Metadatenverwaltung für Multimedia-Content-Management mit OLAP-Funktionalität

Diplomarbeit

Universität Rostock, Fachbereich Informatik

---

vorgelegt von **Stefan Audersch**
geboren am 08.04.1976 in Rostock

Gutachter: Prof. Dr. Andreas Heuer, Universität Rostock
Prof. Dr. Bodo Urban , Universität Rostock, Frauenhofer IGD

Betreuer: Dipl.-Inf. Guntram Flach, ZGDV e.V. Rostock
Dipl.-Inf. Thomas Courvoisier, ZGDV e.V. Rostock
Dipl.-Inf. Astrid Lubinski, Universität Rostock

Abgabedatum: 15.01.2002

Zusammenfassung

Der Hauptschwerpunkt bei der Entwicklung wireless-orientierter, multimedialer Content-Management-Systeme liegt auf der adäquaten Speicherung, Verwaltung und Recherche multimedialer Dokument-Strukturen sowie auf der anwendungsspezifischen Darstellung und Repräsentation dieser Dokumente auf verschiedenen Endgeräten. In diesem Kontext kann die Extensible Markup Language (XML) zur Verbesserung einer großen Bandbreite möglicher Anwendungen beitragen.

Im Rahmen dieser Arbeit wird ein Ansatz für eine universelle Metadatenverwaltung zur Steuerung signifikanter Transformations- und Daten-Analyse-Prozesse entwickelt. Dabei wird eine möglichst enge Kopplung von Data Warehouse-Funktionalität (OLAP) mit einer ausgewählten XML-Framework-Architektur angestrebt. Ziel ist ein erweiterter Architektur-Vorschlag, der die Grundlage für zukünftige offene und erweiterbare Multimedia-Content-Management-Systeme bilden kann. Grundlage dieser Architektur ist ein Metadaten-Repository auf der Basis der Resource Description Framework Spezifikation (RDF).

Abstract

The focus in development of wireless oriented, multimedia centred Content Management Systems is on persistence, management, maintenance and search of multimedia documents as well as on application specific presentation of such documents on various devices. In this context the Extensible Markup Language (XML) provides benefit to a large scale of applications.

This work presents an approach for a universal management of metadata, controlling processes of significant transformation and data analysis. Therefore a tight link between data warehouse functionality (OLAP) and a selected XML framework architecture is intended. This work aims on an extended proposal of an open and scalable Multimedia Content Management System. Fundament of this architecture is a metadata repository on the base of the Resource Description Framework (RDF).

CR-Klassifikation

E.2 Data Storage Representations

H.2.4 Database Management Systems

H.3.3 Information Search and Retrieval

H.5.1 Multimedia Information Service

Keywords

RDF, Metadata, OLAP, Mobile Computing, Query Transformation, XML, Request Processing, Multimedia Information System

Danksagung

An dieser Stelle möchte ich mich herzlich bei den Gutachtern und Betreuen dieser Arbeit für die gegebene Unterstützung, die sachlichen Hinweise und die konstruktive Kritik bedanken. Besonders möchte ich Guntram Flach und Thomas Courvoisier für die gute Betreuung während meiner Hiwi-Tätigkeit im ZGDV danken.

Weiterer Dank gilt meiner Freundin Anja für die umfangreiche Motivations- und Aufbauarbeit, die sie während meines Studiums und speziell in der Diplomphase geleistet hat.

Schließlich möchte ich mich bei meinen Studienkollegen für die angenehme, gemeinsame und interessante Studienzeit bedanken.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Program Interface
CDF	Channel Definition Format
DBMS	Database Managements System
DDL	Data Definition Language
DTD	Document Type Definition
DML	Data Manipulation Language
EBNF	Extended Backus-Naur Form
EER	Extended Entity Relationship Model
FASMI	Fast Analysis of Shared Multidimensional Information
GUI	Graphical User Interface
HOLAP	Hybrid OLAP
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
MCF	Meta Content Framework
MOLAP	Multidimensional OLAP
NF^2	Non First Normal Form
OLAP	On-line Analytical Processing
OLTP	On-line Transaction Processing
OODBMS	Object-Oriented Database Management System
OQL	Object Query Language
ORDBMS	Object-Relational Database Management System
PICS	Platform for Internet Content Selection
RDBMS	Relational Database Management System
RDF	Resource Description Framework
ROLAP	Relational OLAP
RQL	RDF Query Language
SQL	Structured Query Language
TID	Tupel Identifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
URN	Uniform Resource Name
WAP	Wireless Application Protocol
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
XQL	XML Query Language
XSL	XML Stylesheet Language
XSLT	XSL Transformation

1. Einleitung

Die Verwaltung, Aufbereitung und Präsentation von Informationen macht im Allgemeinen einen großen Teil der Aufgaben aus, die von Anwendungssystemen wahrgenommen werden. In diesem Kontext kann die daten-, datensatz- oder dokumentbezogene Trennung von Inhalt, Struktur und Formatierung eines spezifischen Datums zu einer verbesserten Umsetzung von Anwendungssystemen beitragen, die sich insbesondere in Form höherer Flexibilität und Interoperabilität niederschlägt. Flexibilität und Interoperabilität werden beispielsweise dadurch unterstützt, dass einem Dokument jederzeit beliebige Informationen hinzugefügt und Datensätze um beliebige Attribute ergänzt werden können. Weiterhin kann ein Empfänger aus einem Datenstrom oder einem Dokument die für ihn im momentanen Anwendungsfall relevanten Inhalte (Nutzdaten) extrahieren. Hierfür werden neben den eigentlichen Nutzdaten noch zusätzliche Informationen über die Nutzdaten (Metainformationen) zur Verfügung gestellt, die zwischen den Kommunikationspartnern vereinbart sind. Softwareentwickler werden in jüngster Zeit mit der Forderung konfrontiert, den Zugriff auf Informationssysteme über verschiedene Klassen von Endgeräten zu gewährleisten. Zurzeit sind es vor allem mobile Endgeräte, die mit bestehenden Informationssystemen verbunden werden sollen, aber auch weitere Entwicklungen im stationären Bereich sind abzusehen.

Im Kontext wireless-orientierter, multimedialer Content Management-Systeme trägt die Extensible Markup Language (XML) zur Verbesserung einer großen Bandbreite möglicher Anwendungen bei. Offene Content-Management-Architekturen, die die Content Mediation sowohl auf struktureller Ebene (XML/XSLT) als auch durch Daten-Preprocessing (z.B. Multimedia-Erweiterungen von objekt-relationalen Datenbanksystemen) vornehmen, schaffen die Voraussetzung für komplexe, mobile Applikationen.

Metadaten sind die Voraussetzung für die Steuerung der Request-Verarbeitung und Anfrage-Transformation in derartigen komplexen Systeminfrastrukturen. Erste Lösungsansätze auf der Basis der Resource Description Framework-Spezifikation (RDF) bezüglich der Implementierung einer zusätzlichen semantischen Ebene bei der Realisierung eines adäquaten Metadaten-Repositories bilden die Grundlage für aktuelle Forschungs- und Entwicklungsarbeiten.

Schwerpunkt dieser Diplomarbeit ist die Entwicklung eines Ansatzes für eine universelle Metadatenverwaltung zur Steuerung signifikanter Transformations- und Daten-Analyse-Prozesse. Dabei wird eine möglichst enge Kopplung von Data Warehouse-Funktionalität (OLAP) mit einer ausgewählten XML-Framework-Architektur angestrebt. Ziel ist ein erweiterter Architektur-Vorschlag, der die Grundlage für zukünftige offene und erweiterbare Multimedia-Content-Management-Systeme bilden könnte.

Konkrete Fragestellungen in diesem Rahmen sind unter anderem: Wie kann eine adäquate, RDF-basierte Metadatenverwaltung hinsichtlich Datenmodell, Speicherung und Anfragen realisiert werden? Welche Möglichkeiten bestehen bezüglich der Steuerung der Datentrans-

formation, der kontextabhängigen Wissensaufbereitung und der übergreifenden Informationssuche sowie der Nutzung von OLAP-Funktionalität?

Die Diskussion der beschriebenen Probleme und die Darlegung von Lösungsmöglichkeiten im Bereich internetbasierter Portal-Anwendungen (z.B. Europa-MV) sind Bestandteil dieser Diplomarbeit, die am Fachbereich Informatik an der Universität Rostock in Zusammenarbeit mit dem Zentrum für Graphische Datenverarbeitung e.V. Rostock (ZGDV) verfasst wurde.

1.1 Aufbau der Arbeit

Das Thema „Metadatenverwaltung für Multimedia-Content-Management mit OLAP-Funktionalität“ umfasst verschiedene Themenbereiche der Informatik. Grundlegende Begriffe sind Metadaten und OLAP, denen der erste Teil dieser Arbeit gewidmet ist. Intensivere Betrachtung findet hierbei in Kapitel 2 der Metadatenbeschreibungsstandard RDF. Das für die Bereitstellung von OLAP-Funktionalität notwendige multidimensionale Datenmodell sowie die Möglichkeiten der multidimensionalen Anfrage mit SQL sind Gegenstand des dritten Kaptitels.

Ein Schwerpunkt der Diplomarbeit ist die Realisierung einer adäquaten, RDF-basierten Metadatenverwaltung. Sie ist Focus im zweiten Teil der Arbeit. Die Metadatenverwaltung wird eingesetzt zur Steuerung der Request-Verarbeitung und Anfrage-Transformation in einer gewählten XML-Framework-Architektur. Die Architektur sowie der Metadatenbestand werden in Kapitel 4 vorgestellt. Eine RDF-basierte Metadatenverwaltung umfasst Verfahren zu der Persitierung und dem Retrieval von RDF-Daten. Für die beiden Aufgaben werden in Kapitel 5 der Arbeit verschiedene Techniken und Ansätze ausführlich diskutiert. Die daraus resultierende Steuerung der Datentransformation ist in Kapitel 6 geschildert.

Der dritte Teil der Arbeit befasst sich mit der Erweiterung um OLAP-Funktionalität. Im Kapitel 7 werden hierzu die Erweiterung des Metadatenbestandes, verschiedene Realisierungstechniken zur Bereitstellung von OLAP-Funktionalität, die notwendige Kommunikation und die serverseitige Anfragebearbeitung behandelt.

Prototypische Implementierungen, welche die evaluierten und entwickelten Konzepte umsetzten, sind Gegenstand im letzten Teil der Arbeit. In Kapitel 8 werden die Metadatenverwaltung (RDF-Content-Repository) und die Komponente zur Erweiterung um OLAP-Funktionalität (OReSMo) betrachtet.

1.2 Einordnung und Anwendungsszenarien

Die Realisierung dieser Diplomarbeit erfolgt im Rahmen des *XPEA* Projektes am Zentrum für Graphische Datenverarbeitung Rostock e.V.

XPEA ist eine ***X****ML-basierte* ***P****lattform für wireless-orientierte* ***E****-Business- und Content-Managment-* ***A****pplikationen*. Die Plattform besteht aus Werkzeugen und Modulen einer offenen XML-Framework-Architektur, die den generischen Zugriff auf multimediale Dokumente insbesondere in ORDBMS ermöglicht. Dokumentinhalte werden kontext- und endgerätespezifisch aufbereitet. *XPEA* stellt insofern eine Erweiterung des *DigiTed* - Systems [Ditt01]

dar, das bereits Thema einer Diplomarbeit an Universität Rostock war. Die Transformation und Mediation von multimedialen Dokumenten erfolgt im *XPEA*-Framework auf der Basis umfangreicher Metadaten. Die Metadaten beschreiben sowohl Eigenschaften von Datenquellen und Endgeräten als auch semantische Zusammenhänge und Abhängigkeiten zwischen den Daten selber.

Ziel dieser Arbeit ist neben der Realisierung einer adäquaten Metadatenverwaltung, die Erweiterung der XML-Framework-Architektur um OLAP-Funktionalität. Die XPEA-Plattform bietet hierfür eine geeignete Grundlage. Als Anwendungsszenario dient der Datenbestand des Projektes *Europa-MV*, das ebenfalls im ZGDV realisiert wird. *Europa-MV* ist eine Internet-Plattform der Zusammenarbeit und Informationsgewinnung für EU-bezogene Aktivitäten in Mecklenburg-Vorpommern. *Europa-MV* wird als Informations-, Kooperations- und Kommunikationsplattform den Unternehmen, aber auch Kommunen, Hochschuleinrichtungen, Bildungsträgern und anderen Akteuren die Möglichkeit geben, mit ihren jeweiligen Partnern, unabhängig von Ort und Zeit, gemeinsam an Dokumenten zu arbeiten und Projekte zu entwickeln. *Europa-MV* bietet sich daher als Instrument für internationales Projektmanagement an, beispielsweise bei der Herstellung von Erstkontakten zwischen kooperationswilligen Unternehmen und Kommunen oder der Formulierung von EU-Förderanträgen.

Teil des Datenbestandes von *Europa-MV* ist eine Projektdatenbank mit EU-geförderten Projekten in Mecklenburg-Vorpommern. Anhand dieser Projektdatenbank soll die OLAP-Funktionalität der *XPEA*-Plattform demonstriert werden. Anwendungsszenario ist die differenzierte Analyse der Entwicklung EU-geförderter Aktivitäten in Mecklenburg-Vorpommern durch portable Webapplikationen auf der Basis der *XPEA*-Plattform.

Teil I
Grundbegriffe

2. Metadaten

Was sind Metadaten?

Der Begriff „Metadaten“ ist nicht neu, sondern gewinnt im Zusammenhang mit modernen elektronischen Informationssystemen, wie dem World Wide Web, wieder an Bedeutung. Unter Metadaten („Daten über Daten“) versteht man hierbei strukturierte Daten, mit deren Hilfe Informationsressourcen beschrieben werden. Dabei kann es sich sowohl um digitalisierte Daten (z.B. Textdokumente, Videos, ausführbare Dateien oder Archive) als auch um nicht digitalisierte Dinge (z.B. ein gedrucktes Buch oder ein Termin) handeln. Bei einem elektronischen Textdokument sind Metadaten beispielsweise Angaben über Autor, Titel oder Erstellungsdatum, die im einfachsten Fall Attribut-Wert-Paare darstellen, zum Beispiel:

- Autor - Stefan Audersch,
- Erstellungsdatum - 15.01.2001.

Tim Berners-Lee, der Direktor des *World Wide Web Consortiums* (W3C), definiert in [Bern97]: „Metadaten sind maschinenverständliche Informationen über elektronische Ressourcen oder andere Dinge.“ Die Betonung des Satzes liegt hierbei auf „maschinenverständlich“. Durch eine wohldefinierte Semantik und Struktur der Metadaten soll dieses erreicht werden, um somit die Automatisierung von Vorgängen, wie dem Filtern oder Suchen von Ressourcen zu ermöglichen.

Metadaten sind wiederum Daten [Bern97]. Es existiert also keine eindeutige Unterscheidung zwischen Daten und Metadaten. Welche Informationen als Metadaten und welche als Daten aufgefasst werden, muss letztendlich anwendungsabhängig entschieden werden.

2.1 Klassifikation von Metadaten

Die vorhergehende Einleitung machte bereits deutlich, dass es schwer möglich ist, eine exakte, allgemeingültige Definition für Metadaten zu geben. Vielmehr lässt sich nur charakterisieren, welche Informationen als Metadaten angesehen werden können.

Um ein besseres Verständnis zum Begriff Metadaten zu erhalten, ist es hilfreich diese anhand verschiedener Kriterien zu klassifizieren:

Speicherort

Die Speicherung und Verfügbarmachung von Metadaten lässt sich grundsätzlich in drei Varianten unterscheiden [Fiel94]:

- Metadaten werden getrennt vom Dokument gespeichert und können unabhängig von diesem abgerufen werden.
- In einem Container werden die beschriebene Ressource und deren Metadaten verpackt. Der Container stellt die Informationen bereit, wenn sie benötigt werden.
- Die Metadaten sind im Dokument selbst enthalten.

Ein typisches Beispiel für Metadaten innerhalb des Dokuments, sind die HTML-META-Tags im Kopf eines HTML-Dokumentes.

```
<META name="language" content="de">
<META name="author" content="Stefan Audersch">
<META name="keywords" content="Diplomarbeit, RDF, OLAP">
```

Listing 2.1: HTML-META-TAGS

Die Meta-Informationen beschreiben das HTML-Dokument. Suchmaschinen können diese interpretieren und so die Seiten besser indizieren.

Anwendungsbereiche

Die Anwendung von Metadaten beschränkt sich nicht nur auf die Unterstützung von Suchmaschinen oder Filtertechniken. Einige der wichtigsten Anwendungsbereiche für Metadaten sind:

- Unterstützung der Suche von Ressourcen
- Bewertung von Ressourcen
- Beschreibung von Urheberrechten zum Schutz des geistigen Eigentums
- Beschreibung zur Nutzung von Ressourcen
- Zugangskontrolle zu Ressourcen (um zum Beispiel den Zugang zu Seiten mit ungeeignetem Inhalt für Kinder zu blockieren)
- Darstellung von Beziehungen zu anderen Ressourcen
- Einordnung von Ressourcen in ein größeres Umfeld

Verwendungsmöglichkeiten

Entsprechend den Anwendungsbereichen lassen sich Metadaten hinsichtlich ihrer Verwendung in die folgenden Klassen unterteilen [BeSo96]:

- **resource discovery:** Metadaten zu Identifikation und für Nachweiszwecke, auch administrative Daten
- **terms and conditions**: Metadaten für Zugangsbedingungen sowie Nutzungs- und Beschaffungskonditionen
- **structure:** Metadaten zu strukturellen Aspekten
- **context**: Metadaten zum Kontext
- **content:** Metadaten zum Inhalt

- **use history**: Metadaten zur Nutzungs- und Wirkungsgeschichte

Beziehungen zwischen Metadaten und Ressource

Hinsichtlich der Beziehung zwischen der Metadatenbeschreibung und dem Inhalt der zu beschreibenden Ressource lassen sich Metadaten folgendermaßen klassifizieren [Gros97]:

- **content dependent:** Hierbei handelt es sich um die Metadaten, die sich direkt aus dem Inhalt der ursprünglichen Ressource, zum Beispiel durch Schlüsselwortsuche gewinnen lassen. Indizierungstechniken für Texte basieren beispielsweise auf dieser Metadatenklasse.
- **content descriptive:** Inhaltsbeschreibende (*content descriptive*) Metadaten beruhen ebenfalls auf dem Inhalt der ursprünglichen Ressource, gehen jedoch nicht direkt aus ihnen hervor.
- **content independent:** Im Gegensatz zu den vorhergehenden Klassen haben inhaltsunabhängige (*content independent*) Metadaten im Grunde nichts mit dem Inhalt der Ressource zu tun. Im Falle von Dateien sind sie zum Beispiel Angaben über Besitzer, Zugriffsrechte oder Datum der letzten Änderung.

Erzeugungsmethode

Für die Erzeugung von Metadaten existieren verschiedene Herangehensweisen (basierend auf [Gill00]):

- **automatic**: Die Generierung von Metadaten kann automatisch durch eine Maschine erfolgen. Ein typisches Beispiel hierfür ist der Aufbau von Indizes für die Schlüsselwortsuche.
- **manual**: Häufig ist es nicht möglich die notwendigen Metadaten durch einen Computer oder eine Maschine automatisch zu extrahieren. Die Erzeugung der Daten erfolgt in diesem Fall manuell durch einen menschlichen Nutzer.
- **hybrid**: Neben der automatischen und manuellen Erstellung von Metadaten ist es gelegentlich sinnvoll auf hybride Verfahren zurückzugreifen. Maschinell gewonnene Metadaten werden bei Bedarf manuell verbessert, erweitert oder berichtigt.

Neben den hier aufgeführten Klassifikationen, lassen sich weitere Kriterien für die Kategorisierung von Metadaten, beispielsweise hinsichtlich der Struktur, der Persistenz oder des Beschreibungsumfanges, finden. Eine umfangreiche Aufstellung von Klassifikationen kann [Gill00] entnommen werden.

2.2 Standards zur Beschreibung von Metadaten

Die Maschinenverständlichkeit von Metainformationen setzt einen gewissen Standardisierungsgrad voraus. Die gegenwärtige Metadatendiskussion schließt deshalb, neben der Suche nach notwendigen und geeigneten Ressourcenbeschreibungen, auch die Entwicklung eines Standards zur Beschreibung von Metadaten ein. Ein Standard sollte dabei die Darstellung von Metadaten in einem universellen, flexiblen und maschinenverständlichen

Datenformat ermöglichen. Zudem sollte das Datenformat für den effizienten Einsatz in elektronischen Netzen optimiert sein.

Speziell das W3C macht es sich zur Aufgabe, mit dem *Resource Description Framework* einen geeigneten Metadatenbeschreibungsstandard auszuarbeiten.

Resource Description Framework (RDF)

Das W3C rief 1996 eine Arbeitsgruppe, gebildet aus Netscape, Microsoft, IBM, Nokia, OCLC, Reuters, Vertreter der Digital Library Initiativen und weiteren Mitgliedern, ins Leben. Die Arbeitsgruppe hat die Aufgabe, eine neue einheitliche Spezifikation für die Darstellung, Handhabung und Übertragung von Metainformationen zu erstellen und diese zum Standard zu etablieren. Dazu wurden verschiedene Entwicklungen in Hinblick auf Metadaten, wie MCF, XML-Data, PICS oder Dublin Core aufgegriffen. Eine kurze Beschreibung zu diesen findet sich in Anhang A.1.

Ein erster Entwurf für das Resource Description Framework wurde bereits 1997 vorgelegt. Dieser wurde inzwischen mehrfach diskutiert und überarbeitet.

RDF definiert einen Mechanismus zur Beschreibung von Ressourcen und bietet hierzu eine Metasprache für Metadaten. Durch RDF wird eine Infrastruktur zur Codierung, zum Austausch und zur Wiederverwendung von Metadaten zur Verfügung gestellt. Damit ermöglicht RDF eine Interoperabilität zwischen verschiedenen Anwendungen, die auf einem Austausch von Metadaten beruhen. RDF ist nicht starr, sondern bietet einen erweiterbaren Rahmen für die Behandlung von Metadaten.

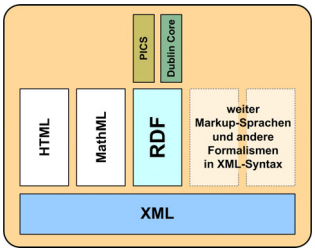

Abbildung 2.1: RDF im XML-Umfeld und RDF-Anwendungsfelder

Die Syntax von RDF kann auf XML basieren. Da XML bereits ein universelles Datenformat darstellt, stellt sich die Frage nach der Notwendigkeit von RDF. Die Antwort liegt in der Maschinenverständlichkeit und lässt sich leicht an folgendem Beispiel demonstrieren:

Die Aussage „*The author of the page is Guntram Hoch*“ kann mit XML verschieden repräsentiert werden (basiert auf [Bern98]):

```
<author>
  <uri>page</uri>
  <name>Guntram Hoch</name>
</author>
```

oder

```
<document href="page">
  <author>Guntram Hoch</author>
</document>
```

oder

```
<document>
  <details>
    <uri>href="page"</uri>
    <author>
```

```
            <name>Guntram Hoch</name>
          </author>
        </details>
      </document>
```

usw.

Alle Repräsentationen sind maschinenlesbar, stellen jedoch unterschiedliche XML-Graphen dar und können so nur mit sehr großem Aufwand semantisch interpretiert werden. RDF definiert bezüglich der Datenstruktur gewissermaßen Einschränkungen, um die am Beispiel gezeigten Mehrdeutigkeiten zu beseitigen.

Da das Resource Description Framework im Rahmen dieser Diplomarbeit eine zentrale Rolle spielt, wird dessen Spezifikation im anschließenden Abschnitt näher vorgestellt.

Ein Überblick zu einigen Anwendungen, bei denen sich RDF bereits etabliert hat, ist in Anhang A.3 zu finden.

2.3 RDF Spezifikation

Die RDF Spezifikation besteht aus zwei grundsätzlichen Teilen. Als theoretische Grundlage dient ein Metadatenmodell, welches in einer XML-basierten Syntax dargestellt werden kann. Dieses wird in der *RDF Model and Syntax Specification* [LaSw99] definiert. Die Semantik der Daten wird über Schemata ausgedrückt. Sie stellen eine Art Klassensystem dar, ähnlich wie man es aus vielen objektorientierten Sprachen kennt. Eine Spezifikation zur Definition der Schemata findet sich in der *RDF Schema Specification* [BrGu00]. Durch die beiden Teile werden zwei im gewissen Sinne gegensätzliche Ziele des RDF verwirklicht: Die Semantik von Objekten ist ausdrückbar und maschinell verwertbar. Ferner wird eine der Dynamik des WWW angemessene Flexibilität und Erweiterbarkeit geboten.

2.3.1 RDF Modell

Grundlage von RDF ist ein Modell in Form eines gerichteten Graphen (*directed labelled graph*), welcher Ressourcen (Texte, Webseiten, Videos etc.) mit Hilfe von Eigenschaften beschreibt. Die Eigenschaften können als Attribute einer Ressource angesehen werden und sind in diesem Sinne als Attribut-Wert-Paar darstellbar.

Das grundlegende Datenmodell von RDF basiert auf drei Objekttypen:

- **Ressourcen** (*resources*) sind Dinge, die durch RDF-Ausdrücke beschrieben werden können. Hierzu gehören elektronische Daten aber auch andere Objekte, die eindeutig mit einer *Uniform Resource Identifier (URI)* [URI01] identifiziert werden können. Die URI stellt eine allgemeine Form der Adressierung dar und umfasst dabei die bekannten *Uniform Resource Locators* (URL) als auch eindeutige, dauerhafte und adressunabhängige Namen für Ressourcen, die *Uniform Resource Names* (URN).
- **Eigenschaften** (*properties*) einer Ressource sind spezielle Aspekte, Charakteristika, Attribute oder Beziehungen. Hierbei wird zwischen dem Typ (*propertyType*) und dessen Wert (*value*) unterschieden. Der Wert kann entweder ein Literal oder eine andere Ressource sein. Jede Eigenschaft hat eine spezielle Bedeutung, Domäne und

Arten von Ressourcen, die sie beschreiben kann. Dieses wird in einem RDF-Schema definiert (siehe Abschnitt 2.3.3).

- **Aussagen** (*statements*) bestehen aus einer Ressource, einer Eigenschaft und einem Wert. Die drei Teile einer Aussage werden auch als Subjekt, Prädikat und Objekt bezeichnet, analog zu den Bestandteilen von Sätzen der natürlichen Sprache (S-P-O).

Eine einfache Aussage, wie zum Beispiel

Michael Ley is the creator of the resource http://dblp.uni-trier.de.

bestehet aus

Subjekt: *http://dblp.uni-trier.de*
Prädikat: *Creator*
Objekt: *Michael Ley*

und lässt sich folgendermaßen in einer graphischen Darstellung des RDF-Modells abbilden:

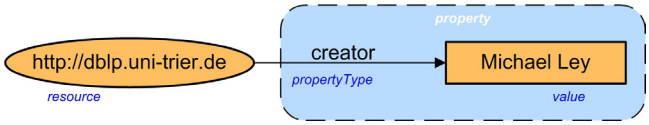

Abbildung 2.2: Einfaches RDF-Statement

Bei der graphischen Notation wird ein Oval für eine Ressource, ein beschrifteter Pfeil für eine Eigenschaft und ein Rechteck für ein Literal verwendet.

Strukturierte Werte

In der Regel ist die Struktur der Metadateninformationen jedoch komplizierter. So könnte beim zuvor genannten Beispiel der Wert der Eigenschaft *creator* nicht nur ein einfacher, sondern ein *strukturierter Wert* sein:

http://dblp.uni-trier.de has creator something and something has name Michael Ley and the email address ley@uni-trier.de.

In RDF übernimmt der strukturierte Wert hierbei die Rolle einer Ressource, die wiederum selbst Eigenschaften besitzt.

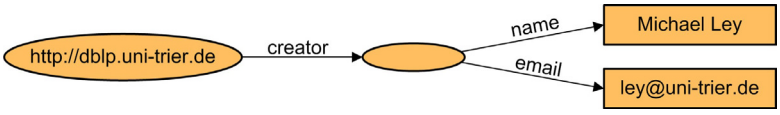

Abbildung 2.3: Strukturierte Werte im RDF-Modell

Dem strukturierten Wert kann eine eindeutige Bezeichnung gegeben werden. In der graphischen Darstellung würde man dazu das leere Oval entsprechend beschriften.

Container

Um einer Eigenschaft mehrere Werte zuzuordnen, werden in der RDF-Spezifikation neben den strukturierten Werten die *Container* definiert. Diese können Ressourcen oder Literale aufnehmen. Es existieren drei Arten von Containern:

- **Bag** ist eine ungeordnete Liste.[1]

- **Sequence** ist eine geordnet Liste.[1]
- **Alternative** ist eine Liste, die Alternativen darstellt.

Container werden in RDF mit Hilfe einer Ressource dargestellt. Diese besitzt die Eigenschaft *rdf:type*, mit der die Art des Containers festgelegt wird. Die einzelnen Listenelemente sind Werte der Eigenschaften *rdf:_1, ... rdf:_n*.

Durch die Verwendung des Containers für eine ungeordnete Liste kann die Aussage

> *The creators of http://itp.de/books/biber-kompakt are Andreas Heuer, Gunter Saake and Kai-Uwe Sattler.*

in RDF, wie in Abbildung 2.4 gezeigt, modelliert werden.

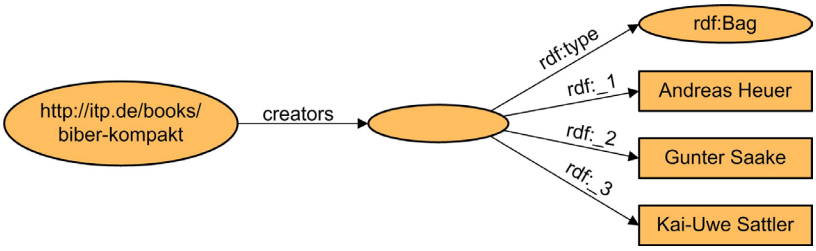

Abbildung 2.4: Container im RDF-Modell

Das RDF-Modell ist unabhängig von der Syntax der Datenrepräsentation. Die vom W3C innerhalb der Spezifikation vorgeschlagene XML-Syntax ist nur eine mögliche Darstellungsform. Beruhend auf dem RDF-Modell und abstrahiert von der Syntax können verschiedene RDF-Ausdrücke miteinander verglichen werden. Zwei Ausdrücke sind äquivalent, wenn deren Modelle identisch sind.

2.3.2 RDF Syntax

Obwohl das Modell Kernpunkt des Resource Description Framework ist, benötigt man für die maschinelle Verarbeitung der Metadaten eine geeignete Syntax. RDF-Dokumente sollen dabei nicht nur von Computern, sondern auch von Menschen gelesen werden können und darüber hinaus leicht über das Netz austauschbar sein. Um den genannten Forderungen gerecht zu werden, schlägt das World Wide Web Consortium XML als Sprache vor. Gründe hierfür sind einerseits die hohe Akzeptanz als auch die Erweiterbarkeit von XML.

Die Beschreibung für das einfache RDF-Modell aus Abbildung 2.2 lautet in der XML-Syntax von RDF:

```
<?xml version="1.0">
<rdf:RDF
  xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://dblp.uni-trier.de"
    <s:creator>Michael Ley</s:creator>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.2: Beschreibung eines RDF-Modells in XML-Syntax [1]

[1] Die Container Bag und Sequence erlauben in der derzeitigen Spezifikation von RDF Duplikate, da hier noch kein Mechanismus festgelegt wurde, der eine Duplikatfreiheit durchsetzen könnte.

Im RDF-Tag werden die Namespaces deklariert, die das Vokabular und die damit verbundenen RDF-Schemata festlegen. Mit *Description* beginnt die eigentliche Beschreibung. Das Attribut *about* kennzeichnet die URI der zu beschreibenden Ressource. Die Eigenschaft sowie deren Wert findet sich im Tag *s:creator*. Wäre der Eigenschaftswert wieder eine Ressource, so könnte mit Hilfe von

```
<s:creator resource="URI-reference"/>
```

auf diese verwiesen werden.

Neben der oben dargestellten *Serialization Syntax* schlägt das W3C die Verwendung einer abgekürzten Syntax, die *Abbreviated Syntax* zur Kodierung von RDF vor. Sie ermöglicht eine kompaktere Form und erlaubt die Einbettung in HTML-Dokumente, da so Browser die RDF-Daten bei der Darstellung komplett ignorieren. Bei der *Abbreviated Syntax* werden die Metadaten in Form von Attributen innerhalb der Tags formuliert:

```
<rdf:RDF
  <rdf:Description about="http://dblp.uni-trier.de"
    s:creator="Miachael Ley" />
</rdf:RDF>
```

Listing 2.3: RDF-Modell in der Abbreviated Syntax

Strukturierte Werte

Die Kodierung von strukturierten Werten kann auf unterschiedliche Art und Weise erfolgen. Es ist möglich die Ressourcenbeschreibungen zu verschachteln oder den strukturierten Wert, der selber wieder eine Ressource darstellt, separat zu beschreiben.

Container

In der XML-Syntax von RDF werden Container durch ein zusätzlicher Tag, welches die Art des Containers festlegt, gekennzeichnet. Bei der Kodierung der einzelnen Elemente muss die Nummerierung nicht unbedingt manuell angegeben werden, sondern kann auch automatisch geschehen, wie im folgenden Beispiel gezeigt:

```
<rdf:RDF
  <rdf:Description about="http://itp.de/books/biber-kompakt">
    <s:creator>
      <rdf:Bag ID="creators">
        <rdf:li>Andreas Heuer</rdf:li>
        <rdf:li>Gunter Saake</rdf:li>
        <rdf:li>Kai-Uwe Sattler</rdf:li>
      </rdf:Bag>
    </s:creator>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.4: Container in XML-Syntax

Werden Aussagen über einen Container verwendet, so können sich diese auf den Container als ganzes beziehen (*about*) oder über die einzelnen Elemente definiert werden (*aboutEach*), was eine abkürzende Schreibweise ermöglicht.

[1] Zur besseren Übersicht werden in den folgenden Beispielen die Namespace-Deklaration und der Prolog für XML weggelassen.

Grundlage für die maschinelle Verarbeitung von RDF-Dokumenten in XML-Syntax sind geeignete Parser. Sie erlauben die syntaktische Analyse von RDF-Dokumenten in XML-Syntax zum Aufbau des darin beschriebenen RDF-Modells und bieten so eine Schnittstellen für die Programmentwicklung. Eine Übersicht der existierenden Parser und deren Bezugsquellen ist in Anhang A.2 aufgeführt. In dieser Arbeit wird der SiRPAC-Parser verwendet.

2.3.3 RDF Schema

Die Nutzung von RDF-Modellen bietet die Möglichkeit, Ressourcen durch Eigenschaften zu beschreiben. Aussagen hinsichtlich der Semantik werden hierbei jedoch noch nicht gemacht. Ein RDF-Schema definiert nun ein Vokabular, welches aus semantischen Eigenschaftstypen besteht. Es legt für jede Eigenschaft fest, welche Bedeutung sie hat, welche Eigenschaftswerte erlaubt sind, welche Arten von Ressourcen diese Eigenschaften besitzen und welche Beziehungen sie zu anderen Eigenschaften haben. Die *RDF Schema Specification* schreibt nun nicht ein Schema vor, sondern spezifiziert, wie RDF-Schemata definiert werden (*Schema Definition Language)*. Dieses stellt eine flexible Lösung dar, da so die Möglichkeit geboten ist, unterschiedliche Vokabulare in Abhängigkeit der Bedürfnisse und Anforderungen aufzustellen. Um zwischen verschiedenen Vokabularen zu unterscheiden, werden diese innerhalb eines Modells durch die Benutzung von XML Namespaces identifiziert, wie bereits in Listing 2.2 gezeigt wurde. So kann einer Eigenschaft eine engere oder weitergehende Bedeutung zugeordnet werden.

Ähnlich dem Klassensystem in objektorientierten Sprachen existieren für *RDF Schema* Klassen und Klassenhierarchien, jedoch mit einem grundlegenden Unterschied: In der Klassendefinition objektorientierter Sprachen wird festgelegt, welche Eigenschaften (Attribute) die Instanzen (Objekte) der Klasse besitzen. In einem RDF-Schema erfolgt dieses in umgekehrter Richtung. Bei jeder Eigenschaft wird festgelegt, für welche Klassen von Ressourcen sie anwendbar sind.

Der objektorientierte Ansatz bietet Möglichkeiten zur Weiterentwicklung von RDF-Vokabularen zwischen denen dann gewisse semantische Beziehungen bestehen. Konzepte hierfür sind die Bildung von Unterklassen sowie die Spezialisierung von Eigenschaften.

Um RDF-Schemata automatisch interpretieren zu können, werden diese wieder in RDF beschrieben. Selbst die *Schema Definition Language* ist in RDF definiert. Deren Vokabular wird für gewöhnlich über den Bezeichner *rdfs* identifiziert. In der Definition von *RDF Schema* sind Klassen und Eigenschaften festgelegt. Die grundlegendsten werden im Folgenden kurz aufgeführt:

classes

- *rdfs:Resource* stellt die Klasse aller Ressourcen dar. Alle Objekte, die mit RDF beschrieben werden, sind Instanzen dieser Klasse.
- *rdfs:Property* ist die Klasse aller Eigenschaften und Unterklasse von *rdfs:Resource*.
- *rdfs:Class* kommt dem generischen Konzept eines Typs gleich, analog dem Konzept der Klasse in objektorientierten Sprachen. Alle Klassen sind Instanzen des Typs *rdfs:Class*, auch *rdfs:Class* selbst.

properties

- *rdf:type* ordnet einer Ressource eine bestimmte Klasse zu. Sie ist somit Instanz der Klasse. Eine Ressource kann Instanz mehrerer Klassen sein.
- *rdfs:subclassOf* spezifiziert eine Teilmengenbeziehung zwischen Klassen und ermöglicht so, den Aufbau einer Klassenhierarchie. Die Relation ist transitiv.
- *rdfs:propertyOf* wird zur Spezialisierung von Eigenschaften verwendet. Eine Eigenschaft kann von einer oder mehreren anderen Eigenschaften abgeleitet sein.

constraintProperties stellen eine Unterklasse von *properties* bereit, die eine Zuordnung von Beschränkungen auf Klassen und Eigenschaften ermöglicht.

- *rdfs:range* legt die gültigen Werte für eine Eigenschaft fest.
- *rdfs:domain* gibt für eine Eigenschaft an, bei welchen Klassen sie angewendet werden kann.

Zum besseren Verständnis kann dem Anhang A.4 ein Beispiel für ein RDF-Schema entnommen werden.

2.4 Zusammenfassung

RDF ermöglicht die Darstellung von Metainformationen unabhängig vom Inhalt oder Format eines Dokumentes. Dieses gestattet die Unterstützung neuerer als auch veralteter Dokumentformate. Neben Dokumenten können auch andere Objekte, die über eine URI identifizierbar sind, beschrieben werden. Metadatenbeschreibungen können sowohl von Personen gelesen als auch vom Computer analysiert werden. RDF-Modelle erlauben eine mächtige Ausdruckskraft, obwohl deren Struktur sehr einfach ist.

Die Spezifikation für RDF-Modelle lässt jedoch eine eindeutige Aussage darüber vermissen, ob der Gebrauch von Statements mit gleichem Subjekt und Prädikat aber unterschiedlichem Objekt zulässig ist, was dem Gebrauch der ungeordneten Liste gleichkommen würde. Bei den Containern kommt das Fehlen der „natürlichen Menge“ hinzu.

RDF-Schemata beschreiben die Semantik und können zur Validierung von Modellen eingesetzt werden. Programme zur Bearbeitung von Metadaten, sind durch Schemainformationen in der Lage, Vorschläge für anwendbare Eigenschaften bzw. für sinnvolle Eigenschaftswerte zu generieren.

RDF ermöglicht die gleichzeitige Verwendung von mehreren Metadatenschemata, deren Definition Metadaten-Gemeinschaften, wie zum Beispiel der Dublin Core Metadata Inititative, überlassen ist. Dabei wird eingestanden, dass es kein allumfassendes Schema gibt, welches die Anforderungen aller Anwendungen erfüllt. Werden unterschiedliche RDF-Schemata von verschiedenen Gemeinschaften entwickelt, so stellt die Abbildung von semantischen Beziehungen zwischen den Vokabularen noch ein Problem dar. Konzepte wie *rdf:subclassOf* oder *rdf:subPropertyOf* bieten derzeit dafür nur Ansatzpunkte. Neben der Bildung von Spezialisierungen bei Klassen und Eigenschaften fehlen Konzepte zur Generalisierung, die beispielsweise die Definition ermöglichen, dass ein Eigenschaftswert vom Typ *X* oder *Y* ist,

selbst wenn diese nicht von einer gemeinsamen Oberklasse (außer *rdfs:Ressource*) abgeleitet sind.

Mit der *Schema Definition Language* werden grundlegende Konzepte zur Definition von RDF-Schemata bereitgestellt. Für die Modellierung komplexer Schemata sind diese jedoch nicht ausreichend. Mit der derzeitigen Version der *RDF Schema Specification* ist es beispielsweise nicht möglich, bei der Verwendung von Containern im Modell, dessen Typ bereits im Schema festzulegen. Kardinalitäten existieren ebenso nicht. Die Spezifikation erlaubt zudem nur die Nutzung eines einzigen Datentyps (*rdfs:Literal*).

Die momentan aktuelle *RDF Schema Specification* ist noch nicht abgeschlossen, so dass voraussichtlich der eine oder andere Mechanismus noch Ergänzungen finden wird.

Erweiterungen bieten bereits Sprachen wie *OIL*[1] [FHH+00] und *DAML*[2] [HeGu01]. Sie bauen auf *RDF Schema* auf und stellen Modellierungskonzepte zur Definition von Ontologien bereit. Die Modellierungssprache DAML (DARPA Agent Markup Language) erweitert *RDF Schema* beispielsweise um Kardinalitäten, XML Schema Datentypen und Boolesche Kombinationen bei der Klassendefinition (vergleiche [Haas01]).

Das in diesem Kapitel vorgestellte Resource Description Framework bildet die Grundlage zur Steuerung der Request-Verarbeitung in der XPEA-Architektur. Mit einem RDF-Modell lassen sich die hierzu notwendigen Metainformationen geeignet beschreiben. RDF-Modelle zeichnen sich durch eine hohe Ausdrucksstärke und Erweiterbarkeit aus, wodurch umfangreiche Möglichkeiten für weitere Entwicklungen geboten werden.

Schwerpunkt dieser Arbeit ist die Entwicklung eines Ansatzes für eine universelle Metadatenverwaltung zur Steuerung von Transformations- und Daten-Analyse-Prozessen. Dabei wird eine möglichst enge Kopplung von Data Warehouse-Funktionalität (OLAP) mit einer ausgewählten XML-Framework-Architektur angestrebt. Ziel ist ein erweiterter Architektur-Vorschlag, der die Grundlage für zukünftige offene und erweiterbare Multimedia-Content-Management-Systeme bilden könnte.

Im Folgenden wird als Grundlage für die weiterführenden Kapitel neben der Einführung in die OLAP-Thematik insbesondere auf die Einbindung des SQL-99-Standards eingegangen und die Leistungsfähigkeit der kommerziellen Systeme IBM/DB2 und Oracle in diesem Zusammenhang untersucht.

[1] OIL. http://www.ontoknowledge.org/oil/index.html

[2] DAPRA Agent Markup Language. http://www.daml.org

3. OLAP

OLAP (On-line Analytical Processing) ist ein Analyseansatz, der die dynamische, multidimensionale Analyse von Daten bezeichnet, mit dem Ziel, neue oder unerwartete Beziehungen zwischen den Variablen zu erkennen.

Mit dieser Definition hat E.F. Codd 1993 in [CoCS93] den Begriff OLAP geprägt. Konzipiert ist OLAP als Antwort auf den steigenden Analysebedarf bei wachsenden Mengen und zunehmender Komplexität gespeicherter Daten. OLAP findet in Gebieten Anwendung, bei denen entscheidungsunterstützende Techniken gefragt sind. Um die Entscheidungen auf einer soliden Basis zu treffen, ist es notwendig, den „Stand der Dinge“ zu kennen. OLAP-Systeme bieten die Möglichkeit innerhalb einer Analyse komplexe Datenbankanfragen einfach und effizient zu stellen. Die Durchführung von Analysen erfolgt als ein interaktiver kreativer Prozess, in dessen Verlauf Hypothesen entwickelt und überprüft werden. Grundlage von OLAP ist ein konzeptuelles multidimensionales Datenmodell. Daraus resultiert eine multidimensionale Sichtweise der Daten.

In diesem Kapitel wird eine Einführung in das Themengebiet OLAP gegeben. Hierzu erfolgt zunächst eine begriffliche Abgrenzung zu OLTP, Data Warehousing und Data Mining. Daran anschließend werden die Codd’schen Regeln und die FASMI-Definition vorgestellt. Ausgehend von einer Darstellung des multidimensionalen Datenmodells werden die verschiedenen OLAP-Operatoren und Speicherungsformen betrachtet. Zentraler Focus des Kapitels ist die Realisierung multidimensionaler Anfragen bei relationaler Speicherung.

3.1 Abgrenzung zu OLTP, Data Warehousing und Data Mining

Die Begriffe OLTP, Data Warehousing und Data Mining werden oft in einem Atemzug mit OLAP genannt. Für ein besseres Verständnis und zur Unterscheidung der Begriffe werden diese hier kurz vorgestellt und zueinander in Beziehung gesetzt.

OLTP

Transaktionale bzw. operative Systeme, oft auch als *On-line Transaction Processing* (OLTP) bezeichnet, werden im Unterschied zu OLAP-Anwendungen nicht zur Entscheidungsunterstützung, sondern zur Bewältigung des Tagesgeschäftes eingesetzt, was hohe Anforderungen an ihre Verfügbarkeit stellt. Beim Einsatz solcher Systeme finden relationale Datenbanken, aufgrund des fortgeschrittenen Entwicklungsstandes, eine hohe Akzeptanz.

Ausgehend von den unterschiedlichen Einsatzgebieten der OLTP- und OLAP-Systeme ergeben sich hinsichtlich diverser Kriterien Unterscheidungsmerkmale, von denen einige in Tabelle 3.1 aufgeführt werden.

Kriterium	*transaktional, operativ - OLTP*	*entscheidungsunterstützend - OLAP*
Datenbestand	detaillierte Daten	verdichtete, aufbereitete Daten
	aktuelle Daten	historische, aktuelle und projizierte Daten
	transaktionsgesteuerte Verarbeitung	analysegesteuerte Verarbeitung
	hohe Anforderungen an Verfügbarkeit	geringe Anforderungen an Verfügbarkeit
	statische Struktur	flexible Struktur
	Kleine Datenmengen pro Prozess	große Datenmengen pro Prozess
Datenmodell	i.d.R. relationales, objektorientiertes oder objektrelationales Datenmodell mit weitgehend normalisiertem Datenbankschema	multidimensionales Datenmodell
Implementierung	hohe Anforderungen an: ❑ Ausfallsicherheit, ❑ Transaktionsraten, ❑ Sicherheitsmechanismen, ❑ Mehrbenutzerbetrieb, ❑ Skalierbarkeit, etc.	hohe Anforderungen an: ❑ effiziente Verarbeitung mehrdimensionaler Datenstrukturen, ❑ schnelle Antwortzeiten für multidimensionale Anfragen

Tabelle 3.1: Unterscheidungsmerkmale zwischen OLTP- und OLAP-Systemen

Data Warehousing

Ein *Data Warehouse* stellt eine einheitlich strukturierte, zentrale Sammelstelle für Unternehmensdaten dar. Darin werden unternehmensübergreifend die Daten aus den operativen Vorsystemen zusammengeführt, integriert und für Analysezwecke aufbereitet. Obwohl der Begriff Data Warehouse eine zentrale Datenbank bezeichnet, liegt der eigentliche Schwerpunkt jedoch auf dem Prozess der Datensammlung und Konsolidierung. Um den Prozess mehr in den Vordergrund zu stellen, spricht man daher auch oft vom *Data Warehousing*.

Das Data Warehousing bezieht in den Prozess der Datenhaltung das komplette Unternehmen ein. Die Datenbasis von OLAP-Analysen ist hingegen eher fachspezifisch und umfasst dem gegenüber nur ein eingeschränktes Datenvolumen. Ein *Data Mart* bildet einen inhaltlich beschränkten Focus des Unternehmens oder einer Abteilung als Teilsicht eines Data Warehouse ab. Er stellt einen mehrdimensionalen strukturierten Informationsspeicher dar, der somit auch als Grundlage für OLAP-Analysen verwendet werden kann.

OLAP-Systeme können als Bestandteil eines Data Warehouse betrachtet werden. Sie sind auf die Analyse von Data Warehouse zugeschnitten, ermöglichen eine einfache Ausführung komplexer Datenbankanfragen und erlauben so den schnellen Transport vorhandener Datenbestände an den Endanwender.

Data Mining

Data Mining ist neben OLAP ein weiterer Analyseansatz, mit dem Ziel, interessante und nützliche Muster beziehungsweise Regeln in großen Datenbanken zu finden. Gegenüber von *Knowledge Discovery in Databases* (KDD), einem nichttrivialen Prozess der Identifikation und Verifikation von gültigen, neuen, potentiell nützlichen und verständlichen Mustern in Datenbeständen [FaPS96] mit Hilfe automatisierter Verfahren, stellt Data Mining die Suche von Mustern im KDD-Prozess dar. Data Mining umfasst den gesamten Prozess von der Bereitstellung der Daten bis zur Anwendung der Kenntnisse und setzt sich aus verschiedenen

Einzelschritten zusammen, die nicht unbedingt sequentiell durchlaufen werden, sondern auch Rücksprünge aufweisen können.

Zum Finden von Mustern in den Datenquellen werden verschiedene Verfahren und Techniken eingesetzt, so zum Beispiel statistische Verfahren, Clusterverfahren, Entscheidungsbäume, interaktive Analysen mittels Visualisierungstechniken und Neuronale Netze. Gesuchte Muster sind beispielsweise [Petr97]: Verbindungsmuster, Sequenzen, Cluster, Abweichungen oder Klassifikationsregeln. Welche Muster interessant sind, hängt dabei von dem Szenario und der Aufgabenstellung ab.

3.2 Codd'sche Regeln und FASMI-Definition

Die Codd'schen Regeln [CoCS93], vorgestellt 1993 anlässlich der Präsentation von Essbase, einem multidimensionalen Datenbanksystem von Arbor Software[1], legen fest, welche Eigenschaften von einem Produkt erfüllt werden müssen, um vollständig dem OLAP-Bereich zugeordnet werden zu können. Um der Kritik einer herstellerabhängigen Definition zu entgegnen und den wachsenden Anforderungen an die multidimensionale Analyse gerecht zu werden, erweiterte E. F. Codd 1995 den Kriterienkatalog um sechs weitere Punkte.

Ein alternatives Klassifikationsmittel bietet die FASMI-Definition [PeCr95], entwickelt von den Herausgebern des OLAP-Reports[2].

Da die Codd'schen Regeln und die FASMI-Definition bereits an verschiedenen Stellen ([BaGü01], [ScBM99], [Clau98]) ausführlich publiziert wurden, wird hier auf eine detaillierte Beschreibung verzichtet. Die einzelnen Punkte der Definitionen sind in Anhang B.1 zu finden.

Generell erlauben die Codd'schen Regeln und die FASMI-Definition eher eine technische Einordnung von Software hinsichtlich der gegebenen OLAP-Funktionalität, als eine Erläuterung des OLAP-Begriffes. Für die Forschungs- und Entwicklungsarbeit im Bereich OLAP stellen sie jedoch eine wesentliche Grundlage dar.

3.3 Multidimensionales Datenmodell

Daten sowie deren Beziehungen zueinander werden für die OLAP-Analyse in multidimensionalen Strukturen dargestellt. Grundlage ist das multidimensionale Datenmodell. Über das Datenmodell existieren teilweise unterschiedliche Auffassungen ([AgGS95], [LeRT96], [LiWa96], [BaPT97], [GyLa97], [GaTo98], [Lehn98], [Vass98], [BaGü01], [VaSe01]). In diesem Abschnitt werden basierend auf [BaGü01] die Elemente und Konzepte des multidimensionalen Datenmodells näher erläutert.

Measures

Measures, auch als Fakten oder Kenngrößen bezeichnet, sind die quantitativen Kennzahlen, die ausgewertet werden sollen. Sie beruhen auf numerischen Werten und beschreiben

[1] jetzt Hyperion: http://hyperion.com

[2] http://www.olapreport.com

gewöhnlich betriebswirtschaftliche Sachverhalte. Typische Beispiele sind Umsatz, Kosten und Stückzahlen. Measures werden durch *Dimensionen* beschrieben und lassen sich durch unterschiedliche Verdichtungsstufen entlang der Dimensionen in verschiedenen Detaillierungsgraden auswerten. Dabei können sie durch einfache Aggregation der atomaren Datenwerte (z.B. Stückzahl = Anzahl der verkaufen Produkte) oder auch aus anderen Measures (z.B. Gewinn = Umsatz – Kosten) berechnet werden.

Dimensionen

Grundprinzip von OLAP ist eine Betrachtung von Daten aus unterschiedlichen Blickwickeln. Innerhalb des Datenmodells definiert eine Dimension eine Auswertungssicht eines Anwendungsbereiches. Der Datenraum (OLAP-Würfel) wird hierzu durch Dimensionen eindeutig, orthogonal strukturiert. So können beispielsweise Umsätze nach Produkt, Zeit und Geographie, den drei Dimensionen des Datenraums, strukturiert werden. Wie in Abbildung 3.1 dargestellt, spannen diese Dimensionen einen dreidimensionalen Würfel auf, in dessen Zellen die Datenwerte gespeichert sind.

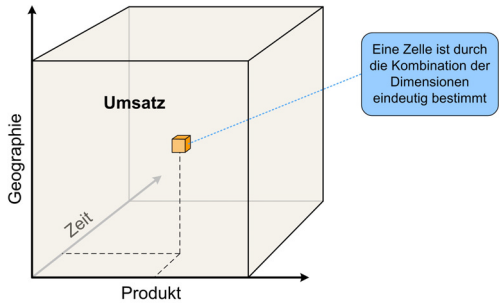

Abbildung 3.1: Dimensionen eines OLAP-Würfels

Eine Dimension besteht aus mindestens zwei Dimensionselementen. In der so genannten *Klassifikationshierarchie* bilden sie die Blätter eines Baumes. Eine Beschreibung der Klassifikationshierarchie erfolgt über ein *Klassifikationsschema*, bestehend aus den *Klassifikationsstufen* (Verdichtungsstufen oder auch Hierarchieebenen), die die *Klassifikationsknoten* (Knoten des Baumes) auf einer Ebene des Baumes repräsentieren. Im Datenraum werden auf der jeweils untersten Klassifikationsstufe (an den Blättern des Baumes) die atomaren Datenwerte gespeichert. Für die Knoten höherer Klassifikationsstufen lassen sich die Werte durch Aggregation ableiten.

Im Allgemeinen besitzen Dimensionen einfache Klassifikationshierarchien, wie beispielsweise die Produktdimension in Abbildung 3.2. Produkte lassen sich in Produktgruppe einteilen und diese wiederum in Produktkategorien.

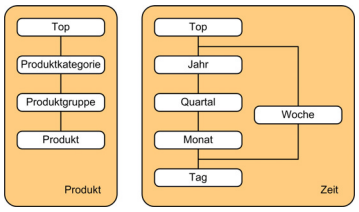

Abbildung 3.2: Klassifikationsschema mit einfacher und paralleler Hierarchie [1]

Nicht immer weisen Dimensionen einfache Hierarchien auf, wie zum Beispiel bei der Dimension Zeit. Das Beispiel in Abbildung 3.2 zeigt eine Verbindung von Tag, Monat, Quartal und Jahr sowie eine parallel Verbindung von Tag und Woche. Man spricht in diesem Fall von so genannten parallelen oder auch multiplen Hierarchien.

Klassifikationsschema, Konsolidierungspfade

Ein Klassifikationsschema dient der Beschreibung einer Klassifikationshierarchie und kennzeichnet die Abhängigkeiten zwischen den Klassifikationsstufen. Bezüglich $\rightarrow$ wird ein Klassifikationsschema als halbgeordnete Menge von Klassifikationsstufen definiert: [2]

$$(\{D.K_0, \ldots, D.K_n\}, \rightarrow)$$

Ein Konsolidierungspfad entspricht einer vollgeordneten Teilmenge des Klassifikationsschemas. Das Klassifikationsschema der Dimension Zeit besteht beispielsweise aus den beiden Konsolidierungspfaden:

$$Z.Tag \rightarrow Z.Monat \rightarrow Z.Quartal \rightarrow Z.Jahr$$

$$Z.Tag \rightarrow Z.Woche$$

Klassifikationshierarchie

Eine Klassifikationshierarchie bezüglich eines Konsolidierungspfades ist ein balancierter Baum, bestehend aus dem Wurzelknoten *ALL* und den Klassifikationsknoten der einzelnen Klassifikationsstufen.

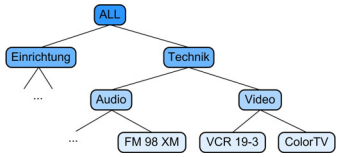

Abbildung 3.3: Ausschnitt einer Klassifikationshierarchie

Würfel

Ein Würfel (engl. cube) besteht aus Datenzellen. Diese beinhalten ein oder mehrere Measures, die als Funktionen der Dimensionen betrachtet werden können. Dabei entsprechen die

[1] In der Dimension Zeit ist eine Zuordnung von Woche zu Jahr konzeptuell unsauber, da ein Jahreswechsel nicht unbedingt von Sonntag zu Montag erfolgt und so eine Woche nicht eindeutig einem Jahr zugeordnet werden kann.

[2] analog den funktionalen Abhängigkeiten

Kenngrößen auf detailliertester Ebene den atomaren Werten und auf darüber liegenden Ebenen deren Aggregationen. Die Kanten des Würfels werden durch die Dimensionen aufgespannt, wobei die Anzahl der Dimensionen als Dimensionalität bezeichnet wird. Besitzt ein Würfel drei Dimensionen, so ergibt sich, wie in Abbildung 3.4 dargestellt, ein Würfel im dreidimensionalen Raum. In der Regel werden beim Aufbau eines OLAP-Würfels jedoch mehr Dimensionen verwendet. Man spricht dann auch von einem Hypercube.

Ein konkreter Würfel lässt sich durch ein multidimensionales Schema beschreiben.

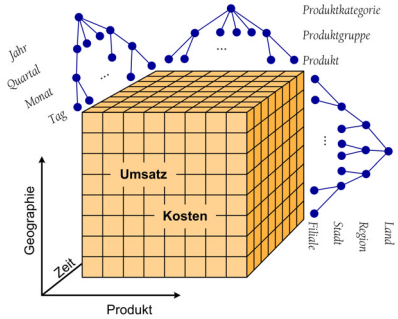

Abbildung 3.4: OLAP-Würfel mit Klassifikationsschemata und -hierarchien

Durch Verwendung von OLAP-Operatoren, bieten OLAP-Systeme dem Nutzer individuelle und flexible Sichtweisen auf den Würfel. Die Operatoren werden im folgenden Abschnitt erklärt.

3.4 OLAP-Operatoren

Die OLAP-Analyse ist ein dynamischer Prozess, bei dem mit Hilfe von multidimensionalen Operatoren durch die Daten navigiert werden kann. Sie erlauben die Hierarchieebenen der Dimensionen zu wechseln, die Isolierung einzelner Sichten aus dem Würfel sowie eine Analyse der Daten aus verschiedenen Perspektiven.

Die einzelnen OLAP-Operatoren werden nun kurz vorgestellt:

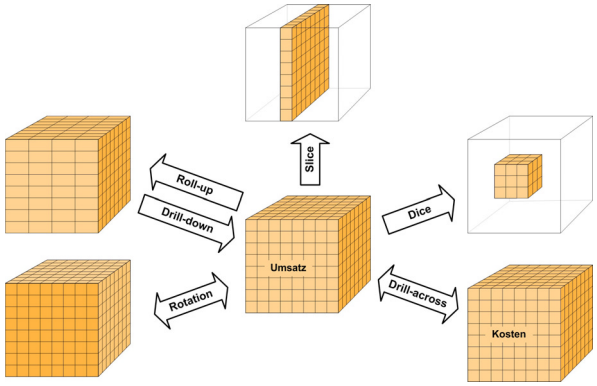

Abbildung 3.5: OLAP-Operatoren

Ein **Roll-up** erlaubt eine abstraktere Sicht auf die Daten, indem entlang der Konsolidierungspfade stärker aggregiert wird, während das Pendant dazu, ein **Drill-down**, ausgehend von den verdichteten Daten auf eine detailliertere Ebene navigiert. Enthält der Würfel mehrere Measures, so kann mit einem **Drill-across** von einer Kenngröße zu einer anderen gewechselt werden. Zur Erzeugung individueller Sichten realisiert ein **Slice** das „Herausschneiden einer Scheibe“ aus dem Würfel. Die Operation **Dice** wird in der Literatur unterschiedlich interpretiert. Auf der einen Seite wird damit das Drehen des Würfels verbunden ([ScBM99], [HeSa99]) und auf der anderen Seite ein „Herausschneiden eines Teilwürfels“ ([BaGü01], [SaCo00], [Lust99]). In dieser Arbeit findet die zuletzt genannte Interpretation Anwendung. Das Drehen des Würfels durch Vertauschen der Achsen wird durch die Operation **Rotation** (auch **Pivotierung** genannt) ermöglicht.

3.5 Speicherkonzepte für multidimensionale Daten

Im Abschnitt 3.3 wurde das für die OLAP-Analyse zu Grunde liegende, multidimensionale Datenmodell vorgestellt. Die konzeptuell multidimensionale Struktur der Daten bedeutet jedoch nicht zwingend, dass die physische Speicherung der atomaren und/oder aggregierten Daten ebenfalls in einer multidimensionalen Form erfolgen muss. Konzepte zur Speicherung eines OLAP-Würfels sind:

- **ROLAP** (relational)

  Werden die Daten in klassischen relationalen Datenbanksystemen verwaltet, so spricht man von Relationalem OLAP (ROLAP). Die Multidimensionalität wird durch entsprechende Modellierungstechniken auf zweidimensionale Tabellen abgebildet.

- **MOLAP** (multidimensional)

  Beim Multidimensionalem OLAP (MOLAP) liegen die Daten hingegen in einer multidimensionalen Speicherstruktur vor.

- **HOLAP** (hybrid)

  Eine Kombination der beiden Formen ist das Hybride OLAP (HOLAP). Detaillierte Daten werden dabei üblicherweise relational verwaltet, wo hingegen häufig benötigte Daten mit einem hohen Aggregationsniveau primär multidimensional gespeichert werden.

Bei der im Rahmen dieser Diplomarbeit betrachteten XML-Framework-Architektur wird von einer relationalen Speicherung der Daten ausgegangen. Zur Abbildung der multidimensionalen Konstrukte in einem relationalen Datenbankmodell existieren verschiedene Möglichkeiten. Die beiden gebräuchlichsten Formen sind das *Star-Schema* und das *Snowflake-Schema*.

Star-Schema

Das Star-Schema [KRRT98] besteht aus einer zentralen Faktentabelle mit den Kenngrößen und Tabellen für die Dimensionen. Einer Dimension ist je eine Tabelle zugeordnet, deren Attribute aus den jeweiligen Klassifikationsstufen bestehen. Neben den Kenngrößen beinhaltet die Faktentabelle die Fremdschlüssel der Dimensionstabellen, die zusammen

einen Primärschlüssel bilden. Durch die Schlüsselbedingungen wird sichergestellt, dass den Kenngrößen existierende Dimensionswerte zugeordnet werden.

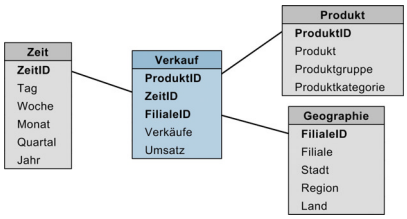

Abbildung 3.6: Star-Schema

Snowflake-Schema

Durch Normalisierung der Dimensionstabellen beim Star-Schema erhält man das Snowflake-Schema [Stan95]. Für jede Klassifikationsstufe einer Dimension wird darin eine separate Tabelle angelegt. Diese beinhaltet neben einer ID der jeweiligen Klassifikationsstufe und den beschreibenden Attribut(en) einen zusätzlichen Fremdschlüssel der benachbarten höheren Klassifikationsstufe.

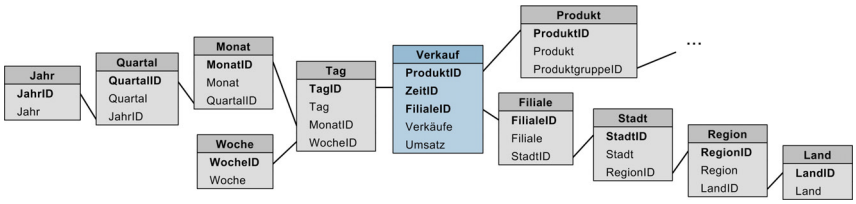

Abbildung 3.7: Snowflake-Schema

Neben den beiden hier vorgestellten Repräsentationsformen sind weitere zu finden, wie zum Beispiel das *Starflake-Schema* [AnMu97] (Mischform aus Star- und Snowflake-Schema), das *Galaxy-Schema* [KRRT98] (mehrere Faktentabellen) oder das *Fact-Constellation-Schema* [KRRT98] (Auslagerung aggregierter Daten in separate Faktentabellen), die hier jedoch nicht weiter betrachtet werden sollen.

Welche der Formen am besten zur Abbildung geeignet ist, hängt stark von den Daten (Datenvolumen, Änderungshäufigkeit) und den Anfragecharakteristiken ab. Generell sollten Entitäten, auf die sehr häufig zugegriffen wird, denormalisiert, also mit der bewussten Inkaufnahme von Redundanzen, gespeichert werden.

3.6 Multidimensionale Anfragen bei einer relationalen Speicherung

Relationale Datenbankmanagementsysteme speichern und verwalten Daten in Relationen. Das Ergebnis von Anfragen sind wiederum Relationen. Eine Sicht auf die Daten ist daher zweidimensional. OLAP hingegen basiert auf einem multidimensionalen Datenmodell, so dass eine zweidimensionale Sicht für OLAP-Anfragen nicht ausreichend ist. Ein Ausweg bietet die direkte multidimensionale Speicherung (siehe MOLAP) mit multidimensionalen Speicherstrukturen. OLAP-Operationen können ohne Umwege auf diesen Strukturen

arbeiten. Aufgrund der technischen Reife werden jedoch häufiger relationale Datenbanksysteme zur Verwaltung der Daten verwendet. Für die Nutzung von OLAP-Funktionalitäten muss eine Schnittstelle zur Abbildung vom Datenmodell des relationalen Datenbanksystems ins multidimensionale Datenmodell und umgekehrt geschaffen werden. Wie eine relationale Speicherung von multidimensionalen Daten erfolgen kann, wurde bereits im vorherigen Abschnitt erläutert. In diesem Abschnitt wird dargelegt, wie multidimensionalen Anfragen mit Hilfe von relationalen umgesetzt werden können. Bei den verwendeten Beispielen wird dabei von dem in Anhang B.2 abgebildeten Tabellen in Form eines Star-Schemas ausgegangen.

Die Umsetzung einer multidimensionalen Abfrage in SQL hängt von der gewählten Abbildungsvariante des multidimensionalen in das relationale Datenmodell ab. In der Regel stellt diese eine dimensionale Reduzierung dar, die aus

- n+1 Verbunden zwischen den n Dimensionen und der Faktentabelle,
- Restriktionen auf den Dimensionen und
- Aggregationen entlang der weggelassenen Dimensionen bestehen.

Das folgende Beispiel zeigt eine entsprechende SQL-Anfrage:

```
SELECT land, monat, produkt, SUM(anzahl)
FROM Filiale, Zeit, Produkt, Verkauf
WHERE Verkauf.filialeID = Filiale.filialeID AND
      Verkauf.tagID = Zeit.tagID AND
      Verkauf.produktID = Produkt.produktID AND
      Jahr = 2000
GROUP BY land, monat, produkt
```

Listing 3.1: *GROUP BY*-Anfrage in SQL

Land	monat	produkt	SUM(anzahl)
Deutschland	11	ColorTV	2
Deutschland	11	Deckenleute Sonja	4
Deutschland	11	FM 98 XM	13
Deutschland	11	VCR 19-3	21
Deutschland	12	FM 98 XM	16
Deutschland	12	VCR 19-3	15
USA	11	ColorTV	34
USA	11	VCR 19-3	11

Tabelle 3.2: Ergebnis der *GROUP BY*-Anfrage

Ergebnis der Anfrage ist die Anzahl der verkauften Produkte pro Monat und Land im Jahr 2001. Neben der im Beispiel gezeigten Summenfunktion sind auch andere Aggregatsfunktionen wie *COUNT( )*, *MIN( )*, *MAX( )* oder *AVG( )* möglich. Angelehnt an die Abbildungsvariante werden die in der Anfrage verwendeten Mehrfachverbunde auch *Star-Joins* genannt, für deren Bearbeitung verschiedene Möglichkeiten der Optimierung existieren.

Die Unterstützung durch den *GROUP BY*-Operator in SQL und Verbundoptimierungen ist zur Gewährleistung geeigneter OLAP-Funktionalität nicht ausreichend. Wichtig für statistische Auswertungen sind die Berechnungen von Teil- und Gesamtsummen. Für deren Berechnung wären eine Vielzahl von *SELECT*-Anfragen nötig, bei einem n dimensionalen Würfel sind das 2^n, die dann jeweils über *UNION* miteinander vereinigt werden. Eine Anfrage und dessen Ergebnis sehen für das Beispiel folgendermaßen aus:

```
  ( SELECT land, jahr, produktKategorie, SUM(anzahl)
    FROM Filiale, Zeit, Produkt, Verkauf
    WHERE  ...
    GROUP BY land, jahr, produktKategorie)
UNION
  ( SELECT NULL, jahr, produktKategorie, SUM(anzahl)
    FROM ...
    WHERE  ...
    GROUP BY jahr, produktKategorie)
UNION
  ( SELECT land, NULL, produktKategorie, SUM(anzahl)
    FROM ...
    WHERE  ...
    GROUP BY land, produktKategorie)
UNION

  ...

UNION
  ( SELECT NULL, NULL, NULL, SUM(anzahl)
    FROM ...
    WHERE  ...)
```

Listing 3.2: Multidimensionale Anfrage in SQL mit mehren Vereinigungen

Land	jahr	produktKategorie	SUM(anzahl)
Deutschland	2000	Einrichtung	4
Deutschland	2000	Technik	67
Deutschland	2000	NULL	71
Deutschland	2001	Einrichtung	1
Deutschland	2001	Technik	36
Deutschland	2001	NULL	37
Deutschland	NULL	Einrichtung	5
Deutschland	NULL	Technik	103
Deutschland	NULL	NULL	101
…			
USA	2000	Einrichtung	0
…			
NULL	2000	Einrichtung	4
…			
NULL	NULL	NULL	187

Tabelle 3.3: Ergebnisrelation der Anfrage aus Listing 2.3

Die Abbildung 3.8 veranschaulicht die Ergebnisrelation in Form einer Kreuztabelle.

Verkäufe		Einrichtung	Technik	Σ
2000	Deutschland	4	67	71
	USA	0	45	45
	Σ	4	112	116
2001	Deutschland	1	36	37
	USA	22	12	34
	Σ	23	48	71
Σ		27	160	187

Abbildung 3.8: Multidimensionale Sicht in Form einer Kreuztabelle

Der SQL-Standard (SQL92) erlaubt keine Verwendung von Funktionen in Verbindung mit dem *GROUP BY*-Operator. Diese werden beispielsweise benötigt, wenn die Zeitangabe nicht durch mehrere Attribute repräsentiert, sondern in einem einzigen Attribut vom Type *date* gespeichert wird. Für eine Aggregation entlang der Klassifikationsstufe *jahr* ist nun eine

Funktion *YEAR(zeit)* erforderlich. Eine Anfrage würde folgendermaßen aussehen, jedoch in der Regel nicht unterstützt werden:

```
SELECT land, YEAR(zeit), produkt, SUM(anzahl)
FROM Filiale, Produkt, Verkauf
WHERE  Verkauf.filialeID = Filiale.filialeID AND
       Verkauf.produktID = Produkt.produktID
GROUP BY land, YEAR(zeit), produkt
```

Listing 3.3: SQL-Statement mit einer Funktion innerhalb der Group by-Klausel

Ein Ausweg lässt sich in der Nutzung von geschachtelten Anfragen finden:

```
SELECT land, jahr, produkt, SUM(anzahl)
FROM ( SELECT  land,
               YEAR(zeit) AS jahr,
               produkt,
               anzahl
        FROM Filiale, Produkt, Verkauf
        WHERE  Verkauf.filialeID = Filiale.filialeID AND
               Verkauf.produktID = Produkt.produktID ) AS foo
GROUP BY land, jahr, produkt
```

Listing 3.4: Geschachtelte SQL-Anfrage

Die beiden oben aufgeführten Problemdarstellungen machen deutlich, dass die Unterstützung von OLAP-Funktionalität zwar mit bekannten relationalen SQL-Anweisungen möglicht ist, zeigen aber auch, wie komplex Anfragen generiert werden müssen und lässt bereits vermuten, wie ineffizient diese verarbeitet werden.

Seitens der Data-Warehouse-Anwender stand somit die Forderung nach einer mächtigeren Sprachunterstützung für SQL, die die komplexe Formulierung von Anfragen vermeidet. Aber auch die Erzielung einer besseren Performance ist nötig. Diese lässt sich dahingehend motivieren, dass bei der Verwendung von Aggregationen und mehreren *GROUP BY*-Operatoren, Teilergebnisse weiter aggregiert werden können und nicht jeweils neu berechnet werden müssen.

In [GCB+96] stellte das Forschungszentrum von Microsoft bereits 1996 auf der ICDE (International Conference on Data Engineering) einen Vorschlag zur Erweiterung des SQL-Standards um OLAP-Funktionalitäten vor. Der dort eingeführte *CUBE*-Operator erlaubt die einfache Formulierung eines SQL-Statements zur Anwendung einer Aggregatsfunktionen auf alle 2^n möglichen Gruppierungskombinationen für eine vorgegebene Menge von Gruppierungsattributen. Neben dem *CUBE*-Operator gab es weitere Vorschläge, wie den *ROLLUP*-Operator und den Grouping Sets.

Die Notwendigkeit, statistische Auswertungen auf Basis von relationalen DBMS zu unterstützen, führte bei den großen Datenbankherstellern dazu, die entsprechende Funktionalität auch im Standard SQL zu verankern, so dass ein gemeinsamer Erweiterungsvorschlag für SQL-99 vorgebracht wurde.

Der folgende Abschnitt stellt die speziell zur Unterstützung von OLAP benötigten Erweiterungen im SQL-99 Standard vor. Daran schließt sich eine kurze Darstellung über Möglichkeiten zur Optimierung von multidimensionalen Anfragen an. Der Abschnitt 3.6.3 legt dar, wie die OLAP-Erweiterungen des SQL-Standards und die Optimierungen in den Datenbankmanagementsystemen DB2 und Oracle umgesetzt werden.

3.6.1 OLAP-Erweiterungen im SQL-99 Standard

Die Unterstützung von OLAP-Funktionalität durch den SQL-99 Standard beruht im Wesentlichen auf der Erweiterung des *GROUP BY*-Operators mittels *CUBE*, *ROLLUP* und *GROUPING SETS*. Sie ermöglichen die einfache Formulierung von Anfragen und geben die Grundlage für eine effiziente Anfrageverarbeitung, zum Beispiel durch Zwischenspeicherung von bereits berechneten Aggregationen. Im Zusammenhang mit der Einführung der Operatoren existiert weiterhin eine Erweiterung bei den Aggregatfunktionen. Die einzelnen Ausdrücke und deren Semantik werden im Folgenden näher erläutert. Eine exakte Syntaxspezifikation ist in ANHANG B.2 nachzulesen.

CUBE

Eine weit verbreitete Präsentationsform ist die Kreuztabelle. Wie in Listing 3.2 gezeigt wurde, ist für deren Berechnung jedoch ein komplexer SQL-Ausdruck (SQL 92) notwendig gewesen. Der in SQL-99 neu aufgenommene *CUBE*-Operator stellt eine Kurzschreibweise dessen dar. Für eine vorgegebene Menge von Gruppierungsattributen werden alle möglichen Gruppierungskombinationen generiert.

```
SELECT land, jahr, produktKategorie, SUM(anzahl)
FROM ...
GROUP BY CUBE (land, jahr, produktKategorie)
```

Listing 3.5: SQL-Anfrage mit Cube-Operator

Die Anfrage aus Listing 3.5 ist äquivalent zur Anfrage in Listing 3.2 und liefert ein identisches Ergebnis. Wie bereits in Tabelle 3.3 angedeutet, hat der Attributwert *NULL* eine besondere Bedeutung. In der Ergebnisrelation steht *NULL* in der Bedeutung „*alle*“ an Stellen, an denen Attribute nicht zur Gruppierung gehören.

Im Zusammenhang mit der multidimensionalen Analyse, erzeugt der *CUBE*-Operator alle Teilsummen und Summen, die für einen multidimensionalen Würfel mit den gegebenen Dimensionen berechnet werden können.

Der *CUBE*-Operator kann auf unterschiedliche Dimensionen angewendet werden. Funktionale Abhängigkeiten werden dabei nicht berücksichtigt, so dass der *CUBE*-Operator auch redundante Kombinationen berechnet. Bei der Attributliste *(produktKategorie, produktGruppe, jahr, monat)* im gegebenen Beispiel würde neben der Gruppierung *(produktKategorie, produktGruppe, jahr, monat)* auch die dazu redundante Kombination *(produktGruppe, monat)* erzeugt werden.

ROLLUP

Ein dem SQL-99 Standard weiter hinzugefügter Operator ist der *ROLLUP*-Operator. Er generiert für eine gegebene Liste von Attributen A_1, ..., A_n nicht alle Gruppierungskombinationen, sondern nur die Gruppierungen $(A_1, \ldots, A_n)$, $(A_1, \ldots, A_{n-1})$, …, (A_1) und $()$. Werden hinter dem Operator verschiedene, durch Komma voneinander getrennte Listen von Gruppierungsattributen angegeben, so erfolgt eine Kombinierung der einzelnen Ergebnisse miteinander. Im Gegensatz zum *CUBE*-Operator bietet die *ROLLUP*-Klausel die Möglichkeit der Berücksichtigung von Klassifikationshierarchien, wobei der funktionale Zusammenhang der Attribute besonders betont wird.

Eine SQL-Anfrage zur Bildung der Teilsummen für die beiden Dimensionen Zeit und Produkt sieht folgendermaßen aus:

```
SELECT produktKategorie, produktGruppe, jahr, monat, SUM(anzahl)
FROM ...
GROUP BY
    ROLLUP (produktKategorie, produktGruppe),
    ROLLUP (jahr, monat)
```

Listing 3.6: SQL-Statement mit Rollup-Operator

Die Bearbeitung der Anfrage liefert jeweils drei Kombinationen für die beiden *ROLLUP*-Operatoren, also insgesamt neun. Der *CUBE*-Operator liefert hingegen für die gegebenen Attribute 16 unterschiedliche Gruppierungen. Das Ergebnis der Anfrage ist in Tabelle 3.4 dargestellt.

produktKategorie	produktGruppe	jahr	monat	SUM(anzahl)
Einrichtungen	Beleuchtung	2000	11	4
Einrichtungen	NULL	2000	11	4
Technik	Audio	2000	11	13
Technik	Video	2000	11	68
Technik	NULL	2000	11	81
NULL	NULL	2000	11	85
…				
NULL	NULL	2001	1	71
Einrichtungen	Beleuchtung	2000	NULL	23
Einrichtungen	NULL	2000	NULL	23
…				
Technik	NULL	NULL	NULL	160
NULL	NULL	NULL	NULL	187

Tabelle 3.4: Ergebnis der Anfrage aus Listing 3.6

Grouping Sets

Eine Möglichkeit zur flexiblen Spezifizierung von Gruppierungen ist im SQL-99 Standard mit dem *GROUPING SETS*-Operator gegeben. Dieser stellt eine Verallgemeinerung der bereits vorgestellten Operatoren dar und gestattet die Aufzählung einer Menge von *GROUP BY*-Klauseln in einem Anfrageausdruck. Die Argumente der Klausel bestehen dabei aus einfachen Gruppierungskombinationen.

Das SQL-Statement zur Berechnung der Verkaufszahlen auf jeder Hierarchiestufe der Dimension Zeit sieht wie folgt aus:

```
SELECT jahr, monat, tag, SUM(anzahl)
FROM ...
GROUP BY GROUPING SETS (
      (jahr, monat, tag),
      (jahr, monat),
      (jahr),
      () )
```

Listing 3.7: Grouping Sets

Die Anfrage liefert ein äquivalentes Ergebnis zu der verkürzten Formulierung:

```
SELECT jahr, monat, tag, SUM(anzahl)
FROM ...
GROUP BY ROLLUP (jahr, monat, tag)
```

Listing 3.8: Äquivalente SQL-Anfrage mit dem ROLLUP-Operator zu Listing 3.7

Neben einfachen Gruppierungskombinationen sind auch komplexe Gruppierungsbedingungen, die sich mittels *CUBE*- oder *ROLLUP*-Operatoren formulieren lassen, als Argumente für Grouping Sets zulässig. Grouping Sets können miteinander kombiniert werden, jedoch nicht selbst Parameter einer *GROUPING SETS*-Anweisung sein. Folgendes Beispiel demonstriert eine Kombination Grouping Sets in Verbindung mit dem *ROLLUP*-Operator:

```
SELECT
    produktKategorie, produktGruppe,
    jahr, monat, woche, tag,
    SUM(anzahl)
FROM ...
GROUP BY
    ROLLUP (produktkategorie, produktGruppe),
    GROUPING SETS (ROLLUP (jahr, monat, tag), (woche)),
    GROUPING SETS ((land), (region))
```

Listing 3.9: SQL-Anfrage mit ROLLUP- und GROUPING SETS-Operator

Der erste Teil der Gruppierung wertet die Produktdimension nur für die Produktkategorie und Produktgruppe aus, während im zweiten Teil die Zeitdimension mit ihrer Parallelhierarchie vollständig ausgewertet wird. Berechnungen auf der Geographiedimension erfolgen hingegen nur auf den Attributen *land* und *region*.

Aggregatfunktionen

Zur Aggregation der Kenngrößen bei der multidimensionalen Analyse stehen im SQL-99 Standard die Set-Funktionen (eher bekannt unter dem Namen Aggregatfunktionen) *AVG*, *COUNT*, *MAX*, *MIN* und *SUM* zur Verfügung. Diese können in Verbindung mit *DISTINCT* verwendet werden, was bei *MAX* und *MIN* allerdings im Ergebnis keine Auswirkungen zeigt. Eine dem SQL Standard neu hinzugefügte Set-Funktion ist *GROUPING*. Sie kann verwendet werden, um die von den oben genannten Operatoren zurückgegebenen *NULL*-Werte mit der Bedeutung „*alle*“ von Standard-*NULL*-Werten zu unterscheiden. Angewandt auf ein Gruppierungsattribut, liefert die *GROUPING*-Funktion den numerischen Wert 1, wenn über dieses Attribut hinweg aggregiert wurde, andernfalls ist der Wert 0.

Gäbe es beispielsweise in dem bisher verwendeten Beispiel für einige Produkte keine Produktkategorie, der Wert ist also *NULL*, so kann eine Unterscheidung der *NULL*-Werte mit deren unterschiedlichen Bedeutungen folgendermaßen geschehen:

```
SELECT produktKategorie, SUM(anzahl), GROUPING(produktKategorie)
FROM ...
GROUP BY ROLLUP (produktKategorie)
```

Listing 3.10: GROUPING-Funktion in SQL

produktKategorie	SUM(anzahl)	GROUPING(produktKategorie)
NULL	11	0
Einrichtung	27	0
Technik	160	0
NULL	198	1

Tabelle 3.5: Ergebnis der Anfrage aus Listing 3.10

Neben der Unterstützung zur Unterscheidung der verschiedenen Bedeutungen von *NULL*-Werten kann die *GROUPING*-Funktion zur Bestimmung des Levels einer Aggregation als auch zur Filterung und Sortierung der Ergebnismenge eingesetzt werden.

Eine Erweiterung der GROUP BY-Klausel im Zusammenhang mit der Verwendung von Funktionen in der Menge der Gruppierungsattribute, wie sie oben beschrieben wurden, bietet der SQL-99 Standard nicht. Sind diese notwendig, so können sie mit Hilfe von geschachtelten Anfragen (siehe Listing 3.4) oder Sichten realisiert werden. Verschiedene Hersteller von Datenbankmanagementsystemen unterstützen zwar die Formulierung solcher Anfragen, was jedoch nicht dem Standard entspricht.

Einen umfassenden Überblick zu den OLAP-Erweiterungen in SQL sowie zum SQL-99 Standard ist [GuPe99] zu entnehmen.

3.6.2 Optimierung von Anfragen

OLAP-Anfragen umfassen in der Regel umfangreiche Aggregationen über bestimmte Datenbereiche. Aus sehr großen detaillierten Datenmengen werden reduzierte Daten erzeugt. Problematisch ist, dass Aggregationen auf großen Datenmengen sehr teuer sind. Für die Beantwortung einer Anfrage müssen die Daten meist komplett gelesen werden. Hinzu kommen die notwendigen Verbunde bei einer Speicherung im relationalen Modell, das Schreiben von Zwischenergebnissen und Sortierungen für die Aggregation. Die effiziente Bearbeitung von Anfragen, die für OLAP-Anwendungen gefordert ist, setzt daher sehr hohe Anforderungen an die Optimierung.

In diesem Abschnitt werden verschiedene Ansätze zur Optimierung von Anfragen aufgezählt. Eine genauere Betrachtung erfolgt hierbei nicht, da die zu Grunde liegende Datenmenge des gewählten Anwendungsszenario Europa-MV eher minimal ist. Ausführliche Informationen zum Thema Optimierung kann [BaGü01] entnommen werden.

Indizierungstechniken

Bei Restriktionen – sollen zum Beispiel nur die Daten für ein bestimmtes Jahr analysiert werden – reduziert sich in der Regel die Anzahl der benötigten Daten erheblich. Durch die Wahl geeigneter Indizierungen kann die Durchführung von *Full Table Scans*, welches dem Laden aller Daten gleichkommt, verhindert werden. Ansätze hierfür sind eindimensionale Indizierungstechniken (speziell B-Bäume), mehrdimensionale Techniken (zum Beispiel der UB-Tree, siehe Anhang C) oder Bitmap-Indizes.

Partitionierungen

Eine Ergänzung zu Indextechniken bieten *Partitionierungen*. Umfangreiche Relationen werden in kleinere einzelne Teilrelationen aufgeteilt. Die Teilrelationen können einzeln gelesen und geschrieben werden, so dass sich verschiedene Transaktionen nicht gegenseitig beeinflussen. Die *Rang-Partitionierung* stellt hierbei die gebräuchlichste Form dar, wobei die Tupel anhand der Werte von zuvor festgelegten Attributen auf die Teilrelationen aufgeteilt werden.

Verbundoptimierungen

Ein weiterer wesentlicher Kostenfaktor ist die Realisierung von Verbunden. Gewöhnlich werden diese intern durch eine Sequenz von paarweisen Verbunden realisiert. Die Aufgabe

des Optimierer besteht darin, eine möglichst kostengünstige Reihenfolge festzulegen. Um diesen Suchraum zu verringern, werden Heuristiken verwendet, die sich im OLTP-Einsatz durchaus bewährt haben, jedoch bei OLAP-Anfragen sehr teuer sein können. Grund hierfür ist meist der sehr große Unterschied zwischen verhältnismäßig kleinen Dimensions- und sehr großen Faktentabellen. Eine Optimierung der Verbunde bei der Verwendung von Star-Joins kann diesem Verhalten entgegenwirken.

Materialisierte Sichten

Beim Einsatz von OLAP-Technologien lässt sich beobachten, dass sich eine Vielzahl von Anfragen auf eine nahezu gleiche Menge von Relationen bezieht. Oft sind die Aggregationen ähnlich und der Datenbestand weitgehend stabil. Durch Vorberechnungen, so genannten *materialisierten Sichten*, können Teilergebnisse von Anfragen mehrfach herangezogen werden, was zu einer deutlichen Reduzierung der Anfragebearbeitungszeit führen kann.

Werden materialisierte Sichten verwendet, sind verschiedene Punkte zu beachten. Die Realisierung muss transparent geschehen, sie darf also keinen Einfluss auf die Anfrageformulierung haben. Da die Sichten redundante Daten darstellen, ist es erforderlich, einen guten Kompromiss zwischen dem zusätzlich benötigtem Speicherplatz und der Reduzierung der Anfragebearbeitungszeit zu finden. Ein weiteres Problem stellt die effiziente Aktualisierung der Sichten dar. Oft lässt sich die gewünschte Performance erst durch den Einsatz von materialisierten Sichten erzielen. Die genannten Probleme, die hierbei auftreten, sind daher auch Gegenstand vieler wissenschaftlicher Untersuchungen.

3.6.3 Kommerzielle Systeme

In Bezug auf die Verwendbarkeit des *CUBE* beziehungsweise *ROLLUP*-Operators sind die kommerziellen Hersteller von Datenbankmanagementsystemen dem Standard bereits voraus. So bieten Hersteller wie Oracle mit ihrem gleichnamigen DBMS, IBM mit DB2 und Microsoft mit dem SQL Server schon seit einiger Zeit OLAP-Unterstützung an. Diese geht dabei in der Regel über den Standard hinaus und schließt unter anderem den Einsatz genannter Optimierungstechniken ein. Einen Überblick zu dem Datenbankmanagementsystem von Oracle und IBM soll in den folgenden beiden Abschnitten gegeben werden. Dabei handelt es sich jedoch nicht um ein Vergleich der beiden Produkte, dieses wurde bereits an anderen Stellen (z.B. [BBM+01], [Gonz00]) vorgenommen.

Oracle

Bereits mit der Version 7.3 stellte Oracle die erste Unterstützung für die multidimensionale Analyse vor. War es hier zunächst nur der Star-Join zur Optimierung von Verbunden beim Star-Schema, so folgte mit Oracle ab der Version 8.1.5 (8i) ein breit gefächertes Spektrum an Hilfsmitteln. Hierzu gehörten SQL-Erweiterungen, wie die *CUBE*-, *ROLLUP*- und *GROUPING*-Klausel, Range- und Hash-Partitionierung sowie die Kombination dieser beiden Verfahren und materialisierte Sichten. Bei der Verwendung existierten jedoch noch Einschränkungen. So durften bei Oracle 8i die *WHERE*-Klausel der materialisierten Sicht außer Join-Bedingungen keine Restriktionen enthalten.

Mit Oracle9i wurden die OLAP-Erweiterungen in Bezug auf den SQL-99 Standard komplettiert. So ist in Oracle9i nun auch das Konzept der Grouping Sets implementiert. *GROUPING SETS*-, *ROLLUP*- und *CUBE*-Klauseln sind nahezu beliebig miteinander kombinierbar.

Neben den Standard-SQL-Erweiterungen finden sich in Oracle9i eine Reihe weiterer Funktionen zur Unterstützung von Data-Warehouse-Anwendungen, wobei Funktionalitäten früherer Versionen zum Teil erweitert wurden. Mit ergänzten Aggregatfunktionen lässt sich beispielsweise die Varianz, die Standardabweichung oder der Korrelationskoeffizient berechnen. *Analytical Functions*, wie sie in Oracle bezeichnet werden, bilden gegenüber den gewöhnlichen Aggregatfunktionen ein neues Sprachkonstrukt. Sie erlauben es völlig unabhängige Unterabfragen zu stellen, die dann zu einem Gesamtergebnis verknüpft werden. In der SQL-Syntax wird hierfür die *OVER*-Klausel verwendet. Eine Funktion zur Ermittlung der Rangfolge lässt sich folgendermaßen formulieren:

```
SELECT filiale,
       anzahl,
       RANK() OVER (ORDER BY anzahl DESC)
FROM ...
```

Listing 3.11: RANK-Funktion bei Oracle9i

Mit der Einführung von Oracle9i sind die Beschränkungen der *WHERE*-Klausel zur Definition von materialisierten Sichten gelockert worden. Für die Aktualisierung der Sichten (*Refreshing*) stehen Automatismen zur Verfügung, die sich bezüglich dem Zeitpunkt (*ON COMMIT*, *ON DEMAND* oder periodisch) und der Art der Aktualisierung (*COMPLETE*, *FAST*) konfigurieren lassen. Die inkrementelle Aktualisierung kann jedoch nur unter bestimmten Voraussetzungen durchgeführt werden. Zusätzlich stellt Oracle Schnittstellen bereit, über die statistische Informationen beziehungsweise Informationen zur Verwendung der materialisierten Sichten ermittelt werden können.

Mit der List-Partitionierung, die eine flexiblere Aufteilung der Daten ermöglicht, stellt Oracle9i eine weitere Partitionierungform zur Verfügung.

Neben den Erweiterungen auf relationaler Ebene bietet Oracle mit *OLAP Service* und *Oracle BI Beans* eine Sammlung von Werkzeugen, die die Entwicklung umfangreicher OLAP-Applikationen unterstützen. Beide sind in den *Oracle Enterprise Manager* integriert. Zu *OLAP Service* gehört unter anderem eine *Java OLAP API*, die durch *Oracle BI Beans* um zusätzliche Funktionalitäten erweitert werden kann.

Einen umfassenden Überblick zur Unterstützung von OLAP-Funktionalitäten in Oracle gibt die Zusammenfassung in [Albr01]. Weitere Informationen können in den Dokumentationen zu Oracle9i [Orac01] nachgelesen werden.

IBM DB2

Das von IBM entwickelte Datenbankmanagementsystem DB2 (DB2 Universal Database v7.2) unterstützt ebenfalls die im SQL-99 Standard spezifizierten OLAP-Erweiterungen.

Die Kernkomponente zur Performancesteigerung in DB2 ist der Anfrageoptimierer. Mit Hilfe von ausgereiften Techniken werden SQL-Anfragen in äquivalente und optimierte Anfragepläne transformiert.

Zur effizienteren Bearbeitung von komplexeren SQL-Anfragen können in DB2 die so genannten *Automatic Summary Tables* herangezogen werden. Sie ermöglichen die Aufnahme von aggregierten Daten über verschiedene Dimensionen aus einer umfangreichen Faktentabelle. Die Aktualisierung der Tabellen kann abhängig von Änderungen an den Basistabellen geschehen (*REFRESH IMMEDIAT*) oder durch ein Kommando bewirkt werden (*DEFERRED*). Der Anfrageoptimierer analysiert die gestellten SQL-Anfragen und veranlasst für deren Bearbeitung gegebenenfalls die Verwendung der *Summary Tables*. Ähnlich wie bei den materialisierten Sichten in Oracle unterstützt DB2 mit den *Automatic Summary Tables* so die transparente Nutzung von vorberechneten Daten.

Neben relationalen OLAP-Erweiterungen in DB2 bietet IBM zusätzliche Werkzeuge zur Unterstützung der multidimensionalen Analyse. Der *DB2 OLAP Server* ist ein Produkt zur Entwicklung von Reporting- und Analyseprogrammen. Er umfasst die Leistungsfähigkeit von *Hyperion Essbase* und erweitert diese um die Option zur Speicherung von multidimensionalen Daten im relationalem Datenbankmodell. Wie der *DB2 OLAP Server* basiert auch der *OLAP Integration Server* auf Techniken von Hyperion. Der *OLAP Integration Server* bietet eine graphische Benutzerschnittstelle zur Definition von OLAP Strukturen auf einer relationalen Datenbasis. Das *DB2 OLAP Starter Kid* stellt eine Teilmenge der Funktionalität des *OLAP Server* und des *OLAP Integration Server* bereit. Weiterhin stehen mit dem *Data Warehouse Center* umfangreiche Funktionen zur Erstellung eines Data Warehouse, auf dessen Grundlage OLAP-Analysen durchgeführt werden können, zur Verfügung.

Weiterführende Informationen zur Unterstützung der multidimensionalen OLAP-Analyse in DB2 sind in [WhWh99] zu finden.

3.7 Zusammenfassung

In diesem Kapitel wurden grundlegende Konzepte und Techniken aus dem Bereich OLAP vorgestellt. Intensive Betrachtung fanden hierbei das multidimensionale Datenmodell sowie die Anfragenunterstützung für multidimensionale Strukturen bei relationaler Speicherung.

Das multidimensionale Datenmodel bildet die Grundlage zur OLAP-Analyse. Die OLAP-Analyse ist ein dynamischer Prozess, bei dem der Anwender mithilfe spezieller Operatoren durch die multidimensionalen Datenstrukturen navigieren kann, um so individuelle Sichten auf den Würfel zu erhalten.

Für die Speicherung multidimensionaler Daten gibt es verschiedene Realisierungsmöglichkeiten. Im Rahmen dieser Diplomarbeit steht dabei die relationale Speicherung im Vordergrund. Die OLAP-Erweiterungen in SQL-99 bilden die Grundlage für eine effiziente Bearbeitung multidimensionaler Anfragen bei einer relationalen Speicherung. Deren Unterstützung ist durch die kommerzielle wie Oracle oder IBM/DB2 gegeben.

Teil II
Metadatenverwaltung

4. XML-Framework-Architektur

Metadaten sind die Voraussetzung für die Steuerung der Request-Verarbeitung und Anfrage-Transformation in komplexen Systeminfrastrukturen. Ein Lösungsansatz auf der Basis von RDF bezüglich der Implementierung einer zusätzlichen semantischen Ebene bei der Realisierung eines adäquaten Metadaten-Repositories bildet den Schwerpunkt für die nachfolgenden Abschnitte. Dieser Lösungsansatz ist eingebettet in die offene und erweiterbare XML-Framework-Architektur der XPEA-Referenz-Implementierung.

4.1 Systemarchitektur

Im Rahmen des XPEA-Projektes wird eine Systemarchitektur entwickelt, die eine kontext-abhängige Suche von Daten in verschiedenen Datenbanken ermöglicht. Hierbei wird speziell die generische Integration von Datenquellen, die Unterstützung unterschiedlicher Dokumentenformate und die Bereitstellung von Schnittstellen für die verschiedensten Endgeräte berücksichtigt.

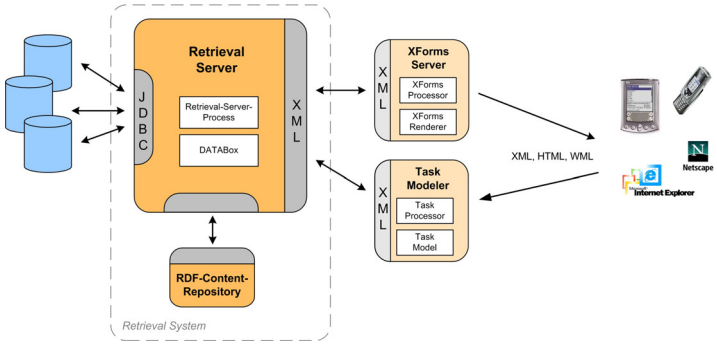

Abbildung 4.1: XPEA-Architektur

Die XPEA-Plattform baut auf den Prototypen DigTeD [Ditt01] auf, der im Rahmen des XPEA-Projektes um den *Task Modeler* und den *XForms Server* erweitert wird. Die Aufgabe der

erweiterten Komponenten sind die Generierung endgerätespezifischer Dialogstrukturen und die Interaktionssteuerung. Im Zuge des XPEA-Projektes erfolgt zudem eine Erweiterung der DATABox, zur Metadaten-gesteuerten Transformation der Datenbankanfrageergebnisse in XML-Dokumente (XML-Map). Die erweiterten Komponenten befinden sich derzeit in der Entwicklung.

Retrieval System

Zentraler Kern der XPEA-Plattform ist das *Retrieval System*, dessen Architektur die Grundlage der zu lösenden Aufgaben in der Diplomarbeit bildet.

Die an das Retrieval System gestellten Anforderungen sind:

- Skalierbarkeit,
- Flexibilität hinsichtlich Systemerweiterungen,
- generische Integration verschiedener Datenbanken mit adäquater Zugriffssteuerung,
- kontextabhängige Steuerung der Anfragebearbeitung und Ergebnis-transformation (-mediation),
- Endgeräteunabhängigkeit und
- Modularität.

Für die Umsetzung wurde eine Architektur gewählt, die zur Zugriffs-, Anfrage-, Transformations- und Darstellungssteuerung auf Metadaten basiert. Die Grundlage zur Beschreibung der Metadaten stellt das Resource Description Framework dar.

Durch die Nutzung von Metadatenbeschreibungen ergeben sich für das Retrieval System zwei unterschiedliche Formen des Retrievals:

- Das Dokumentenretrieval ist eigentliches Ziel der Architektur, bei der Informationen oder Dokumente in den integrierten Datenbanken des Systems gesucht werden.
- Unterstützt wird diese Suche durch Metadaten. Ergebnis ist die Lieferung passender und für die Anfragetransformation des Dokumentenretrieval notwendiger Metadaten.

Retrieval Server und RDF-Content-Repository

Die Architektur des Retrieval Systems, dargestellt in Abbildung 4.1, setzt sich aus dem Retrieval Server und dem RDF-Content-Repository zusammen.

Die Aufgaben des Retrieval Servers sind die kontextabhängige Anfrage- und Ergebnistransformation. Der Server verfügt hierzu über verschiedene Schnittstellen und Komponenten. Die Schnittstellen dienen dem Zugriff auf notwendige Datenbestände sowie der Weiterleitung der Anfrageergebnisse. Die Schnittstellen und Komponenten des Retrieval Servers werden nun kurz vorgestellt:

- Eine parametrisierte JDBC-Schnittstelle ermöglicht die Verbindung zu den verschiedenen integrierten Datenbanken. Die Informationen für den jeweiligen Datenbankzugriff sind Bestandteil der Metadaten und werden über das RDF-Content-Repository ermittelt.
- Über die XML-Schnittstelle erfolgt die Kommunikation mit dem XForms- beziehungsweise Application Server. Im Gegensatz hierzu sind im DigTeD-Ansatz die

Kommunikationspartner mobile Endgeräte (Palm) oder ein HTTP-Server, der als Zwischenkomponente die Kommunikation zu einem Browser oder ein WAP-fähigen Mobiltelefon organisierte.

Da es sich aus Sicht des Retrieval Servers bei den Kommunikationspartnern um verschieden Clients handelt, soll folgend davon abstrahiert und diese allgemein als Client bezeichnet werden.

- Die Schnittstelle zum RDF-Content-Repository stellt einen Zugriff auf die Metadaten bereit.
- Die Steuerung des Dokumentenretrieval erfolgt über eine zentrale Prozessteuerung des Retrieval Servers.
- Dokumente und Informationen werden mit Hilfe von SQL-Ausdrücken in den Datenbanken gesucht. Die DataBOX organisiert die einzelnen Anfragen und generiert aus den Ergebnissen XML-Dokumente.

Aufgabe des RDF-Content-Repository ist die Verwaltung der für die Applikation notwendigen Metadaten. Deren Verwaltung wurde im Prototypen DigTeD eher proprietär gelöst. Ein Ziel dieser Diplomarbeit ist die Entwicklung einer adäquaten, RDF-basierten Metadatenverwaltung. Die grundlegenden Konzepte für deren Umsetzung wird in Kapitel 5 erläutert.

Im folgenden Abschnitt erfolgt ein Überblick zu den anwendungsspezifischen Metadaten und deren Modellierung.

4.2 Metadatenbestand der Retrieval Systems

Voraussetzung für die Integration verschiedener Datenquellen in das System, die semantische Beschreibung von multimedialen Daten zur Content Transformation auf struktureller Ebene sowie auf den Datenelementen selbst und die Anbindung verschiedener Endgeräte sind Metadaten. Diese können anhand von zwei Kriterien (siehe Klassifikation in Abschnitt 2.1) den folgenden Klassen zugeordnet werden:

Kriterium: Verwendung

- terms and conditions
- structure
- context
- content

Kriterium: Beziehungen zwischen Metadaten und Ressource

- content descriptive
- content independent

Die Beschreibung der Metadaten erfolgt in einem RDF-Modell. Aus welchen Informationen sich das Modell zusammensetzt, illustriert die Abbildung 4.2.

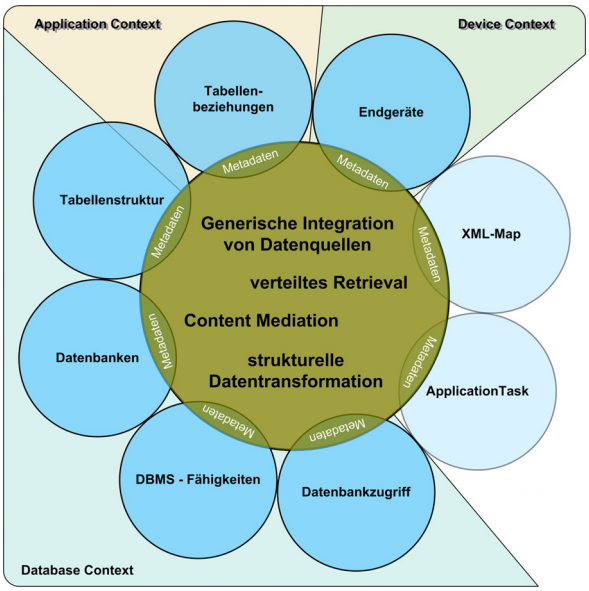

Abbildung 4.2: Metadaten bei der XML-Framework-Architektur

Die verschiedenen Gruppen von Metainformationen spiegeln sich in den folgenden drei Kontexten wieder:

Database Context

Der *Database Context* beinhaltet Informationen zu den unterschiedlichen Datenbankmanagementsystemen und den integrierten Datenbanken. Hierzu gehören:

- Fähigkeiten des DBMS (multimediale Fähigkeiten, wie z.B. Skalierung von Bilddaten),
- JDBC-Treiberinformationen
- Namen und Beschreibungen der Datenbanken
- Informationen zum Datenbankzugriff (URL, Benutzername und Passwort)
- Namen und Beschreibungen der Relationen
- Attribute der Relationen einschließlich Datenbankdatentyp, Datentyp (MIME-Type), maximale Datengröße sowie anwendbare Transformationsmethoden

Application Context

Die Metadaten zur Beschreibung der Beziehungen zwischen den Relationen innerhalb der Datenbanken und der Eigenschaften relevanter Attribute bezüglich eines gewählten Kontexts sind Bestandteil des *Application Context*. Eine Trennung von Database- und Application Context findet unter dem Gesichtspunkt statt, dass die hier vorgestellte Systemarchitektur für unterschiedliche Anwendungen eingesetzt werden kann. Es ist so die Möglichkeit gegeben, für jeden Kontext eine spezielle Sicht auf den Datenbestand zu definieren. Der Application Context umfasst folgende Informationen:

- Beziehungen zwischen den Tabellen (Fremdschlüsselbeziehungen und darüber hinaus definierte Zusammenhänge)
- Attribute, über denen das Retrieval und über denen die Ergebnisprojektion erfolgt

Device Context

Für die Unterstützung verschiedener Arten von Endgeräten enthält der jeweilige *Device Context* Metadaten in Form von gerätespezifischen Parametern. Relevante Daten sind hierbei Auflösung und Farbtiefe des Displays, Arbeitsspeicher und unterstützte Dokumenttypen.

In der Abbildung 4.2 treten zudem die Punkte *XML-Map* und *ApplicationTask* auf. Die unter diesen Punkten gespeicherten Metadaten sind Erweiterungen in der XPEA-Architektur. Sie dienen hier zur Steuerung der Transformation von Daten in das XML-Format und der Beschreibung des für den Endbenutzer aufbereiteten Graphical User Interface (GUI). Diese Abbildungssteuerungen sind in der derzeitig entwickelten Architektur noch proprietär gelöst.

Die Abbildung 4.3 veranschaulicht den Aufbau des RDF-Modells anhand eines kleinen Ausschnittes für den Database Context.

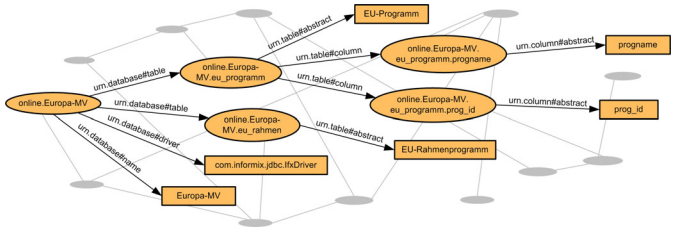

Abbildung 4.3: Ausschnitt aus dem Metadatenbestand

4.3 Erzeugung von Metadaten

Der vorherige Abschnitt verdeutlichte bereits, welcher umfangreiche Metadatenbestand zur Steuerung des Retrievals notwendig ist. Die Erzeugung des zugrunde liegenden RDF-Modells kann durch automatische Verfahren unterstützt werden. Dieses ist jedoch nicht immer möglich. Gründe hierfür sind

- die Komplexität der zu beschreibenden Daten,
- das fehlendes Kontextwissen bei den Verfahren sowie
- nicht maschinell erfassbare Informationen.

Für das Retrievalsystem wurden Techniken entwickelt, die den automatischen als auch den manuellen Gewinnungsprozess der Metadaten unterstützen. So können beispielsweise die Relationsschemata und die Beziehungen zwischen den einzelnen Relationen maschinell erfasst werden. Zur Beschreibung der Endgeräte bietet sich hingegen eine benutzerfreundliche Bedieneroberfläche an. Im Rahmen dieser Arbeit wurden hierzu zwei Module entwickelt:

Metadatenextraktion der Datenbankschemata

Mit Hilfe eines JDBC-Treibers stellt das Modul eine Verbindung zu den integrierten Datenbanken auf. Die JDBC-API ermöglich das Auslesen von Datenbankmetadaten. So lassen sich durch das Modul die vorhandenen Relationen, deren Attribute, die dazugehörigen Typen, Datenbankfähigkeiten und Fremdschlüsselbeziehungen automatisch ermitteln.

Metadaten der Endgeräte

Das Modul stellt eine graphische Oberfläche zur Beschreibung der Endgeräte bereit (siehe Abbildung 4.4). Darin können die gerätespezifischen Parameter festgelegt werden.

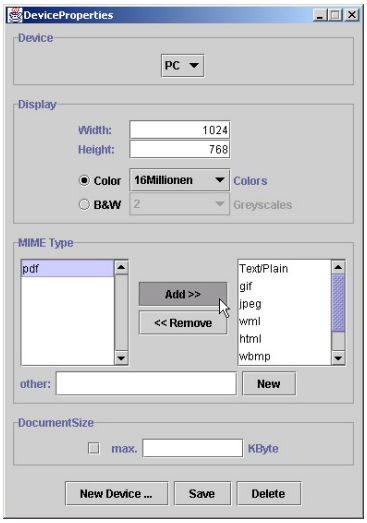

Abbildung 4.4: Benutzerschnittstelle zur Beschreibung der Endgeräte

Nachdem in diesem Kapitel die Systemarchitektur, der Aufbau des Metadatenbestandes und die Module zur Metadatengenerierung vorgestellt wurden, wird im anschließenden Kapitel erläutert, wie eine geeignete Speicherung des RDF-Modells erfolgen kann und wie Anfragen möglich sind.

5. RDF-basierte Metadatenverwaltung

In Form des RDF-Content-Repository ist eine adäquate, RDF-basierte Metadatenverwaltung realisiert worden. Deren Aufgaben sind:

- die Persitierung von RDF-Daten,
- das Retrieval von RDF-Daten und
- die Bereitstellung geeigneter Unterstützungstechniken zur Erstellung von Metadaten.

Wie die Gewinnung von Metadaten unterstützt werden kann, wurde bereits in Abschnitt 4.2 dargelegt. In diesem Kapitel werden Techniken zur Speicherung und zum Retrieval von RDF-Daten vorgestellt. Um einen allgemeinen Überblick zu diesen Themen zu geben, werden innerhalb der Abschnitte 5.1 und 5.2 die Speicherung und das Retrieval von RDF-Daten teilweise losgelöst vom Anwendungsszenario betrachtet.

5.1 Speicherung von RDF

Die Nutzung von RDF-Beschreibungen auf Grundlage von low-level APIs und der Speicherung von RDF-Daten in Dateien, gewährleistet keine optimale Entwicklung, Wartung und Erneuerung von realen, RDF-basierten Applikationen. Gegebene RDF-APIs bieten die Möglichkeiten RDF-Daten zu parsen und daraus RDF-Modelle aufzubauen, welche manipuliert und wieder seriallisiert werden können. Die Modelle werden jedoch nur im Hauptspeicher gehalten. Die Verarbeitung der Daten in flachen Dateien und im Hauptspeicher ohne eine geeignete Datenhaltungskomponente zieht erhebliche Defizite nach sich. Sie unterstützt beispielsweise keine effiziente Verwaltung großer Datenmengen und bietet zudem keine Ausfallsicherheit. Eine adäquate, persistente Speicherung der RDF-Daten mit Manipulations- und Anfragemöglichkeiten ist somit wünschenswert.

Datenbanken zeichnen sich durch die folgend genannten Eigenschaften aus:

- Verwaltung großer Datenmengen
- Unterstützung des Mehrbenutzerprinzips durch Transaktionskonzepte
- Datenunabhängikeit
- Gewährleistung von Datenschutz durch Benutzerkontrolle
- Datensicherheit durch Recovery-Mechanismen

Außerdem bieten Datenbanken Indizierungsmechanismen und Anfrageoptimierungen für eine effiziente Abarbeitung von Anfragen. Es liegt also nahe, die Leistungsmerkmale vorhandener Datenbanksysteme für eine adäquate, RDF-basierte Metadatenverwaltung zu nutzen und darauf aufzubauen.

5.1.1 Vorhandene Implementierungen von RDF-Datenbanken

Zur Speicherung von RDF existieren bereits einige Implementierungen. Diese bauen in der Regel auf bestehende DBMS auf. Einige interessante RDF-Datenbanken sollen hier kurz vorgestellt werden.

RDFSuite

Die *RDFSuite*[1] [ACK+01] ist eine Entwicklung des *FORTH Institute of Computer Sience*, die aus mehreren Komponenten besteht: *Validating RDF Parser* (VRP), *RDF Schema Specific Database* (RSSDB) und *RDF Query Language* (RQL).

Bei dem VRP handelt es sich um ein API zum Parsen von RDF-Daten in XML-Syntax. Neben der syntaktischen Analyse der Statements in einem Dokument unterstützt die Bibliothek auch die semantische Überprüfung gegenüber einem RDF-Schema. Die RSSDB ist eine Datenbank zur Speicherung von RDF-Beschreibungen auf Basis einer objekt-relationalen Datenbank. Die objekt-relationale Repräsentation der RDF-Daten wird automatisch auf Grundlage definierter RDF-Schemata generiert. Durch die Kombination des VRP und der RSSDB wurde eine Architektur zu Speicherung mit Konsistenzprüfung verwirklicht. Darauf aufgesetzt ist eine Implementierung der RQL. Die Anfragesprache RQL wird im Abschnitt 5.2.7 näher vorgestellt.

Sesame

Sesame[2] ist eine Architektur zur Speicherung und Anfrage von RDF-Daten sowie von Schema-Informationen. Entwickelt wird Sesame von *Aidministrator Nedeerland bv* im Zusammenhang mit dem EU-Projekt On-To-Knowledge. Die Architektur wurde so konzipiert, dass sie auf verschiedenen Datenbankmanagementsystemen (relational oder objektorientiert) aufbauen kann. Die derzeitige Implementation basiert auf PostgreSQL. Anfragesprache von Sesame ist – wie bei der RDFSuite – RQL. Ein detaillierter Architekturüberblick ist in [BrKa01b] zu finden.

rdfDB (JrdfDB)

Eine sehr einfache, „open-source" Datenbank ist *rdfDB* von R. V. Guha [Guha00]. Die Implementierung von rdfDB ist in C++ geschrieben und Client-Server-basiert. Für die internen Verwaltung der RDF-Daten wird Sleepycat's Berkeley DB verwendet. Zur Datenbankanbindung existieren Schnittstellen für C, Perl und Java (JrdfDB[3]). Anfragesprache der RDF-Datenbank ist rdfDB Query (siehe Abschnitt 5.2.7).

Neben den hier aufgeführten Datenbanken existieren weitere. Ein Überblick gibt das W3C unter *http://www.w3.org/2001/05/rdf-ds/DataStore*.

[1] RDFSuite. http://139.91.183.30:9090/RDF/index.html

[2] Aidministrator Nederland bv: Sesame. http://sesame.aidministrator.nl/

[3] JrdfDB. http://4xt.org/downloads/JrdfDB/

Problem der vorhandenen RDF-Datenbanken ist zu meist deren Verfügbarkeit. Viele befinden sich noch im Entwicklungsstadium, stellen lediglich Online-Demonstrationen zur Verfügung oder bestehen derzeit nur aus Teilkomponenten. Andere hingegen bieten nur eingeschränkte Möglichkeiten zur Kopplung mit dem Retrieval Server oder zur Anfrageformulierung.

Im Folgenden werden verschiedene Ansätze der Speicherung von RDF-Daten vorgestellt und diskutiert. Ziel ist die Selektion einer geeigneten Speicherungsform der RDF-Daten für RDF-Content-Repository der XPEA-Architektur.

5.1.2 Anforderungen an eine RDF-Datenbank

Bevor auf verschiedene Architekturansätze eingegangen werden soll, wird ein Anforderungskatalog für eine resultierende RDF-Datenbank aufgestellt und erläutert. Bei der Speicherung der RDF-Daten sollen folgenden Kriterien erfüllt sein:

- **Skalierbarkeit**: Die Datenbank muss RDF-Daten persistent speichern können. Hierbei sollte eine effiziente Verwaltung kleiner, mittlerer sowie großer Datenmengen gewährleistet sein. Ferner soll die Möglichkeit gegeben sein, mehrere RDF-Modelle zu speichern.
- **Anfrageoperationen**: Für die Nutzung der RDF-Daten muss die Datenbank über eine geeignete, generische Anfragesprache verfügen. Die Formulierung simpler Anfragen soll einfach sein, komplizierte Anfragen ermöglicht werden. Das Thema RDF-Anfragesprachen wird im Abschnitt 5.2 genauer diskutiert.
- **Effizienz und Optimierbarkeit**: Die Datenbank soll eine effiziente Anfragebearbeitung gewährleisten. Diese kann unter anderem durch Indizierungsmechanismen des Datenbanksystems unterstützt werden. Zudem sollte das Datenbanksystem eine Optimierung von Anfragen bieten, wie es bei herkömmlichen DBMS der Fall ist.
- **Datenunabhängigkeit**: Neben der physikalischen und logischen Datenunabhängigkeit, die durch die 3-Ebenen-Architektur des zu Grunde liegenden DBMS geben sein sollte, ist eine Datenunabhängigkeit zwischen dem RDF-Modell und der Abbildung dessen in der Basisdatenbank gefordert. Diese schließt die Unabhängigkeit sowohl vom Datenbankmodell als auch vom Datenbankschema ein.
- **Organisation**: Der mit der Speicherung der RDF-Daten verbundene Overhead durch zusätzliche IDs oder Ähnlichem soll so gering wie möglich ausfallen. Besonders im Zusammenhang mit der relationalen Speicherung ist es notwendig, einen geeigneten Kompromiss zwischen der Duldung von Redundanzen innerhalb der Basisdatenbank, der Normalisierung des Datenbankschemas und der Inkaufnahme von natürlichen Verbunden für die Anfragebearbeitung zu finden.

Neben den speziell auf RDF-Daten zugeschnittenen Anforderungen soll die Datenbank natürlich auch übliche Leistungsmerkmale von Datenbanksystemen, wie den Mehrbenutzerbetrieb, Datensicherheit, Datenintegrität und Datenschutz bieten. Ein allgemeiner Architekturansatz ist daher, ein vorhandenes Datenbankmanagementsystem zu nutzen und dieses um eine Schnittstelle zur Speicherung und Anfrage von RDF-Daten zu erweitern.

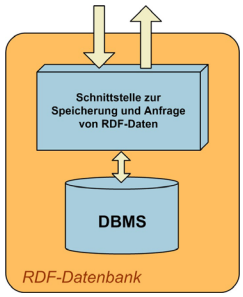

Abbildung 5.1: Realisierungsansatz für eine RDF-Datenbank

Mit der dargestellten Architektur ist es möglich, die Leistungsmerkmale des zu Grunde liegenden Datenbanksystems zu übernehmen und um weitere RDF-spezifische Merkmale zu ergänzen. Die Aufgabe besteht nun darin, das RDF-Modell in die der Datenbank gearteten Struktur geeignet abzubilden.

5.1.3 Klassifikation der Ansätze

Für die Speicherung von RDF-Daten gibt es verschiedene Ansätze. Diese beruhen auf verschiedenen Datenbankmodellen sowie unterschiedlichen Datenbankschemata und lassen sich hinsichtlich der Art der Speicherung und der Validierung anhand des RDF-Schemas klassifizieren. Dieser Abschnitt gibt eine Darstellung der Klassifikationen. Daran anschließend werden verschiedene Realisierungen auf Basis des objektorientierten, des objekt-relationalen und des relationalen Datenbankmodells vorgestellt. Der Focus der Betrachtung liegt hierbei auf der Speicherung von RDF-Daten in relationalen Datenbanksystemen.

Generische Speicherung vs. schemagebundene Speicherung

Ein Ansatz zur relationalen Speicherung des in Abbildung 5.2 dargestellten RDF-Modells ist beispielsweise die Eigenschaften als Relationen mit den Attributen *Subject* und *Object* im Datenbankschema zu modellieren. Das Schema umfasst für das gegebene Beispiel drei Relationen (siehe Abbildung 5.3).

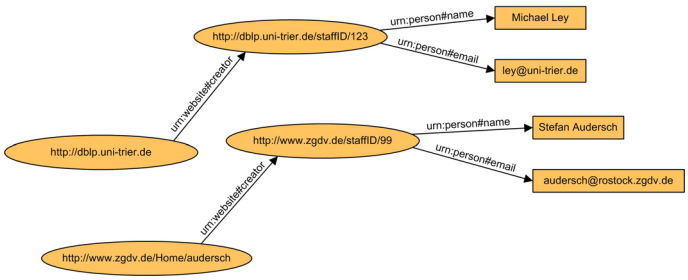

Abbildung 5.2: RDF-Modell

urn:person#eMail	
Subject	**Object**
http://dblp.uni-trier.de/staffID/123	*ley@uni-trier.de*
http://www.zgdv.de/staffID/99	*audersch@rostock.zgdv.de*

urn:person#name	
Subject	**Object**
http://dblp.uni-trier.de/staffID/123	*Michael Ley*
http://www.zgdv.de/staffID/99	*Stefan Audersch*

urn:website#creator	
Subject	**Object**
http://dblp.uni-trier.de	*http://dblp.uni-trier.de/staffID/123*
http://zgdv.de/Home/audersch	*http://www.zgdv.de/staffID/99*

Abbildung 5.3: Schemagebundene Speicherung des RDF-Modells aus Abbildung 5.2

Problematisch bei der Modellierung eines geeigneten Datenbankschemas ist jedoch, dass die vorkommenden Relationen erst durch die RDF-Modelle selbst oder durch ein gegebenes RDF-Schema definiert werden. Das würde heißen, dass sich das relationale Datenbankschema bei unbekanntem RDF-Schema abhängig von den hinzugefügten RDF-Statements gegebenenfalls ändern müsste. Eine derartige Modellierung des Datenbankschemas kann also nur in Erwähnung gezogen werden, wenn das RDF-Schema bekannt und keinen Änderungen unterworfen ist. Wird unter gegebenen Voraussetzungen eine dargestellte Speicherung gewählt, kann es darüber hinaus zu einer starken Fragmentierung der RDF-Daten durch eine hohe Anzahl von Relationen kommen. Ansätze zur Abhilfe bieten gegebenenfalls geeignete Transformationen der RDF-Schemata.

Mit Hilfe eines generischen Ansatzes zur Speicherung von RDF-Daten soll es möglich sein, jedes beliebige RDF-Modell in der Datenbank abzulegen. Eine einfache Art und Weise RDF-Daten generisch in einer relationalen Datenbank zu speichern, kann durch die Verwendung einer Relation

$$R_{tripel}\,(resource, property, value, valueIsResource)$$

verwirklicht werden. Diese Realisierung erlaubt zwar die generische Speicherung, hat jedoch zur Folge, dass sich alle Tripel in einer einzigen Relation befinden. Bei sehr großen Modellen führt dieses zwangläufig zu starken Einbußen bei der Indizierung und demzufolge zu einer schlechten Performance bei Anfrageoperationen auf der Relation. Wie die Daten bei relationaler Speicherung auf mehrere Relationen aufgeteilt werden können und welche Vor- und Nachtteile sich hierbei ergeben, ist dem Abschnitt 5.1.5 zu entnehmen.

Aufgrund der mangelnden Flexibilität bei der Nutzung einer schemagebundenen Speicherung wird im weiteren Verlauf lediglich die generische Speicherung von RDF-Daten in Betracht gezogen.

Mit Schemavalidierung vs. ohne Schemavalidierung

Die Speicherung von RDF-Daten lässt sich mit oder ohne Validierung der Daten realisieren. Bei der Schemavalidierung werden die in die Datenbank neu hinzugefügten Statements anhand eines oder mehrerer RDF-Schemata zuvor geprüft. Entsprechen die Daten dem Schema, werden sie in die Datenbank aufgenommen. Das RDF-Schema fungiert in diesem Fall als Integritätsbedingung, welches eine im Hinblick auf das Schema konsistente Datenhaltung gewährleistet.

In einer Realisierung mit Schemavalidierung ist es notwendig, neben Techniken zur Haltung von RDF-Daten auch Techniken zur Speicherung von RDF-Schemata und Verfahren zur Validierung der RDF-Daten anhand dieser Schemata bereitzustellen. Des Weiteren muss die

Datenbank nachträgliche Änderungen an den RDF-Schemata durch Verfeinerungen, Verbesserungen beziehungsweise Erweiterungen unterstützen.

Eine RDF-Datenbank mit Schemavalidierung bietet beispielsweise das *FORTH Institute of Computer Sience* mit der *RDFSuite* (siehe Abschnitt 5.1.1).

Entgegen den Vorteilen, die Metadatenverwaltung mit der Schemavalidierung zu kombinieren, findet dieses in der näheren Betrachtung und speziell in der Implementierung keine weitere Berücksichtigung. Hierfür sprechen verschiedene Argumente:

- Ist ein RDF-Schema Veränderungen unterworfen, treten spezielle Probleme auf. Werden bei der Schemaevolution beispielsweise neue Eigenschaften definiert, so stellt sich die Frage, wie bei vorhandenen RDF-Modellen in der Datenbank diesbezüglich vorgegangen werden soll. Gleiches gilt für Änderungen von Eigenschaften, deren Wert zuvor ein Literal war und nun eine Ressource ist. Noch komplexer sieht die Situation bei Manipulationen der Klassen-Unterklassen-Beziehungen aus.
- Die *RDF Schema Specification* befindet sich derzeit im Status „*Candidate Recommendation*“ und kann daher noch nicht als normative Referenz angesehen werden. Des Weiteren bietet *RDF Schema* gegenwärtig nicht die gewünschte Ausdrucksstärke. So lässt sich beispielsweise im RDF-Schema der Typ eines Containers nicht festgelegen.
- Oftmals sind für die Beschreibung von RDF-Modellen gar keine Schemata vorhanden. Dieses liegt unter anderem auch an der fehlenden Aussagekraft der RDF-Schemata.
- Neben logischen Problemen existieren auch praktische Einschränkungen, wie Performanceeinbußen bei der Nutzung der Schemavalidierung als Integritätsbedingung. Es sind somit zusätzliche Bearbeitungsschritte beim Einfügen neuer RDF-Daten in Kauf zunehmen.
- Die XPEA-Architektur verlangt lediglich die Speicherung einer verhältnismäßig übersichtlichen Menge von RDF-Daten, deren Konsistenz bereits durch die Applikation gegeben ist. Die Schemavalidierung neu hinzugefügter Daten erfordert hierbei nur zusätzliche Bearbeitungsschritte.

5.1.4 Speicherung in OODBMS

Objektorientierte Datenbankmanagementsysteme (OODBMS) zeichnen sich durch besonders geeignete Fähigkeiten zur Datenmodellierung aus. Dieses ermöglicht eine adäquate Modellierung der Anwendungsdaten im Datenbanksystem. Unterstützend wirken hierbei Konzepte wie Typkonstruktoren, Objektidentitäten, Klassen, Beziehungen oder Strukturvererbungen. Das in Abbildung 5.4 dargestellte objektorientierte Datenbankschema verwendet verschiedene objektorientierte Konzepte und ermöglicht die Verwaltung von RDF-Daten entsprechend der *RDF Model and Syntax Specification*.

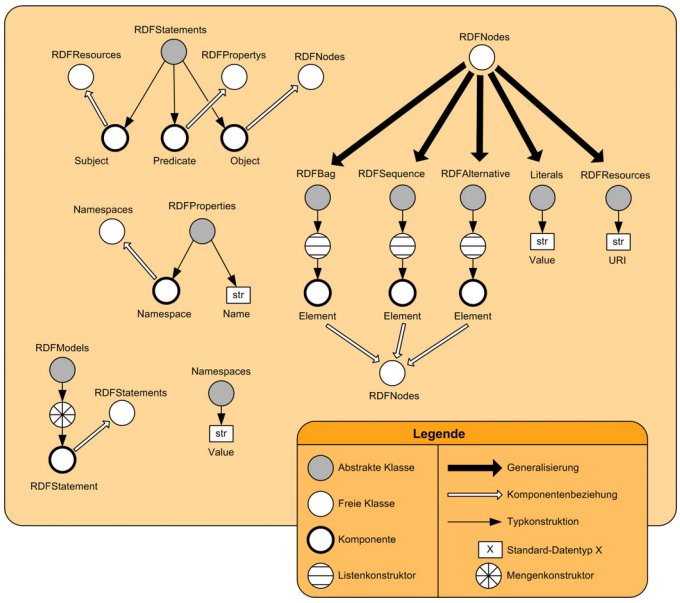

Abbildung 5.4: OODB-Schema zur Speicherung von RDF-Daten

Das abgebildete Datenbankschema verwirklicht die Speicherung mehrerer RDF-Modelle und ermöglicht die direkte Abbildung von Containern.

Bei Bearbeitung von Anfragen auf dem Schema muss das Datenbanksystem in der Regel auf mehrere Objekte über Klassen-Komponentenklassen-Beziehungen hinweg zugreifen. Zur Verbesserung der Anfrageperformance ist es empfehlenswert, die Anfragen durch so genannte Pfadindexe oder Multiindexe zu unterstützen.

Neben konzeptuellen Erweiterungen bieten OODBMS die Vermeidung des Impedance Mismatch in Kooperation mit objektorientierten Programmiersprachen. Dieser Fakt wirkt sich gerade im Hinblick der Entwicklung einer RDF-Datenbank in einer objektorientierten Programmiersprache wie Java vorteilhaft aus.

Entgegen den genannten positiven Eigenschaften soll in dieser Arbeit jedoch nicht weiter auf die Realisierung einer Metadatenverwaltung mit Hilfe eines OODBMS eingegangen werden.

5.1.5 Speicherung in RDBMS

Dieser Abschnitt gibt einen Überblick über Verfahren zur Speicherung von RDF-Daten in relationalen Datenbankmanagementsystemen (RDBMS). Hierbei sollen verschiedene Wege der relationalen Speicherung diskutiert werden, mit der Zielstellung den besten Ansatz zu identifizieren. Ferner sollen Lösungen gefunden werden, die für bestimmte Anforderungen besonders geeignet sind.

Bevor Realisierungen auf einzelnen relationalen Datenbankschemata erörtert werden, soll zunächst ein Vorschlag in einem semantischen Datenbankmodell (SDM), dem Extended Entity-Relationship-Model (EER-Model) gegeben werden (siehe Abbildung 5.5). Das Modell ist dienlich in der Entwurfsphase des relationalen Schemas.

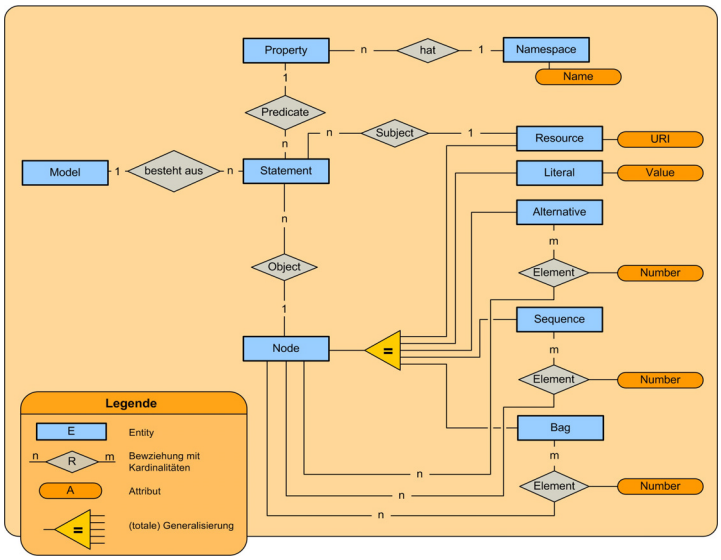

Abbildung 5.5: EER-Schema zur Speicherung von RDF-Daten [1]

Die Speicherung von RDF-Daten kann auf Basis verschiedener relationaler Datenbankschemata verwirklicht werden, die sich bezüglich der Kriterien Skalierbarkeit, Effizienz und Organisation voneinander unterscheiden. Im Folgenden werden drei unterschiedliche Ansätze vorgestellt.

Datenbankschema A

Das Datenbankschema beruht auf der RDF-Spezifikation und ermöglicht die Abbildung von RDF-Modellen in einer relationalen Datenbank. Ziel ist zusätzlich die Vermeidung von Redundanzen. Eine Möglichkeit dieses zu erreichen, ist die mehrfache Speicherung von Ressourcen, Properties usw. zu umgehen, indem diese über IDs referenziert werden. In der Regel ist der Speicherbedarf der IDs wesentlich kleiner, so dass bei mehrfacher Verwendung gleicher Elemente eine geringe Datenmenge anfällt.

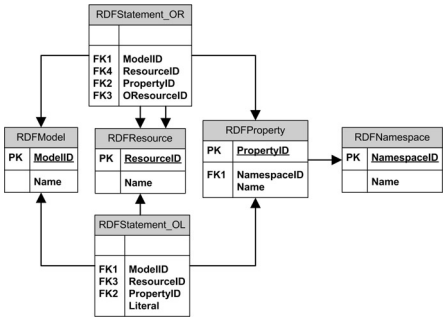

Abbildung 5.6: Datenbankschema A zur Speicherung von RDF-Daten

[1] Im dargestellten EER-Schema sind Container (Alternative, Sequence, Bag) als eigenständige Entities aufgeführt. Es ist ebenso möglich diese als Ressource zu verwalten, analog zur RDF-Spezifikation. Ein Container wird hierbei durch eine Ressource dargestellt, die eine interne URI bekommt und anhand der Property rdf:type typisiert wird. Die Elemente des Containers sind dann die Werte der Eigenschaften rdf:_1 … rdf:_n. Demnach können die Entities Alternative, Sequence und Bag im EER-Schema auch vernachlässigt werden, wenn eine separate Speicherung dieser nicht gewünscht ist.

Die Objekttypen Ressource und Property werden in jeweils separaten Relationen gespeichert, wobei der Namespace der Properties in eine weitere Relation ausgelagert ist. Die Speicherung der Statements erfolgt in der Tabelle *RDFStatements_OL* beziehungsweise *RDFStatemens_OR*, je nachdem ob es sich bei dem Objekt des Statements um eine Ressource oder ein Literal handelt. Mit der Tabelle *RDFModel* ist es möglich mehrere RDF-Modelle unter verschiedenen Namen zu verwalten.

Typische Anfragen an die Metadatenverwaltung der XPEA-Architektur sind:

- *„gebe mir alle zu einer Ressource X zugehörigen Properties und Values“* oder
- *„gebe mir zur Ressource X den Wert der Property Y“.*

Listing 5.1 zeigt die SQL-Anfrage für das zuerst genannte Beispiel.

```
  SELECT  RDFModel.Name as ModelName,
          concat(RDFNamespace.Name, RDFProperty.Name) as Property,
          RDFStatement_OL.Literal as Value,
          "f" as ValueIsResource
  FROM    RDFResource, RDFProperty, RDFNameSpace, RDFLiteral,
          RDFStatement_OL, RDFModel
  WHERE   RDFStatement_OL.ModelID = RDFModel.ModelID AND
          RDFStatement_OL.ResourceID = RDFResource.ResourceID AND
          RDFStatement_OL.PropertyID = RDFProperty.PropertyID AND
          RDFProperty.NamespaceID = RDFNamespace.ID AND
          RDFModel.Name = "RDF-Model_1"
          RDFResource.Name = "online.Europa-MV.de"
UNION
  SELECT  RDFModel.Name as ModelName,
          concat(RDFNamespace.Name, RDFProperty.Name) as Property,
          r2.Name as Value,
          "t" as ValueIsResource
  FROM    RDFResource as r1, RDFProperty, RDFNamespace, RDFResource as r2,
          RDFStatement_OR, RDFModel
  WHERE   RDFStatement_OR.ModelID = RDFModel.ModelID AND
          RDFStatement_OR.ResourceID = r1.ResourceID AND
          RDFStatement_OR.PropertyID = RDFProperty.PropertyID AND
          RDFProperty.NamespaceID = RDFNamespace.ID AND
          RDFStatement_OR.OResourceID = r2.ResourceID AND
          RDFModel.Name = "RDF-Model_1"
          r1.Name = "online.Europa-MV.de"
```

Listing 5.1: Anfrage von RDF-Statemens beim Datenbankschema A

Die SQL-Anfrage macht deutlich, dass für eine einfache Anfrage, bei der die Eigenschaften einer Ressource zurückgegeben werden sollen, zur Bearbeitung der Teilanfragen vier beziehungsweise fünf Verbunde notwendig sind. Auch bei der Annahme, dass nicht alle Eigenschaften, sondern nur ein bestimmter Eigenschaftstyp verlangt ist, verringert sich die Anzahl der Verbunde nicht.

Der Verbund, eine binäre Operation, ist eine der wichtigsten Basisoperatoren im relationalem Datenbankmodell. Er wird durch spezielle Algorithmen besonders unterstützt, erfordert jedoch einen gewissen Aufwand. Dessen Komplexität beträgt im idealen Fall $O(n + m)$, kann aber auch bis zu $O(n \times m)$ im schlechtesten Fall sein und sollte daher in der Performancebetrachtung nicht unterschätzt werden.

RDF-Modelle sind gerichtete Graphen. Die Ressourcen und Eigenschaftswerte entsprechen den Knoten und bei den Eigenschaftstypen handelt es sich um gerichtete, bezeichnete Kanten. Ein Tripel besteht aus einer Ressource, dem zugehörigem Eigenschaftstyp sowie dessen Eigenschaftswert und stellt somit ein RDF-Statement dar. Die Speicherung von RDF-Modellen kommt daher der Abbildung von gerichteten Graphen in einer Datenbank gleich. Die beiden

folgenden Schemavorschläge basieren auf der Darstellung der RDF-Modelle durch eine Menge von Tripel, die auf verschiedene Weise relational gespeichert werden können.

Datenbankschema B

Im Gegensatz zum zuvor beschriebenen Schema werden bei diesem Datenbankschema die RDF-Daten in nur einer Relation verwaltet. Die Tripel des RDF-Graphen sind in den Attributen *Resource*, *Property* und *Value* gespeichert. Ob es sich bei dem Eigenschaftswert um eine Ressource oder ein Literal handelt, ist in *ValueIsResource* festgelegt. Zur Unterstützung mehrerer RDF-Modelle dient das Attribut *ModelName*.

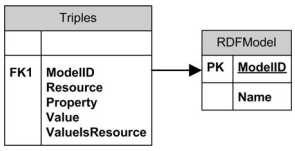

Abbildung 5.7: Datenbankschema B zur Speicherung von RDF-Daten

Die Ermittlung der Eigenschaften einer Resource kommt nun, gegenüber dem vorherigen Datenbankschema, mit einem Verbund aus. Jedoch folgt durch die simple Haltung der Daten das Vorhandensein vieler Redundanzen, da die Informationen von Knoten beziehungsweise Kanten mehrfach in der Relation auftreten. Gerade bei der Verwaltung voluminöser RDF-Daten führt dies zu großem Speicherbedarf der Relation. Durch die daraus resultierende Verteilung der Relation auf viele Sekundarspeicherseiten kann eine akzeptable Bearbeitungszeit von Anfragen, ohne geeignete Indizierung nicht gewährleistet werden. Wie eine solche Indizierung aussehen könnte, wird in Abschnitt 5.1.8 erörtert.

Neben der Wahl einer geeigneten Indizierung kann für die Verwaltung mehrerer RDF-Modelle die Verteilung der RDF-Daten auf mehre Relationen nützlich sein. Hierbei könnte pro RDF-Modell eine separate Relation gewählt werden.

Datenbankschema C

Verglichen mit dem vorherigen Ansatz werden bei diesem Datenbankschema die Eigenschaftswerte in eine separate Relation ausgelagert. Eine Verknüpfung der Tabellen *Triples* und *RDFValue* erfolgt über das Attribut *ValueID* beziehungsweise *TriplesID*.

Die Verwendung des Schemas bietet sich bei zwei unterschiedlichen Situationen an:

- Treten bei den RDF-Modellen innerhalb der Statements mehrere Objekte häufiger auf, so kann deren wiederholte Speicherung vermieden werden. Speziell wenn es sich bei den redundanten Objekten um speicherintensive Literale handelt und/oder auf viele Ressourcen sehr oft referenziert wird, ist die Speicherung in dieser Form zu empfehlen. Die *ObjectID* in der *Value*-Relation muss hierbei eindeutig sein.

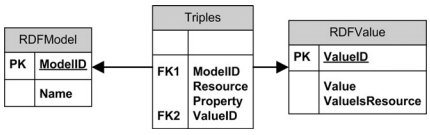

Abbildung 5.8: Datenbankschema C (1) zur Speicherung von RDF-Daten

- Auf der anderen Seite kann auch die wiederholte Speicherung von identischen Subjekt-Prädikat-Kombinationen umgangen werden. Hierbei ist die duplikatfreie Verwendung der *TripleID* in der Tabelle *Triples* sicherzustellen.

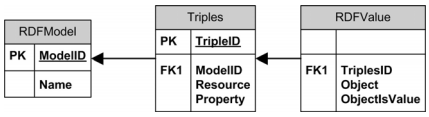

Abbildung 5.9: Datenbankschema C (2) zur Speicherung von RDF-Daten

Eine Anfrage für das Lesen von RDF-Statements bedarf für deren Bearbeitung zwei Verbunde. Wie auch beim vorherigen Datenbankschema kann die Bearbeitungszeit von Anfragen durch eine geeignete Wahl von Indizierungen und die Verteilung der RDF-Modelle auf separate Relationen optimiert werden. Für die Verteilung sind dementsprechend zwei Relationen pro Modell notwendig.

Die hier erörterten Formen der persistenten Speicherung von RDF-Daten in relationalen Datenbanken stellen nur einige Möglichkeiten der Speicherorganisation dar. Neben diesen können eine Vielzahl weiterer Datenbankschemata aufgezählt werden. Problem ist allerdings immer wieder, dass der Wert einer Eigenschaft ein Literal oder eine Ressource sein kann.

Weitere Ansätze der relationalen Speicherung von RDF-Daten sind in [Meln00] zu finden.

5.1.6 Speicherung in ORDBMS

Eine weitere Gruppe von Datenbanksystemen bilden die objektrelationalen Datenbankmanagementsysteme (ORDBMS). Ein Ansatz zur Speicherung von RDF-Daten in einem ORDBMS soll hier nur kurz dargestellt werden.

Objektrelationale Datenbanksysteme stellen eine Erweiterung relationaler Systeme um objektorientierte Konzepte dar. Zu den erweiterten Konzepten gehören unter anderem Typen, Typkonstruktoren oder auch Objektidentitäten im Strukturteil des Datenbankmodells. Diese können, je nach Entwicklungstand des verwendeten ORDBMS, für die Verwaltung von RDF-Daten genutzt werden. In Abbildung 5.10 findet sich beispielsweise die Darstellung einer NF^2-Relation mit RDF-Daten, die so durch Verwendung von Typkonstruktoren in einem objektrelationalen Datenbankschemas abgebildet werden kann.

Resource	Properties		
	PropertyType	Values	
		Value	ValueIsResource
.../staffID/99	.../name	Stefan Audersch	False
	.../phone	0172-1373835	False
		0381-9469183	False
.../diplomaID/1199	.../creator	.../staffID/99	True
...	...	...	...

Abbildung 5.10: RDF-Daten in einer NF^2-Relation

Das Schema erlaubt implizit die Speicherung mehrerer Eigenschaften zu einer Ressource, denen wiederum mehrere Werte zugeordnet werden können.

Die RDF-Datenbank der RDFSuite (siehe Abschnitt 5.1.1) verwendet beispielsweise eine objektrelationale Datenbank zur Verwaltung der RDF-Daten. Nähere Informationen zu dem dabei verwendeten Datenbankschema sind in [ACK+01] nachzulesen.

5.1.7 Speicherung in XML-Datenbanken

Da RDF in XML-Syntax darstellbar ist, besteht neben der Verwaltung von RDF-Daten in relationalen, objektorientierten und objekt-relationalen DBMS die Möglichkeit, RDF-Modelle in XML-Datenbanken zu speichern. Im Gegensatz zu den genannten DBMS ist es hierbei nicht notwendig, die RDF-Daten in die jeweilig geartete Datenstruktur zu konvertieren. Die Software AG bietet mit dem Informationsserver Tamino[1] beispielsweise ein Datenbanksystem, das XML-Dokumente in nativer Form speichert und somit auch RDF in XML-Syntax. Anfragen auf XML-Dokumente sind mit XPath oder XQL möglich.

Werden RDF-Daten in XML-Form gespeichert, ist darauf zu achten, dass für ein und dasselbe RDF-Modell verschiedene Darstellungen in XML existieren. Das Modell aus Abbildung 5.2 (Seite 42) kann zum Beispiel durch die folgenden zwei XML-Ausdrücke beschrieben werden:

```
<rdf:RDF>
  <rdf:Description about="http://dblp.uni-trier.de">
  <s:creator>
    <rdf:Description about="http://dblp.uni-trier.de/staffID/123">
      <v:Name>Michael Ley</v:Name>
      <v:Email>ley@uni-trier.de</v:Email>
    </rdf:Description>
  </rdf:Description>
</rdf:RDF>
```

oder

```
<rdf:RDF>
  <rdf:Description about="http://dblp.uni-trier.de">
    <s:creator rdf:resource="http://dblp.uni-trier.de/staffID/123"/>
  </rdf:Description>
  <rdf:Description about="http://dblp.uni-trier.de/staffID/123">
      <v:Name>Michael Ley</v:Name>
      <v:Email>ley@uni-trier.de</v:Email>
  </rdf:Description>
</rdf:RDF>
```

Speziell die verschiedenen Darstellungsformen identischer RDF-Modelle (im Beispiel wurden hier nur zwei angegeben) müssen bei der Speicherung von RDF-Daten in XML-Datenbanken berücksichtigt werden. Wie in Abschnitt 5.2.5 noch zu sehen ist, gilt dieses ebenso bei der Verwendung von XML-Anfragesprachen. Einziger Ausweg hierfür ist die Speicherung von RDF-Daten in „normierten“ XML-Dokumenten. Aus den geschilderten Gründen stellt die effiziente Speicherung von RDF-Daten auf Basis von XML-Datenbanken eine eher ungünstige Wahl dar.

Fazit Focus dieser Arbeit ist die universelle Metadatenverwaltung auf Basis des relationalen Datenbankmodells. Im Abschnitt 5.1.5 wurden hierzu drei verschiedene Ansätze diskutiert.

Der XPEA-Architektur liegt nur eine verhältnismäßig geringe Menge von Metadaten-beschreibungen zu Grunde, so dass eine gegebenenfalls redundante Speicherung von Informationen im Datenbankschema B durchaus akzeptabel ist. Da Änderungen auf dem

[1] Software AG: Tamino. http://www.softwareag.com/tamino/

Metadatenmodell selten und unter kontrollierten Bedingungen erfolgen, ist die Gefahr für Änderungsanomalien in dem Datenbankschema gering. Das Datenbankschema zeichnet sich zudem durch dessen Einfachheit aus. Es ermöglicht einen unkomplizierten Zugriff auf die RDF-Statements. Unter der der Bedingung, dass für mehrere RDF-Modelle jeweils separate Relationen verwendet werden, sind keine Verbunde zur Ermittlung eines RDF-Statements notwendig. Hierbei ist jedoch zu beachten, dass bei navigierenden Anfragen über den Graphen des RDF-Modells, zum Beispiel bei Pfadausdrücken, Verbunde benötigt werden. Deren Anzahl entspricht in diesem Fall der Pfadtiefe. Dieser Umstand tritt aber auch bei den anderen Ansätzen auf, wobei sich hier die Zahl der Verbunde entsprechend vervielfacht.

Für eine adäquate, RDF-basierte Metadatenverwaltung wird das Datenbankschema B zur Speicherung gewählt. Da für die Metadatenverwaltung der XML-Framework-Architektur nur ein RDF-Modell notwendig ist, wird nur eine Relation verwendet.

5.1.8 Indizierung

Um eine Anfrage effizient zu beantworten, sollte die Datenbank möglichst nur die tatsächlich benötigten Datensätze vom Sekundärspeicher laden. Hierbei werden diese durch geeignete Indexstrukturen unterstützt. Einen umfassenden Überblick zu verschiedenen Indizierungstechniken als auch Dateiorganisationen wird in [HeSa99] gegeben.

Der folgende Abschnitt gibt Aufschluss über geeignete Indizierungen basierend auf der im vorherigen Abschnitt gewählten relationalen Speicherung.

Wie kann eine adäquate Indizierung bei der anvisierten Speicherung von RDF-Daten nun aussehen? Bevor diese Frage beantwortet werden kann, ist es notwendig, die primär an das Datenbanksystem gestellten Anfragen zu analysieren. Diese differieren von Anwendung zu Anwendung, so dass eine allgemein gültige Lösung für die optimale Indizierung nicht gegeben werden kann.

Typische Anfragen an die RDF-Datenbank

Im Anwendungsszenario Europa-MV werden RDF-Daten zur Beschreibung von bereitgestellten Datenbanksystemen, den darin gespeicherten Daten oder zur Klassifizierung von Endgeräten benötigt. Die Informationen bilden die Grundlage für Datenretrieval als auch Datentransformation. Häufig gestellte Anfragen sind:

- *Welche Relationen beinhaltet die Datenbank X und wie sind diese miteinander verbunden?*
- *Ist das Attribut X durchsuchbar?*
- *Welcher Treiber und welche URL sind zur Kommunikation mit der Datenbank X notwendig?*
- *Welche Datenbanken sind für die Applikation verfügbar?*
- *Wie sieht die Hierarchie einer Dimension X des OLAP-Würfels Y aus?* [1]
- *Zu welchem OLAP-Würfel passt des Schlüsselwort X?*[1]

[1] Das für die Anfrage zugrunde liegende RDF-Modell ist in Abschnitt 7.3 beschrieben.

Es ist festzustellen, dass bei zahlreichen Anfragen das Subjekt, das Prädikat oder auch beide gegeben sind. Einige Anfragen setzen das Objekt voraus. Eine ideale Indizierung sollte also die Selektion für gegebene Subjekte, Prädikate oder die Kombination aus beiden grundsätzlich unterstützen. Eventuell sollten auch Suchabfragen auf Objekte effizient ermöglicht werden.

Herkömmliche Indexe, welche überwiegend in RDBMS zu finden sind, unterstützen gewöhnlich nur eindimensionale Anfragen. Zwar können diese auch über einen zusammengesetzten Schlüssel (*composite index*) definiert werden, jedoch optimieren sie die Anfragen nur, wenn alle Schlüsselattribute (oder die ersten) gegeben sind. Dieses hängt damit zusammen, dass die betreffenden Attributwerte einfach miteinander verkettet werden und darüber indiziert wird. Der Index ist also primär nach dem ersten Attribut sortiert. Wird ein Primärindex verwendet, so werden die Daten in dieser Reihenfolge auf dem Sekundärspeicher geclustert. Das erlaubt einen effizienten Zugriff bei *partial-match*-Anfragen über den Primärindex, jedoch liegen die Daten bezüglich der verwendeten Sekundärindexe verstreut vor.

Mehrdimensionale Speichertechniken

Abhilfe können mehrdimensionale Speichertechniken bieten. Sie kombinieren die Speicherorganisation der Datensätze mit der Indizierung und bieten neben der Unterstützung von Anfragen, bei denen alle dafür definierten Attribute gegeben sind, auch Hilfe bei teilweiser Angabe. Zu den Techniken gehören mehrdimensionale Baumverfahren, mehrdimensionales Hashen und Grid-Files. Grundgedanke ist dabei, die Datensätze auf einen mehrdimensionalen Raum abzubilden.

Eine Nutzung des UB-Trees, einem mehrdimensionalen Baumverfahren (vorgestellt in Anhang C), zur Strukturierung und Indizierung der Daten im gewählten Datenbankschema wird folgend diskutiert.

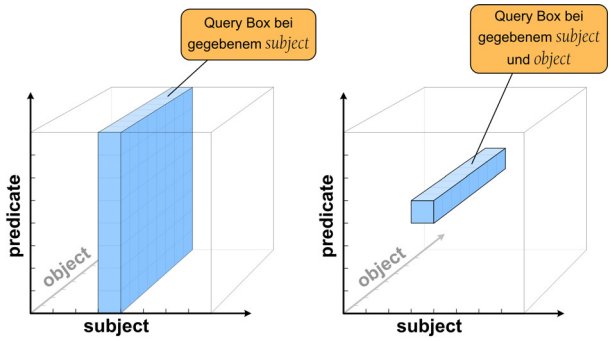

Abbildung 5.11: Query Box bei RDF-Anfrage

Die Verwendung des UB-Trees als mehrdimensionale Speichertechnik für die RDF-Datenbank erlaubt die nachbarschaftserhaltende Sekundärspeicherung von Tripeln der RDF-Graphen. Das Anwendungsszenario benötigt zwar kaum partial-match-Anfragen, für deren effiziente Bearbeitung sich der UB-Tree auszeichnet. Dennoch ist durch die Speichertechnik ebenfalls eine Unterstützung von exact-match-Anfragen gegeben. Bei deren Bearbeitung für einen definierten 3-dimensionalen Raum entartet die Query Box je nach Anzahl der gegebenen

Werte in eine Ebene oder eine Gerade (siehe Abbildung 5.11). Der Vorteil des UB-Trees liegt durchaus nicht im Bearbeitungsaufwand von eindimensionalen exact-match-Anfragen gegenüber eindimensionalen Indexen, dagegen unterstützt er aber ein- als auch mehrdimensionale Anfragen ohne dafür diverse Indexe zu verwenden. Dieses erfordert einen deutlich geringeren Aufwand für die Indexpflege und weniger zusätzliche Sekundärspeicher. Zudem wird eine nachbarschaftserhaltende Speicherung aus mehrdimensionaler Sicht geboten, welche sich beispielsweise bei der Anfrage auf alle Eigenschaften eines Endgerätes oder auch bei Anfragen auf alle Tabellen einer gegebenen Datenbank im RDF-Modell des Anwendungsszenarios durchaus bezahlt macht. Es empfiehlt sich, die Indizierung über die Attribute *Resource* und *Property* vorzunehmen. Sind vermehrt Anfragen über gegebene Eigenschaftswerte erforderlich, sollte die Schlüsselmenge um das Attribut *Value* erweitert werden.

Der UB-Tree ist derzeit lediglich in das kommerzielle Datenbanksystem TransBase HyperCube[1] integriert und steht somit leider nicht für die Nutzung bei der RDF-Metadatenverwaltung dieser Diplomarbeit zur Verfügung.

Eindimensionale Speichertechniken

Da eine Unterstützung von Anfragen auf die RDF-Daten im Basisdatenbanksystem nicht durch eine mehrdimensionale Speichertechnik gegeben werden kann, ist es notwendig, vorhandene eindimensionale Zugriffstrukturen zu nutzen. Ausschlaggebend für die Wahl einer optimalen Indizierung sind auch hier, die in der Anwendung vorkommenden Anfragen. In der Regel sind diese beim Anwendungsszenario durch gegebene Ressourcen und/oder Eigenschaftstypen gekennzeichnet. Folgend werden zwei Realisierungen zur Indizierung der Basisdatenbank angegeben.

Indizierung mit Mehr-Attribut-Index

Die erste Realisierung basiert auf der Verwendung eines zusammengesetzten Schlüssels (*composite index*). Hierbei ist zu beachten, dass nicht alle DBMS deren Nutzung bei unvollständiger Angabe der Attribute unterstützen. Ist dieses jedoch gegeben, kann ein dünnbesetzter Primärindex auf Basis eines B^*-Baumes mit dem zusammengesetzten Sekundärschlüssel aus den Attributen *Resource* und *Property* gewählt werden. Das häufiger gegebene Attribut sollte an erster Stelle stehen. Die Verwendung eines Primärschlüssels ist nicht möglich, da die Duplikatfreiheit für den zusammengesetzten Schlüssel nicht garantiert ist. Durch die Wahl des Primärindex ist ein gleichmäßiger Zugriff auf die einzelnen Datensätze sowohl bei gegebener Ressource und Eigenschaftstyp als auch bei gegebenem erstem Attribut, mit einer Komplexität von $O(\log_m(n))$ gewährleistet. Ergibt eine Selektion mehrere Ergebnisse, so kann der Primärindex die Dateiorganisation der internen Relation nutzen. Zur Beschleunigung von Anfragen mit Selektion auf dem zweiten Attribut, ist darauf ein B^+-Baum als Sekundärindex zu empfehlen. Die Datenzugriffe bei *exact-match*-Anfragen entsprechen hierbei ebenso der oben genannten Komplexität. Bei Anfragen mit mehreren Ergebnissen kann allerdings die interne Sortierung der Datensätze nicht verwendet werden, da diese nicht nach dem Sekundärindex geclustert sind. Für die zusätzliche Optimierung von Anfragen bei

[1] TransAction Software Gmbh: Transbase HyperCube. http://www.transaction.de/engl/prod/hypercube.htm

gegebenem Wert des *Value*-Attributs kann darauf ein weiterer Sekundärindex definiert werden. Es sei jedoch darauf hingewiesen, dass bei Verwendung mehrerer Indexe dessen Anpassungsaufwand bei Änderungsoperationen der Basisdaten zunimmt. Obendrein kann sich die Indizierung des *Value*-Attributs als negativer Einfluss herausstellen, da das Attribut größere Daten enthalten kann.

Indizierung mit mehreren Ein-Attribut-Indexen

Für die Beantwortung von Anfragen mit mehreren gegebenen Werten können mehrere Sekundärindexe, falls vorhanden, herangezogen werden. Für jeden Wert ermittelt der jeweilige Index eine Liste von TIDs (Tupel-Identifier). Die Schnittmenge dieser liefert die TIDs der Ergebnisdatensätze, die die gesamte Anfragebedingung erfüllen. Ist ein Index in Form eines Primärindex realisiert und bietet bei gestellter Anfrage eine hohe Selektivität, so kann ausschließlich dieser für die Anfragebearbeitung verwendet werden. Das DBMS lädt dazu alle Datensätze, die der Einschränkung des Attributwertes genügen, sequentiell und filtert diese mit der restlichen Bedingung nach. Der Primärindex sollte also über ein Attribut definiert werden, das eine niedrige Selektionskardinalität[1] aufweist und/oder dessen Wert häufig in Anfragen gegeben ist. So kann bei mehreren Ergebnisdatensätzen die Clusterung bezüglich des Primärindexes ausgenutzt werden.

Die genannten Eigenschaften sind beim Attribut *Resource* am besten gegeben, so dass darauf ein Primärindex definiert wird. Zur Indizierung des *Property*-Attributs ist ein Sekundärindex heranzuziehen. Für das Attribut *Object* gelten auch hier die im Abschnitt zuvor genannten Einschränkungen.

Fazit: Die Auswahl einer geeigneten Indizierung für das gewählte Datenbankschema ist abhängig von der Anfragecharakteristik und der gegeben Indexunterstützung des Basisdatenbanksystems. Bei der XPEA-Architektur erfolgen Anfragen an den Metadatenbestand in der Regel mit gegeben Werte für das *Resource*- und/oder *Property*-Attribut, so dass eine Unterstützung von Anfragen durch Indizierung auf diesen beiden Attributen gewünscht ist. Günstig wäre eine mehrdimensionale Speichertechnik. Die RDF-Datenbank der XPEA-Architektur baut auf das DBMS Oracle8i auf. In dieser Version steht keine mehrdimensionale Speichertechnik zur Verfügung. Die Verwendung eines zusammengesetzten Schlüssels zur Anfragebearbeitung erfolgt nur wenn alle Attribute des Schlüssels in der Anfrage gegeben sind. Für die Realisierung des RDF-Content-Repository werden daher zwei eindimensionale Indexe genutzt. Die Indizierung des *Resource*-Attributs erfolgt über einen geclusterten Index. Für das *Property*-Attribut wird ein Sekundärindex verwendet.

5.2 RDF-Anfragesprachen

Für die Verarbeitung der RDF-Daten ist das Vorhandensein einer Anfragesprache essentiell. Eine RDF-Anfragesprache sollte hierbei der strukturellen und inhaltsbasierten Suche dienen.

[1] Ist die Selektionskardinalität eines Attributs `A` niedrig, so ergibt sich bei Selektion $\sigma_{A=x}(r)$ über dem Attribut eine hohe Selektivität.

Es existiert bereits eine große Anzahl von Anfragesprachen. Das wohl bekannteste Beispiel ist SQL, die Standardanfragesprache für relationale Datenbanken. Wie im folgendem Abschnitt noch festzustellen ist, eignen sich Anfragesprachen herkömmlicher Datenmodelle aufgrund der Eigenheiten von RDF jedoch nicht oder nur bedingt.

In diesem Abschnitt werden Kriterien für eine RDF-Anfragesprache aufgestellt, eine Klassifizierung der Sprachen gegeben und verschiedene Vorschläge zu Anfragesprachen erörtert.

5.2.1 Kriterien für RDF-Anfragesprachen

Eine adäquate RDF-Anfragesprache sollte gewissen Kriterien und Anforderungen genügen, um ein effizientes Retrieval über vorhandene RDF-Beschreibungen zu ermöglichen.

In [HeSa97], [BrFH00] und [AbBS99] finden sich für unterschiedlichen Datenmodellen Kriteriensammlungen für Anfragesprachen. Hiervon ausgehend wird Kriterienkatalog aufgestellt, der speziell an die Eigenschaften des semistrukturierten Datenmodells von RDF angepasst ist:

- **Ad-hoc:** Eine Anfrage sollte formuliert werden können, ohne dafür ein vollständiges Programm schreiben zu müssen.
- **Deskriptiv:** In einer Anfrage ist zu formulieren „Was man haben will“ und nicht „Wie man an die gefragten Informationen kommt“. Die Navigation entlang des eigentlichen RDF-Graphen soll dabei nicht verhindert werden.
- **Effizient:** Anfragen sollen effizient zu bearbeitet werden.
- **Othogonal:** Sprachkonstrukte sollen in ähnliche Situationen auch ähnlich anwendbar sein.
- **Abgeschlossen:** Die Ergebnisdarstellung der Anfragen soll im gleichen Modell erfolgen, wie das der Anfragesprache zu Grunde liegende Datenmodell. Basiert die Anfragesprache also auf einer Menge von Tripeln, so soll auch das Ergebnis eine Menge von Tripeln sein. Dieses ermöglicht die Formulierung komplexer Anfragen, in denen Teilanfragen verwendet werden können.
- **Passend zum Datenmodell:** RDF-Daten stellen einen gerichteten Graphen dar. Unabhängig wie dieser in einem Datenmodell der Anfragesprache repräsentiert wird, müssen Grundoperationen auf dem Graphen ermöglicht werden, um eine Unterstützung für RDF-Modelle bereitzustellen.
- **Kompakt:** Die Anfragesprache sollte mit einem Minimum an Basisoperationen auskommen, ohne dabei deren Mächtigkeit zu verlieren.
- **Sicher:** Keine Anfrage mit korrekter Syntax darf in eine Endlosschleife geraten oder ein unendliches Ergebnis liefern.
- **Semantisch fundiert:** Um die Bearbeitung von Anfragen und die Optimierung dieser zu ermöglichen, ist eine genaue Definition der Semantik der Anfragesprache (gegenüber deren Syntax) erforderlich.
- **Vollständig:** Die Anfragesprache sollte Operationen wie Selektion, Projektion, Verbund und Vereinigung bieten.

- **RDF-Schema:** Die Anfragesprache muss den Zugang zu RDF-Schemabeschreibungen ermöglichen.
- **RDF-Schema korrekt:** Die Anfragesprache sollte die speziellen Eigenschaften von RDF Schemata unterstützen. So sind beispielsweise *rdfs:subClass* oder *rdfs:subProperty* transitiv, was bei der Semantik der Anfragesprachen zu berücksichtigen ist.

Als optionales Kriterium wäre hierbei noch zu nennen, dass neben der Abgeschlossenheit der Anfrageergebnisse auch die Möglichkeit zur Umstrukturierung oder die Darstellung des Endergebnisses in einem anderen Format wünschenswert ist. So sollte beispielsweise die Ausgabe einer Menge von Eigenschaftswerten bei einer Anfragesprache, die auf einem Graphen als Datenmodell arbeitet, möglich sein.

Bei der konkreten Betrachtung der einzelnen Anfragesprachen wird sich jedoch herausstellen, dass die Sprachen die genannten Kriterien in diesem Umfang meisst nicht erfüllen.

5.2.2 Klassifikation

In [BrFH00] findet sich eine Klassifikation, die die Anfragesprachen in die zwei folgenden Klassen gliedert:

- **SQL/XQL-Syntax basierter Ansatz**: Bei diesem Ansatz werden die RDF-Beschreibungen als gespeicherte Daten in einer relationalen oder XML-Datenbank angesehen. Zu dieser Klasse wird die *RDF Query Spezifikation* [MaSu98] genannt.
- **Deklarativer Ansatz:** Im deklarativen Ansatz werden die RDF-Beschreibungen als Knowledge Base angesehen, auf der Techniken zur Wissensrepräsentation und Methoden zur Schlussfolgerung von neuem Wissen angesetzt werden. Dieser Klasse werden sowohl *A Query and Inferencing Service for RDF* und *Metalog* als auch *RQL* zugeordnet.

In der vorliegenden Arbeit soll eine andere Klassifikation als Grundlage dienen, da diese eine passendere Differenzierung der verschiedenen Ansätze von RDF-Anfragesprachen zulässt:

- **XML-basieter Ansatz:** Der Klasse sind XML-Anfragesprachen zugeordnet. Diese basieren auf einer Repräsentation der RDF-Beschreibungen in XML-Syntax und lich sich somit auch als RDF-Anfragesprache einsetzen. Wie jedoch bereits in Abschnitt 5.1.7 angedeutet, ist dieses nur bedingt gegeben.
- **Logik-basierter Ansatz:** RDF-Beschreibungen sind als Regeln und Fakten in der Knowledge Base eines Logikprogrammes abgebildet. Mit Hilfe von Inferenztechniken können Anfragen auf Basis der Logik beantwortet werden.
- **Graphen-basierter Ansatz:** In dieser Klasse von Ansätzen bauen Anfragesprachen auf ein Datenbankmodell auf, welches die RDF-Beschreibungen in einem Graphen oder als Menge von Tripeln repräsentiert. Hierbei soll nicht weiter zwischen Graphen- und der Tripeldarstellung unterschieden werden, da sich ein Graph leicht durch eine Menge von Tripeln darstellen lässt. In der Regel stellt eine Anfrage in diesem Ansatz eine Selektion von Teilgraphen dar.

Die *RDF Query Specification* soll hier außen vor und als ein allgemeiner Vorschlag für RDF-Anfragesprachen vom W3C betrachtet werden.

Ausgehend von der in Abschnitt 5.1 gewählten Speicherung der RDF-Beschreibungen im relationalen Datenbankmodell ist in Abbildung 5.12 der Unterschied zwischen den beiden wichtigsten Klassen von Ansätzen für eine Anfragesprache noch einmal graphisch verdeutlicht.

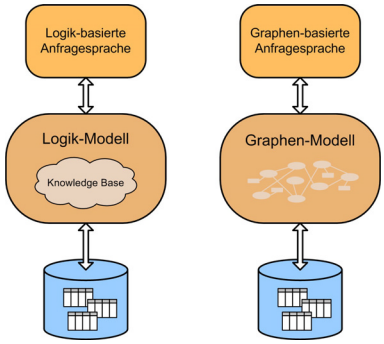

Abbildung 5.12: Ansätze für RDF-Anfragesprachen

5.2.3 RDF Query Specification

In der *RDF Query Specification* [MaSu98] stellt das W3C vor, wie eine RDF-Anfragesprache aussehen könnte. Dieses soll jedoch keine Spezifikation sein, sondern eher eine Richtung für die Syntax und Semantik einer möglichen Anfragesprache geben.

Als Ansatzpunkt für die vorgestellte Sprache wird SQL genommen. Warum SQL selbst nicht als Anfragesprache geeignet ist wird im Abschnitt 5.2.4 näher erläutert.

Bei der *RDF Query Specification* arbeiten RDF-Anfragen auf einem Container von Ressourcen, wobei die Ressourcen von unterschiedlichem Typ sein können und gegebenenfalls verschiedene Eigenschaften besitzen. Die Eigenschaftswerte können wiederum Ressourcen sein. Das Ergebnis einer Anfrage ist wieder ein Container von Ressourcen, eine Teilmenge der Ausgangsdaten.

Wie die konkrete Syntax der Anfragesprache aussehen könnte, wird in [MaSu98] anhand von Beispielen verdeutlicht. Hierbei wird eine Darstellung in RDF mit XML-Syntax und *rdfq* als Namespace verwendet. Die in [MaSu98] aufgeführten Beispiele demonstrieren die Nutzung folgender Operationen:

- Selektion von Ressourcen (*select*)
- Selektion mit Bedingungen (*condition*)
- Selektion innerhalb von RDF-Containern
- Geschachtelte Anfragen
- Projektion
- Aggregation (*count(*)*, *max()*, *min()*)
- Vereinigung von Anfrageergebnissen (*union*)
- Gruppierung nach Eigenschaftswerten (*group*)
- Sortierung von Anfrageergebnissen (*order*)
- Verwendung von Quantoren (*exist*, *forAll*)

In den Klammern sind die für die Operatoren verwendeten Tags angegeben. Diese lassen bereits eine Ähnlichkeit zu SQL vermuten, was durch das folgende Beispiel noch näher verdeutlicht wird.

```
<rdfq:rdfquery>
  <rdfq:Union>
    <rdfq:From eachResource="http://www.rostock.zgdv.de/people/">
      <rdfq:Select>
        <rdfq:Property name="Publication"/>
      </rdfq:Select>
    </rdfq:From>
    <rdfq:From eachResource="http://www.uni-rostock.de/people">
      <rdfq:Select>
        <rdfq:Condition>
          <rdfq:and>
            <rdfq:equals>
              <rdfq:Property name="Fachbereich"/>
              <rdf:String>Informatik</rdf:String>
            </rdfq:equals>
            <rdfq:greaterThan>
              <rdfq:Property name="PublicationsCount"/>
              <rdf:Integer>10</rdf:Integer>
            </rdfq:greaterThan>
          <rdfq:and>
        <rdfq:Condition>
      </rdfq:Select>
    </rdfq:From>
  </rdfq:Union>
</rdfq:rdfquery>
```

Listing 5.2: Anfrage von RDF-Daten nach der *RDF Query Specification*

Die Anfrage in Listing 5.2 ermittelt:

> *alle Personen des ZGDV Rostock, die etwas publiziert haben, und alle Personen der Universität Rostock am Fachbereich Informatik, die mehr als 10 Veröffentlichungen haben.*

5.2.4 Warum nicht SQL?

Die Structured Query Language (SQL) stellt den de facto Standard für Anfragesprachen in relationale DBMS dar. SQL impliziert die Unterstützung großer Datenmengen, Anfrageoptimierung etc. Es wäre daher wünschenswert, diese robuste und häufig verfügbare Technologie auch für Anfragen auf semistrukturierte Daten wie RDF verwenden zu können. Jedoch ist SQL als Anfragesprache auf RDF-Daten nicht geeignet. Primärer Grund ist hierfür, dass SQL auf dem relationalem Datenmodell basiert. RDF ist dagegen Graphen-basiert und semistrukturiert. Selbst wenn eine interne relationale Speicherung von RDF-Modellen gewählt ist, bietet sich die SQL-Anfragesprache des Basisdatenbanksystems nicht an, da so die gewünschte Datenunabhängigkeit (siehe Abschnitt 5.1.2) verletzt wird.

Im Folgenden werden die aus heutiger Sicht wichtigsten Vorschläge für eine RDF-Anfragesprache vorgestellt. Anhand der Beschreibung von zwei XML-basierten Anfragesprachen wird dargelegt, warum diese sich nicht für das Retrieval auf RDF-Daten eignen.

5.2.5 XML-basierte Anfragesprachen

XPath

XPath (XML Path) [ClDe99] entstand bei der Teilung von *XML Stylesheet Language* (XSL) in *XSL Transformation* (XSLT) und XPath im Juni 1999. Sie stellt eine Spezifikation dar, die Elemente oder Attribute in einer XML Datei referenzieren kann. XPath-Ausdrücke beschreiben dazu auf der Basis der XML-Baumstruktur, wie die gesuchten Objekte eines Dokumentes erreicht werden. Im Abschnitt 5.1.7 wurde bereits dargestellt, dass ein RDF-Modell verschieden in XML beschrieben werden kann. Durch den navigierenden Ansatz eignet sich XPath als Anfragesprache auf RDF-Daten daher nur bedingt.

XQL

Eine weitere Anfragesprache für XML-Dokumente ist *XML Query Language* (XQL) [RoLS98]. Sie ermöglicht die Suche nach Objekte mit Hilfe von boolschen Ausdrücken, Filtern, Indizierungen innerhalb von Collections etc. Wie XPath basiert auch XQL auf der Baumstruktur des XML-Dokumentes und kommt daher ebenfalls als RDF-Anfragesprache nicht in Betrachtung. Ein weiterer Grund für die mangelnde Eignung von XQL als RDF-Anfragesprache ist die fehlende Unterstützung Verbunden. Geht man davon aus, dass sich Ressourcenbeschreibungen am Wurzelknoten befinden und von Eigenschaftswerten auf diese verwiesen werden, so ist die Verwendung eines Verbundes notwendig.

Neben den beiden dargestellten XML-Anfragesprachen existieren einige weitere, wie zum Beispiel Quilt, XML-QL, XQuery, Lorel u.a. Deren Anfragemächtigkeiten auf RDF-Daten in XML-Syntax soll hier jedoch nicht weiter diskutiert werden. Nur kurz erwähnt sei, dass es sich bei *XQuery* [FFM+01] um die wahrscheinlich vielversprechendste XML-Anfragesprachen in Bezug auf RDF handelt, da sich XQuery eine Menge Eigenschaften anderer Sprachen (SQL, OQL, XML-QL u.a.) zu Nutze macht.

Einen Überblick zu verschiedenen XML-Anfragesprachen liefern [Meye99], [Hert01] oder die *XML Query working group*[1].

5.2.6 Logik-basierte Anfragesprachen

A Query and Inferencing Service for RDF (F-Logic)

Der in [DBSA98] beschriebene Ansatz erläutert wie RDF-Anfragen auf Basis von FLogic, eine an die Frame Logik angelehnte Sprache, realisiert werden können. F-Logic ist eine objektorientierte Datenbanksprache, die Konzepte zur Darstellung einer Ontologie und die Ausdruckskraft deduktiver Datenbanksprachen bietet. Die Datenbankanfrage erfolgt mit Hilfe von deklarativen Regeln im Datalog-Stil. Die Konzepte und Möglichkeiten von F-Logic werden in [Onto01] und [KiLW95] im Detail beschrieben.

F-Logic entspricht syntaktisch gesehen einer Obermenge der Prädikatenlogik 1. Stufe, welche um objektorientierte Konzepte erweitert wurde. Ein F-Logic-Programm besteht aus einer

[1] XML Query working group. http://www.w3.org/XML/Query

Sammlung von Fakten und Regeln. Wird F-Logic zum Retrieval auf RDF-Daten angewendet, müssen zunächst das RDF-Modell und die jeweiligen Schemainformationen auf Fakten und Regeln abgebildet werden.

Ein RDF-Modell lässt sich beispielsweise folgendermaßen in F-Logic ausdrücken:

RDF-Beschreibung in XML-Syntax:

```
<rdf:RDF>
  <rdf:Description about="Datenbanken: Konzepte und Sprachen">
    <v:Year>1995</v:Year>
  </rdf:Description>

  <rdf:Description about="Andreas Heuer">
    <s: Publications rdf:resource="Datenbanken: Konzepte und Sprachen"/>
  </rdf:Description>
</rdf:RDF>
```

Darstellung in F-Logic:

```
"Datenbanken – Konzepte und Sprachen"[Year->>1995]
"Andreas Heuer"[publications->>"Datenbanken: Konzepte und Sprachen"]
```

Beispiel 5.1: Abbildung von RDF-Daten in F-Logic

Das Beispiel 5.2 zeigt die F-Logic-Befehle zum Laden eines einfachen RDF-Schemas.

Klassendefinition:

```
Person:: Objekt.
Author:: Person.
Book:: Objekt.
```

Attributdefinition:

```
Person[name=>> Literal].
Author[publications=>> Book; cooperatesWith=>> Author].
Book[year=>> Integer].
```

Beispiel 5.2: Abbildung eines RDF-Schemas in F-Logic

Zusätzlich ist es möglich, weitere Inferenzregeln dem F-Logic-Programm hinzuzufügen. Zum Beispiel ist bekannt, wenn ein Autor A mit einem Autor B zusammenarbeitet, dass auch B mit A kooperiert:

```
FORALL A, B
  A:Author[cooperatesWith ->> B] <- B:Author[cooperatesWith ->> A]
```

Auf Basis einer Abbildung der RDF-Daten auf Fakten und Regeln, wird durch buttom-up Evaluation des Programms eine Knowledge Base erzeugt. Diese kann anschließend abgefragt werden. Anfragen bestehen in F-Logic ähnlich zu Datalog aus einer Regel mit leerem Regelkopf. Eine Anfrage zur Ausgabe aller Objekte, die die Eigenschaft *Publications* haben, sieht beispielsweise folgendermaßen aus:

```
FORALL X, Y <- X[Publications ->> Y]
```

Logiksprachen zeichnen sich durch ihre Anfragemächtigkeit aus. So sind mit F-Logic natürlich auch kompliziertere Anfragen möglich. Hierzu können Basisanfragen mit Variablen und Booleschen Ausdrücken kombiniert werden. Das folgende Beispiel zeigt eine komplexe Anfrage in F-Logic:

Ermittle alle Ressourcen und alle Angestellten des W3C für die gilt, dass der Angestellte der Erzeuger der Ressource ist aber keine Beziehungen zu Nokia oder Xerox hat.

```
FORALL Res, Pers
  <- Res[Creator->Pers:Employees[affiliation->>"http://www.w3.org"]]
    AND FORALL Prop, T
      Employee[Prop=>>T] AND
```

```
( NOT Pers[Prop->>"http://www.research.nokia.com"].
  OR Pers[Prop->>"http://www.parc.xerox.com"] )
```

Beispiel 5.3: Anfrage von Metadaten mit F-Logic [DBSA98]

F-Logic als Anfragesprache auf RDF-Beschreibungen erlaubt rekursive Anfragen und bietet außerdem die Möglichkeit, Anfragen an das Schema zu stellen. Zum Beispiel können alle Autoren, die eine Eigenschaft mit dem Wert *„Harry Potter“* haben, ermittelt werden (ausgehend davon, dass auch andere Ressourcen eine Eigenschaft mit dem identischen Wert haben können):

```
FORALL A, Prop, T <- A:Author[Prop=>>T] AND
  A[Prop->>"Harry Potter"]
```

Für die Bearbeitung der Beispielanfrage wird also geprüft, bei welchen Autoren das Attribut *Publications* oder *CooperatesWith* dem Wert *„Harry Potter“* entspricht.

Die Verwendung einer Regelbasierten Anfragesprache, wie sie in diesem dargestellten Ansatz zum Zugriff auf Metadatenbeschreibungen in RDF vorgeschlagen wird, bringt jedoch einige Probleme mit sich [BrFH00]:

- Es können nicht alle RDF-Ausdrücke direkt in F-Logik ausgedrückt werden. Zum Beispiel ist die Beschreibung von Containern nicht direkt möglich.
- Die Inferenz basiert auf manuell hinzugefügten Regeln.
- Die Semantik von RDF selbst ist nicht definiert. Es existieren keine generischen Inferenzregeln für *RDF-Model* und *RDF-Schema*.
- F-Logic ist die interne Darstellungssprache von Ontobroker. Ontobroker nimmt die Existenz einer vordefinierten zentralen Ontologie an, wobei das Laden eines individuellen RDF-Schemas nicht unterstützt wird. Dafür sind unter anderem spezielle Inferenzregeln Bestandteil der Ontologie.

Eine weitere Einschränkung stellt das Abbilden der RDF-Schemata in einem anderem Formalismus oder sogar das Ignorieren zugehöriger RDF-Schema-Informationen dar. Die Darstellung äquivalenter Inferenzregeln in F-Logic ist meist nicht vollständig möglich.

F-Logic bietet trotz der genannten Probleme eine mächtige Anfragesprache, die eine benutzerfreundliche Formulierung von Anfragen erlaubt.

Der *Simple Logic-based RDF Interpreter*[1] (SiLRI) ist eine Implementation zur Nutzung von F-Logic-Anfragen auf RDF-Daten. Die Inferenzmaschine stellt einen wesentlichen Teil der Frame-Logic bereit. Sie ist Bestandteil des kommerziellen Systems Ontobroker von Ontoprice, jedoch separat frei verfügbar. SiLRI ist in Java implementiert und baut auf den SiRPAC-Parser auf. Beim Einsatz von SiLRI werden zunächst die RDF-Daten in den Hauptspeicher geladen, wobei jedes RDF-Statement in Fakten der Form *setatt_(Resource, Property, Value)* übersetzt wird. Auf diesen Statements können Regeln definiert werden, die beispielsweise für Anfragen an die Ontologie benutzt werden können.

[1] Ontoprice GmbH: Simple Logic-based RDF Interpreter (SiLRI). http://www.ontoprise.de/com/co_silri.htm

Metalog

Metalog [MaSa98] [MaSa99a] ist ein Ansatz zur Darstellung einer „logischen“ Sicht auf Metadaten und wird als „next-generation query system for metadata“ [MaSa99b] etikettiert. In vielen Punkten ähnelt Metalog dem zuvor dargestellten Ansatz. Jedoch geht Metalog einige Schritte weiter. Ziel ist die Realisierung einer Anfragesprache, die es erlaubt, Metadaten, Inferenzregeln und Anfragen in einer der englischen Sprache angelehnten Syntax zu definieren. Daneben sollen Anfragen auch in RDF möglich sein.

Der Ansatz Metalog besteht aus drei Schichten:

Metalog Model Layer

- Erstellung eines erweiterten RDF Metalog Schemas
- Erweiterung des RDF-Schemas um logische Relationen (*and*, *or*, *not*, *implies*) und Variablen

Hierdurch ist es möglich, Inferenzregeln in RDF-Schemata zu formulieren.

Metalog Logic Layer

- „logische Interpretation“ der RDF-Daten in Logik

Dieses ermöglicht die Abbildung der Informationen in Logikprogramme.

Metalog Language Layer

- Schaffung einer Sprachschnittstelle zur Formulierung von strukturierten Daten und Anfrageregeln

Die dritte Schicht bildet eine syntaktische Schnittstelle zwischen dem Benutzer und dem Metalog Model Layer. RDF-Anfragen oder natürlich-sprachliche Anfragen können auf diesen Weg logisch interpretiert werden.

In [MeSa99a] findet sich ein Vorschlag zur Syntax von Metalog. Wörter in Großbuchstaben entsprechen Variablen. Kleinschreibungen werden Schlüsselwörtern (zum Beispiel *then*, *and*, *not*, *implies* u.a.) zugeordnet, wenn diese vorhanden sind. Ansonsten werden sie ignoriert. Werte bzw. Literale sind durch Anführungszeichen gekennzeichnet. Die Anfragen unterscheiden sich durch ein abschließendes Fragezeichen von Inferenzregeln. Sie werden nicht der Knowledge Base hinzugefügt, sondern durch eine Inferenzmaschine ausgewertet.

Eine Aussage kann wie folgt interpretiert werden:

Die zu interpretierende Aussage ist:

```
if SHE and HE have a "degree" in "math" and "science" and "John" "is" "good" then …
```

Nach dem Entfernen nichtnotwendiger Wörter entsteht folgender Ausdruck:

```
SHE and HE "degree" "math" and "science" and "John" "is" "good" then …
```

Durch Parsen des Ausdrucks und Bestimmung der einzelnen Ebenen innerhalb dessen entsteht der logische Ausdruck:

```
degree(Bag(SHE, HE), Bag("math", "science")) and is ("John", "good") => …
```

Beispiel 5.4: Interpretation einer Anfrage bei Metalog [MeSa99a]

Für eine Implementierung von Metalog ist die Nutzung von Datalog anvisiert.

Derzeit stellt Metalog lediglich einen Weg vor, Inferenzregeln durch logische Operatoren in RDF-Schemata einzubetten. Eine tatsächliche Abfragesprache ist noch nicht vorgeschlagen. Ferner ist die Architektur eines Systems, welches Metalog und binäre Prädikate zur Anfrage auf RDF-Daten verwendet, nicht offensichtlich. (aus [BrFH00])

Eine Implementierung, die die Ideen der Autoren in der Praxis zeigt, steht zudem noch aus.

5.2.7 Graphen-basierte Anfragesprachen

RQL

Eine an die SQL-Syntax angelehnte Anfragesprache ist die *RDF Query Language* (RQL), die in [KCPA99] vorgestellt wird. RQL ermöglicht auf einfache Art und Weise Anfragen über RDF-Modelle und Schemainformationen zu stellen. Ziel ist dabei, Anfragen auf RDF-Daten mit minimalen Kenntnissen über vorhandene RDF-Schemata zu ermöglichen.

Das zu Grunde liegende Datenbankmodell ist ein semistrukturierter Graph. Dieser ist angelehnt an der *RDF Model and Syntax* und *RDF Schema Specification* des W3C. Er beinhaltet die RDF-Daten aus dem RDF-Modell und schließt Schema-Beschreibungen ein, die auf einem oder mehreren RDF-Schemata beruhen können. Das Modell erlaubt somit die Unterstützung von RDF Schemata, insbesondere *subClassOf* und *subPropertyOf*.

RQL knüpft an den funktionellen Ansatz von OQL an. Zusätzlich werden verallgemeinerte Pfadausdrücke und die Nutzung von Variablen für Knoten und Kanten unterstützt. Für RQL existiert eine definierte Menge von Basisanfragen, Filtern und Iteratoren. Diese können kombiniert werden, um komplexe Anfragen zu stellen.

Die Basisanfragen bieten ein Mittel, um auf Informationen des RDF-Modells und der RDF-Schemata zuzugreifen. Zum Beispiel liefert die Anfrage

```
property
```

die URIs aller verfügbaren Eigenschaftstypen. Hierbei ist *property* die Bezeichnung eines Metadatenobjektes, das die Namen aller Kanten im Graph enthält.

Zur Unterstützung der Klassen-, Eigenschaftstyp- beziehungsweise Typhierarchien sind in RQL verschiedene Funktionen vorgesehen, zum Beispiel *subClassOf* und *subClassOf^*, für indirekte und direkte Unterklassen. Die Anfrage

```
subClassOf^(Person)
```

ermittelt beispielsweise die direkten Unterklassen von *Person*.

RQL erlaubt nicht nur Anfragen an das Schema, sondern auch an das eigentliche RDF-Modell. Sollen alle Resource-Value-Paare zu einem gegebenen Eigenschaftstyp *create* ermittelt werden, so lautet die Anfrage hierzu:

```
create
```

Hierbei ist unter anderem die Bildung von Vereinigung und Durchschnitt möglich. Neben den Binären Operationen erlaubt RQL die Verwendung von Booleschen Prädikaten, wie =, <, > und *LIKE*. Diese können auf URIs ebenso wie Literale angewendet werden. Zudem existieren in RQL die Booleschen Operatoren *AND*, *OR* und *NOT*.

Ein Zugriff auf RDF-Container geschieht mit speziellen Operatoren, wie *in*, *element* oder *[ ]*.

Eine der wichtigsten Konzepte sind die so genannten Filter, die es erlauben Selektionen und Projektionen durchzuführen. Ein Filter besteht aus den von SQL bekannten *SELECT-FROM-WHERE*-Blöcken. Ein Beispiel für eine Anfrage mit Filter wäre:

Ermittle die Ressourcen, die Autor von etwas mit dem Titel „Datenbanken: Konzepte und Sprachen" sind und eine Email haben, die mit „@informatik.uni-rostock.de" endet.

```
SELECT W
FROM {X}Title{Y}, {Z}author{W}.eMail{Q}
WHERE X = Z
    AND Y = "Datenbanken: Konzepte und Sprachen"
    AND Q LIKE "*@informatik.uni-rostock.de"
```

Beispiel 5.5: RQL-Anfrage mit Filter

Neben den im Beispiel 5.5 gezeigten Pfadausdrücken, werden verallgemeinerte Pfadausdrücke[1] unterstützt:

Finde die Ressourcen die eine Eigenschaft besitzen, dessen Typ mit „size" endet.

```
SELECT X
FROM $P property {Y}$P{X}
WHERE $P LIKE "*size"
```

Beispiel 5.6: RQL-Anfrage mit verallgemeinertem Pfadausdruck

Eine detaillierte Beschreibung der Funktionalität, Syntax und Semantik von RQL ist [KCPA00], [Karv00] oder [FICS01] zu entnehmen. In Anhang D.1 findet sich eine Syntaxbeschreibung von RQL in EBNF.

RQL bietet geeignete Konzepte, wie Basisanfragen und Filter, die eine Vielfalt von Anfragen ermöglichen und auch die Formulierung komplexer Anfragen zu lassen.

Das Datenbankmodell von RQL ist ein semistrukturierter Graph, der sich an der *RDF Model Specification* orientiert und auf einer theoretischen Grundlage fundiert. RQL hat eine wohldefinierte Semantik. Durch die genannten Eigenschaften ist eine Basis zur Untersuchung des Determinismus von Anfragen als auch ein Ansatz zur Anfrageoptimierung gegeben. Wie bei einer RQL-Anfrage beispielsweise die Auswertung von Pfadausdrücken optimiert werden kann, ist in [KCPA00] gezeigt.

Bezüglich der Semantik der RDF-Schemata gegenüber der *RDF Schema Specification* existieren bei RQL jedoch einige kleine Einschränkungen, welche in [BrKa01a] nachzulesen sind.

Eine Implementierung von RQL ist in der vom FORTH Institute of Computer Science entwickelten RDFSuite zu finden (siehe Abschnitt 5.1.1). Der *RQL Interpreter* ist in C++ geschrieben und setzt sich zusammen aus den Modulen: *Parser* (Analyse der RQL-Anfrage), *Graph Constructor* (Bearbeitung der Anfrage) und *Evaluation Engine* (Zugang zu den RDF-Beschreibungen in der Datenbank mit SQL3). Derzeit werden vom Interpreter Funktionen, wie arithmetische Operationen oder Quantoren noch nicht unterstützt.

[1] Bei verallgemeinerten Pfadausdrücken werden die Variablen für Schemata, Daten und Klassen anhand von Präfixen ($ und @) unterschieden.

Neben der RDFSuite existiert eine Implementierung der RDF-Anfragesprache für Sesame (siehe Abschnitt 5.1.1), die einige Veränderungen zum *RQL Interpreter* der RDFSuite aufweist. Diese sind in [BrKa01b] näher beschrieben.

SquishQL

Die Sprache SquishQL [Mill01] stellt im Gegensatz zu RQL eine sehr einfache Anfragesprache für RDF dar. Sie wurde am *Institute for Learning and Research Technology* (ILRT) entwickelt, stark beeinflusst von [GLMB98] und der rdfDB Query Language, die im folgenden Abschnitt vorgestellt wird.

Der RDF-Anfragesprache SquishQL liegen die im Folgenden genannten Konzepte zu Grunde:

- **Einfach**: SquishQL unterstützt keine transitive Vererbung innerhalb der Eigenschaftstyp- oder Klassenhierarchie. Dieses ermöglicht den Einsatz eines einfachen zugrunde liegenden Datenmodells und die unkomplizierte Bearbeitung von Anfragen. Wiederum die Einfachheit schlussfolgert jedoch einen gewissen Verlust an Semantik.
- **Tripel-basiert**: Das Datenbankmodell von SquishQL ist eine Menge von Tripeln, welches eine sehr nahe Abbildung des RDF-Modells erlaubt. Neben dem Datenbankmodell beruht auch die Formulierung der SquishQL-Anfragen auf Tripel.
- **Lesbar**: Anfragen in SquishQL sind so konstruiert, dass sie leicht für Menschen lesbar sind. Die Anfragesprache ist hierzu an die von SQL bekannte Syntax angelehnt und verwendet die *SELECT-FROM-WHERE*-Blöcke.

Eine Anfrage in SquishQL sieht beispielsweise folgendermaßen aus:

```
select ?titel, ?authorName,
where  (fbi::book#publicationYear ?book ?year)
       (fbi::book#title ?book ?title)
       (fbi::book#creator ?book ?author)
       (fbi::person#name ?author ?authorName)
and ?year > 1995
using fbi for http://informatik.uni-rostock.de/
```

Finde die Titel der Bücher die nach 1995 veröffentlicht wurden und die Namen derer Autoren.

Hinter der *WHERE*-Klausel findet sich eine Liste von Tripeln. Durch Angabe von URIs oder Eigenschaftswerten statt Variablen erfolgt hierauf eine Selektion. Variablen unterscheiden sich von Werten durch ein vorangestelltes Fragezeichen. Ein Verbund Tripelmengen ist durch die Verwendung gleichnamiger Variablen in verschiedenen Tripeln erreichbar. Die Variablen zur Projektion werden wie bei SQL im *SELECT*-Block angegeben. Hinter *AND* kann eine einfache Bedingung formuliert werden. Leider erlaubt die derzeitige Spezifikation hier nur die Verwendung einer einzigen Vergleichsoperation mit =, <, >, <=, >= oder ~. Das Zeichen ~ dient dem Vergleich von Zeichenketten, analog dem *like*-Operator in SQL. Boole'schen Prädikate wie *AND*, *OR* und *NOT* werden nicht unterstützt. Die *USING*-Klausel bietet eine Möglichkeit, die Formulierung von Anfragen zu vereinfachen, zum Beispiel bei der Verwendung von Namespaces. In einem optionalen *FROM*-Block der Anfrage kann eine Liste von URLs angegeben werden. Diese erlaubt es, Anfragen auch an RDF-Daten zu stellen, die sich nicht in der Datenbank, sondern sich an einer beliebigen Stelle im World Wide Web befinden.

Eine Syntaxbeschreibung von SquishQL kann zum besseren Verständnis dem Anhang D.2 entnommen werden.

SquishQL bietet längst nicht die Möglichkeiten Anfragen zu stellen, wie es RQL oder Logik-basierte Anfragesprachen tun. Speziell die Fähigkeiten, Selektionen auf Tripelmengen zu definieren, ist sehr begrenzt. Des Weiteren ist eine Unterstützung von RDF-Schemata nur im geringen Maße gegeben. Zwar ist es möglich über gespeicherte RDF-Schema-Beschreibungen einfache Anfragen zu stellen, jedoch müssen komplexere Anfragen von der Applikation ausgewertet werden. Auf Grund der genannten Einschränkungen kann SquishQL nur als Alternative zu den bereits vorgestellten Anfragesprachen gesehen werden, wenn innerhalb einer Applikation keine komplexen Anfrageprädikate notwendig sind und die Nutzung von RDF-Schemata für die Anwendung keine wesentliche Rolle spielt. In der Regel sind allerdings gerade einfache Anfragen die häufigsten.

Die Entwicklung der Anfragesprecher steht im engen Zusammenhang mit der im folgenden Absatz beschriebenen Implementation, die eine einfache Anfrageschnittstelle für verschiedene RDF-Datenbankmodelle darstellt. Auf Grund der unterschiedlichen Datenrepräsentationen kann diese nicht alle RDF-Konzepte unterstützen. Eine Anfragesprache wie SquishQL findet daher durchaus ihre Berechtigung.

Für SquishQL existiert der experimentelle Anfrageinterpreter *Inkling*[1]. Dieser wurde in Java implementiert und baut auf dem SiRPAC-Parser auf. Inkling verwendet dessen Interface-Klassen, um die RDF-Daten intern darzustellen.

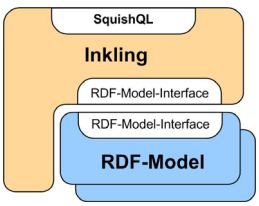

Abbildung 5.13: Inkling Architektur

Ziel der Entwicklung von Inkling war es, eine Implementation eines Anfrageinterpreters für SquishQL zu schaffen, die mit verschiedenen RDF-Datenbankimplementierungen verwendet werden kann. Um dieses zu verwirklichen, nutzt die Implementierung das Treiberprinzip. Intern lädt Inkling einen Treiber, der über eine definierte Schnittstelle den Zugriff auf die RDF-Daten ermöglicht. Nach außen stellt Inkling einen JDBC-Treiber bereit, um so den Interpreter in eine Applikationen einzubinden.

Durch die modulare Implementierung der Anfragesprache ist es möglich, auf einfache Art und Weise in Applikationen Anfragen mit SquishQL zu stellen, wobei die RDF-Daten in einem beliebigen Datenmodell verwaltet werden können.

[1] Inkling: RDF query using SquishQL. http://swordfish.rdfweb.org/rdfquery/

rdfDB Query

Die Anfragesprache *rdfDB Query* ist im Zusammenhang mit *rdfDB* (siehe Abschnitt 5.1.1) entwickelt worden. Die Anfragesprache unterscheidet sich von SquishQL kaum, so dass hier die Unterschiede nur kurz genannt werden. In Anhang D.3 findet sich eine komplette Syntaxbeschreibung von rdfDB Query.

Mit rdfDB können mehrere RDF-Datenbanken angelegt werden. So wird in der FROM-Klausel im Gegensatz zu SquishQL der Name einer entsprechenden Datenbank angegeben, über die die Anfrage gestellt wird. Die Anfragesprache rdfDB Query ist sehr simpel. Die AND- und USING-Klausel entfallen hier.

Neben einer Anfragesprache bietet rdfDB eine stark vereinfachte DDL (Data Definition Language) und DML (Data Manipulation Language).

Fazit: Die vorgestellten Sprachen unterscheiden sich bezüglich deren Ansätze, Anfragemächtigkeit und Konzepte. Für einen Vergleich der Anfragesprachen sind in der Tabelle 5.1 die unterstützten Operatoren gegenüber der *RDF Query Specification* vom W3C aufgeführt.

	F-Logic	RQL	SquishQL	rdfDB Query
RDF-Schema Unterstützung	(x)	x	-	-
Selektion	x	x	x	((x))
Projektion	x	x	x	x
Verbund	x	x	x	x
Selektion innerhalb von Containern	(x)	x	(x)	(x)
Geschachtelte Anfragen	(x)	(-)	-	-
Aggregationen	-	(x)	-	-
Boolesche Operatoren	x	x	-	-
Quantoren	x	x	-	-
Vereinigung	(-)	x	-	-
Sortierung	-	-	-	-
DDL und DML	(x)	-	-	(x)

Tabelle 5.1: Vergleich der RDF-Anfragesprachen

Wie der Tabelle zu entnehmen ist, zeigen F-Logic und RQL eine Umsetzung vieler der gewünschten Operatoren. Auch wenn die beiden Sprachen nicht alle der zu Beginn des Kapitels genannten Kriterien in vollem Umfang erfüllen, so stellen sie doch flexible Möglichkeiten der Anfrageformulierung bereit, mit denen auch komplexe Anfragen ausgedrückt werden können. Die Wahl einer geeigneten Anfragesprache für das RDF-Content-Repository sollte daher auf eine der beiden Sprachen fallen. Jedoch ist für die Auswahl nicht nur die Eignung der Sprache, sondern auch deren Verfügbarkeit und Umsetzung in einer Implementation ausschlaggebend. Eine eigene Implementation eines geeigneten Anfrageinterpreters ist im Rahmen dieser Diplomarbeit nicht vorgesehen.

Für den Logik-basierten Ansatz mit F-Logic ist eine Implementation (SiRIL) vorhanden. Zur Auswertung der Anfragen werden jedoch in der derzeitigen Umsetzung alle Daten im Hauptspeicher verwaltet, welches auf die buttom-up Evaluation des F-Logic-Programmes zurückzuführen ist. Die Anpassung der Inferenzmaschine an die Speicherung im relationalen Modell, bei dem zudem noch das gewählte Datenbankschema verwendet wird, käme daher einer kompletten Neukodierung gleich. Eine Anfragebearbeitung bei der zuvor alle Daten in den Hauptspeicher geladen werden müssen, ist für das RDF-Content-Repository nicht akzeptabel.

Für RQL existieren verschiedene Implementierungen. Allerdings stehen diese nur für Online-Demos bereit. Eine frei verfügbare Implementierung der RQL stellte das ILTR erst im November 2001 bereit, so dass diese bei der Auswahl nicht mehr berücksichtigt werden konnte.

Aus den genannten Gründen verbleiben für die Entwicklung des RDF-Content-Repository nur noch die Anfragesprachen SquishQL und rdfDB Query. Von den beiden verbleibenden wurde SquishQL gewählt. SquishQL bietet gegenüber rdfDB Query die Formulierung einfacher Bedingungen.

Ausgehend vom Anwendungsszenario ist diese Wahl durchaus akzeptabel, da sich hier die Anfragen an den RDF-Metadatenbestand weitgehend auf einfache Anfragekonstrukte beschränken, die mit SquishQL möglich sind.

5.3 Zusammenfassung

Ziel des Kapitels war die Entwicklung einer geeigneten Metadatenverwaltung für die XML-Framework-Architektur. Zunächst wurden Techniken zur Speicherung von RDF-Daten evaluiert. Eine wesentliche Erkenntnis dabei war, dass für die Verwaltung von RDF-Daten deren Repräsentation in XML-Syntax nicht geeignet ist. Neben Speicherungsformen in OODBMS und ORDBMS sind primär Ansätze auf Basis relationaler Modelle untersucht worden. Es wurde gezeigt, dass sich zur Persitierung ohne RDF-Schema-Informationen die einfache Speicherung von Statements in ihrer Tripelform als sinnvoll erweist. Darauf aufbauend wurden Vorschläge zur Unterstützung von Anfragen durch Indizierungstechniken diskutiert. Eine Indizierung in Form einer mehrdimensionalen Speichertechnik, speziell der UB-Tree, stellte sich dabei als geeignet heraus. Auf Grund der Verfügbarkeit kann allerdings nur auf eindimensionale Indizierungstechniken zurückgegriffen werden.

Neben der Speicherung von RDF-Daten spielen Anfragen eine wesentliche Rolle. Im Abschnitt 5.2 sind hierzu verschiedene RDF-Anfragesprachen vorgestellt und verglichen worden. Für das RDF-Content-Repository wird letztendlich SquishQL verwendet.

Wie die in diesem Kapitel gewonnenen Kenntnisse konkret im RDF-Content-Repository umgesetzt wurden, ist Thema das Abschnittes 8.1 im vierten Teil der Arbeit.

6. Steuerung der Datentransformation

In den vorherigen Kapiteln wurde der Aufbau der XML-Framework-Architektur, der zugrunde liegende Metadatenbestand dargelegt. Hiervon ausgehend wurde im Kapitel 5 eine Realisierung für eine Metadatenverwaltung vorgestellt. Wie eine metadatengesteuerte Request-Verarbeitung und Anfragetransformation am Retrieval Server erfolgt, wird in dieses Kapitel gezeigt.

Grundlegend basiert die Bearbeitung von clientseitigen Anfragen auf den in [Ditt01] vorgestellten Retrieval-Server-Prozess, wobei hier jedoch der Zugriff auf Metadaten über das in dieser Arbeit entwickelte RDF-Content-Repository erfolgt. Der Prozess setzt sich aus einer Reihe von Transformationen, Anfragen an die integrierten Datenbanken und Anfragen an das RDF-Content-Repository zusammen. Diese werden im Folgenden anhand ihrer zeitlichen Reihenfolge beim Eintreten einer clientseitigen Anfrage erörtert. Die Abbildung 6.1 gibt einen allgemeinen Überblick zum Retrieval-Server-Prozess.

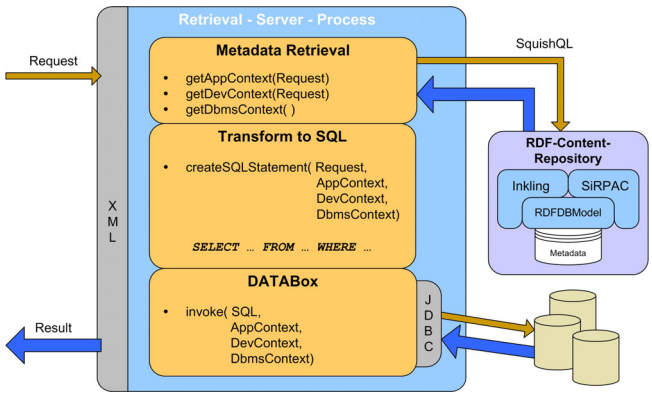

Abbildung 6.1: Retrieval-Server-Prozess

Ist der Endanwender zu einem bestimmten Thema (Kontext) an Dokumenten interessiert, so stellt er über sein Endgerät (dem Client) eine Anfrage an das System. Der Client sendet hierzu an den Retrieval Server eine Anfrage in XML-Syntax. Darin enthalten sind Festlegungen zum gewählten Application Context, zum verwendeten Endgerät (Device Context) und etwaige Suchbegriffe. Hinzu kommen noch Informationen wie gewünschte Sprache und Nutzeridentifikation. Die entsprechende DTD zur Clientanfrage ist in [Ditt01] zu finden.

Nach dem Eintreffen der Anfrage am Server beginnt der eigentliche Retrieval Prozess. Zunächst wird die Anfrage syntaktisch analysiert und gegenüber der DTD validiert. Mit den aus der Anfrage extrahierten Informationen werden die für die Bearbeitung notwendigen Metadaten über das RDF-Content-Repository ermittelt. Hierzu gehören Metadaten-beschreibungen zum jeweiligen Application Context (z.B. relevante Relationen, Verbundbeziehungen, durchsuchbare Attribute), zum entsprechendem Device Context

(endgerätespezifische Parameter) und zum Database Context (benötigte DBMS, JDBC-Treiber, Namen der Relationen etc.). An einem einfachen Beispiel soll das Metadatenretrieval verdeutlicht werden:

```
SELECT ?dbName, ?jdbcDriver, ?url, ?userID, ?pass
WHERE (urn:database#Name  databaseUri  ?dbName)
      (urn:database#Url  databaseUri  ?url)
      (urn:database#JdbcDriver  databaseUri  ?jdbcDriver)
      (urn:database#userId  databaseUri  ?userId)
      (urn:database#pass  databaseUri  ?pass)
```

Listing 6.1: SquishQL-Anfrage zur Ermittlung der Datenbank-Verbindungsinformationen [1]

Die in Listing 6.1 angegebene SquishQL-Anfrage ermittelt die Verbindungsinformationen für benötigte integrierte Datenbanken. Welche Datenbank(en) für die Bearbeitung der Anfrage relevant ist (sind), ergibt sich aus dem Application Context und soll hier als gegeben betrachtet werden.

Im zweiten Schritt des Retrieval-Server-Prozesses erfolgt die Generierung eines oder mehrerer SQL-Statements. Grundlage sind die zuvor ermittelten Metainformationen. Das folgende Statement veranschaulicht den generellen Aufbau der SQL-Anfrage:

```
SELECT SeA_1, SeA_2, ...
FROM Rel_1, Rel_2, ...
WHERE JC_1, JC_2, ...
  AND (SA_1 OR SA_2 OR ...)
```

*SeA_* : selektierte Attribute (Gegebenfalls können hierbei für verschiedenen Multimediadatentypen auch DBMS-spezifische Funktionen, wie Skalierung, Konvertierung oder Komprimierung angewandt werden)

*Rel_* : benötigte Relationen

*JC_* : Verbundbeziehungen zwischen den Relationen

*SA_* : durchsuchbare Attribute mit den entsprechenden Vergleichsbedingungen

Listing 6.2: Genereller Aufbau der SQL-Anfrage beim Retrieval-Server-Prozess

Objektrelationale Datenbanksysteme wie Informix, DB2 oder Oracle bieten durch erweiterte Datentypen und Multimediaerweiterungen, zumeist in Form von vorgefertigten Modulen (DataBlades, DB2 Extenders, Cartridges), die Fähigkeit multimediale Daten geeignet zu verwalten. Mit Hilfe von Funktionen in Datenbankanfragen können automatische Transformationen auf entsprechende Datentypen erfolgen (Daten-Preprocessing). Diese Konzepte werden bei der Anfragegenerierung am Retrieval Server berücksichtigt. Welche Transformationsfunktion angewendet wird ergibt sich aus den im vorherigen Schritt ermittelten Metainformationen.

Um die Anfrage an die Datenbank zu senden, baut die DATABox nach der Generierung des SQL-Statements eine Verbindung über die JDBC-Schnittstelle mit dem festgelegten Treiber und dazugehöriger Konfiguration zur Datenbank auf. Die ermittelte Ergebnisrelation wird innerhalb der DATABox in ein XML-Dokument umgewandelt und gegebenenfalls mittels XSL-Stylesheets in ein für das Endgerät lesbares Format transformiert. Bei der derzeitigen Realisierung des Systems erfolgen diese Transformationen noch proprietär. In einer

[1] Der Bezeichner *databaseUri* steht für die URI der Datenbank, für die die Verbindungsinformationen gesucht werden.

weiterentwickelten Version sollen auch diese Umformungen durch den Metadatenbestand parametrisiert ablaufen.

Resultat des Retrieval-Server-Prozesses ist Ergebnisdokument welches an den Client zurückgegeben wird.

Die serverseitige Request-Verarbeitung durch den Retrieval-Server-Prozess basiert auf einem zweistufigen Ansatz. Eine Datentransformation erfolgt hierbei sowohl auf struktureller Ebene (DATABox), als auch den Datenelementen selbst (Daten-Preprocessing am DBMS).

Mit der Erweiterung des Retrieval Systems durch das in dieser Arbeit entwickelte RDF-Content-Repository ergeben sich für die Request-Verarbeitung, Anfrage- und Datentransformation gegenüber dem in [Ditt01] vorgestellten Vorschlag kaum Änderungen. Anpassungen sind lediglich auf der Ebene des Metadatenretrievals erforderlich. Im bisherigen System erfolgte der Zugriff auf den Metadatenbestand durch eine low-Level API. Mit dem RDF-Content-Repository ist eine geeignete, RDF-basierte Metadatenverwaltung verwirklicht worden, bei der die Ermittlung der benötigten Metainformationen mit Hilfe einer RDF-Anfragesprache erfolgt.

Zentrales Thema im zweiten Teil dieser Arbeit waren Betrachtungen zur Bereitstellung einer adäquaten, RDF-basierten Metadatenverwaltung. Schwerpunkt im dritten Teil ist die geeignete Erweiterung der XPEA-Architektur um OLAP-Funktionalität.

Teil III
OLAP-Erweiterung

7. Erweiterter Architekturvorschlag zur Nutzung von OLAP-Funktionalität

Neben der Entwicklung einer geeigneten Metadatenverwaltung besteht das Ziel dieser Arbeit in der Erweiterung der gewählten XML-Framework-Architektur um OLAP-Funktionalitäten. Der Begriff OLAP und die damit verbundene Technologien wurden hierzu in Kapitel 3 vorgestellt.

Grundlage für die Erweiterung des Systems ist die Erweiterung des Metadatenbestandes und die adäquate Veränderung der Steuerung des Content Repositories. Die entwickelten Konzepte und Systembausteine werden anhand des Anwendungsszenarios „Europa-MV“ validiert. Hier sollen aufsetzend auf vorhandene Datenbestände signifikante OLAP-Methoden und spezifische Datenstrukturen in die endgeräte- und nutzerorientierte Informationsbereitstellung und -Aufbereitung einbezogen werden.

7.1 OLAP am Beispiel von Europa-MV

Im Vorfeld der Entwicklung einer erweiterten Architektur zur Unterstützung von OLAP-Funktionalitäten wurden im Rahmen der Diplomarbeit Untersuchungen zur multidimensionalen Analyse durchgeführt. Ziel dabei war die Evaluation geeigneter multidimensionaler Schemata für die OLAP-Analyse auf Grundlage einer zur Verfügung stehenden Datenbasis.

Eine Untersuchung wird hier kurz vorgestellt.

Die Europa-MV Plattform vereinigt verschiedene Datenbanken unter verschiedenen Anwendungsaspekten. Ein Ziel von Europa-MV ist es, Unternehmen und Instituten die Teilnahme an EU-weiten Projekten durch den Zugriff auf Ausschreibungen und Projektunterlagen zu vereinfachen. Hierzu existiert eine umfangreiche Datenbank über

(laufende und abgeschlossene) EU-geförderte Projekte in Mecklenburg-Vorpommern. Die Projektdatenbank umfasst unter anderem Informationen über Förderungsmenge, Projektvolumen, -partner, -beginn und -ende sowie eine Gliederung der Projekte nach EU-Programmen und EU-Rahmenprogrammen.

Die in der Projektdatenbank vorhandenen Daten eignen sich zum Aufbau eines multidimensionalen Würfels. Aus diesem Grund wurde in dieser Diplomarbeit die Projektdatenbank als Datenbasis gewählt. Zur Vereinfachung der Schemamodellierung und OLAP-Analyse wurden die Analysis Service-Komponente des SQL Server 2000 von Microsoft verwendet, die folgend kurz vorgestellt wird.

SQL Server 2000 Analysis Service

Microsoft bietet mit dem *SQL Server 2000 Analysis Service* einen Server zur multidimensionalen Analyse von Daten. Analysis Service ist eine Erweiterung der OLAP Service-Komponente (eingeführt im SQL Server Version 7.0), bei der neben OLAP-Funktionalitäten auch Dataminingfunktionen zur Verfügung stehen.

Durch verschiedene Assistenten, Editoren und Tools ermöglicht das Analysis Services-System eine einfache Erstellung und Verwaltung von multidimensionalen Würfeln. Basisdaten können sowohl aus OLE DB- oder ODBC-Datenquellen extrahiert werden. Für die Speicherung der Datenwürfel werden drei Formen unterstützt: multidimensional (MOLAP), relational (ROLAP) und hybrid (HLOAP). Mittels der Sprache MDX (Multidimensional Expressions), eine definierte OLAP-Erweiterungen in OLE DB, ist der multidimensionale Datenzugriff auf den Würfel möglich. MDX ist angelehnt an SQL, bietet jedoch eine umfangreichere Syntax für das Abrufen und Bearbeiten multidimensionaler Daten.

Durch die Nutzung verschiedener, bereitgestellter Werkzeuge in Analysis Service ist eine unkomplizierte Erstellung von Dimensionen und OLAP-Würfeln auf den unterschiedlichsten Basisdaten möglich, so auch auf der Projektdatenbank von Europa-MV.

Grundlage der Modellierung eines multidimensionalen Würfels auf relational gespeicherten Daten bei Analysis Services ist die Bereitstellung der Daten in Form eines Star-Schemas oder Snowflake-Schemas. Durch das vorhandene Datenbankschema der Projektdatenbank ist dieses nicht gegeben. Daher wurden zunächst verschiedene Sichten erzeugt, die geeignete Dimensions- und Faktentabellen darstellen. Die folgende Abbildung veranschaulicht das entwickelte Star-Schema.

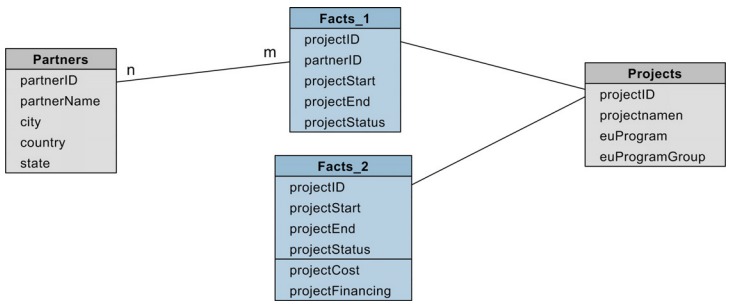

Abbildung 7.1: Star-Schema auf Basis der Projektdatenbank

Hinweise zum verwendeten Star-Schema:

- Die Dimension *Projektstatus* besteht nur aus einem Level und kann daher direkt in der Faktentabelle gespeichert werden.
- Analysis Service erlaubt den einfachen Aufbau einer Dimension auf den Datentype *date*. So können die Dimensionen Projektstart und -ende direkt in der Faktentabelle gespeichert werden.
- Da ein Projekt mehrere Partner haben kann, kommen in der Faktentabelle *facts_1* Projekte mehrfach vor. Die Tabelle *facts_2* beinhaltet dagegen jedes Projekt nur einmal. Dafür besteht jedoch keine Beziehung zur Dimensionstabelle *partners*.
- Die Faktentabelle facts_2 enthält zur besseren Übersicht nur Projekte mit definierten Werten für die Attribute *projectCost* und *projectFinancing*.

Das Schema besteht aus zwei verschiedenen Faktentabellen. Unter der Verwendung der Factentabelle *facts_1* wurde ein Würfel mit folgendem Schema aufgebaut:

Measures	
Name	**Aggregation**
Projektanzahl	COUNT(DISTINCT projectID)

Dimensionen	
Name	**Klassifikationsschema**
Partner	Partnername → Stadt → Region → Land
Projekt	Projektname → EU-Programm → EU-Programmgruppe
Projektstart	Tag → Monat → Quartal → Jahr
Projektende	Tag → Monat → Quartal → Jahr
Projektstatus	Projektstatus

Abbildung 7.2: Schema des Würfels *Projektanzahl*

Mit Hilfe von MDX erlaubt Analysis Service neben der Verwendung einfach berechneten Measures die Definition von Kenngrößen, die sich aus anderen Measures berechnen. Dieses findet Anwendung im zweiten Würfel, der über der Faktentabelle *facts_2* definiert ist:

Measures	
Name	**Aggregation**
Projektanzahl	COUNT(projectID)
Projektvolumen	SUM(projectCost)
Projektfinanzierung	SUM(projectFinancing)
Prozentuale Projektfinanzierung	100 / [Measures][Projektvolumen] x [Measures][Projektfinanzierung]

Dimensionen	
Name	**Klassifikationsschema**
Projekt	Projektname → EU-Programm → EU-Programmgruppe
Projektstart	Tag → Monat → Quartal → Jahr
Projektende	Tag → Monat → Quartal → Jahr
Projektstatus	Projektstatus

Abbildung 7.3: Schema des Würfels *Projektkosten*

Die beiden Würfel zeigen mögliche Modellierungen multidimensionaler Datenstrukturen auf der Projektdatenbank. Sie erlauben die Beantwortung von Fragestellungen wie z.B. „*Wie viele Projekte wurden im Jahr 2000 abgeschlossen?*“ oder „*Welche Finanzierung liegt den einzelnen EU-Programmgruppen zu Grunde?*“.

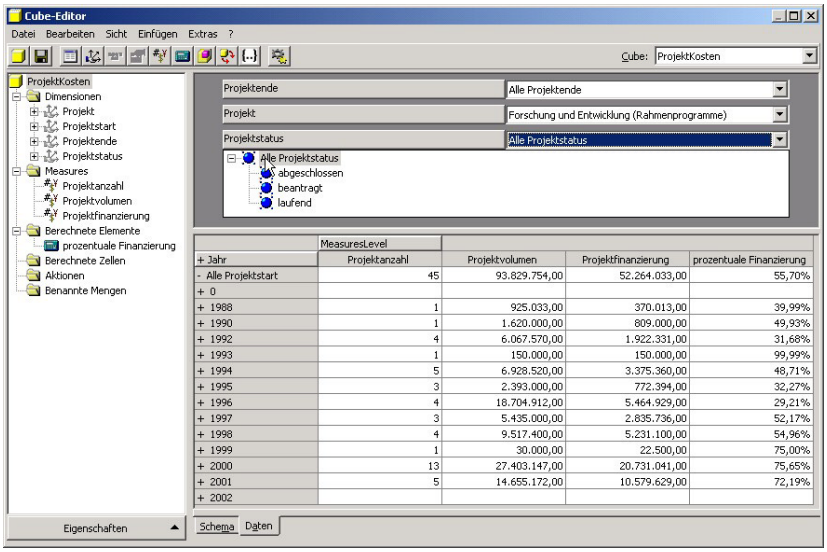

Abbildung 7.4: Analysis Service – Cube Editor

Die Untersuchung mit Hilfe von Analysis Service dient zunächst nur der Definition sinnvoller OLAP-Würfel. Wie eine Integration von OLAP-Funktionalität in die gegebene XML-Framework-Architektur erfolgen kann, zeigen die folgenden Abschnitte.

7.2 Anforderungen und Einschränkungen

In der OLAP-Einführung (Kapitel 3) wurden die Konstrukte des multidimensionalen Modells, die multidimensionalen Operatoren und die unterschiedlichen Speicherungsformen vorgestellt. Die in dieser Arbeit vorgestellte Erweiterung der Architektur soll eine geeignete Unterstützung von OLAP-Funktionalität bieten. Dieses bedarf die besondere Berücksichtigung verschiedener Punkte:

- Im Unterschied zum Dokumentenretrieval ist OLAP ein dynamischer Analyseprozess, basierend auf einer komplexeren Interaktion.
- Soll die OLAP-Unterstützung für verschiedene Endgerätegruppen zur Verfügung stehen, so sind speziell die Besonderheiten mobiler Endgeräte zu berücksichtigen.
- Die Visualisierung multidimensionaler Daten kann auf unterschiedliche Weise erfolgen. Angemessen an der Performance der Endgeräte ist es notwendig, geeignete Festlegungen für die Visualisierung aufzustellen.
- Nutzt ein Endgerät bereitgestellte OLAP-Funktionalität, ist Grundlage hierfür die Bereitstellung von Kommunikation zwischen Server und Client. Den Kommunikationsaufwand und die zu übertragenen Daten gilt es geeignet zu minimieren.

Mit der Integration von OLAP-Funktionalität in die bestehende XPEA-Architektur ist nicht die Entwicklung eines vollständigen OLAP-Werkzeuges angestrebt, so dass hier zur Vereinfachung an verschiedenen Punkten Einschränkungen vorgenommen werden sollen:

- **Relationale Speicherung mit Star-Schema**

  Für die Erweiterung des Systems wird von einer relationalen Speicherung der multidimensionalen Daten ausgegangen. Zur Abbildung der multidimensinalen Strukturen in das relationale Model eignen sich das Snowflake- und Star-Schema. Bietet das Datenbankschema der Basisdatenbank keine der beiden Schemaformen, so kann diese in der Regel durch zusätzlich definierte Sichten in der Datenbank bereitgestellt werden. Diese Vorgehensweise wurde beispielsweise auch für die Untersuchung in Abschnitt 7.1 angewendet.

  Da sich ein Snowflake-Schema prinzipiell durch Denormalisierung der einzelnen Dimensionstabelle (dieses kann auch durch geeignete Sichtendefinition geschehen) auf ein Star-Schema abbilden lässt, soll nunmehr nur das Star-Schema in Betracht gezogen werden.

- **Dimension mit einfachen Hierarchien**

  Bei der Entwicklung einer erweiterten Architektur wird von Dimensionen mit einfachen Hierarchien ausgegangen.

- **Einfache Aggregationen bei den Measures**

  Measures können sich durch einfache Aggregation aus den atomaren Datenwerten oder durch Berechnung aus anderen Kenngrößen ergeben. Folgend werden nur einfach aggregierte Measures betrachtet.

- **Sichten in Form einer Kreuztabelle mit zwei einbezogenen Dimensionen**

  Um dem Endanwender verständliche Sichten auf einen OLAP-Würfel visualisieren zu können und eine Unterstützung mobiler Endgeräte zu gewährleisten, wird bei der Visualisierung der Sichten von einer Kreuztabelle mit zwei Dimensionen ausgegangen. Da eine Kreuztabelle auch in Form eines Diagramms dargestellt werden kann, stellt die Festlegung der Darstellungsform eigentlich keine Einschränkung dar.

- **Keine Betrachtung von Nutzerschnittstellen**

  Die Umsetzung einer geeigneten clientseitigen Nutzerschnittstelle ist nicht Aufgabe dieser Diplomarbeit und soll daher nicht weiter betrachtet werden. Ziel ist die Bereitstellung von OLAP-Funktionalität und einer damit verbundenen Kommunikationsschnittstelle auf dem Server. Bei der im Rahmen des XPEA-Projektes erweiterten Architektur werden die Nutzerschnittstellen durch eine Nutzung von Technologien wie XForms realisiert.

7.3 Metadaten für OLAP-Würfel – Erweitertes RDF-Modell

Für die Definition von multidimensionalen Datenstrukturen sind Metadaten unabdingbar. Unter Metadaten sind hierbei Informationen zu verstehen, wie die Schemabeschreibungen des OLAP-Würfels oder Schemadaten der dazugehörigen Datenbank, die die Basisdaten enthalten.

Dieser Abschnitt gibt einen Überblick zu Metadaten, die für den Entwurf, die Konstruktion und die Benutzung von multidimensionalen Datenmodellen am Retrieval Server der verwendeten XML-Framework-Architektur notwendig sind.

7.3.1 Metadaten und Data-Warehousing

Im Bereich des Data-Warehousing – dieses schließt OLAP ein – sind Metadaten, Metadatenmodelle, Austauschformate für Metadaten sowie Metadatenmanagementsysteme (Data-Warehouse-Repositories) zentrale Themen der Entwicklung und Forschung. Zur Verbesserung der Interoperabilität zwischen OLAP-Werkzeugen existieren verschiedene Standardvorschläge. Als Vorschläge für Metadatenmodelle sind hierbei das *Open Information Model* (OIM) von der *Meta Data Coalition* (MCD)[1] und das *Common Warehouse Metamodel* (CWM) von der *Object Management Group* (OMG)[2] zu nennen. Beide basieren auf der *Unified Modeling Language* (UML) und verwenden XML als Austauschformat.

Die genannten Standardvorschläge stellen Ansätze für komplexe Metadatenmodellen zum Einsatz bei umfangreichen Data-Warehouse-Applikationen vor. Die Austauschformate bieten Vorausetzungen für die Interoperabilität zwischen Repositories. Ziel der Vorschläge ist die Involvierung des Metadatenmodells für den gesamten Bereich des Data-Warehousing. Dieses schließt beispielsweise auch Punkte wie Datenschutz, Sicherheitsaspekte und Unterstützungen für Analysetechniken wie Data-Mining ein.

Die im Rahmen der Diplomarbeit zu entwickelnde Erweiterung der ausgewählten Architektur strebt keine vollständige Kopplung von Data-Warehouse-Techniken an. Die den Standards zu Grunde liegenden Metadatenmodelle wären viel zu umfangreich. Ziel ist zudem die Nutzung des zur Verfügung stehenden RDF-Content-Repositories für die Verwaltung der Metadaten und die Integration dieser Metadaten in das bestehende RDF-Metadatenmodell der XML-Framework-Architektur. Aus diesen Gründen werden die genannten Standards nicht weiter betrachtet.

Folgend wird ein Vorschlag für ein geeignetes, applikationsspezifisches, RDF-basiertes Metadatenmodell gegeben.

7.3.2 Erweitertes, RDF-basiertes Metadatenmodell

Eine OLAP-Anwendung analysiert Daten in multidimensionaler Form. Wesentliche Begriffe in diesem Kontext sind Würfel, Measures, Dimensionen und Hierarchien. Diese gilt es mit Hilfe eines geeigneten Metadatenmodells zu beschreiben. Die folgenden Anforderungen werden hierbei an das Modell gestellt:

- **Flexibel:** Das Modell muss eine Beschreibung von unterschiedlich definierten, multidimensionalen Schemata unterstützen.
- **Vollständig:** Vom Modell müssen alle für die Bereitstellung der OLAP-Funktionalität notwendigen Metainformationen aufgenommen werden können.
- **Minimal:** Das Modell soll dabei nur die notwendigen Metadaten enthalten.
- **Strukturiert:** Eine geeignete Strukturierung des Modells bietet eine gute Übersichtlichkeit, so dass die Definition von multidimensionalen Schemata vereinfacht wird.

[1] Meta Data Coalition (MCD). http://www.mcdinfo.com

[2] Object Management Group (OMG). http://www.omg.org

- **Funktional:** Für eine optimale Unterstützung der OLAP-Analyse sollen die Metainformationen im Modell geeignet zur Formulierung multidimensionaler Anfragen abgebildet werden.
- **RDF-basiert:** Zur Beschreibung der Metadaten soll RDF verwendet werden. Dieses ermöglicht eine geeignete Integration in die XML-Framework-Architektur und die Verwaltung der Metadaten im entwickelten RDF-Content-Repository.
- **Adäquat:** Die zur Transformations- und Anfragesteuerung liegt der XPEA-Architektur eine umfangreiches RDF-Modell zugrunde. Das Metadatenmodell zur Beschreibung eines OLAP-Würfels soll adäquat zu dem bestehenden Modell sein.

Mit dem Metadatenmodell müssen verschiedene Objekte (Konstrukte des multidimensionalen Datenmodells und Speicherungsform der Daten) beschrieben werden. Die folgenden Objekte sind hierbei zu berücksichtigen:

- **Würfel:** Die Architektur soll die Möglichkeit zur Verwaltung verschiedener *OLAP-Würfel* bieten. Das Suchen eines Würfels ist zudem wünschenswert. Der Würfel wird hierzu mit den Informationen, wie einem *Namen*, einer *Beschreibung* und *Schlüsselwörtern* beschrieben.
- **Measures:** Ein OLAP-Würfel kann ein oder mehrere *Measures* besitzen. Measures haben einen *Namen*. Bei einer relationalen Speicherung in einem Star-Schema basieren diese auf einem *Attribut* in einer *Faktentabelle* und einer über dem Attribut definierten *Aggregation*.
- **Dimension:** Neben den Measures sind *Dimensionen* wesentlicher Bestandteil eines OLAP-Würfels. Dimensionen besitzen einen *Namen* und ihnen liegt eine *Dimensionstabelle* zu Grunde. Zur Beschreibung einer Dimension gehört ein *Klassifikationsschema*, welches aus einer geordneten Menge von *Klassifikationsstufen* mit einem *Namen* und einem dazugehörigen *Attribut* in der Dimensionstabelle besteht. Die Klassifikationshierarchie einer Dimension ist ein balancierter Baum, deren Knoten sich aus den Daten in der Tabelle ergeben. Im Bereich OLAP werden die Beschreibungen der Klassifikationshierarchien in der Regel als Metadaten aufgefasst. Dieses soll hier nicht der Fall sein.
- **Star-Schema:** Das *Star-Schema* stellt eine geeignete relationale Speicherung von multidimensionalen Daten dar. Es besteht aus einer *Faktentabelle* und mehreren *Dimensionstabellen*. Zwischen den Tabellen bestehen *Beziehungen*.

Die Abbildung E.1[1] in Anhang E.1 zeigt auf Grundlage der oben genannten Punkte und unter Berücksichtigung der Anforderungen den generellen Aufbau eines RDF-basierten Metadaten-modell zur Schemabeschreibung eines OLAP-Würfels. Das Modell zu Beschreibung des OLAP-Würfels strukturiert sich anhand der oben genannten Objekte.

[1] Durch die Verwendung des SquishQL-Anfrageinterpreters Inkling ergaben sich bei der Umsetzung des RDF-Content-Repository verschiedene Probleme (siehe Abschnitt 8.1). Für ein effizienteres Metadatenretrieval wurde daher zur Beschreibung eines Würfels auf ein vereinfachtes Metadatenmodell (Abbildung E.2 in Anhang E.1) zurückgegriffen.

Die Metabeschreibungen der Würfel lassen sich in den Metadatenbestand der XPEA-Plattform integrieren. Sie stellen dabei eine Erweiterung des Application Context dar und können durch das RDF-Content-Repository verwaltet werden.

7.4 Bereitstellung von OLAP-Funktionalität

Im vorherigen Abschnitt wurde gezeigt, wie die Verwaltung und der Zugriff durch die Verwendung von Metadaten zur Beschreibung des multidimensionalen Schemas auf dem Retrieval Server der XML-Framework-Architektur realisiert werden kann. Soll nun von dem System OLAP-Funktionalität bereitgestellt werden, so kann dieses auf verschiedene Art und Weise erfolgen. Wie eine geeignete Realisierung aussehen kann, zeigt dieser Abschnitt.

7.4.1 Clientseitiger Zugriff auf multidimensionale Daten

Um dem Endgerät den Zugriff auf einen OLAP-Würfel zu ermöglichen, sind verschiedene Ansätze denkbar. Drei Ansätze sollen hier vorgestellt werden:

Laden des gesamten OLAP-Würfels

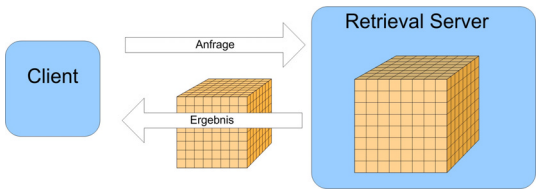

Abbildung 7.5: Laden des gesamten OLAP-Würfels

Der multidimensionale Würfel liegt auf dem Server bereit und kann für Analysen von einem Client geladen werden. Hierbei werden die Schemabeschreibungen und die gesamten Daten des Würfels an den Client übertragen. Der Analyseprozess erfolgt dann nur auf dem Client. Die Umsetzung der Logik für die Analyse ist Bestandteil des Clients.

Gegen die Nutzung des Ansatzes in diesem Anwendungsfall sprechen verschiedene Gründen:

- Der komplette OLAP-Würfel muss an den Client übertragen werden, obwohl gegebenenfalls nur einige Sichten bei der Analyse erforderlich sind.
- Der Datenumfang eines OLAP-Würfels kann sehr groß sein und auch mal mehrere Gigabytes umfassen. Dieses Datenvolumen kann bereits für einen gewöhnlichen Desktop-PC eine Herausforderung bedeuten, ganz abgesehen von mobilen Endgeräten.
- Die Anwendungslogik für den Analyseprozess muss auf dem Client umgesetzt werden. Dieses setzt einerseits voraus, dass die Endgeräte über genügend Rechenleistung verfügen, und bedeutet auf der anderen Seite die Entwicklung verschiedener Programme für unterschiedliche Endgeräte.

Bereitstellung des OLAP-Würfels auf einem Webserver

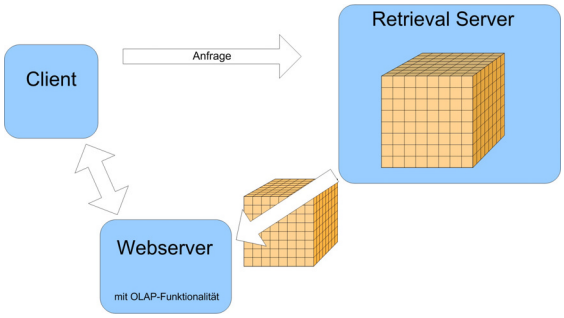

Abbildung 7.6: Bereitstellung des OLAP-Würfels auf einem Webserver

In diesem Ansatz wird der Würfel über einem Webserver zur Verfügung gestellt. Hierzu muss der Würfel in ein geeignetes Format konvertiert und beim Webserver abgelegt werden. Bei der Konvertierung ist es möglich nur eine Teilmenge des Würfels (Data Mart) zu erstellen. Eine Selektion der für die Analyse notwendigen Informationen ist jedoch nur sehr schwer möglich. Die eigentliche OLAP-Funktionalität (Bearbeitung der OLAP-Operationen etc.) bietet der Webserver.

Online Analyse mit der Anfrage von Sichten

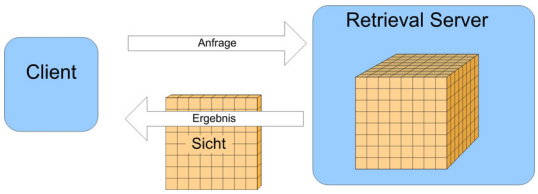

Abbildung 7.7: Online Analyse mit der Anfrage von Sichten

Ein anderer, in dieser Arbeit favorisierter Ansatz ist die Integration der OLAP-Funktionalität in den Retrieval Server. Der Server stellt eine Anfrageschnittstelle bereit, mit deren Hilfe Sichten auf den Würfel vom Endgerät angefragt werden können. Der Ansatz ermöglicht die Online Analyse ohne Zwischenkomponente, bei der, durch die clientseitige Generierung von Anfragen, die zu übertragende Datenmenge auf die gerade nur benötigten Daten reduziert wird. Es besteht zudem die Möglichkeit, die vom Server generierten Sichten endgeräteabhängig aufzubereiten.

Die Anfrage von Sichten bei einer Online Analyse lässt sich verschiedenartig realisieren. Drei Realisierungsformen werden im nächsten Abschnitt vorgeschlagen.

7.4.2 Online Analyse – Anfrage von Sichten

Eine clientseitige Anfrage von Sichten ist in den Folgend dargestellten Formen realisierbar:

Nutzung vordefinierter Sichten

Eine sehr einfache Realisierung ist die Bereitstellung von vordefinierten Sichten. Der Administrator des Systems richtet verschiedene Sichten auf den Würfel ein, die vom Endanwender angefragt werden können und auf Basis der aktuellen Daten generiert werden.

Bei dieser Realisierung geht jedoch die Flexibilität der eigentlichen OLAP-Analyse durch die statisch definierten Sichten verloren.

Sichtdefinierende Anfragen

Im Vorfeld der Analyse erhält der Client die notwendigen Metainformationen (Measures, Dimensionen, Klassifikationsschema, Klassifikationshierarchie). Mit Unterstützung eines GUI kann der Endanwender Sichten auf den Würfel definieren und diese vom Server anfragen.

Problematisch an dieser Realisierung ist die hohe Datenmenge der Metainformation, die zur Definition einer Sicht notwendig ist und an den Client übertragen werden muss. So wird zur Beschreibung der Dimensionen unter anderem die komplette Informationsmenge der einzelnen, normierten Dimensionstabellen (siehe Snowflake-Schema, Abschnitt 3.5) benötigt. Aufgrund der hohen Datenmenge ist dieser Ansatz daher für Endgeräte mit umfangreicher Speicherausstattung und breitbandigem Netzwerkzugang geeignet.

Durch Weglassen der Klassifikationshierarchie kann das Datenaufkommen wesentlich reduziert werden, so dass dieser Realisierungsansatz auch für Endgeräte mit eingeschränkten Ressourcen, wie z.B. mobile Endgeräte, möglich wäre. Hierbei sind jedoch gewisse Kenntnisse des Endanwenders über den Würfel Vorausetzung.

Operationale Anfragen

Ein für mobile Endgeräte besser geeigneter Ansatz ist der hier vorgestellte. Bei der ersten Anfrage innerhalb eines OLAP-Analyseprozesses erhält der Client eine vordefinierte Sicht auf den Würfel, inklusive einiger Metainformationen zum Würfel. Auf die Sicht des Würfels können vom Endanwender OLAP-Operationen ausgeführt werden. Zur Reduktion der zu übertragenden Metainformationen werden die OLAP-Operationen in gewisser Hinsicht eingeschränkt und nur die Metabeschreibungen an den Client gesendet, die für die nächst möglichen Operationen notwendig sind. Einschränkungen auf den OLAP-Operationen sind:

- Ein Roll-up auf einer in der Kreuztabelle enthaltenen Dimension kann nur um eine Ebene erfolgen. Die Operation wird dabei gleichzeitig mit einem Slice & Dice verbunden. In der visuellen Darstellung der Tabelle kann der Endanwender demnach eine Spalte beziehungsweise Zeile auswählen, für die eine detailliertere Darstellung erfolgen soll.
- Ein Drill-down erfolgt analog zum Roll-up in umgekehrter Richtung, mit einer damit verbundenen Aufhebung von Slice & Dice für eine Ebene.

- Die Slice & Dice Operation für Dimensionen die nicht Bestandteil der Kreuztabelle sind sowie deren Umkehrung erfolgt ebenfalls nur über eine Ebene.
- Operationen wie Drill-across und Rotation bleiben unverändert.
- In einem Analyseschritt kann nur **eine** Operation ausgeführt werden.

Zur Ausführung der Operationen sendet der Client an den Retrieval Server innerhalb der Anfrage Metadaten, die die aktuelle Sicht und die gewählte Operation beschreiben. Der Server kann anhand dieser Informationen die neue Sicht generieren und diese angereichert um Metainformationen für die darauf folgenden, ausführbaren Operationen als Antwort zurückgeben. Dem Endanwender ist so die Möglichkeit gegeben, verschiedene Sichten auf den Würfel schrittweise zu erfragen. Vorteile an dieser Realisierung sind:

- Das zu übertragende Datenvolumen ist gegenüber dem zuvor vorgestellten Ansatz stark reduziert.
- Das Endgerät muss für die Analyse keine Metainformationen des Würfels speichern.
- Die Programmlogik zur Bereitstellung des Analyseprozesses befindet sich auf dem Server. Das Endgerät benötigt lediglich ein Programm zur Ergebnisdarstellung und Anfragegenerierung.

Dagegen wirken sich die eingeschränkte Funktionalität der OLAP-Operatoren (nur schrittweise Navigation) und die gegebenenfalls redundante Übertragung von Metainformationen nachteilig aus. Unter dem Gesichtpunkt, dass die Visualisierung von Tabellen mit einer umfangreichen Spalten- beziehungsweise Zeilenanzahl auf einem mobilen Endgerät mit geringer Bildschirmauflösung nur schwer realisierbar ist, stellt die Einschränkung hinsichtlich der Verbindung der Roll-up-Operation mit einem Slice & Dice (analog Drill-down) keinen wesentlichen Nachteil dar.

Fazit: Die aufgeführten Vorschläge zeigen unterschiedliche Realisierungsmöglichkeiten zur Bereitstellung von OLAP-Funktionalität auf verschiedenen Endgeräten. Statt einer direkten Datenübertragung zwischen Server und Client, wie sie oben dargestellt wurden, kann die Kommunikation auch über eine Zwischenkomponente erfolgen, zum Beispiel über einem HTTP-Server, der die Informationen in HTML-Dokumente einbindet.

In dieser Arbeit werden die beiden zuletzt genannten Vorschläge realisiert. Die Umsetzung mit vordefinierten Sichten ist als trivial anzusehen und soll daher keine weitere Betrachtung finden.

7.5 Kommunikation zwischen Server und Client

Gemäß den zuvor diskutierten Realisierungsformen (Online Analyse mit sichtdefinierenden und operationalen Anfragen) ist die Kommunikation zwischen Server und Client zu organisieren. Hierbei soll die Kommunikationsschnittstelle der bestehenden XML-Framework-Architektur nicht neu entwickelt, sondern erweitert werden. Wie eine geeignete Erweiterung aussehen kann, wird in diesem Abschnitt dargestellt.

Im bestehenden System erfolgen die Anfragen und Antworten in XML-Syntax. Deren Struktur ist in DTDs (siehe [Ditt01]) festgelegt.

Für die Suche von OLAP-Würfeln sind hinsichtlich der Kommunikation keine Anpassungen vorzunehmen. Neben den bereits vom System unterstützen Dokumenten, wie Text, Bild und andere, kann auch ein OLAP-Würfel zunächst als ein Dokument aufgefasst werden. Ergebnis des Retrievals wären dann eindeutige Identifikatoren der passenden Würfel.

Im Gegensatz zu den bisher unterstützten Dokumenten des Retrieval Servers erfolgt der Zugriff auf einen OLAP-Würfel nicht in einem einzigen Ladevorgang, sondern durch eine schrittweise Anfrage von Sichten. Clientseitig kann dabei zwischen sichtdefinierenden oder operationalen Anfragen als Zugriffsform gewählt werden. Deren Unterstützung erfordert Veränderungen bei der Kommunikation und die damit verbundenen Erweiterungen der DTDs. In den DTDs sind Strukturen zu definieren, die die Übertragung von

- Beschreibungen des multidimensionalen Würfels,
- Sichten auf den Würfel,
- Beschreibungen von Sichten und
- Beschreibungen von OLAP-Operationen

erlauben. Vorschläge zur XML-basierten Beschreibung von multidimensionalen Würfeln finden sich in [Hard01] und [MiWe02]. Die beiden Ansätze sind in deren Form jedoch für die hier benötigten Beschreibungen nicht anwendbar, so dass eigene Strukturen zur XML-basierten Beschreibung der oben genannten Informationsobjekte entwickelt wurden.

Für die Kommunikation wurden die in Anhang E.2 dargestellten DTDs definierten. Die Abbildungen E.1 und E.2 illustrieren zusätzlich den daraus folgenden, grundlegenden Aufbau der Anfrage- und Ergebnisdokumente.

Der bei der Online Analyse auftretende Datentransfer zwischen Server und Client wird für die beiden Zugriffsformen nun kurz erläutert.

7.5.1 Kommunikation bei Sichtdefinierenden Anfragen

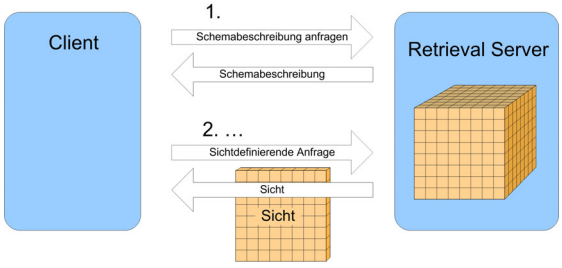

Abbildung 7.8: Kommunikation bei sichtdefinierenden Anfragen

Grundlage für die Anfrage von Sichten sind Kenntnisse über den Aufbau des OLAP-Würfels. Vom Client werden daher zunächst die Schemabeschreibungen angefragt (*CubeMetaData*). Entsprechend der in Abschnitt 7.4.2 beschriebenen Zugriffsform bieten sich für den Server in Abhängigkeit des *Device Context* zwei verschiedene Formen der Antwort:

- Eine detaillierte Schemabeschreibung (Measures, Dimensionen mit Klassifikationsschema und Klassifikationshierarchie) eignet sich für Endgeräte mit ausreichenden Ressourcen.

- Für mobile Endgeräte ist dagegen eine reduzierte Beschreibung vorteilhafter. Hierbei wird auf die Klassifikationshierarchie der Dimensionen verzichtet.

Nachdem der Client die Schemabeschreibungen geladen hat, können darauf basierend Sichten definiert und angefragt werden. In der Anfrage (*CubeView*) spezifiziert der Client die folgenden Parameter einer Sicht auf den multidimensionalen Würfel:

- Measure der Sicht (*Measure*)
- Dimensionen der Kreuztabelle (*TableDimension*) mit Einschränkungen entlang des Hierarchiepfads (*HierachyPath*), Klassifikationsstufe in der Sicht und Ausrichtung in der Tabelle
- Dimensionen (*Dimension*) mit Einschränkungen entlang des Hierarchiepfads (*HierarchiePath*)

Die vom Retrieval Server darauf berechnete Antwort (*CubeView*) besteht aus den Informationen:

- Measure der Sicht (*Measure*)
- Dimensionen der Kreuztabelle (*TableDimensionLarge*) mit Beschreibungen zur sichtbaren Klassifikationshierarchie und zusammengefassten Aggregationen in der Tabelle
- Tabelle mit Aggregationen (*Table*)
- Gesamtaggregation (*TotalAggregation*)

Auf Grundlage der im ersten Schritt der Online Analyse ermittelten Schemabeschreibungen kann die Anfrage von Sichten beliebig oft vom Client wiederholt werden.

7.5.2 Kommunikation bei Operationalen Anfragen

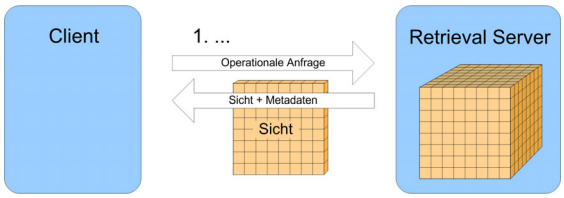

Abbildung 7.9: Kommunikation bei operationalen Anfragen

Die Kommunikation bei operationalen Anfragen unterscheidet sich von der bei sicht-definierenden Anfragen zunächst dadurch, dass im ersten Anfrageschritt eine clientseitige Ermittlung der Schemabeschreibung des Würfels entfällt. Stattdessen stellt der Client eine Anfrage (CubeOperation) mit leerem Anfrageblock. Diese veranlasst den Retrieval Server, die abstrakteste Sicht auf den OLAP-Würfel zu generieren und als Ergebnis an den Client zu senden. Neben den Daten und Beschreibungen der Sicht beinhaltet das Ergebnis notwendige Informationen für darauf folgende OLAP-Operationen, die bereits in Abschnitt 7.4.2 für operationale Anfragen vorgestellt wurden. Das Ergebnis (*CubeOperationView*) besteht aus folgenden Teilen:

- Measure der Sicht (*Measure*)

- Dimensionen der Kreuztabelle (*TableDimensionSmall*) mit Beschreibungen zur sichtbaren Klassifikationshierarchie und zusammengefassten Aggregationen in der Tabelle
- alle weiteren Dimensionen (*Dimension*) mit dem aktuellen Klassifikationspfad (*HierarchyPathTreeWithNextLevel*) und den Knoten der Klassifikationshierarchie der darauf folgenden Klassifikationsstufe (*Leaf*).
- Tabelle mit Aggregationen (*Table*)
- Gesamtaggregation (*TotalAggregation*)
- vorhandene Measures für den Würfel (*Measures*)

Mit den Informationen kann clientseitig die Sicht auf den OLAP-Würfel in Form einer Kreuztabelle dargestellt werden. Soll eine OLAP-Operation auf die aktuell dargestellte Sicht ausgeführt werden, so sendet der Client eine Anfrage (*CubeOperation*) an den Retrieval Server mit einer Beschreibung der Sicht (*CurrentCubeMetaData*) sowie Informationen über die auszuführende Operation (*RollUp_SliceDice*, *DrillDown_UndoSliceDice*, *SliceDice*, *UndoSlice-Dice*, *DrillAcross* oder *Rotation*). Der Server setzt die Operation auf die Sicht um und sendet das Ergebnis mit den notwendigen Informationen für sich daran anschließende, ausführbare OLAP-Operationen an das Endgerät.

Der Vorgang kann beliebig oft wiederholt werden, wobei der Client jedes Mal eine Anfrage, bestehend aus Zustandsbeschreibung und der auszuführenden Operation, an den Server sendet. Dieser liefert daraufhin die neue Sicht sowie die zusätzlichen Informationen für die Beschreibung weiterer ausführbarer OLAP-Operationen.

Fazit: Für die beiden Zugriffsformen wurden geeignete Wege zur Übertragung der notwendigen Daten beschrieben. Die vorgestellten Kommunikationsabläufe ermöglichen die Unterstützung der multidimensionalen Analyse auf dem Client in Form eines iterativen Analyseprozesses. Wie die vom Client gestellten Anfragen serverseitig interpretiert und bearbeitet werden, ist Thema des nächsten Abschnittes.

7.6 Anfragebearbeitung und Transformation

In den vorhergehenden Abschnitten wurde gezeigt, wie ein multidimensionaler Würfel serverseitig verwaltet, OLAP-Funktionalität auf dem Client bereitgestellt und die Kommunikation zwischen Server und Client aufgebaut werden kann. Für eine Bearbeitung der clientseitigen Anfragen müssen diese vom Server geeignet interpretiert und in Anfragen an das RDF-Content-Repository und die integrierten Datenbanken transformiert werden. Aus den ermittelten Daten ist vom Retrieval Server ein Ergebnis in Form eines XML-Dokumentes zu generieren, welches an das Endgerät gesendet wird.

In diesem Abschnitt wird ein Überblick über die einzelnen Arbeitsschritte im Retrieval-Server-Prozess für die Bearbeitung der verschiedenen Anfragen gegeben.

Bevor auf die Bearbeitung der einzelnen Anfragen näher eingegangen wird, fasst die Tabelle 7.1 zur besseren Übersicht noch einmal zusammen, auf welche Informationsquellen der Retrieval Server zur Bearbeitung der unterschiedlichen Anfragen zugreift.

	RDF-Content-Repository	Integrierte Datenbank(en)
Würfel suchen	x	
Schemabeschreibung ermitteln	x	x [1]
Sichtdefinierende Anfrage	x	x
Operationale Anfrage	x	x

Tabelle 7.1: Informationsquellen zur Anfragebearbeitung

Würfel suchen

Im Gegensatz zum Dokumentenretrieval erfolgt eine Suche nach OLAP-Würfeln nicht auf Basis der integrierten Datenbanken, sondern nur über Beschreibungen der OLAP-Würfel im Metadatenbestand des RDF-Content-Repositories. Ist beispielsweise eine OLAP-Analyse zum Thema *Projekte* gewünscht, so können mit der SquishQL-Anfrage aus Listing 7.1 über das RDF-Content-Repository die entsprechenden Würfel gesucht werden.

```
SELECT ?cubeId, ?name, ?abstract
WHERE  (urn:cube#keywords ?cubeId ?keyword)
       (urn:cube#name ?cubeId ?name)
       (urn:cube#abstract ?cubeId ?abstract)
AND ?keyword ~ Projekte
```

Listing 7.1: SquishQL-Anfrage zum Suchen eines OLAP-Würfels

Ergebnis der Anfrage sind die IDs, Namen und textuelle Beschreibungen der passenden Würfel. Die Informationen können in ein XML-Dokument eingebettet und als Ergebnis der Suchanfrage an den Client übertragen werden.

Schemabeschreibungen ermitteln

Im ersten Schritt der Nutzung sichtdefinierender Anfragen ist die Ermittlung des multidimensionalen Schemas des OLAP-Würfels erforderlich. Entsprechend dem Endgerät entscheidet der Retrieval Server, wie detailliert die Beschreibung erfolgt.

Für eine reduzierte Beschreibung werden die Measures und Dimensionen mit den dazugehörigen Klassifikationsschemata ermittelt. Das Listing Listing 7.2 zeigt, wie beispielsweise ausgehend von der gegebenen ID des Würfels die Informationen der Dimensionen vom Metadatenbestand des Systems angefragt werden können.

```
SELECT ?dimName, ?levelName, ?levelOrdinal
WHERE  (urn:cube#dimensions online.Europa-MV.cube ?dimId)
       (urn:dimension#name ?dimId ?dimName)
       (urn:dimension#classificationSchemaLevel ?dimId ?levelId)
       (urn:classificationLevel#name ?levelId ?levelName)
       (urn:classificationLevel#ordinal ?levelId ?levelOrdinal)
```

Listing 7.2: SquischQL-Anfrage zur Ermittlung der Dimensionsinformationen

[1] Daten aus der integrierten Datenbank werden nur zur Beschreibung der Klassifikationshierarchie bei Endgeräten mit ausreichender Performance benötigt.

Ergebnis der Anfrage sind die Namen der Dimensionen und deren Klassifikationsschemata, bestehend aus den Klassifikationsstufen und dessen Positionen. Die Informationen werden in XML folgendermaßen dargestellt:

```
...
<Cube cubeId="online.Europa-MV.cube">
  <CubeMetaData>
    <Measures>
      ...
    </Measures>
    <Dimensions>
      ...
      <DimensionShort name="Projekt">
        <ClassificationSchema>
          <Level name="EU-Programmgruppe">
            <Level name="EU-Programm">
              <Level name="Projektname"/>
            </Level>
          </Level>
        </ClassificationSchema>
      </DimensionShort>
...
```

Listing 7.3:

Für eine detaillierte Schemabeschreibung werden über das RDF-Content-Repository zusätzlich die zur Dimension zugehörigen Relationen, die Attributnamen der Klassifikationsstufen und die Informationen zum Datenbankzugriff ermittelt. So können in einem weiteren Arbeitsschritt die Daten aus den Dimensionstabellen der integrierten Datenbank zum Aufbau der Klassifikationshierarchie erfragt werden.

Sichtdefinierende Anfragen

Ergebnis einer sichtdefinierenden Anfrage ist eine erstellte Sicht in Form einer Kreuztabelle auf den OLAP-Würfel. Um diese zu generieren, werden multidimensionale Anfragen benötigt. Grundlagen zur Formulierung solcher Anfragen sind mit den OLAP-Erweiterungen in SQL-99 (siehe Abschnitt 3.6.1) gegeben. Der allgemeine Aufbau der SQL-Anfrage, die zur Bearbeitung sichtdefinierender Anfragen verwendet wird, sieht folgendermaßen aus:

```
SELECT dim_1, dim_2, aggregation(measure)
FROM factTable, dimTable_1, dimTable_2 [, dimTable_x ...]
WHERE joinCondition
      AND dimRestriction
GROUP BY CUBE (dim_1, dim_2)
```

`dim_1, dim_2`	:	Attributnamen der gewählten Klassifikationsstufe für die beiden in Kreuztabelle enthaltenen Dimensionen
`aggregation`	:	Aggregationsfunktion des Measures
`measure`	:	Attributname des gewählten Measures
`factTable`	:	Name der Faktentabelle
`dimTable_1, dimTable_2`	:	Tabellennamen der Dimension, die in der Kreuztabelle enthalten sind
`dimTable_x`	:	Tabellennamen weiterer Dimensionen auf denen Einschränkungen festgelegt wurden
`joinCondition`	:	Verbundbedingungen der Tabellen anlog dem zu Grunde liegendem Star-Schema
`dimRestriction`	:	Einschränkungen auf Dimensionen

Listing 7.4: Allgemeiner Aufbau der SQL-Anfrage

Ist seitens der Datenbank keine Unterstützung der SQL-Erweiterung gegeben, so ist die Anfrage, wie in Listing 3.2 (Seite 24) beispielsweise gezeigt, durch Verwendung von *Union* und drei *Select*-Anfragen zu emulieren.

Bevor jedoch eine SQL-Anfrage an eine integrierte Datenbank gestellt werden kann, sind verschiedene Arbeitsschritte zu durchlaufen. In diesen werden:

1. durch Parsen der Anfrage vom Client die Beschreibung der Sicht (Measure, Dimensionen der Kreuztabelle etc.) extrahiert,
2. die Schemabeschreibungen des Würfels (Aggregation des Measures, Tabelle und Attribute für Dimensionen und Measure, Tabellenbeziehungen) sowie Datenbankzugriffsinformationen aus dem Metadatenbestand ermittelt und
3. die Anfrageparameter der sichtdefinierenden Anfrage vom Client auf Parameter der SQL-Anfrage abgebildet.

Die Abbildung 7.10 veranschaulicht, wie die Transformation einer sichtdefinierenden Anfrage in eine SQL-Anfrage der oben vorgestellten Form geschehen kann.

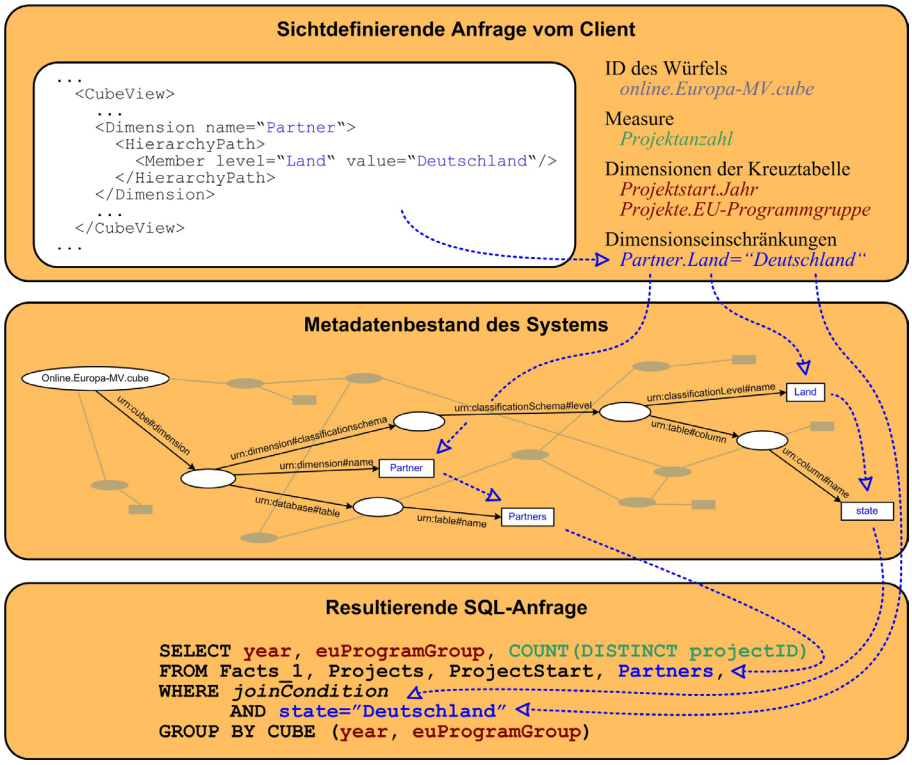

Abbildung 7.10: Anfragetransformation bei sichtdefinierenden Anfragen

Die Ergebnisrelation der SQL-Anfrage enthält die Daten der gewählten Sicht. Entsprechend der vorgegebenen Struktur können diese in einem XML-Dokumente abgebildet und an den Client zurückgegeben werden.

Durch die im Gegensatz zum Dokumentenretrieval komplexere Struktur des Ergebnis-dokumentes, erfolgt die Erstellung des Dokumentes nicht über die DATABox, sondern über spezielle Methoden.

In einer Weiterentwicklung der XPEA-Architektur (XML-Map) ist vorgesehen, die Generierung der Ergebnisdokumente mit Hilfe von Metadaten zu steuern.

Operationale Anfragen

Die serverseitige Bearbeitung operationaler Anfragen stellt gegenüber den sichtdefinierenden Anfragen keinen durchweg anderen Bearbeitungsprozess dar, sondern lässt sich vielmehr durch Hinzufügen verschiedener Arbeitschritte auf den zuvor vorgestellten Vorschlag zurückführen.

Bestandteile einer operationalen Anfrage sind die Beschreibung der Sicht auf den Würfel und Informationen über die darüber auszuführende Operation. Nach einer Analyse der Anfrage und geringfügig geändertem Metadatenretrieval auf dem Server kann aus den Informationen der Anfrage sehr einfach eine Bestimmung der neuen Sicht erfolgen. Stellt ein Client beispielsweise folgende operationale Anfrage:

```
...
<FetchCube cubeId="online.Europa-MV.cube">
  <CubeOperation>
    <CurrentCubeMetaData>
      ...
      <CurrentTableDimension name="Projekte" id="projects01" alignment="vertical">
        <HierarchyPath>
          <Member level="EU-Programmgruppe" value="Forschung und Entwicklung"/>
        </HierarchyPath>
      </CurrentTableDimension>
      ...
    </CurrentCubeMetaData >
    <RollUp_SliceDice tableDimension="projects01" upMemberValue="BIOTECH 2"/>
  </CubeOperation>
</FetchCube>
...
```

so kann mit Hilfe des im Metamodel gespeicherten multidimensionalen Schemas des OLAP-Würfels die hinzukommende Einschränkung

```
Projekte.EU-Programm="BIOTECH 2"
```

entlang der *Projekte*-Dimension ermittelt werden. Alle anderen Parameter für die neue Sicht ergeben sich aus der Beschreibung der aktuellen Sicht. Analog zu sichtdefinierenden Anfragen kann nun eine SQL-Anfrage an die integrierte Datenbank gestellt werden, um so die erforderlichen Daten der neuen Sicht auf den OLAP-Würfel zu gewinnen. Wie bei sichtdefinierenden Anfragen wird das Ergebnis der SQL-Anfrage in ein XML-Dokument eingebettet, wobei hier jedoch zusätzlich notwendige Schemainformationen zur Ausführung folgender OLAP-Operationen in das Dokument mit aufgenommen werden. Die Informationen hierzu lassen sich aus dem Metadatenbestand und den Dimensionstabellen ermitteln.

Teil IV
Implementierungen

8. Prototypische Implementierungen

Die in den vorherigen Kapiteln vorgeschlagenen Realisierungsansätze sind sowohl für die RDF-basierte, adäquate Metadatenverwaltung als auch für die serverseitige Bereitstellung von OLAP-Funktionalität prototypisch umgesetzt worden. Hauptziel der prototypischen Implementierungen ist die Demonstration der evaluierten Konzepte.

Ausgehend von den Schwerpunkten dieser Arbeit wurden die beiden folgenden Komponenten entwickelt:

- Das *RDF-Content-Repository* verwaltet den Metadatenbestand der XPEA-Plattform und stellt mit SquishQL eine Schnittstelle zum Metadatenretrieval bereit. Die Integration des Repositories erforderte notwendige Anpassungen beim Retrieval-Server-Prozess.
- Das *OLAP Retrieval Server Modul* (*OReSMo*) erweitert den Retrieval Server der XML-Framework-Architektur um OLAP-Funktionalität. Die Transformation der Ergebnisdaten, ermittelt durch das Metadatenretrieval über den Metadatenbestand des Systems und durch relationale Anfragen an die integrierte Datenbank, in XML-Dokumente erfolgt über eine Teilkomponente von OReSMo, der *OLAPBox*[1].

Die genannten Komponenten werden nun näher vorgestellt.

8.1 Metadatenverwaltung – RDF-Content-Repository

Zur adäquaten Speicherung von RDF-Daten und zur Bereitstellung einer RDF-Anfrageschnittstelle basiert das RDF-Content-Repository auf den in Kapitel 5 vorgeschlagenen Ansätzen:

- Relationale Speicherung der RDF-Daten in Tripelform
- Eindimensionale Indizierung mit einem zusammengesetzten Primärindex (*resource*, *property*) und einem Sekundärindex (*property*)

[1] Der Name OLAPBox soll die Bezüge zur OLAP-Technologie und zu der bereits entwickelten DATABox des Retrieval Servers verdeutlichen.

- SquishQL als RDF-Anfragesprache

Die umgesetzte Architektur des RDF-Content-Repository veranschaulicht Abbildung 8.1.

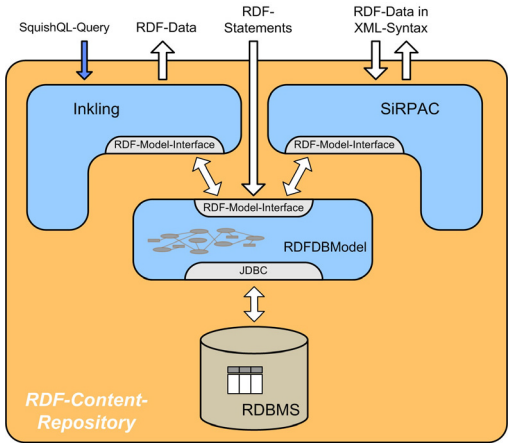

Abbildung 8.1: Architektur des RDF-Content-Repositories

Zentraler Kern der Architektur ist eine Implementierung des RDF-Modells (*RDFDBModel*). Die eigentlichen RDF-Daten werden hierbei in einer relationalen Datenbank verwaltet. Über das *RDF-Model-Interface* kann sowohl der *SiRPAC*-Parser als auch der SquishQL-Anfrage-interpreter *Inkling* auf das RDF-Modell zugreifen. Der SiRPAC-Parser ermöglicht das Hinzufügen von RDF-Daten in das Modell sowie das komplette Auslesen des RDF-Modells in ein XML-Dokument. Das RDF-Content-Repository erlaubt mit Hilfe von SquishQL-Anfragen ein Metadatenretrieval über den Metadatenbestand. Über einen direkten Zugriff auf das RDF-Model-Interface ist zudem das Einfügen, Löschen und Ändern von RDF-Statements möglich.

Probleme

Bei der Umsetzung des RDF-Content-Repository musste leider festgestellt werden, dass eine Bearbeitung von SquishQL-Anfragen durch die Referenzimplementierung Inkling keineswegs optimiert erfolgt. Bei der Anfragebearbeitung werden zum Beispiel die passenden RDF-Statements zu den Tripeln der SquishQL-Anfrage unabhängig voneinander ermittelt und erst im Hauptspeicher miteinander verknüpft.

```
SELECT ?dimName, ?levelName, ?levelOrdinal
WHERE  (urn:cube#dimensions online.Europa-MV.cube ?dimId)
       (urn:dimension#name ?dimId ?dimName)
```

Listing 8.1: SquishQL-Anfrage zu Ermittlung der Dimensionsnamen eines Würfels

Für die Anfrage aus Listing 8.1 bedeutet dieses, dass die Namen aller Dimensionen im RDF-Modell angefragt werden, obwohl die dem Würfel zugehörigen Dimensionen mit dem ersten Tripel der SquishQL-Anfrage selektiert werden. Ein weiteres Problem ergibt sich bei der Verwendung von Selektionsbedingungen im *AND*-Block einer SquishQL-Anfrage. Die Vergleichsbedingungen werden nicht an das Modell weitergeleitet, sondern vom Interpreter im Hauptspeicher ausgewertet.

Aufgrund dieser Problemstellungen stellt sich die Frage, wie bei einer adäquaten Speicherung der RDF-Daten sowie einem optimiertem Datenzugriff durch Indizierungstechniken auf Ebene

des Basisdatenbanksystems, eine effiziente RDF-Anfragebearbeitung gewährleistet werden kann, wenn die Implementierung der RDF-Anfragesprache dieses nicht zu nutzen vermag.

Ausgehend von den festgestellten Problemen ergeben sich für den Anfrageinterpreter Inkling folgende Verbesserungsvorschläge:

- Durch die Nutzung von Datenwerten aus Teilergebnissen für weitere Anfragen an das RDF-Modell können die dabei ermittelten Ergebnisse reduziert werden.
- Mit der Verwendung von statistischen Informationen des RDF-Modells kann die Reihenfolge von Teilanfragen an das Modell optimiert werden.
- Selektionsbedingungen durch Vergleichsoperatoren wie <, > oder ~ können direkt an das implementierte RDF-Modell weitergegeben werden. Diese können dort durch Datenbankfunktionalitäten abgebildet werden.
- Durch die gleichzeitige Anfrage mehrerer Tripel kann ein Verbund effizient und optimiert vom Basisdatenbanksystem durchgeführt werden.

Eine eigene Implementierung eines Anfrageinterpreters für SquishQL soll im Rahmen dieser Arbeit nicht vorgenommen werden.

Um die ineffiziente Bearbeitung von Anfragen durch den Anfrageinterpreter Inkling teilweise zu umgehen, sind an verschiedenen Stellen Anpassungen vorgenommen worden. So wird beispielsweise absichtlich auf die Verwendung von RDF-Containern im Metadatenbestand des Systems verzichtet. Durch den Verzicht können innerhalb der SquishQL-Anfrage Tripel mit drei Variablen vermieden und somit Ermittlungen von sehr umfangreichen Teilergebnissen umgangen werden.

8.2 Komponente zur Bereitstellung von OLAP-Funktionalität – OReSMo

In Kapitel 7 wurden Konzepte zur Bereitstellung von OLAP-Funktionalität auf dem Server vorgeschlagen. Im Rahmen dieser Arbeit wurde eine serverseitige Unterstützung der Online Analyse im Prototypen OReSMo umgesetzt.[1] OReSMo stellt eine Komponente zur Erweiterung des Retrieval Servers, speziell des Retrieval-Server-Prozesses, der XPEA-Plattform dar und bietet so die notwendige Funktionalität.

[1] In der derzeitigen Version bearbeitet OReSMo zunächst nur Anfragen zur Ermittlung der Schemabeschreibung eines Würfels und sichtdefinierende Anfragen.

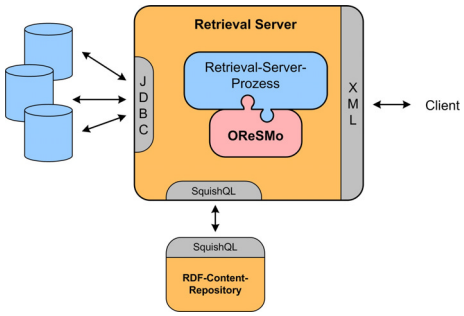

Abbildung 8.2: Erweiterung des Retrieval Server mit OReSMo

Clientseitige Anfragen zur Online Analyse werden vom Retrieval Server an OReSMo weitergeleitet. Entsprechend den in Abschnitt 7.6 vorgestellten Arbeitsschritten erfolgt in dem Modul die Interpretation der Anfragen durch den *OLAP-Retrieval-Server-Prozess*, der in Abbildung 8.3 veranschaulicht ist.

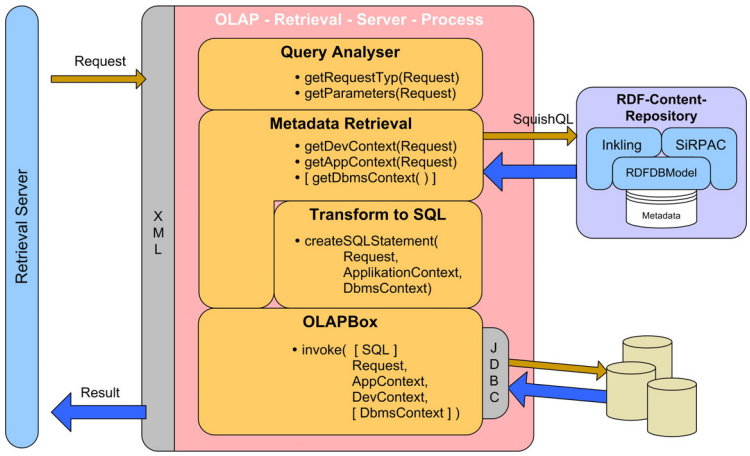

Abbildung 8.3: OLAP-Retrieval-Server-Prozess

Der Anfragebearbeitungsprozess erfolgt in folgenden Schritten:

- **Analyse der Anfrage**

  Im ersten Schritt der Bearbeitung wird die clientseitig Anfrage durch einen XML-Parser (JAXP[1]) syntaktisch analysiert. Dabei werden der Typ und die Parameter aus der Anfrage extrahiert.

- **Metadatenretrieval**

  Entsprechend dem Anfragetyp werden die notwendigen Metainformationen zur Beschreibung des Endgerätes (Device Context), des Würfels (Application Context) und der Datenbank (Database Context) über das RDF-Content-Repository ermittelt.

- **Erzeugung einer SQL-Anfrage**

  Handelt es sich um eine sichtdefinierende Anfrage, wird anhand der Daten aus der Anfrageanalyse und dem Metadatenretrieval ein SQL-Statement generiert (siehe

[1] Java™ API for XML Processing (JAXP) v1.2 EA1. http://java.sun.com/xml/javaxmlpack.html

Abbildung 7.10, Seite 88). Ist von der entsprechend integrierten Datenbank keine Unterstützung des *CUBE*-Operators gegeben, besteht das SQL-Statement aus drei vereinigten *SELECT-FROM-WHERE*-Blöcken.

Bei einer Anfrage zur Ermittlung der ausführlichen Schemabeschreibung eines Würfels erfolgt in diesem Schritt die Erzeugung von SQL-Anfragen an die jeweiligen Dimensionstabellen.

Anfragen zur Ermittlung einer reduzierten Schemabeschreibung, welche beispielsweise von mobilen Endgeräten erfolgen, benötigen in diesem Schritt keine Bearbeitung.

- Datenbankanfrage und Ergebnistransformation

  Im letzten Schritt des Bearbeitungsprozesses baut die OLAPBox, wenn es erforderlich wird, eine Verbindung zur Datenbank auf und ermittelt die Ergebnisse der SQL-Statements. Die gewonnenen Informationen aus den Datenbankanfragen und dem Metadatenbestand bildet die OLAPBox in einem XML-Dokument entsprechend der festgelegten Struktur (siehe Anhang E.2.2) als Anfrageergebnis ab und gibt dieses für den Client an den Retrieval Server weiter.

Die prototypische Implementierung OReSMo demonstriert die Anwendung der entwickelten Konzepte zur Bereitstellung von OLAP-Funktionalität. Durch den modularen Aufbau ist die Komponente leicht in den Retrieval Server zu integrieren.

9. Zusammenfassung

In der vorliegenden Arbeit wurden Konzepte für die adäquate Verwaltung von Metadaten zur Steuerung der Request-Verarbeitung und Informationsaufbereitung für eine XML-Framework-Architektur entwickelt. Dabei wurde insbesondere eine enge Kopplung mit OLAP-Funktionalität angestrebt. Als Realisierungsrahmen wurde die im ZGDV e.V. entwickelte XPEA-Plattform gewählt.

Zur Steuerung der Datenanalyseprozesse in dieser Architektur sind Metadaten von großer Bedeutung. Als Datenformat zur Beschreibung der Metadaten wird in der XPEA-Architektur RDF genutzt. Daraus folgt die Notwendigkeit einer adäquaten, RDF-basierten Metadaten-verwaltung, die Verfahren zur Speicherung und zum Retrieval der genutzten RDF-Daten bietet.

In dieser Arbeit wurden verschieden Verfahren für Speicherung von RDF-Daten diskutiert. Hierbei wurde zunächst festgestellt, dass eine Repräsentation der RDF-Daten in XML für die Speicherung ungünstig ist. Eine Speicherung von RDF-Daten kann sowohl schemagebunden als auch generisch erfolgen, wobei in dieser Arbeit generische Verfahren bevorzugt werden. Für die Verwaltung der Daten bietet sich aufgrund bestehender Leistungsvorteile die Verwendung vorhandener DBMS an. Dabei wurden für relationale, objektrelationale und objektorientierte Datenmodelle entsprechende Konzepte für die Abbildung von RDF-Daten aufgezeigt, wobei der Focus auf dem relationalen Datenbankmodell lag. Als Resultat wurde im weiteren Verlauf der Arbeit eine einfache tripelbasierte Speicherung favorisiert.

Für eine effiziente Anfragebearbeitung ergibt sich die Notwendigkeit einer geeigneten Indizierung. Entsprechende Zugriffsstrukturen wurden untersucht, wobei sich eine mehrdimensionale Speichertechnik beispielsweise auf Grundlage eines UB-Trees anbietet. Aufgrund von Verfügbarkeitsproblemen wurde für diese Arbeit jedoch letztendlich auf eindimensionale Ein-Attribut-Indexe zurückgegriffen.

RDF-Anfragesprachen lassen sich nach Logik- und Graphen-basierte Ansätze klassifizieren. Nach Evaluation existierender Anfragesprachen wurden F-Logic (Logik-basierte) oder RQL (Graphen-basierte) wegen ihrer Ausdrucksstärke favorisiert. Aufgrund unvollständiger beziehungsweise ungeeigneter Implementierungen wurde für diese Diplomarbeit das Graphen-basierte SquishQL verwendet.

Die entwickelten Konzepte erlauben die Umsetzung einer RDF-basierten Metadaten-verwaltung (RDF-Content-Repository). Für die Requestverarbeitung und Anfrage-transformation der gewählten XML-Framework-Architektur ergeben sich durch den Einsatz des RDF-Content-Repositorys kaum Änderungen. Anpassungen sind nur auf der Ebene des Metadatenretrievals notwendig.

Weiterer Schwerpunkt dieser Arbeit ist die Entwicklung eines Ansatzes zur Bereitstellung von OLAP-Funktionalität für die gewählte XML-Framework-Architektur.

Allgemeine Grundlage für OLAP-Analysen bildet ein multidimensionales Datenmodell. Die multidimensionalen Datenstrukturen lassen sich im relationalem Modell abbilden.

Für die Beschreibung eines konkreten OLAP-Würfels sind Metadaten notwendig, um die der Metadatenbestand der Architektur erweitert werden muss. Hierfür wurde ein RDF-basiertes Modell zur Beschreibung eines Würfels entwickelt.

Für die Bereitstellung der OLAP-Funktionalität empfiehlt sich die Online-Analyse, die das Anfragen von multidimensionalen Sichten ermöglicht. Für deren Umsetzung wurden drei Ansätze vorgestellt. Eine Nutzung vordefinierter Sichten bringt Einbußen hinsichtlich der Flexibilität mit sich, wogegen sichtdefinierende und operationale Anfragen eine flexible Analyse des multidimensionalen Datenraums erlauben. Bei den beiden letztgenannten Ansätzen handelt es sich um interaktive Prozesse, woraus sich die Notwendigkeit geeigneter Kommunikation ergibt. Dafür wurden entsprechende Kommunikationsabläufe und -formate definiert.

Aufbauend auf den vorgestellten Ansätzen wurden Mechanismen zur serverseitigen Anfragetransformation entwickelt. Durch den Einsatz der OLAP-Erweiterungen im SQL-99 Standard, kann eine effiziente Anfragebearbeitung gewährleistet werden.

Die entwickelten Konzepte wurden prototypisch als Erweiterung des Retrieval Systems der XPEA-Plattform in der Komponente OReSMo umgesetzt.

Anhang A – Metadatenbeschreibung

A.1 Weitere Standards zur Beschreibung von Metadaten

Meta Content Framework (MCF)

Einer der ersten Standardvorschläge war das *Meta Content Framework* (MCF), welches 1996 von Apple Computer und einer Reihe von Partnern aus der Industrie angekündigt wurde. Später wurde dieser Standard von Netscape, zunächst durch Erweiterung auf XML [BrGu97] und anschließend durch Vorlage eines formalen Vorschlages beim World Wide Web Consortium, weitergeführt.

MCF wird als Strukturbeschreibungssprache bezeichnet. Die Informationen sind eine Menge von gerichteten Graphen (*directed labelled graph*), die aus folgenden Grundelementen bestehen:

- **Knoten** (*node*)
- **Labels** (*property type*) und
- **Kanten** (*arcs*), bestehend aus Quell- und Zielknoten sowie einem Label.

Knoten repräsentieren Dinge wie Webseiten, Bilder, Channels oder auch Objekte der realen Vorstellungswelt (z.B. Personen, Gebäude oder Ereignisse). Kanten beschreiben dagegen Charakteristiken der Objekte oder auch Beziehungen zu anderen Objekten. Jedes Label ist ein Knoten, jedoch nicht umgekehrt. Wird beispielsweise ein Label *Author* zur Beschreibung eines Dokumentes verwendet, so hätte man auch einen gleichnamigen Knoten.

Neben den Grundelementen werden im MCF-Vorschlag eine Menge von Kanten mit vordefinierter Semantik, wie *typeOf*, *domain*, *name* oder *description* spezifiziert.

Abbildung A.1 veranschaulicht die Darstellung einiger Knoten und deren zugeordneten Eigenschaften.

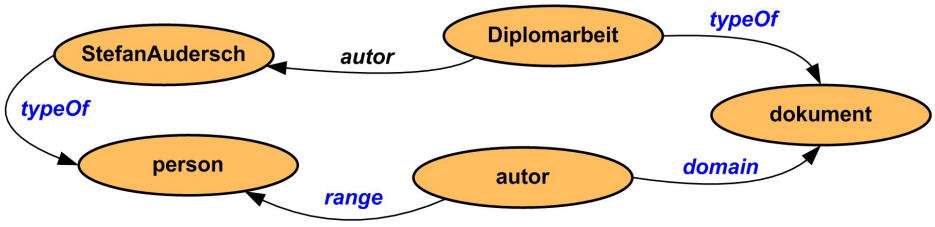

Abbildung A.1: Beispiel für ein MCF-Datenmodell

Das Modell aus Abbildung A.1, erweitert um einige Beschreibungen, sieht in MCF-Syntax folgendermaßen aus:

```
<xml-mcf>
<mcf-ref XML-LINK="SIMPLE" ROLE="XML-MCF-BLOCK"
      href="http://www.woauchimmer.org/notwendiges.mcf"/>
  <person id="StefanAudersch">
    <name>Stefan Audersch</name>
    <description>Student der Universität Rostock</description>
    <email>audersch@informatik.uni-rostock.de</email>
  </person>
  <dokument id="Diplomarbeit">
    <name>Metadatenverwaltung für ...</name>
    <autor unit="StefanAudersch">
    <erstellungsDatum>15.01.2002</erstellungsDatum>
  </dokument>
</xml-mcf>
```

Listing A.1: Personen- und Dokumentenbeschreibung in MCF-Syntax [1]

Der strukturbeschreibende Charakter erlaubt es MCF, seine eigene Schemadefinitionssprache zu sein, wodurch eine dynamische Erweiterung eines Metadatenmodells durch Autoren oder Applikationen ermöglicht wird.

Eine umfassende Einführung in MCF wird in [GuBr97] gegeben.

XML-Data

Neben Netscape hat auch Microsoft mit der *Specification for XML-Data* [LJM+97] einen Entwurf für einen Metadatenbeschreibungsstandard beim W3C eingereicht. Im Gegensatz zum MCF arbeitet XML-Data nicht mit gerichteten Graphen, sondern mit Bäumen.

XML-Data basiert auf *Web Collections in XML* [HBH+97] und arbeitet mit dem *Channel Definition Format* (CDF) [Elle97] zusammen. *Web Collections in XML* und CDF stammen ebenfalls von Microsoft. Im Unterschied zu CDF werden Beschreibungselemente in XML-Data nicht mit Hilfe einer *Document Type Definition* (DTD) deklariert, sondern durch ein spezielles Schema in XML. Elemente sind vererbbar und können erweitert werden. Neben der Vererbung bei Elementen ist es auch möglich, ein ganzes Schema zu vererben.

Listing A.2 zeigt ein Metadatenschema für Bücher und die Beschreibung eines Buches in XML-Data.

```
<?XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882/" as="s"/?>
<s:schema>
  <elementType id="author">
    <string/>
  </elementType>

  <elementType id="title">
    <string/>
  </elementType>

  <elementType id="Book">
    <element type="#title" occurs="OPTIONAL"/>
    <element type="#author" occurs="ONEORMORE"/>
  </elementType>
</s:schema>
...
<Book>
  <author>Peter Glulutzan</author>
  <author>Trudy Pelzer</author>
  <title>SQL-99 Complete, Really</title>
</Book>
```

Listing A.2: Beschreibung eines Buches mit XML-DATA

[1] Die Knoten *name* und *description* gehören zu den vorgegebenen Schlüsselwörtern, während *email*, *autor*, und *erstellungsDatum* in *notwendiges.mcf* definiert sind.

Platform for Internet Content Selection (PICS)

Ein erster Ansatz des W3C, um Metadaten über Webseiten zur Verfügung zu stellen, war 1996 die *Platform for Internet Content Selection* [PICS96]. PICS bietet die Möglichkeit, Aussagen über den Inhalt von Webseiten zu treffen, ist jedoch primär nur für deren Beurteilung nach verschiedenen Kriterien geeignet. Ursprüngliches Ziel war insbesondere die Kennzeichnung von jugendfreien Internet-Inhalten beziehungsweise die Herausfilterung unerwünschter Inhalte.

PICS geht von einer freiwilligen Kategorisierung der Seiten durch deren Anbieter, ähnlich wie bei Videofilmen oder Computerspielen, aus. Alternativ kann auch ein unabhängiger PICS-Service, der PICS-Zertifikate für Inhalte ausstellt, in Anspruch genommen werden. Eltern und Lehrer können festlegen, auf welche Inhalte Kinder zugreifen dürfen und der Browser stellt dann nur Webseiten dar, die ein entsprechendes PICS-Label aufweisen.

PICS-Labels können im HTML-Dokument über den RFC-822-Header oder vollkommen getrennt vom Dokument transportiert werden. Ein Beispiel für die Einbettung in ein HTML-Dokument zeigt Listing A.3.

```
<head>
  <META http-equiv="PICS-Label" content='
  (PICS-1.1 "http://www.gcf.org/v2.5"
  labels on "1994.11.05T08:15-0500"
  until "1995.12.31T23:59-0000"
  for "http://w3.org/PICS/Overview.html"
  ratings (suds 0.5 density 0 color/hue 1)) '>
</head>
```

Listing A.3: PICS-Label in einem HTML-Dokument [PICS96]

Die PICS-Auszeichnung erfolgt mit einem META-Tag innerhalb des HTML-Headers. Zunächst müssen Pflichtangaben, wie die PICS-Version, spezifiziert werden. Die Angaben *labels on*, *until* und *for* sind optional. Entscheidend ist das Attribut *ratings*, in der die eigentliche Inhaltskategorisierung notiert wird.

PICS fehlt jedoch für ein universelles Metadaten-System eine gewisse Flexibilität. So wurde im Rahmen der Metadaten-Aktivitäten des W3C daran gearbeitet, auch die PICS-Spezifikation vollständig im Resource Description Framework abzubilden.

A.2 RDF Parser

RDF Parser ermöglicht die syntaktische Analyse von RDF-Daten in XML-Syntax. Für die verschiedensten Entwicklungsumgebungen existieren bereits RDF Parser. In der Tabelle A.1 werden einigen Parser und deren Bezugsquelle aufgeführt.

Java	
• SiRPAC *(Bestandteil der RDF-API)*	http://www-db.stanford.edu/~melnik/rdf/api.html
• Profium RDF Parser *(ehemals Promenade)*	http://www.profium.com/gb/products/developers
• Another RDF Parser (ARP) *(Bestandteil der Jena-API)*	http://www-uk.hpl.hp.com/people/jjc/arp/
• Validating RDF Parser (VRP) *(Bestandteil der RDFSuite)*	http://139.91.183.30:9090/RDF/VRP/index.html
C	
• Libwww	http://www.w3.org/Library/src/HTRDF
• Repat	http://injektilo.org/rdf/repat.html
Perl	
• Perlib	http://www.w3.org/1999/02/26-modules/
Prolog	
• SWI-Prolog RDF Parser	http://www.swi.psy.uva.nl/projects/SWI-Prolog/packages/sgml/online.html

Tabelle A.1: RDF-Parser und deren Quellen

Ein weit verbreiteter RDF Parser ist SiRPAC, der auch im Rahmen dieser Diplomarbeit zum Einsatz kommt.

A.3 RDF Anwendungen

RDF stellt ein universelles Metadatenformat bereit, welches Voraussetzungen bietet, sich in vielen Anwendungen zu etablieren. Die Stärken von RDF liegen bei der Metadaten-beschreibung von Daten mit heterogenen Strukturen, die an verschiedenen Orten vorliegen und eine relativ häufige Anzahl von Bezügen zueinander haben. Dieses macht RDF speziell für Web-Suchmaschinen interessant. Aber auch für andere Anwendungsfelder ist RDF von hohem Interesse.

Folgend werden drei verschiedene RDF-basierte Anwendungen vorgestellt und anhand eines Beispiels ein Ausblick darauf gegeben, wofür RDF noch eingesetzt werden könnte.

Dublin Core

Mit dem ursprünglichen Ziel, Dokumente im World Wide Web zu beschreiben und über Suchmaschinen gezielt mit Hilfe bestimmter Kategorien durchsuchbar zu machen, begann ab 1995 in einer Reihen von Workshops die Entwicklung des *Dublin Core*[1]. Mittlerweile hat sich das Anwendungsumfeld erweitert, so dass dieser Beachtung von verschiedensten Gruppen findet, die sich mit Metadatenbeschreibungen befassen.

[1] Dublin Core Metadata Initiative. http://dublincore.org/

Dublin Core ist verhältnismäßig einfach angelegt, damit die Metadatenelemente von einer Vielzahl von Erzeugern akzeptiert und angewandt werden. Die Metadatenelemente umfassen 15 Grundelemente, das so genannte *Dublin Core Metadata Element Set*. Sie können in Form eines RDF-Schemas als kleinster gemeinsamer Nenner für verschiedene RDF-basierte Anwendungen dienen [KoSc01].

Open Directory Project

Das Ziel des *Open Directory Project*[1] ist der Aufbau eines möglichst umfassenden Weg-Verzeichnisses. Die Pflege und Organisation des Katalogs erfolgt dezentral, vom Internet selbst, um so dem Wachstum des WWW gerecht zu werden. Jeder kann einen kleinen Teil des Gesamtkataloges bearbeiten. Die gesamten Verzeichnisinformationen stehen frei zur Verfügung und können weiterverwendet werden. Suchmaschinen, wie zum Beispiel Netscape Search, AOL Search, Google, Lycos oder HotBot, nehmen diesen Service zur Erweiterung ihres eigenen Angebots in Anspruch. Als Datenaustauschformat dient hierbei RDF in XML-Syntax.

Reggie Metadata Editor

Der *Reggie Metadata Editor*[2] ist ein Java Applet, welcher die einfache Erzeugung von Metadaten ermöglicht. Entwickelt wurde der Editor am *Distributed Systems Technology Centre* (DSTC). Anhand eines gewählten Schemas, hier ist auch die Definition eigener Schemata möglich, bietet der Reggie Metadata Editor eine graphische Benutzerschnittstelle zur Eingabe der Beschreibungsinformationen. Die eingegebenen Daten können letztendlich im HTML 3.2 und 4.0 Format als auch in XML-Syntax (Serialization- oder Abbreviated Syntax) von RDF ausgegeben werden.

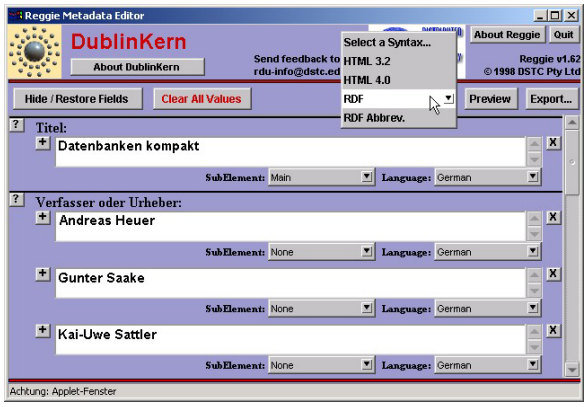

Abbildung A.2: Reggie Metadata Editor

[1] Open Directory Project. http://domz.org

[2] Distributed Systems Technology Centre (DSTC): Reggie Metadata Editor. http://metadata.net/dstc/

MPEG 7

MPEG steht für Motion Picture Experts Group, die 1989 gegründet wurde. Mit MPEG-7 [Marti01], dem *Multimedia Content Description Interface,* soll die Entwicklung eines Standards für die Beschreibung verschiedener Typen multimedialer Informationen erzielt werden. Hierzu wird eine Menge von Deskriptoren bereitgestellt, die mit Hilfe der MPEG-7-DDL (Description Definition Language) spezifiziert werden. Zur syntaktischen Repräsentation von MPEG-7-Beschreibungen wird XML eingesetzt.

Neben XML bietet auch das Resource Description Framework genügend grundlegende Konzepte zur Darstellung von MPEG-7-Informationen. Wie eine Abbildung von MPEG-7 auf RDF aussehen kann, ist unter anderem Thema einer derzeit laufenden Diplomarbeit von Michael Drews an der Universität Rostock [Drew02].

Ein umfassender Überblick zu MPEG-7 findet sich in [Marti01] oder [Rust01].

A.4 Beispiel für ein RDF-Schema

Das folgende Beispiel demonstriert den grundlegenden Aufbau eines RDF-Schemas.

```
<rdf:RDF xml:lang="en"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdfs:Class rdf:ID="Person">
  <rdfs:comment>Klasse der Personen.</rdfs:comment>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Student">
  <rdfs:comment>Klasse der Studenten.</rdfs:comment>
  <rdfs:subClassOf  rdf:resource="#Person"/>
</rdfs:Class>

<rdf:Property ID="name">
  <rdfs:comment>Name einer Person.</rdfs:comment>
  <rdfs:range
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property ID="matrikelNr">
  <rdfs:comment>Matrikelnummer eines Studenten.</rdfs:comment>
  <rdfs:range
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="#Student"/>
</rdf:Property>

<rdf:Property ID="bestFriend">
  <rdfs:comment>Beste(r) Freund(in) einer Personen.</rdfs:comment>
  <rdfs:range rdf:resource="#Person"/>
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
</rdf:RDF>
```

Listing A.4: RDF-Schema zur Beschreibung von Personen und Studenten

Das Schema stellt die Ressourcenklassen *Person* und *Student* (diese wurde von *Person* abgeleitet) zur Verfügung. Personen, somit auch Student, haben einen Namen und einen besten Freund, der wiederum eine Person ist. Studenten besitzen zusätzlich eine Matrikelnummer.

Anhang B – OLAP

B.1 Codd'sche Regeln und FASMI-Definition

Codd'sche Regeln

Die Codd'schen Regeln, definiert von E.F. Codd 1993 in [CoCS93], umfassen die folgenden zwölf Punkte:

1. **Multidimensionale konzeptionelle Sichtweise:** Betrachtung von Kenngrößen aus Sicht verschiedener Dimensionen
2. **Transparenz:** transparenter Zugriff auf Daten unterschiedlicher Quellen
3. **Zugriffsmöglichkeit:** Zugriff auf unternehmensinterne und -externe Datenquellen
4. **Gleichbleibende Antwortzeit bei der Berichterstellung:** Antwortzeit unabhängig von der Anzahl der Dimensionen und Menge der gespeicherten Daten
5. **Client/Server-Architektur:** Trennung von Speicherung, Verarbeitung und Darstellung
6. **Generische Dimensionalität:** einheitliche Behandlung von Dimensionen
7. **Dynamische Behandlung dünn besetzter Matritzen:** automatische Anpassung des physischen Schemas an die Dimensionalität und Datenverteilung
8. **Mehrbenutzer-Unterstützung:** gleichzeitiges Arbeiten mehrerer Anwender
9. **Uneingeschränkte kreuzdimensionale Operationen:** automatische Ableitung der Berechnungen, die sich aus den Hierarchiebeziehungen der Dimensionen ergeben
10. **Intuitive Datenbearbeitung:** ergonomische und intuitive Benutzeroberfläche, flexible Navigation durch die Daten
11. **Flexible Berichterstattung:** Berichterstellung mit beliebiger Datenanordnung
12. **Unbegrenzte Anzahl von Dimensionen und Klassifikationsebenen:** keine Einschränkungen hinsichtlich der Anzahl unterstützter Dimensionen und deren Aggregationsebenen

Erweiterte Codd'sche Regeln

Den ursprünglichen Kriterienkatalog erweiterte E.F. Codd 1995 um sechs weitere Regeln (aus [Clau98]):

1. **Datenintegration:** zusätzlich zur multidimensionalen Struktur transparenter Zugriff auf darunter liegende Datenquellen

2. **Unterstützung verschiedener Analysemodelle:** Unterstützung des *categorical* (beschreibend), *exegetical* (erklärend), *contemplative* (bedenkend) und *formularical* (formelbasiert) Analysemodells
3. **Trennung zwischen analyseorientierten und operativen Daten:** keine Auswirkung bei Änderung der Analysedaten auf Quellsysteme
4. **Trennung der Speicherorte:** Ausführung von Schreiboperationen auf OLAP-Daten, nicht auf produktiven Datenbestand propagieren
5. **Unterscheidung zwischen Null- und Fehlwerten:** Unterscheidung zwischen fehlenden Werten und Werten mit der Bedeutung NULL
6. **Behandlung von fehlenden Werten:** effiziente Verwaltung leerer Felder (Speicherplatzoptimierung)

FASMI

Mit dem Ziel eine einfache und herstellerunabhängige Möglichkeit zur Bewertung eines Produktes und dessen Einordnung in die Kategorie OLAP zu entwickeln, wurde im OLAP-Report von 1995 die FASMI-Definition [PeCr95] vorgestellt. FASMI steht hierbei für die fünf Schlüsselwörter ***F**ast **A**nalysis of **S**hared **M**ultidimensional **I**nformation*, die sich wie folgt beschreiben lassen:

1. **Geschwindigkeit:** Antwortzeit durchschnittlich 5, maximal 20 Sekunden
2. **Analysemöglichkeit:** anwenderfreundliche und intuitive Analyse
3. **Sicherheit:** Mehrbenutzerbetrieb mit Zugriffsrechten und Sperrverfahren
4. **Multidimensionalität:** multidimensionale konzeptuelle Sicht unabhängig vom zugrunde liegenden Datenbankmodell
5. **Kapazität:** keine Begrenzung der Dimensionalität und des Datenvolumens

B.2 Star-Schema für multidimensionale Anfrage

product

productID	productName	productGroup	productCategory
1	VCR 19-3	Video	Technik
2	ColorTV	Video	Technik
3	Deckenleute Sonja	Beleuchtung	Einrichtung
4	FM 98 XM	Audio	Technik

time

dateID	day	week	month	year
1	11	45	11	2000
2	24	51	12	2000
3	10	2	1	2001

store

storeID	storeName	city	country	state
1	HIFImarkt	Rostock	MV	Deutschland
2	Trödelmarkt	Rostock	MV	Deutschland
3	Technik Meier	Dresden	Sachsen	Deutschland
4	Shopping Center	Seatle	Washington	USA

sale

storeID	dateID	productID	count
1	1	1	21
1	1	4	13
1	3	1	9
2	1	2	2
2	1	3	4
2	2	4	3
2	3	3	1
3	2	1	15
3	2	4	13
3	3	1	27
4	1	1	11
4	1	2	34
4	3	3	22
4	3	2	12

Abbildung B.1: Star-Schema für multidimensionale Anfrage

B.2 OLAP-Erweiterung in SQL99

(basierend auf [GuPe99])

GROUP BY-Klausel in SQL99

```
SELECT columnExpressions
  FROM tableReferences
  [WHERE searchConditions
  [GROUP BY <groupingSpecification> [HAVING conditions]]

<groupingSpecification>::=
    <groupingColumnReferenceList> |
    ROLLUP (<groupingColumnReferenceList>) |
    CUBE (<groupingColumnReferenceList>) |
    GROUPING SETS (<groupingSetList>) |
    ( ) |
    <groupingSet>,  <groupingSetList>

<groupingColumnReferenceList>::=
    {columnReference [ collateclause ] } [,…]

<groupingSetList>::=
    <groupingSet> [ {, <groupingSet>} … ]

<groupingSet>::=
    <groupingColumnReference> |
    <groupingColumnReferenceList> |
    ROLLUP (<groupingColumnReferenceList>) |
    CUBE (<groupingColumnReferenceList>) |
    ( )
```

Aggregatfunktionen in SQL 99

```
<setFunctionSpecification>::=
    <generalSetFunction> |
    COUNT(*) |
    <groupingOperation>

<generalSetFunction>::=
    <setFunctionType> ([ {DISTINCT | ALL} ] columnExpression)

<setFunctionType>::=
    COUNT | MAX | MIN | SUM | AVG | EVERY | ANY | SOME

<groupingOperation>::=
    GROUPING (columnReference)
```

Anhang C – Mehrdimensionale Indizierung

UB-Tree

Der *Universal B-Tree* (UB-Tree) wurde 1996 von Rudolf Bayer vorgeschlagen [Baye97]. Die Idee beim UB-Tree ist, den mehrdimensionalen Raum so auf einen eindimensionalen Schlüssel abzubilden, dass benachbarte Punkte im Raum auch nah beieinander liegende Schlüsselwerte ergeben. Der eindimensionale Schlüssel berechnet sich durch bitweise Verschränkung (Bit-Interleaving) der zu indizierenden Attribute, analog dem Verfahren beim mehrdimensionalen Hashen. Hierbei ist es notwendig, dass die lexikographische Ordnung für die Bitrepräsentation der Attribute sich an der natürlichen Ordnung des jeweiligen Datentyps orientiert. Die berechneten Schlüssel, auch Z-Werte genannt, bestimmen eindeutig einen Punkt im Raum. Werden die Z-Werte in aufsteigender Reihenfolge miteinander verbunden, bilden sie die so genannte Z-Kurve.

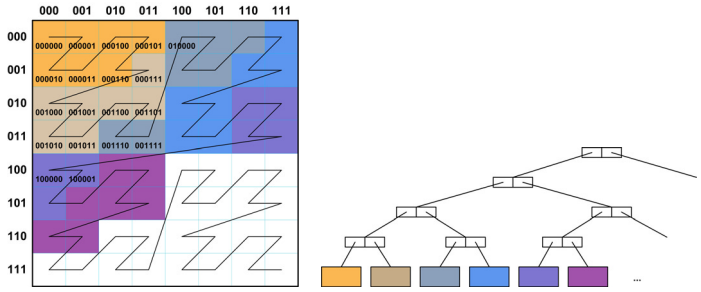

Abbildung C.1: Z-Kurve, Z-Regionen und UB-Tree

Im Wesentlichen erhält die Z-Kurve, bis auf vereinzelte Ausnahmen (zum Beispiel bei den Z-Werten *001010* und *10000*), die räumliche Nähe der Punkte. Benachbarte Punkte im mehrdimensionalen Raum weisen auch nah beieinander liegende Z-Werte auf. Die Datensätze werden über einen gewöhnlichen B*-Baum mit den Z-Werten als Schlüssel verwaltet, dem UB-Tree. Die Nachbarschaftsbeziehungen werden so auch weitestgehend auf dem Sekundärspeicher erhalten. Jedes Blatt des UB-Trees enthält ein bestimmtes Intervall von Z-Werten, die bei Abbildung auf den mehrdimensionalen Raum Z-Regionen bilden, die nicht unbedingt räumlich zusammenhängen.

Eine mehrdimensionale Bereichsanfrage stellt im Raum einen Hyperquader (Query Box) dar. Entscheidend für die Anfrageperformance ist ein Algorithmus, der möglichst nur diejenigen Blöcke vom Sekundärspeicher lädt, die Datensätze aus der Query Box enthalten können. Das heißt, es gilt hierbei nur die Z-Regionen zu ermitteln, welche die Query Box schneiden. Die Query Box ist durch einen Startpunkt *qa* (obere linke Ecke im 2-dimensionalem Raum) und einen Endpunkt *qe* (untere rechte Ecke im 2-dimensionalem Raum) spezifiziert.

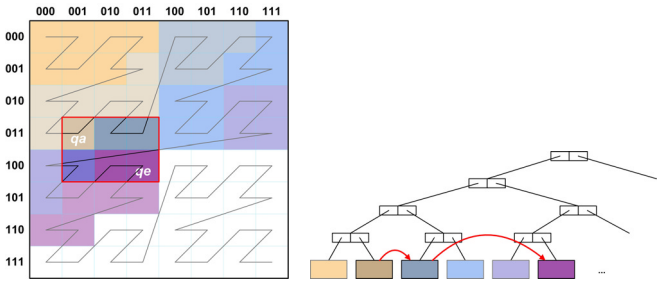

Abbildung C.2: Query Box und RQ-Algorithmus beim UB-Tree

Der vorgeschlagene *Algorithm for range queries* (RQ-Algorithmus) [Baye97] bestimmt über den UB-Tree den Datenblock der den Schlüsselwert *qa* enthält und sucht darin nach Datensätzen, die den Anfragebedingungen entsprechen. Zur Bestimmung weiterer Datensätze geht der Algorithmus nicht nur rein sequentiell den UB-Tree bis zum Endpunkt *qe* durch, sondern vollzieht Sprünge, wenn die Z-Kurve die Query Box verlässt. So werden nur Z-Regionen geladen, die auch potentielle Datensätze enthalten.

Durch die Kombination von UB-Tree-Clustering und RQ-Algorithmus führen Bereichsanfragen zu einer nahezu idealen Ausführung. Da der UB-Tree auf den B*-Baum aufsetzt, ergeben sich für Änderungsoperationen äquivalente Aufwände gegenüber dem B*-Baum.

Anhang D – RDF-Anfragesprachen

D.1 Syntax von RQL in BNF

(basierend auf [FICS01] [1])

```
<query>::=
     (<query>) |
     {SUBCLASSOF | SUBPROPERTYOF} [^] (<query>) |
     {DOMAIN | RANGE} (<query>) |
     <query> {<setOp> | <boolOp> | <compOp> | IN} <query> |
     NOT <query> |
     {COUNT | AVG | MIN | MAX} ( <query> ) |
     <query> "[" <query> [ : <query> ] "]" |
     ^<query> |
     <constant> |
     <var> |
     identifier |
     CLASS |
     PROPERTY |
     {BAG | SEQ} ( <query>, [ <query> ] ) |
     <selectFromWhereQuery> |
     {EXISTS | FORALL} <var> <query> : <query>

<selectFromWhereQuery>::=
     SELECT <projList> FROM <rangeList> [ WHERE <query> ]

<compOp>::= < | <= | > | >= | = | != | LIKE

<setOp>::= UNION | INTERSECT | MINUS

<boolOP>::= AND | OR

<constant>::=
     integerLiteral |
     realLiteral |
     quotedStringLiteral |
     quotedCharLiteral |
     date |
     TRUE
     FALSE
     & identifier

<var>::= <dataVar> | <classVar> | <typeVar> | <propertyVar>

<dataVar>::= identifier

<classVar>::= $ identifier

<typeVar>::= $$ identifier

<propertyVar>::= @ identifier

<projList>::= * | <query> {,<query>}...

<rangeList>::= <pathExpression> {,<pathExpression>}...

<pathExpression>::= <pathElement> {.<pathElement>}...

<pathElement>::= ["{"<fromTo>"}"] <query> ["{"<fromTo>"}"]

<fromTo>::=
     [<dataVar>] [:{<classVar> | <typeVar> | identifier}] |
     <classVar> |
     <typeVar>
```

[1] siehe http://139.91.183.30:9090/RDF/RQL/bnf.html

D.2 Syntax von SquishQL in EBNF

(basierend auf [Mill01][1])

```
<query>::=
    SELECT { <variable>+ | * }
    [ FROM <url> [, <url>]*]
    WHERE { (<predicate> <subject> <object>) }+
    [ AND <constraint> ]
    [ USING {prefix FOR predicateUrl}+ ]

<predicate>::= { [prefix::]value | <variable> }

<subject>::= { [prefix::]value | <variable> }

<object>::= { [prefix::]value | <variable> }

<variable>::= ? name

<constraint>::= <variable> { = | < | > | <= | >= | ~ } {value | <variable> }
```

D.3 Syntax von rdfDB Query in EBNF

(basierend auf [Guha00])

```
<statement>::= <query> | <createDatabase> | <insert>  | <load>

<query>::=
    SELECT { <variable>+ | * }
    FROM database
    WHERE { (<predicate> <subject> <object>) }+

<predicate>::= { [prefix::]value | <variable> }

<subject>::= { [prefix::]value | <variable> }

<object>::= { [prefix::]value | <variable> }

<variable>::= ? name

<createDatabase>::= CREATE DATABASE databaseName

<insert>::= INSERT INTO databaseName (predicateValue, subjectValue, objectValue)

<load>::= LOAD RDF_XML FILE fileUrl databaseName
```

[1] siehe http://swordfish.rdfweb.org/rdfquery/ql.html

Anhang E – OLAP-Erweiterung

E.1 Metadatenmodel zur Beschreibung eines OLAP-Würfels

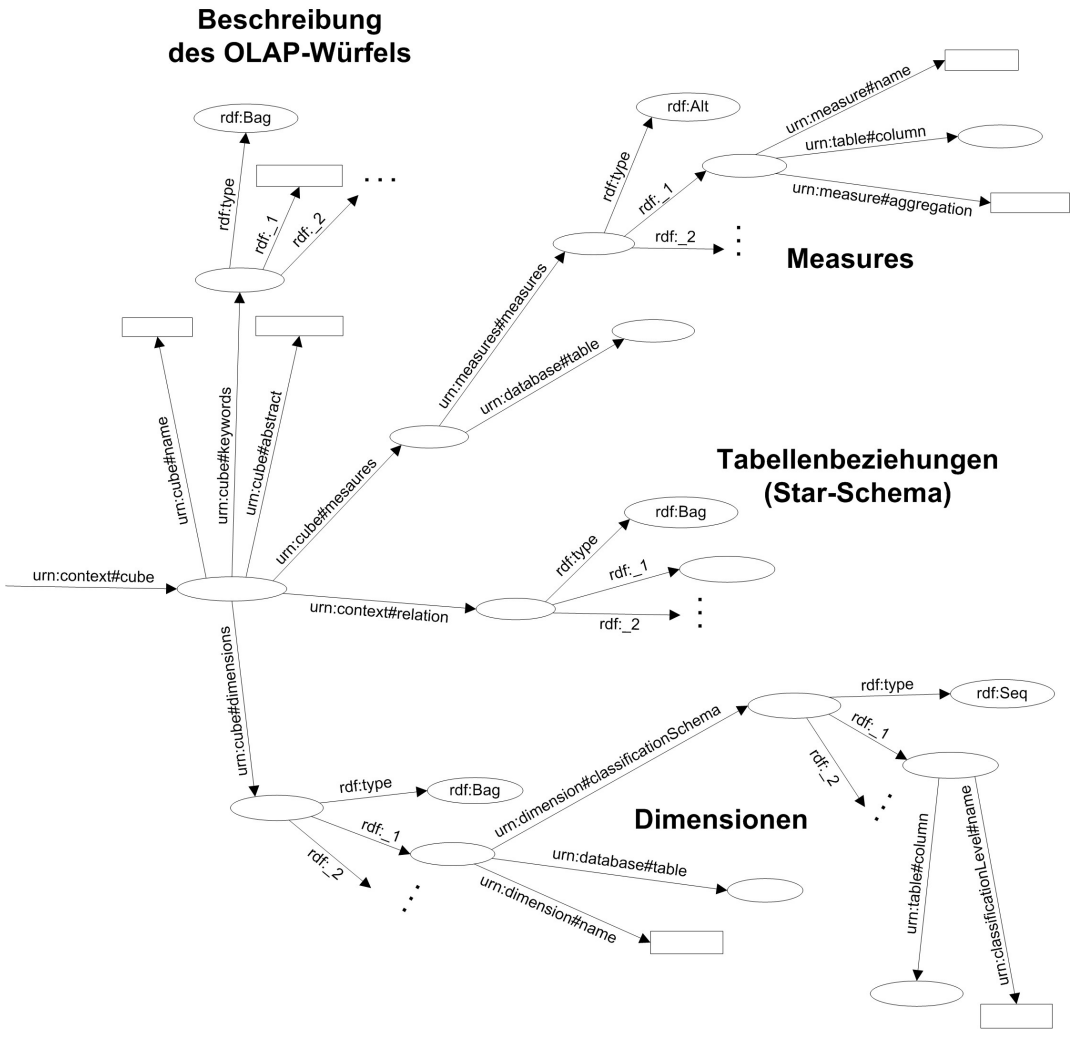

Abbildung E.1: RDF-Modell zur Beschreibung eines OLAP-Würfels [1]

[1] Dem Retrieval System liegt ein umfangreiches RDF-Modell zu Grunde. Zur besseren Übersicht zeigt die Abbildung E.1 nur einen Ausschnitt aus dem gesamten RDF-Modell.

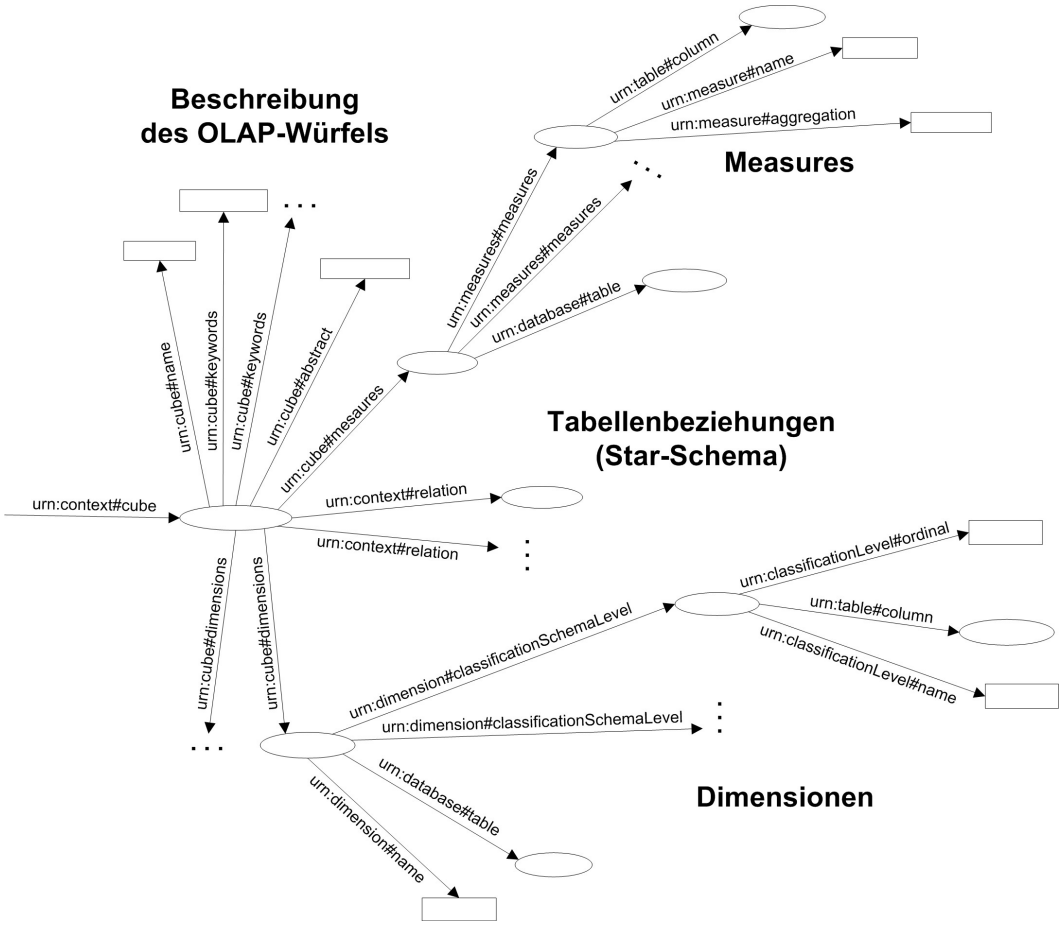

Abbildung E.2: vereinfachtes RDF-Modell zur Beschreibung eines OLAP-Würfels [1]

[1] Dem Retrieval System liegt ein umfangreiches RDF-Modell zu Grunde. Zur besseren Übersicht zeigt die Abbildung E.2 nur einen Ausschnitt aus dem gesamten RDF-Modell.

E.2 DTDs zur Kommunikation bei der XPEA-Architektur

Die nachfolgenden DTDs bilden die Grundlage zu Kommunikation zischen Server und Client für eine Kopplung von OLAP-Funktionalität mit XPEA-Architketur.

E.2.1 Anfrage vom Client

Für eine bessere Übersicht ist in Abbildung E.3 die allgemeine Struktur des XML-Dokumentes, die sich anhand der clientToServer.dtd ergibt, graphisch dargestellt.

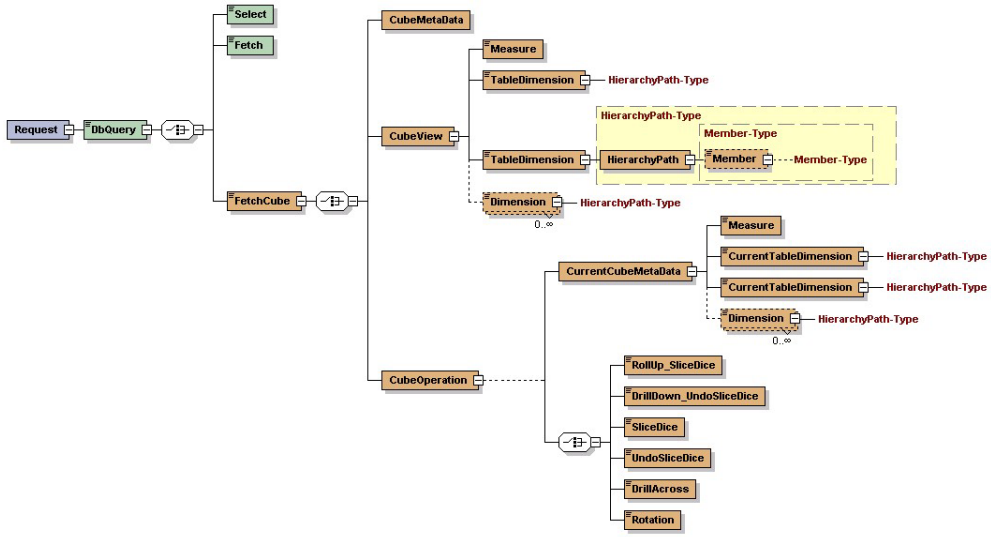

Abbildung E.3: Struktur des Anfragedokumentes vom Client

ClientToServer.dtd

```
<?xml version="1.0" encoding="UTF-8"?>

<!--DTD zur Beschreibung der ClientAnfragen-->
<!ELEMENT Request (DbQuery)>
<!ATTLIST Request
    device (pc | palm | handy) #REQUIRED>

<!--Kennzeichnet eine Anfrage an den Retrieval Server-->
<!ELEMENT DbQuery (Fetch | Select | FetchCube)>
<!ATTLIST DbQuery
    user CDATA #REQUIRED
    pass CDATA #REQUIRED>

<!--Anfrage-Typ: Suchen von Dokumenten und Information-->
<!ELEMENT Select (#PCDATA)>
<!ATTLIST Select
    Rubric CDATA #IMPLIED
    Language (german | english) #IMPLIED
    Results (all | 10 | 20 | 30 | 40 | 60 | 100) #IMPLIED>

<!--Anfrage-Typ: Dokument mit entsprechender ID anfragen-->
<!ELEMENT Fetch EMPTY>
<!ATTLIST Fetch
    id CDATA #REQUIRED>

<!--Anfrage-Typ: Daten eines OLAP-Cube mit entsprechender ID erfragen-->
<!ELEMENT FetchCube (CubeMetaData | CubeOperation | CubeView)>
<!ATTLIST FetchCube
```

```
    cubeId CDATA #REQUIRED>

<!--Cube-Anfrage-Typ: Anfrage der Metadaten eines OLAP-Cube -->
<!ELEMENT CubeMetaData EMPTY>

<!--Cube-Anfrage-Typ: Sichtdefinierende Anfrage  (Hinweis: Dieser Anfragetyp ist für Endgeräte mit
ausreichender Performance (z.B. PC,  NoteBook,  SubNoteBook) vorgesehen.)-->
<!ELEMENT CubeView (Measure, TableDimension, TableDimension, Dimension*)>

<!ELEMENT TableDimension (HierarchyPath)>
<!ATTLIST TableDimension
    name ID #REQUIRED
    alignment (horizontal | vertical) #REQUIRED
    level CDATA #REQUIRED>

<!ELEMENT Dimension (HierarchyPath)>
<!ATTLIST Dimension
    name ID #REQUIRED>

<!ELEMENT HierarchyPath (Member?)>
<!ELEMENT Member (Member?)>
<!ATTLIST Member
    level CDATA #REQUIRED
    value CDATA #REQUIRED>

<!--Cube-Anfrage-Typ: Operationale Anfrage  (Hinweis: Dieser Anfragetyp ist für Endgeräte mit
geringer Performance (z.B. PDA) vorgesehen.)-->
<!ELEMENT CubeOperation (CurrentCubeMetaData, (RollUp_SliceDice | DrillDown_UndoSliceDice |
SliceDice | UndoSliceDice | DrillAcross | Rotation))?>

<!ELEMENT CurrentCubeMetaData (Measure, CurrentTableDimension, CurrentTableDimension, Dimension*)>
<!ELEMENT Measure EMPTY>
<!ATTLIST Measure
    name CDATA #REQUIRED>

<!ELEMENT CurrentTableDimension (HierarchyPath)>
<!ATTLIST CurrentTableDimension
    name CDATA #REQUIRED
    dimensionID ID #REQUIRED
    alignment (horizontal | vertical) #REQUIRED>

<!-- Operationale Anfrage: Rollup + Slice&Dice-->
<!ELEMENT RollUp_SliceDice EMPTY>
<!ATTLIST RollUp_SliceDice
    TableDimension IDREF #REQUIRED
    upMemberValue CDATA #REQUIRED>

<!--Operationale Anfrage: Ein Drill-down + UndoSlice&Dice-->
<!ELEMENT DrillDown_UndoSliceDice EMPTY>
<!ATTLIST DrillDown_UndoSliceDice
    TableDimension IDREF #REQUIRED>

<!--Operationale Anfrage: Slice & Dice (Hinweis: kann nur auf einer nicht sichtbaren Dimension
ausgeführt werden.)-->
<!ELEMENT SliceDice EMPTY>
<!ATTLIST SliceDice
    Dimension IDREF #REQUIRED
    upMemberValue CDATA #REQUIRED>

<!--Operationale Anfrage: UndoSlice&Dice-->
<!ELEMENT UndoSliceDice EMPTY>
<!ATTLIST UndoSliceDice
    Dimension IDREF #REQUIRED>

<!--Cube-Operation: Drill-across-->
<!ELEMENT DrillAcross EMPTY>
<!ATTLIST DrillAcross
    newMeasure CDATA #REQUIRED>

<!--Cube-Operation: Rotation-->
<!ELEMENT Rotation EMPTY>
<!ATTLIST Rotation
    newDimension IDREF #REQUIRED
    alignment (horizontal | vertical) #REQUIRED>
```

E.2.2 Ergebnis vom Retrieval Server

Für eine bessere Übersicht ist in Abbildung E.4 die allgemeine Struktur des XML-Dokumentes, die sich anhand der ServerToClient.dtd ergibt, graphisch dargestellt.

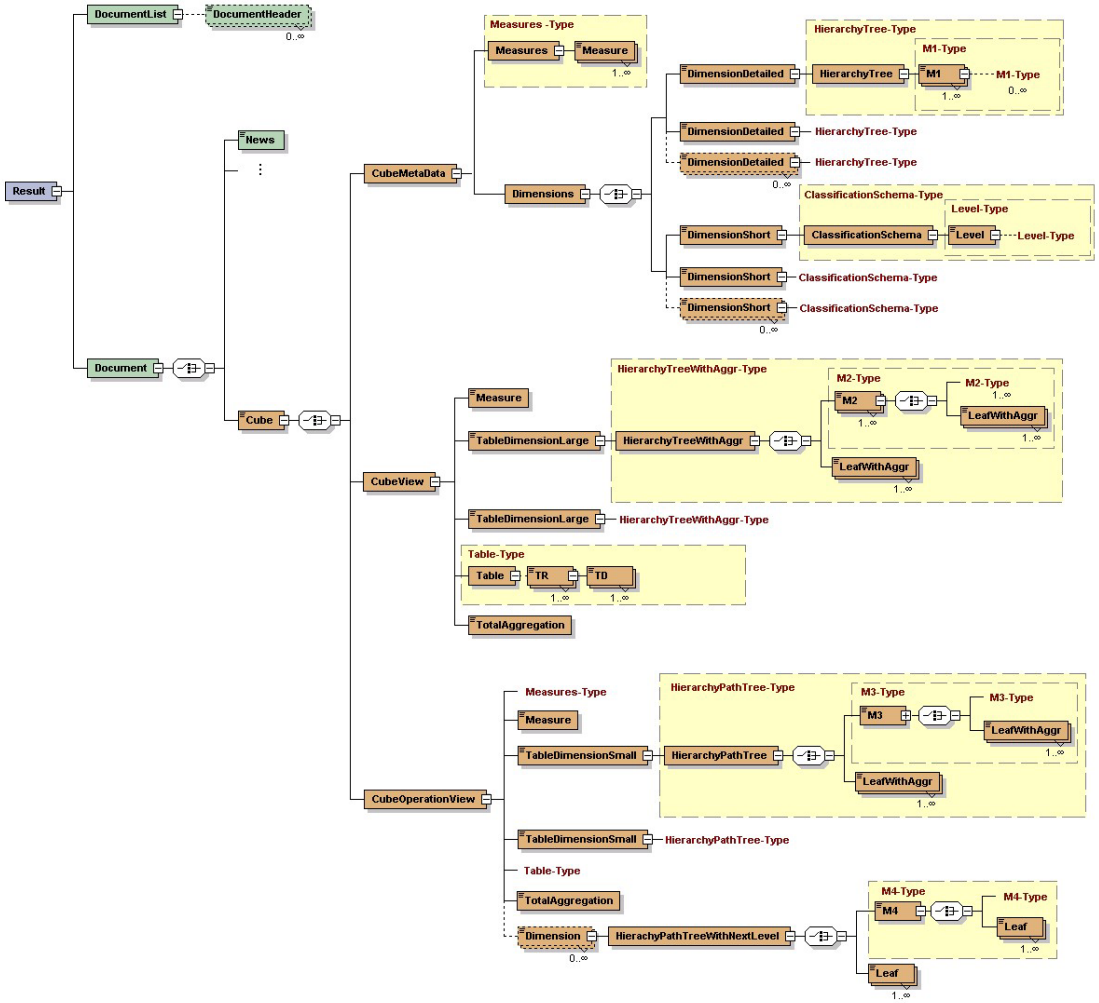

Abbildung E.4: Struktur des Ergebnisdokumentes vom Server

ServerToClient.dtd

```
<?xml version="1.0" encoding="UTF-8"?>

<!--DTD zur Beschreibung des Ergebniss vom Server-->
<!ELEMENT Result (DocumentList | Document)>

<!ELEMENT DocumentList (DocumentHeader*)>

<!ELEMENT DocumentHeader EMPTY>

<!ATTLIST DocumentHeader
    id CDATA #REQUIRED
    mimeType (HTML | text-Plain | pdf | image | cube) #REQUIRED
    titel CDATA #REQUIRED>

<!ELEMENT Document (News | Cube)>

<!ELEMENT News (#PCDATA)>
```

```
<!ELEMENT Cube (CubeMetaData | CubeView | CubeOperationView)>
<!ATTLIST Cube
    cubeId CDATA #REQUIRED>

<!--Metadaten eines Würfels-->
<!ELEMENT CubeMetaData (Measures, Dimensions)>
<!ATTLIST CubeMetaData
    abstract CDATA #REQUIRED>

<!ELEMENT Measures (Measure+)>

<!ELEMENT Measure EMPTY>
<!ATTLIST Measure
    name CDATA #REQUIRED>

<!ELEMENT Dimensions ((DimensionDetailed, DimensionDetailed, DimensionDetailed*) | (DimensionShort,
DimensionShort, DimensionShort*))>

<!ELEMENT DimensionDetailed (HierarchyTree)>
<!ATTLIST DimensionDetailed
    name CDATA #REQUIRED>

<!ELEMENT HierarchyTree (M1+)>

<!ELEMENT M1 (M1*)>
<!ATTLIST M1
    level CDATA #REQUIRED
    value CDATA #REQUIRED
    ordinal CDATA #REQUIRED>

<!ELEMENT DimensionShort (ClassificationSchema)>
<!ATTLIST DimensionShort
    name CDATA #REQUIRED>

<!ELEMENT ClassificationSchema (Level)>

<!ELEMENT Level (Level?)>
<!ATTLIST Level
    name CDATA #REQUIRED>

<!--Ergebniss eine Sichtdefinierenden Anfrage-->
<!ELEMENT CubeView (Measure, TableDimensionLarge, TableDimensionLarge, Table, TotalAggregation)>

<!ELEMENT TotalAggregation (#PCDATA)>
<!ATTLIST TotalAggregation
    value CDATA #REQUIRED>

<!ELEMENT TableDimensionLarge (HierarchyTreeWithAggr)>
<!ATTLIST TableDimensionLarge
    name CDATA #REQUIRED
    alignment (horizontal | vertical) #REQUIRED>

<!ELEMENT HierarchyTreeWithAggr (M2+ | (LeafWithAggr+))>

<!ELEMENT M2 (M2+ | (LeafWithAggr+))>
<!ATTLIST M2
    level CDATA #REQUIRED
    value CDATA #REQUIRED
    ordinal CDATA #REQUIRED>

<!ELEMENT LeafWithAggr EMPTY>
<!ATTLIST LeafWithAggr
    level CDATA #REQUIRED
    value CDATA #REQUIRED
    ordinal CDATA #REQUIRED
    aggregationValue CDATA #REQUIRED>

<!ELEMENT Table (TR+)>

<!ELEMENT TR (TD+)>
<!ATTLIST TR
    ordinal CDATA #REQUIRED>

<!ELEMENT TD (#PCDATA)>
<!ATTLIST TD
    ordinal CDATA #REQUIRED>

<!--Ergebnis einer Operationalen Anfrage-->
```

```
<!ELEMENT CubeOperationView (Measures, Measure, TableDimensionSmall, TableDimensionSmall, Table,
TotalAggregation, Dimension*)>

<!ELEMENT TableDimensionSmall (HierarchyPathTree)>
<!ATTLIST TableDimensionSmall
    name CDATA #REQUIRED
    alignment (horizontal | vertical) #REQUIRED>

<!ELEMENT HierarchyPathTree (M3 | (LeafWithAggr+))>

<!ELEMENT M3 (M3 | (LeafWithAggr+))>
<!ATTLIST M3
    level CDATA #REQUIRED
    value CDATA #REQUIRED>

<!ELEMENT Dimension (HierachyPathTreeWithNextLevel)>
<!ATTLIST Dimension
    name CDATA #REQUIRED>

<!ELEMENT HierachyPathTreeWithNextLevel (M4 | (Leaf+))>

<!ELEMENT M4 (M4 | (Leaf+))>
<!ATTLIST M4
    leval CDATA #REQUIRED
    value CDATA #REQUIRED>

<!ELEMENT Leaf EMPTY>
<!ATTLIST Leaf
    level CDATA #REQUIRED
    value CDATA #REQUIRED>
```

Literaturverzeichnis

AbBS99 S. Abiteboul, P. Bruneman, D. Suciu: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann. ISBN 1-55860-622-X, 1999.

ACK+01 S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, K. Tolle: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. Proceedings of the Second International Workshop on the Semantic Web (SemWeb2001), 2001.

AgGS95 R. Agrawal, A. Gupta, S. Sarawagi: Modeling Multidimensional Databases. IBM Research Report, IBM Almaden Research Center, 1995.

AnMu97 S. Anahory, D. Murray: Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems. Addison Wesley. ISBN 0201175193, 1997.

Albr01 J. Albrecht: Business Intelligence mit Oracle9i. Datenbank Spektrum 1. Seite 36-43, dpunkt.verlag, 2001.

BaGü01 A. Bauer, H. Günzel: Data-Warehouse-Systeme: Architektur Entwicklung Anwendung. dpunkt.verlag, 2001.

BaPT97 E. Baralis, S. Paraboschi, E. Teniente: Materialized View Selection in a Multidimensional Database. 23rd VLDB Conference, 1997.

Baye97 R. Bayer: The Universal B-Tree for Multidimensional Indexing: general Concepts. Proc. Worldwide Computing and Its Applications, International Conference, (WWCA'97), Seite 198-209, Springer Verlag, 1997.

BBM+01 J. Bloemen, G. Brunner, W. Miller, B. Nelson, S. O. Schulz, B. Ströhle: IBM DB2 UDB V 7.2 and Oracle9i: A technical comparison. PASS Consulting Group Frankfurt a. Main, 2001.

Bern97 T. Berners-Lee: Metadata Architecture. W3C, 1997. *http://www.w3.org/TR/NOTE-rdf-simple-intro-971113.html*

Bern98 T. Berners-Lee: Why RDF model is different from the XML model. W3C, 1998. *http://www.w3.org/DesignIssues/RDF-XML.html*

BeSo96 D. Bearman, K. Sochats: Metadata Requirements for Evidence, Automating 21st Century Science. The Legal, Regulatory, Technical and Social Aspects of Electronic Laboratory Notebooks and Collaborative Computing, TeamScience, 1996.

BrFH00 J. Broekstra, C. Fluit, F. van Harmelen: Representation and Query Languages for Semistructured Data. On-To-Knowledge (EU-IST-199910132) Project deliverable 8, Aidministrator Nederland, 2000.
http://www.ontoknowledge.org/countd/countdown.cgi?draftdel8.pdf

BrGu00 D. Brickley, R.V. Guha: Resource Description Framework (RDF) Schema Specification. W3C Candidate Recommendation, 2001.
http://www.w3.org/TR/rdf-schema

BrGu97 T. Bray, R. V. Guha. An MCF Tutorial. Netscape, 1997.
http://www.w3.org/TR/NOTE-MCF-XML/MCF-tutorial.html

BrKa01a J. Broekstra, A. Kampman: Representation and Query Languages for Semistructured Data. On-To-Knowledge Project deliverable, 2001.
http://sesame.aidministrator.nl/doc/del9.pdf

BrKa01b J. Broekstra, A. Kampman: Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. On-To-Knowledge Project deliverable, 2001.
http://sesame.aidministrator.nl/doc/sesame-final.pdf.

Clau98 N. Clausen: OLAP Multidimensionale Datenbanken: Produkte, Markt, Funktionsweisen und Implementierung. Addison Wesley Verlag Bonn, ISBN 382731402X, 1998.

ClDe99 J. Clark, S. DeRose: XML Path Language (XPath) Version 1.0, W3C Recommendation, 1999.
http://www.w3.org/TR/xpath

CoCS93 E.F. Codd, S.B. Codd, C.T. Salley, Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical Report, Arbor Software Cooporation, 1993.

DBSA98 S. Decker, D. Brickley, J. Saarela, J. Angele: A Query and Inference Service for RDF. Proceedings of Query Languages Workshop (QL 1998), MA. 1998.

Ditt01 C. Dittberner: Zugriff auf multimedia XML-Dokumente in Objekt-relationalen Datenbanksystemen. Diplomarbeit an der Universität Rostock. 2001.

Drew02 M. Drews: Personalisierte Verwaltung und universeller Zugriff auf multimediale Dokumentstrukturen auf Basis von MPEG-7. Diplomarbeit an der Universität Rostock, 2002.

Elle97 C. Ellerman: Channel Definition Format (CDF). Microsoft Corp., 1997. *http://www.w3.org/TR/NOTE-CDFsubmit.html*

FaPS96 U. Fayyad, G. Piatesky-Shapiro, P. Smyth: From Data Mining to Knowledge Discovery: An Overview. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, Cambridge, MA, Seite 1-34, 1996.

FFM+01 P. Fankhauser, M. Fernández, A. Malhotra, M. Rys, J. Siméon, P. Wadler: XQuery 1.0 Formal Semantics. W3C Working Draft, 2001. *http://www.w3.org/TR/query-semantics/*

FHH+00 D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, M. Klein: OIL in a nutshell. Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), 2000.

FICS01 FORTH Institute of Computer Science: The RDF Query Language. 2001. *http://139.91.183.30:9090/RDF/RQL/index.html*

Fiel94 R. T. Fielding: Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web. Chapter 6, The Need for Visible Metainformation, 1994.

GaTo98 L. Cabbibo, R. Torlone. A Logical Approach to Multidimensional Databases. In 6th EDBT, 1998.

GCB+96 J. Gray, S. Chaudhuri, A.Brosworth, A. Layman, D. Reichard, M. Venkataro, F. Pellow, H. Pirahesh: Data Cube: A Relational Aggregation Operator. Generalizing Group-By, Cross-Tab and Sub-Totals. In: Proceedings of the 12th International Conference on Data Enginieering (ICDE 96, New Orleans (LA), U.S.A., 26. Feb.-1.März). 1996.

Gill00 A. J. Gilliland-Swetland: Introduction to Metadata Setting the Stage. 2000. *http://www.getty.edu/research/institute/standards/intrometadata/pdf/swetland.pdf*

GLMB98 R.V. Guha, O. Lassila, E. Miller, D. Brickley: Enabling Inferencing. 1998. *http://www.w3.org/TandS/QL/QL98/pp/enabling.html*

Gonz00 M.L. Gonzales: Last One Standing. CMP Media LLC. 2000. *http://www.intelligententerprise.com/000929/feat1.shtml*

Gros97 N. Großer: InfoHarness: Was sind Metadaten? 1997. *http://www2.informatik.uni-erlangen.de/IMMD-II/Lehre/SS97/internet/www/vortrag3/nsgrosse/META.HTM*

GuBr97 R.V. Guha, T. Bray: Meta Content Framework Using XML. W3C Technical Report, 1997.
http://www.w3.org/TR/NOTE-MCF-XML-970624/

Guha00 R.V. Guha: rdfDB: An RDF Database. 2000.
http://web1.guha.com/rdfdb/

GuPe99 P. Gultzan, T. Pelzer: SQL-99 Complete, Really: An Example-Based Reference Manual of the New Standard. R&D Books ISBN 0-87930-568-1, 1999.

GyLa97 M. Gyssens, L.V.S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In 23rd VLDB Conference, Athens, 1997.

Haas01 P. Haase: Situationsgesteuerter mobiler Zugriff auf Digitale Bibliotheken. Diplomarbeit an der Universität Rostock. 2001

Hard01 G. Harde: XCube: Konzept für eine XML-basierte Beschreibung von Datenwürfeln zur Realisierung eines föderativen Data-Warehouse-Netzwerkes. Föderierte Datenbanken 2001. Seite 64-77, 2001.

HBH+97 A. Hopmann, S. Berkun, G. Hatoun u.A.: Web Collections using XML. Microsoft Corp., 1997.
http://www.w3.org/TR/NOTE-XMLsubmit.html

HeGu01 J. Hendler, D. L. McGuinness: The DARPA agent markup language. IEEE ntelligent Systems, 2001.

Hert01 F. Hertel: XML Anfragesprachen: XML-QL, XQL, Quilt und XQuery. 2001.
http://www.fmi.uni-passau.de/~fhertel/Seminar/Seminar.ps

HeLu98 A. Heuer, A. Lubinski: Data Reduction - an Adaptation Technique for Mobile Environments. Proceedings of Interactive Applications of mobile Computing (IMC'98), 1998.

HeSa97 A. Heuer, G. Saake: Datenbanken: Konzepte und Sprachen. MITP-Verlag Bonn, ISBN 3-8266-0619, 1997.

HeSa99 A. Heuer, G. Saake: Datenbanken: Implementierungstechniken. MITP-Verlag Bonn, ISBN 3-8266-0513-6, 1999.

Karv00 G. Karvounarakis: A Declarative RDF Metadata Query Language for Community Web Portals. Master's thesis, University of Crete, 2000.

KCPA00 G. Karvounarakis, V. Christophides, D. Plexousakis, S. Alexaki: Querying Community Web Portals. Technical Report, Institute of Computer Science, FORTH, Heraklion, Greece, 2000.

KCPA99 G. Karvounarakis, V. Christophides, D. Plexoussakis : Querying Semistructured (Meta)Data and Schemas on the Web: The case of RDF and RDFS. Technical Report, Institute of Computer Science, FORTH, Heraklion, Greece, 1999.

KiLW95 M. Kifer, G. Lausen, J. Wu. Logical foundations of object-oriented and frame-based languages. Journal of the ACM, 42(4). Seite 741-843, 1995.

KoSc01 S. Kokkelink, R. Schwänzl: Expressing Qualified Dublin Core in RDF / XML. Dublin Core Metadata Initiative Recommendation, 2001. *http://dublincore.org/documents/2001/11/30/dcq-rdf-xml/*

KRRT98 R. Kimball, L. Reeves, M. Ross, W. Thornwaite: The Data Warehouse Lifecycle Toolkit. Wiley, New York, 1998.

LaSw99 O. Lassila, R. Swick: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 1999. *http://www.w3.org/TR/REC-rdf-syntax*

Lehn98 W. Lehner: Modeling Large Scale OLAP Scenarios. 6th EDBT, 1998.

LeRT96 W. Lehner, T. Ruf, M. Teschke: CROSS-DB: A Feature-extended Multi-dimensional Data Model for Statistical and Scientific Databases. 5th International Conference on Information and Knowledge Management, (CIKM 96, Rockville, Maryland), Seite 253-260, 1996.

LiWa96 C. Li, X. Sean Wang: A Data Model for Supporting On-Line Analytical Processing. CIKM 1996, 1996.

LJM+97 A. Layman, E. Jung, E. Maler, H. S. Thompson, J. Paoli, J. Tigue, N. H. Mikula, S. D. Rose: XML-Data. W3C Technical Report, 1997. *http://www.w3.org/TR/1998/NOTE-XML-data-0105/*

Lubi00 A. Lubinski: Small Database Answers for Small Mobile Resources. Intelligent Interactive Assistance & Mobile Multimedia Computing (IMC2000), Rostock, Seite 9. – 10, 2000.

LuHe00 A. Lubinski, A. Heuer: Configured Replication for Mobile Applications. Proceedings of the BalticDB&IS'2000, 1.-5.5.2000, Vilnius, Litauen, 2000.

Lust99 M. Lusti: Data Warehousing und Data Mining: Eine Einführung in entscheidungsunterstützende Systeme. Springer Verlag Berlin, ISBN 3-540-66221-9, 1999.

Marti01 J. M. Martinez: Overview of the MPEG-7 Standard. International Organisation For Standardisation (ISO), 2001. *http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm*, 2001.

MaSa98 M. Marchiori, J. Saarela: Query + Matadata + Logic = Metalog. 1998. *http://www.w3.org/TandS/QL/QL98/pp/metalog.html*

MaSa99a M. Marchiori, J. Saarela: Towards the Semantic Web: Metalog. 1999. *http://www.w3.org/RDF/Metalog/CIKM-050299.html*

MaSa99b M. Marchiori, J. Saarela: Metalog - The RDF querying system. 1999. *http://www.w3.org/RDF/Metalog/*

MaSu98 A. Malhotra, N. Sundaresan: RDF Query Specification. W3C Working Draft, 1998. *http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html*

Meln00 S. Melnik: Storing RDF in relational database. 2000. *http://www-db.stanford.edu/~melnik/rdf/db.html*

Meye99 H. Meyer: XML-Anfragesprachen: XQL, XML-QL und XML Lore. 1999.

Mill01 L. Miller: Inkling: RDF query using SquishQL. Institute for Learning and Research Technology (ILRT), 2001.

MiWe02 L. Milewski, I. Weber: Storing OLAP Cube Metadata in XML Dokuments. Studienarbeit an der Universität Rostock. 2002.

Onto01 Ontoprice GmbH: How to write F-Logic Programs: A Tutorial for the Language F-Logic. 2001. *http://www.ontoprise.de/download/*

Orac01 Oracle9i Data Warehousing Guide Release 1 (9.0.1): Chapter 18 SQL for Aggregation in Data Warehouses. 2001.

PeCr95 N. Pendse, R. Creeth: The OLAP Report: Succeeding with On-Line Analytical Processing. Vol. 1, Business Intelligence, 1995.

Petr97 J. Petrak: Data Mining Methoden und Anwendung. Technical Report TR-97-15. Österreichisches Forschungsinstitut für AI, 1997.

PICS96 Platform for Internet Content Selection (PICS). W3C, 1996. *http://www.w3.org/PICS/*

RoLS98 J. Robie, J. Lapp, D. Schach: XML Query Language (XQL). 1998. *http://www.w3.org/TandS/QL/QL98/pp/xql.html*

Rust01 M. Rust: Multimedia Content Management für Interaktive TV-Applikationen unter Einbeziehung des MPEG-7 Standards. Diplomarbeit an der Universität Rostock, 2001.

SaCo00 K.-U. Sattler, S. Conrad: Vorlesung Data-Warehouse-Technologien. Folien zur Vorlesung WS2000/01, Universität Magdeburg, 2000.

ScBM99 H. Schinzer, C. Bange, H. Mertens: Data Warehouse und Data Mining: Marktführende Produkte im Vergleich. 2., völlig überarbeitete und erweiterte Auflage, Verlag Franz Vahlen München, ISBN 3800624664, 1999.

Stan95 Standford Technology Group (Hrsg.): Designing the Data Warehouse on Relational Database. Whitepaper, Standford, 1995.

URI01 URIs, URLs, and URNs: Clarifications and Recommendations. W3C/IETF URI Planning Interest Group Technical Report, 2001. *http://www.w3.org/TR/uri-clarification*

VaSe01 P. Vassiliadis, T. Sellis: A Survey on Logical Models for OLAP Databases Technical Report DWQ NTUA 301, accepted for publication at SIGMOD RECORD, 2001.

Vass98 P. Vassiliadis: Modeling Multidimensional Databases, Cubes and Cube Operations. 10th SSDBM Conference, Italy, 1998.

WhWh99 M. Whitehorn, M. Whitehorn: Business Intelligence: The IBM Solutions. Springer-Verlag London, 1999.

Abbildungsverzeichnis

Thesen

1. Das Resource Description Framework bietet ein Instrument zur maschinenlesbaren Formulierung von Metadaten, die dank des offenen W3C-Standards geeignet sind, semantische Informationen über unstrukturierte und semistrukturierte Daten zu speichern. Diese Informationen können nicht nur für die Verwaltung sondern auch zur Verarbeitung der Information genutzt werden.

2. Die Anfragebearbeitung und Daten-Transformation basiert auf einem zweistufigen Ansatz bei der die Daten-Transformation (Content-Mediation) sowohl auf struktureller Ebene (XML/XSLT) als auch auf den Datenelementen selbst stattfindet. Die Funktionsbausteine werden dabei direkt durch die Device- und Kontext-Informationen des RDF-Content-Repositories gesteuert. Dadurch kann eine endgerätespezifische Ergebnistransformation gewährleistet werden.

3. Die Verwaltung von RDF-Daten in XML-Syntax ist nicht geeignet. Neben Speicherungsformen in OODBMS und ORDBMS sind primär Ansätze auf Basis relationaler Modelle untersucht worden. Es hat sich gezeigt, dass zur Persistierung ohne RDF-Schema-Informationen die einfache Speicherung von Statements in ihrer Tripelform als sinnvoll herausgestellt hat. Darauf aufbauend wurden Vorschläge zur Unterstützung von Anfragen durch Indizierungstechniken diskutiert. Eine Indizierung in Form einer mehrdimensionalen Speichertechnik, speziell der UB-Tree, stellte sich als geeignet heraus. Auf Grund der Verfügbarkeit konnte allerdings nur auf eindimensionale Indizierungstechniken zurückgegriffen werden.

4. RDF-Anfragesprachen lassen sich nach Logik- und Graphen-basierten Ansätzen klassifizieren. Nach Evaluierung existierender Anfragesprachen wurden F-Logic (Logik-basiert) oder RQL (Graphen-basiert) wegen ihrer Ausdrucksstärke favorisiert. Aufgrund unvollständiger beziehungsweise ungeeigneter Implementierungen wurde für die Entwicklungsarbeiten das Graphen-basierte SquishQL verwendet. Dies hat sich im Rahmen der Anwendungsumgebung als ausreichend herausgestellt. Als problematisch hat sich allerdings die Tatsache erwiesen, dass der verwendete SquishQL-Anfrageinterpreter die eindimensionale Indizierungstechniken nicht nutzt, was sich im Falle größerer Datenmengen zu Performance-Verlusten führt.

5. Die Erweiterung der XML-Framework-Architektur um OLAP-Funktionalität hat sich als grundsätzlich tragfähig erwiesen. Spezifische Anforderungen und Einschränkungen ergeben sich bzgl. des multidimensionalen Modells, der multidimensionalen Operatoren und der unterschiedlichen Speicherungsformen. Die Leistungsfähigkeit der entwickelten Konzepte konnten prototypisch als Erweiterung des Retrieval Systems der XPEA-Plattform in der Komponente OReSMo nachgewiesen werden.

6. Mit Hilfe der entwickelten Lösung wurde gezeigt, wie ein multidimensionaler Würfel serverseitig verwaltet, OLAP-Funktionalität auf dem Client bereitgestellt und die Kommunikation zwischen Server und Client aufgebaut werden kann. Für eine Bearbeitung der clientseitigen Anfragen müssen diese vom Server geeignet interpretiert und in Anfragen an das RDF-Content-Repository und die integrierten Datenbanken transformiert werden. Aus den ermittelten Daten wird vom Retrieval-Server ein Ergebnis in Form eines XML-Dokuments generiert, welches an das Endgerät gesendet wird. Die Tragfähigkeit dieses neuartigen Ansatzes konnte anhand des Anwendungsszenarios „Europa-MV“ nachgewiesen werden.

7. Die OLAP-Erweiterungen des SQL-99 Standards bieten die Grundlage einer effizienten Bearbeitung multidimensionaler Anfragen bei einer relationalen Speicherung von OLAP-Würfeln. Die untersuchten kommerzielle System Oracle und IBM/DB2 implementieren die OLAP-Erweiterungen des SQL99-Standards adäquat und bieten darüber hinaus systemspezifische Erweiterungen bezüglich physikalischer Speicherstrukturen und spezielle Indizierungstechniken.

8. Für die Bereitstellung der OLAP-Funktionalität empfiehlt sich die Online-Analyse, die das Anfragen von Sichten ermöglicht. Für deren Umsetzung wurden drei Ansätze nachgewiesen. Eine Nutzung vordefinierter Sichten bringt Einbußen hinsichtlich der Flexibilität mit sich, wogegen sichtdefinierende und operationale Anfragen eine flexible Analyse des multidimensionalen Datenraumes erlauben.

9. Die Verwendung von Java (JDK 1.2) als Implementierungsprache für XML-basierte Anwendungen hat sich bewährt. Bei der Performance der verfügbaren XML-Prozessoren ergaben sich allerdings erhebliche Unterschiede. So ist der Einsatz des XML-Parsers von Oracle bei der Verarbeitung größerer Datenmengen nicht zu empfehlen. Die beste Performance unter den getesteten Prozessoren boten JAXP von Sun und XALAN von Apache.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Quellen angefertigt habe.

Stefan Audersch

Rostock, 15. Januar 2002