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/HX')
           FROM Regions
           WHERE ID = Region2);
SET LY2 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LY')
           FROM Regions
           WHERE ID = Region2);
SET HY2 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/HY')
           FROM Regions
           WHERE ID = Region2);
SET LX3 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LX')
           FROM Regions
           WHERE ID = Region3);
SET HX3 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/HX')
           FROM Regions
           WHERE ID = Region3);
SET LY3 = (SELECT db2xml.extractInteger(
                            Metadata,'/Metainfo/Rectangle/LY')
```

```
              FROM Regions
              WHERE ID = Region3);
   SET HY3 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/HY')
              FROM Regions
              WHERE ID = Region3);
   IF (    ((LX1>LX2) AND (LX1<LX3) AND (LY1>LY2) AND (LY1<LY3))
        OR ((LX1<LX2) AND (LX1>LX3) AND (LY1<LY2) AND (LY1>LY3))
        OR ((Right(Region1,Region2)=1) AND Left(Region1,Region3)=1)
        OR ((Under(Region1,Region2)=1) AND Above(Region1,Region3)=1)
        OR ((Right(Region1,Region3)=1) AND Left(Region1,Region2)=1)
        OR ((Under(Region1,Region3)=1) AND Above(Region1,Region2)=1))
      THEN RETURN 1;
      ELSE RETURN 0;
   END IF;
END

CREATE FUNCTION Left (Region1 Int, Region2 Int)
RETURNS SMALLINT
BEGIN ATOMIC
   DECLARE LX1 Int;
   DECLARE HX1 Int;
   DECLARE LY1 Int;
   DECLARE HY1 Int;
   DECLARE LX2 Int;
   DECLARE HX2 Int;
   DECLARE LY2 Int;
   DECLARE HY2 Int;
   SET LX1 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/LX')
              FROM Regions
              WHERE ID = Region1);
   SET HX1 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/HX')
              FROM Regions
              WHERE ID = Region1);
   SET LY1 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/LY')
              FROM Regions
              WHERE ID = Region1);
   SET HY1 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/HY')
              FROM Regions
              WHERE ID = Region1);
   SET LX2 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/LX')
```

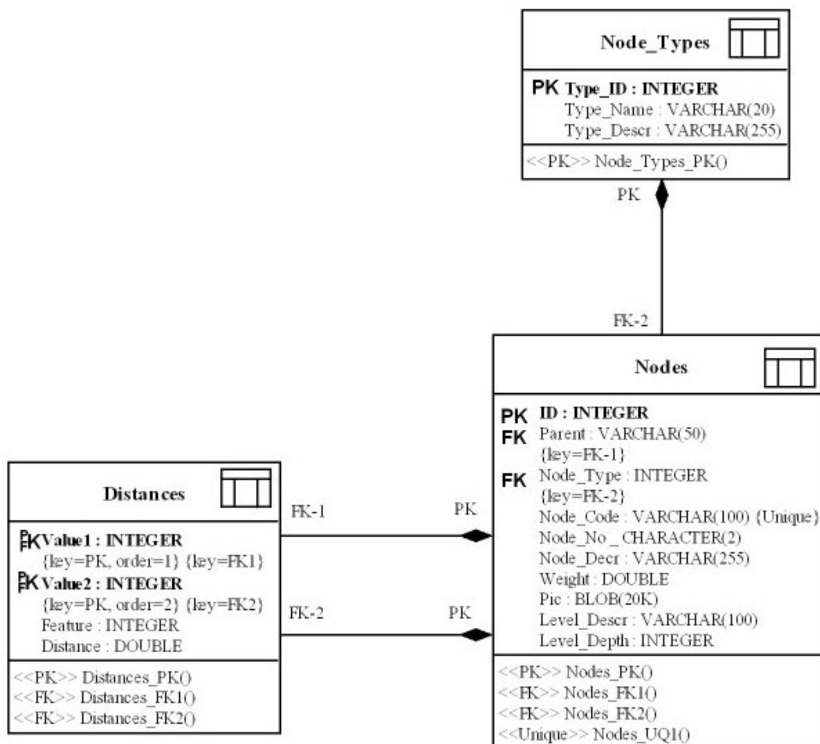
```
              FROM Regions
              WHERE ID = Region2);
   SET HX2 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/HX')
              FROM Regions
              WHERE ID = Region2);
   SET LY2 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/LY')
              FROM Regions
              WHERE ID = Region2);
   SET HY2 = (SELECT db2xml.extractInteger(
                               Metadata,'/Metainfo/Rectangle/HY')
              FROM Regions
              WHERE ID = Region2);
   IF (HX1 < HX2
         AND (   LY1 Between LY2 and HY2
              OR HY1 Between LY2 and HY2
              OR LY2 Between LY1 and HY1
              OR HY2 Between LY1 and HY1))
       THEN RETURN 1;
       ELSE RETURN 0;
    END IF;
END

CREATE FUNCTION Near(C1 VARCHAR(10), C2 VARCHAR(10))
RETURNS DOUBLE
LANGUAGE SQL
BEGIN ATOMIC
   DECLARE num1 DOUBLE;
   DECLARE num2 DOUBLE;
   DECLARE numGes DOUBLE;
   SET numGes = (SELECT COUNT(*)
                 FROM Component
                 WHERE CAST(Document AS VARCHAR(10))
                 =
                (SELECT CAST(Document AS VARCHAR(10))
                 FROM Component
                 WHERE CAST(oid AS VARCHAR(20))= C1));
   SET num1 = (SELECT Metadata->Dummy
               FROM Component
               WHERE CAST(oid AS VARCHAR(10))=C1);
   SET num2 = (SELECT Metadata->Dummy
               FROM Component
               WHERE CAST(oid AS VARCHAR(10))=C2);
   RETURN POWER((numGes/(numGes-abs(num1-num2))),2);
END
```

```
CREATE FUNCTION Picture (IC VARCHAR(10))
RETURNS INTEGER
LANGUAGE SQL
RETURN
   (SELECT ID
    FROM Images
    WHERE CAST(ImageComp as VARCHAR(10)) = IC)

CREATE FUNCTION PicturesFromComp (C VARCHAR(10))
RETURNS TABLE (IC VARCHAR(10))
LANGUAGE SQL
RETURN
   (SELECT CAST(oid AS VARCHAR(10))
     FROM ImageComponent
     WHERE CAST(ParentComponent AS VARCHAR(10)) = C
        OR CAST(ParentComponent->ParentComponent AS VARCHAR(10)) = C)

CREATE FUNCTION PicturesFromDoc (Doc VARCHAR(10))
RETURNS TABLE(IC VARCHAR(10))
LANGUAGE SQL
RETURN
   (SELECT CAST(oid AS VARCHAR(10))
    FROM ImageComponent
    WHERE CAST(Document AS VARCHAR(10)) = Doc)

CREATE FUNCTION Regions (ImageID INTEGER)
RETURNS TABLE(RegionID INTEGER)
LANGUAGE SQL
RETURN
   (SELECT ID
    FROM Regions
    WHERE Image = ImageID)

CREATE FUNCTION Right (Region1 Int, Region2 Int)
RETURNS SMALLINT
LANGUAGE SQL
RETURN Left(Region2,Region1)

CREATE FUNCTION Under (Region1 Int, Region2 Int)
RETURNS SMALLINT
LANGUAGE SQL
RETURN  Above(Region2,Region1)
```

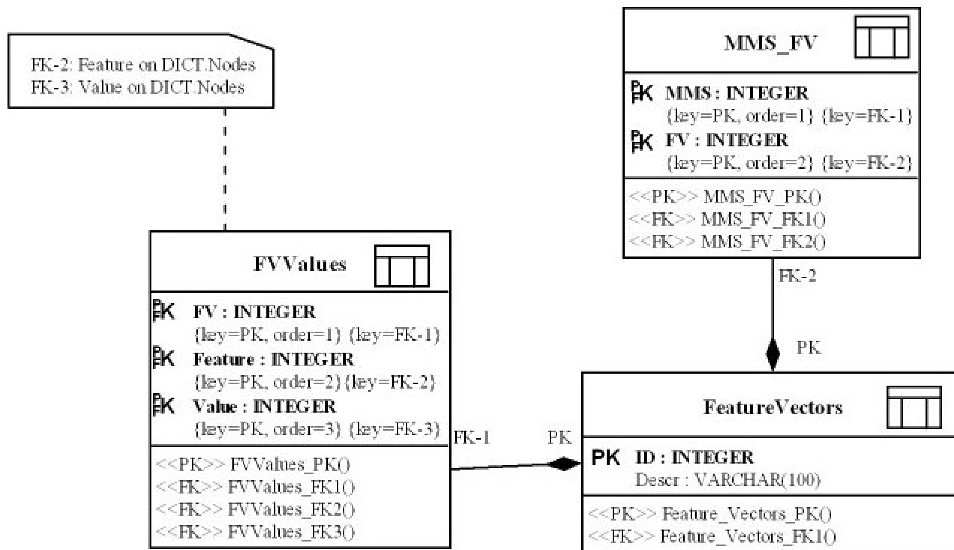
Anhang C

Schemata der eNoteHistory Datenbank

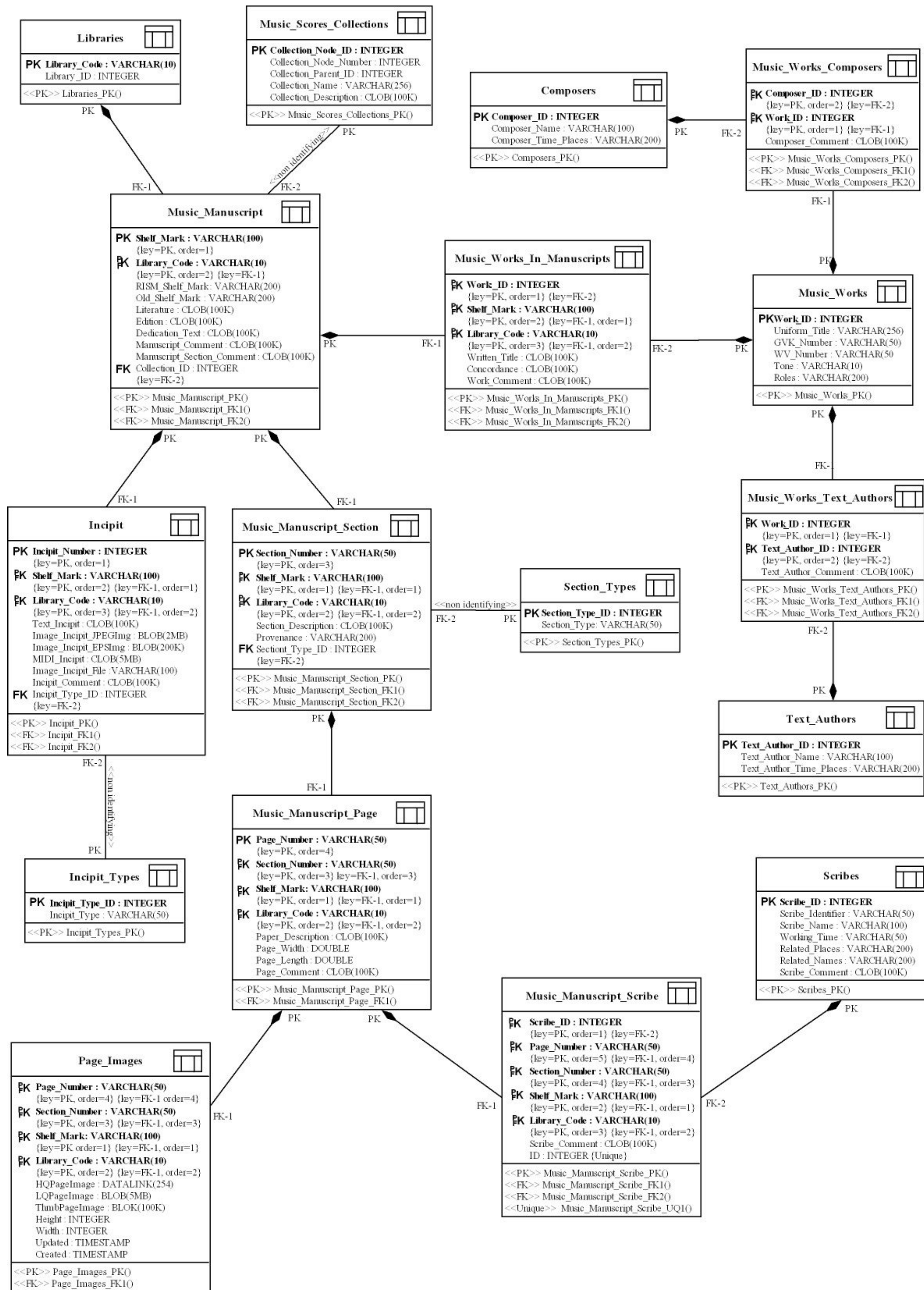
C.1 Dict



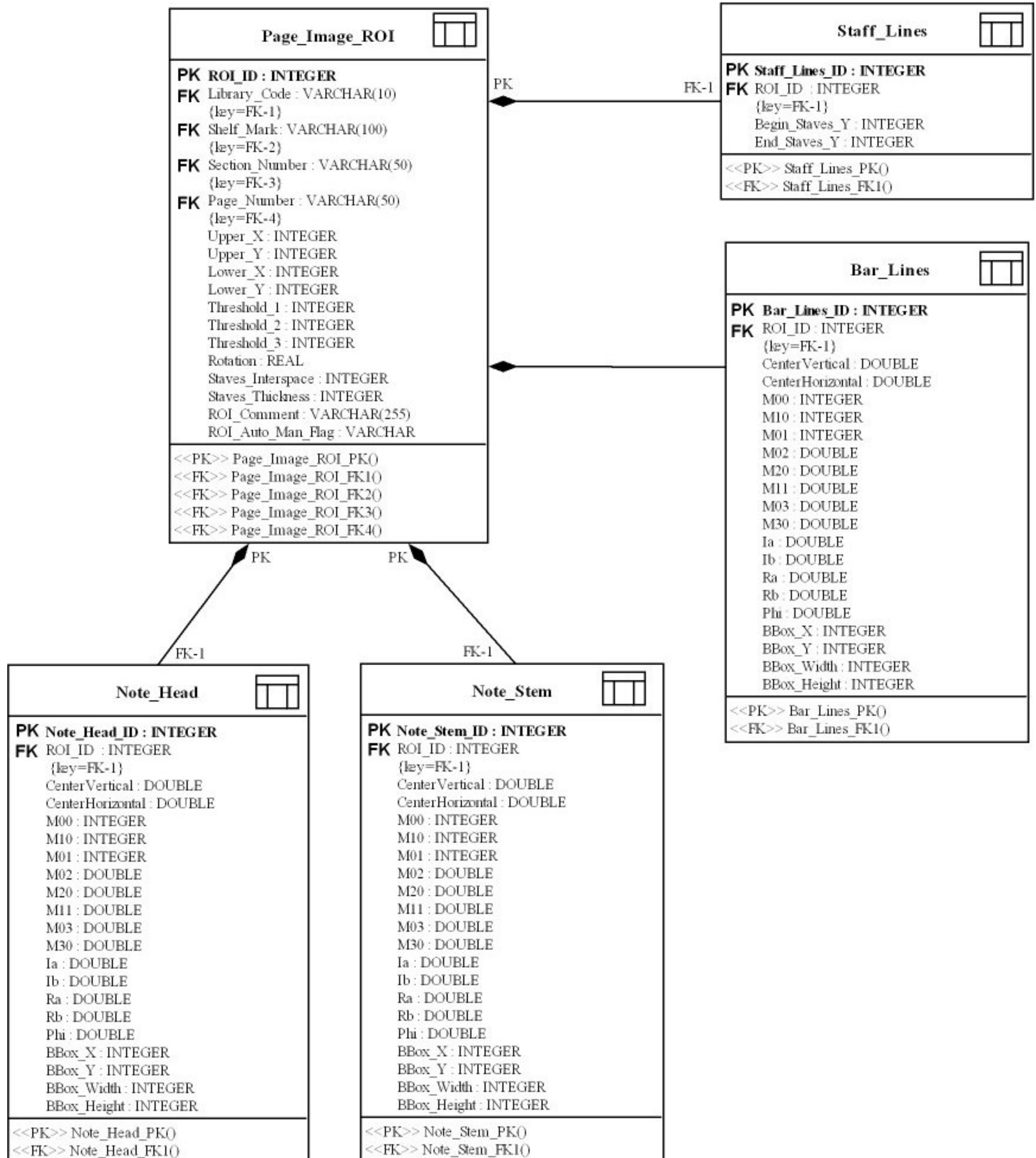
C.2 Features



C.3 Metadata



C.4 IPFV



Selbständigkeitserklärung

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

Rostock, den 15.12.2005

Thesen

1. Retrieval-Konzepte existieren für einzelne Medien, ein Retrieval über das Gesamtobjekt eines multimedialen Dokuments wird bisher nur rudimentär unterstützt oder basiert ausschließlich auf einer anwendungsabhängigen Plattform.
2. Eine Backend-Lösung, die vollständig auf bestehenden Standards basiert, ist anwendungsunabhängig und kann allgemein als Grundlage verwendet werden.
3. Durch Nutzung der ISO-Standards SQL:2003 und SQL/MM kann eine kombinierte Anfrageverarbeitung auf Struktur und Inhalt multimedialer Dokumente realisiert werden.
4. Die Möglichkeit der Integration extern gespeicherter Daten, die von spezialisierten Medien-Servern verwaltet werden, wird durch SQL bereitgestellt. Dadurch werden spezialisierte Verarbeitungsfunktionalitäten, effiziente Verarbeitung hoher Datenaufkommen oder Verwendung von Spezialhardware ermöglicht.
5. Anfragen auf Struktur können in die Klassen geometrische, unscharfe, logische modellbezogene und logische selbstdefinierte Anfragen klassifiziert werden.
6. Anfragen auf Inhalte bestehen aus textbezogenen, bildbezogenen und metadatenbezogenen Anfragen.
7. Eine beliebige Kombination von strukturellen und inhaltsbezogenen Anfragen kann durch verschachtelte SFW-Blöcke realisiert werden.
8. Die objektrelationalen Datenbankmanagementsysteme von IBM und Oracle bieten laut Dokumentation eine ausreichende Unterstützung der SQL-Standards, um eine praktische Umsetzung des Multimedia-Datenbanksystems zu gewährleisten. Allerdings bestehen teilweise auch Lücken in der Standardunterstützung, die jedoch kompensiert werden können.
9. Der vom eNoteHistory-Projekt bereitgestellte Bestand multimedialer Dokumente kann verwendet werden, um eine praktische Anwendung kombinierter Anfragen auf Struktur und Inhalt zu realisieren.