Konzeption und Umsetzung eines interaktiven,
multimedialen Produktinformationssystems

Studienarbeit

Universität Rostock, Fachbereich Informatik
Lehrstuhl für Datenbank- und Informationssysteme

vorgelegt von: Christoph Dittberner
geboren am 05.11.1973 in Rostock

Betreuer: Prof. Dr. Andreas Heuer
Dipl.-Inf. Guntram Flach
Dipl.-Inf. Oliver Hein

Abgabedatum: 15. Mai 2000

Aufgabenstellung

Die schnelle Entwicklung und Verbreitung des World Wide Web hat zu einem Wandel in der Geschäftsstrategie vieler Unternehmen geführt. Das Web wird zunehmend als geeignetes Medium für den Handel und die Werbung entdeckt, aber auch als Möglichkeit, damit unternehmensweite Netze (Intranet) aufzubauen. Daraus ergeben sich aber gerade für die Datenbanksystem-Architekturen eine Reihe von Anforderungen, um diesen multimedialen Anwendungen gerecht zu werden.

Im Rahmen dieser Studienarbeit sollen zunächst verschiedene Strategien zur Entwicklung von datenbankbasierten Multimediaanwendungen untersucht werden. Dabei sind sowohl evolutionäre objekt-relationale als auch rein objektorientierte Ansätze anhand relevanter Anforderungskriterien zu berücksichtigen.

Das von Computer Associates (CA) entwickelte Datenbanksystem Jasmine wurde durchgängig objektorientiert entworfen, um Multimedia- und Web- Anwendungen zu ermöglichen. Dieses Multimedia-DBMS ist hinsichtlich relevanter Anforderungen (zuvor aufgestellte Kriterien) zu bewerten und anderen Lösungen gegenüberzustellen (**INFORMIX Universal Server**).

Als Anwendungsszenario wird ein Produkt-Informationssystem vorgeschlagen, das herkömmliche Produktinformationen (Preis, Hersteller, etc.) mit multimedialen Darstellungen (u.a. Bild, Video, Ton) in Kombination zulässt und intra-/ internetweit zur Verfügung stellt. Zur Umsetzung werden sowohl die Entwicklungswerkzeuge von Jasmine als auch die Programmiersprache Java eingesetzt. Für vergleichende Untersuchungen können wahlweise die INFORMIX-Entwicklungswerkzeuge genutzt werden.

Danksagung

Ich möchte mich bei Prof. Dr. Andreas Heuer und bei Dipl. Inf. Guntram Flach für ihre nützliche Kritik und hilfreiche Anregungen und Hinweisen bedanken.

Ferner bin ich Dipl. Inf. Oliver Hein für die fachlichen Diskussionen sowie Hinweise zum Thema Jasmine sehr dankbar.

Mein besonderer Dank gilt Martin Löffler, der mich sehr bei der Realisierung der Benutzeroberfläche der Prototypen unterstützt hat.

Inhaltsverzeichnis

1. Einführung

1.1 Motivation

Das Internet hat in den vergangenen Jahren immer mehr an Bedeutung gewonnen. Durch immer mehr Firmen, Institutionen und private Haushalte, die in das World Wide Web (WWW) streben, wächst die Größe dieses weltweiten Netzes ständig an. Schätzungen zufolge sind heute ca. 44 Millionen Rechner im Internet miteinander verbunden.

Für Firmen und andere Institutionen ist es somit wichtiger denn je geworden, sich im WWW zu präsentieren, um so neue Märkte zu erschließen und den gegenseitigen Erfahrungsaustausch voran zu treiben. Auf dem heutigen wettbewerbsorientiertem Markt sind WWW- und E-Commerce-Anwendungen ein wichtiges strategisches Mittel geworden. Dabei spielt die Präsentation von Informationen aus verschiedensten internen und externen Quellen eine tragende Rolle. Betrachten wir zum Beispiel einmal die Online-Kataloge. Sie bieten beispielsweise Möglichkeiten zur Suche nach Produktinformationen inklusive Preisen und Referenzen und sollen die Kooperation mit externen Anbietern und Lieferanten ermöglichen. Wesentliche Schwerpunkte bei solch einer Präsentation liegen natürlich auch auf einer ausgefeilten Darstellung der persönlichen Daten. Dazu gehört auch, dass neben den eigentlichen Daten multimediale Elemente wie Grafiken, erläuternde Filmen oder beschreibender Ton mit eingebunden werden.

Auf dem Datenbanksektor hat sich in letzter Zeit viel getan. Dominierten früher die relationalen Datenbankmanagementsysteme (DBMS), so werden heute mehr und mehr objektorientierte bzw. objekt-relationale DBMS von Firmen zur Datenverwaltung eingesetzt. Durch die Ausweitung des Internets ergeben sich neue Gesichtspunkte in Hinblick auf die Funktionalität, die ein Datenbanksystem heute zu gewährleisten hat. Neben der Einbindung der Multimediakomponenten liegt ein wesentlicher Schwerpunkt in der Anbindung an das World Wide Web.

1.2 Zielstellung und Vorgehensweise

Im Rahmen dieser Studienarbeit werde ich nun verschiedene Strategien zur Entwicklung datenbankbasierter Multimediaanwendungen untersuchen. Ich werde Kriterien aufstellen, worauf bei der Konzeption und Gestaltung eines Produktinformationssystems zu achten ist. Im weiteren zeige ich Ihnen als ein spezielles Anwendungsszenario den Prototyp des Produktinformationssystems ProInfo-MV und werde erläutern wie bei der Umsetzung dieses Systems vorgegangen wurde. Darüber hinaus war ich an der Realisierung zweier Prototypen[1] zur Text- und Bilderkennung beteiligt, bei der Content-Based-Retrieval-Techniken evaluiert werden konnten.

[1] Die Prototypen „Text- und Image-Retrieval“ sind Realisierungen, die im Rahmen von Projekten am ZGDV Rostock entstanden sind.

2. Grundlagen

Zum besseren Verständnis dieser Studienarbeit möchte ich im ersten Teil dieses Kapitels erläutern, was man sich unter einem Produktinformationssystem (PIS) vorstellen könnte und welche Kriterien man bei der Umsetzung beachten muss.

Im zweiten Teil erläutere ich kurz den Begriff Multimedia und zeige, was sich hinter einem Multimedia Datenbankmanagementsystem (MMDBMS) verbirgt bzw. welche Kriterien erfüllt sein müssen. Abschließend werde ich die objekt-relationalen- und objektorientierten Datenbankmanagementsysteme (ORDBMS und OODBMS) vorstellen und hinsichtlich ihrer Umsetzung der objektorientierten Konzepte bewerten.

2.1 Produktinformationssysteme

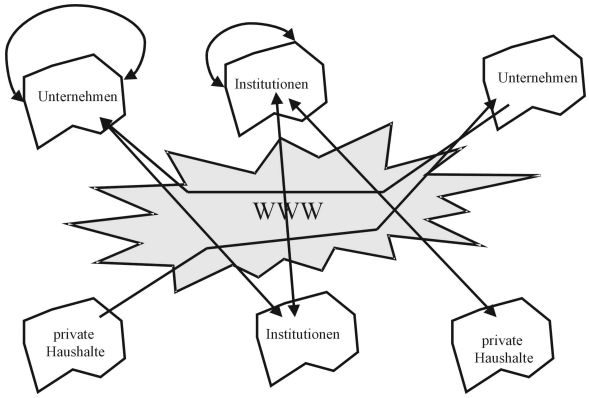

Abbildung 2.1 Die Kommunikation im Internet

Durch das steigende Angebot von immer leistungsfähigeren Rechnern mit CD-ROM Laufwerken und Internetanbindung haben sich die Möglichkeiten elektronischer Präsentationen auf CD-ROM und über das Internet stark verbessert. Weitere Vorteile gegenüber dem herkömmlichen Papierkatalog sind vor allem die Einbindung multimedialer Darstellungen, die Aktualität und der schnelle Zugriff auf die Informationen.

Für Firmen, die sich im WWW präsentieren wollen, spielt zusätzlich die Einbindung ihrer vorhandenen Datenbestände eine wichtige Rolle. Gerade für sie ist eine ansprechende Präsentation besonders wichtig, soll hier doch der Kunde einerseits einen umfassenden Firmenüberblick bekommen und andererseits die Möglichkeit erhalten, sich die Produktinformationen anzuschauen bzw. die einzelnen Produkte zu bestellen. Wesentliche Schwerpunkte für Firmen sind dann

- eine ausgefeilte Darstellung der Daten,
- die Migration, Verwaltung, Integration und Präsentation unternehmenspezifischer Daten (auch alter Datenbestände),
- die Erweiterung dieser Informationen um erklärende Texte, Audio, Video und Grafiken,

- Möglichkeiten zur inhaltsbasierten Suche und
- Realisierung des Datenaustausches zwischen Kunde und Unternehmen
- unter Umständen auch die Portierbarkeit auf CD-ROM o.ä.

Damit diese Kriterien erfüllt und umgesetzt werden können, muss man das Produkt-Informationssystem in zwei Komponenten unterteilen, die Präsentations- und die Angebotskomponente.

2.1.1 Die Präsentationskomponente

Diese Komponente dient primär dem Werbezweck. Hier geht es vorrangig um die multimediale Aufbereitung der firmenspezifischen Daten, eine umfassende Firmendarstellung sowie Interaktionsmöglichkeiten mit den Benutzern des Informationssystems. Diese Komponente ist durch eine individuelle Gestaltung charakterisiert, so enthält sie Informationen wie Adressen, Telefonnummern, Ansprechpartner und textuelle Informationen über das Unternehmen sowie Videos und Animationen, die mit gesprochenem Text bzw. Musik untermalt sind. Außerdem werden hier Angaben zu den Geschäftsbedingungen und Serviceleistungen sowie über die Unternehmenshistorie gemacht. Auch andere Angaben wie Umweltschutz und Qualität der angebotenen Produkte (Testberichte, Zeitungsartikel etc.) oder kleine Gewinnspiele sind hier denkbar.

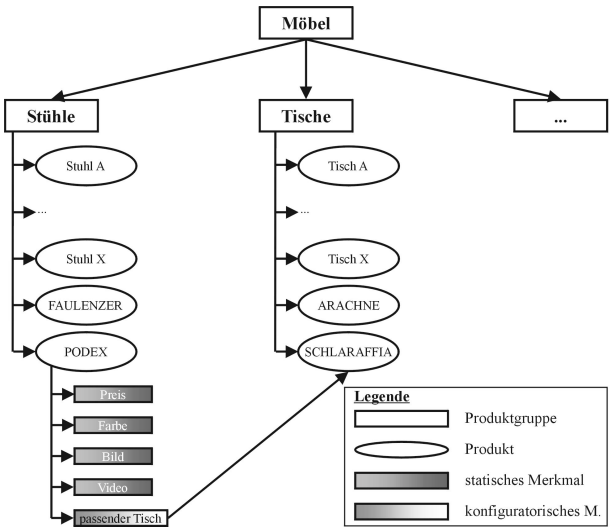

Abbildung 2.2 Die Produkthierarchie der Angebotskomponente

2.1.2 Die Angebotskomponente

Die Angebotskomponente sollte umfassende Produktinformationen enthalten und kann durch die Anbindung von Mehrwertdiensten wie Bestellung, Beratung oder auch Finanzierung aufgewertet werden. Hier werden vorwiegend textuelle Daten dargestellt und entsprechend komplexe Suchmechanismen zur Volltextrecherche angeboten. Einen Vorschlag zur Struktur der Daten möchte ich am Beispiel eines fiktiven Einrichtungshauses folgendermaßen vorstellen. Die einzelnen Produkte wie „Tisch SCHLARAFFIA“, „Tisch ARACHNE“, „Stuhl PODEX“ oder „Stuhl FAULENZER“ werden in Produktgruppen („MÖBEL“) zusammengefasst, die wiederum in andere Produktgruppen („TISCHE“, „STÜHLE“) untergliedert werden können

und dadurch eine hierarchische Struktur definieren. In diese Gruppen werden dann die einzelnen Produkte eingegliedert. Gekennzeichnet bzw. charakterisiert sind diese Produkte wiederum durch statische Merkmale wie Preis, Farbe, Form sowie durch konfiguratorische Merkmale, die die Beziehung eines Produkts zu anderen Produkten/ Produktgruppen festlegen. So ist zum Beispiel der „Stuhl FAULENZER“ ein Zusatzprodukt zu dem „Tisch ARACHNE“. Um die Präsentations- und Verkaufsfunktion zu verbessern, sollten zusätzliche statische multimediale Merkmale wie Bilder, Video, Audio oder auch Elemente aus dem Bereich „Virtual Reality“ mit eingebracht werden. In Abbildung 2.2 werden diese Zusammenhänge verdeutlicht.

2.1.3 Navigation

Dem Benutzer des Produktinformationssystems stehen zwei Möglichkeiten zur Verfügung, um auf die Produktinformationen zugreifen zu können: der selektive und der explorative Zugriff.

Der selektive Zugriff auf die Daten stellt eine Möglichkeit dar, über einfache oder komplexe Suchanfragen auf die Informationen zuzugreifen. Durch Angabe von Produktmerkmalen können ein oder mehrere Produkte durch den Endbenutzer gezielt ausgewählt werden.

Eine weniger zielorientierte Methode erfolgt durch den explorativen Zugriff. Der Benutzer verfolgt hierbei einen der Verweise, die ihm durch das System angeboten werden. Ausgehend von einem Inhaltsverzeichnis und durch die Möglichkeit, auf die nächste oder die vorherige Produktseite zu gelangen, kann man sich hier ein lineares Navigieren ähnlich dem Blättern in einem Katalog vorstellen. Dieser „erkundende“ Zugriff lässt sich auch über die konfiguratorischen Produktmerkmale realisieren.

Eine Kombination aus selektivem und explorativem Zugriff, derart dass der Benutzer über seine Anfrage in die passende Produkthierarchie gelangt, ist auch denkbar. Ein weiterer Ansatz zur Verfeinerung der Suchergebnisse wäre die Fähigkeit des Systems anhand der passenden Produkte, durch den Benutzer auszuwählen, die Suchanfrage so zu erweitern, dass möglichst viele dieser passenden Produkte gefunden werden können. Diese Ansätze werden innerhalb dieser Arbeit nicht weiter verfolgt.

2.1.4 Präsentation/ Vertrieb

Wie und wodurch soll der Benutzer erreicht werden? Die Wahl des entsprechenden Mediums zum Vertrieb des Produktinformationssystems kann über Erfolg und Misserfolg entscheiden. Durch den steigenden Einfluss von E-Commerce-Anwendungen bietet sich das Internet geradezu an. Aber den Anwender ohne Internet-Zugang sollte man auch weiterhin berücksichtigen und so eine Möglichkeit des Vertriebs auf CD-ROM vorsehen.

2.1.5 Erweiterungen (optional)

Weitere Kriterien bei der Gestaltung und Entwicklung eines Produktinformationssystems wären die Unterstützung von Mehrsprachigkeit und das Anbieten von Mehrwertdiensten.

Diese Notwendigkeit der Mehrsprachigkeit ist dabei natürlich vom Einsatzrahmen des Systems abhängig. Will man zum Beispiel nur den deutschen Markt ansprechen, macht es eigentlich keinen Sinn zusätzliche Sprachen anzubieten.

Mehrwertdienste wie zum Beispiel ein Kundenberatungsassistent, eine Anforderungsanalyse oder eine Finanzierungsberatung wären zwar denkbar, zum Beispiel in Zusammenarbeit mit Kreditinstituten, sind aber nicht zwingend notwendig.

2.2 Multimedia

Multimedia (MM) ist das Schlagwort der 90er. Jede Anwendung, die eine Kombination aus Bildern, Sound und Videos in irgendeiner Art und Weise präsentiert und gewisse Interaktionsmöglichkeiten bietet, klassifiziert sich meistens als Multimedia-System. Doch was bedeutet das eigentlich?

Als MM-Daten werden Text (1 Seite: ≈2 KB), Audio-Informationen (zeitabhängig; 1 min WAV-Format: ≈9,5 MB), Bilddaten (1 Seite: 10 KB bis 100 MB je nach Auflösung), Video (zeitabhängig; Folge von Bildern/ Frames; 1 min AVI-Format: 4 MB bis 50 MB je nach Auflösung) und auch Vektorgraphiken (2D/ 3D) bezeichnet. MM-Informationen sind, wie man hier schon sieht, sehr platzaufwendig. Neuere Formate, die die Daten nach speziellen Komprimierungsstrategien auf einen Bruchteil (1:10 und mehr) packen, haben in jüngster Zeit weite Verbreitung erfahren: z.B. MP3-Format (Audio; 5 min ≈4-5 MB). Ein typisches Multimedia-System (im herkömmlichen Sinne) besteht aus dem Anwendungsprogramm und den Multimediadaten, wird auf eine oder mehrere CD-ROMs gepresst und dient vielleicht als Präsentationssystem, durch das der Benutzer mittels eines speziellen Browsers navigieren kann.

Was wir aber wirklich benötigen, sind inhaltsbasierten Retrieval-Techniken. So können Daten über Schlüsselwörter bzw. Meta-Informationen klassifiziert werden. Hier wird zwischen manueller (Schlüsselwörter werden vom Benutzer per Hand vergeben; ein wichtiger Aspekt liegt dabei auch auf der hierarchischen Struktur der Meta-Informationen), halbautomatischer (durch eine vom Benutzer vorgenommene Klassifikation lernt das System und klassifiziert automatisch die nicht beschriebenen Bereiche) bzw. automatischer Merkmalsextraktion (ausgehend von einer vorgegebenen Klassifikation; analog zu halbautomatisch) unterschieden. Ein weiterer Ansatz beruht auf der Ähnlichkeitsmessung. So wird die Farbähnlichkeitssuche über Farbhistogramm, Farbdominanz bzw. Farbverteilung in Bezug auf ein Beispielbild die Suche durchgeführt. Solche Mechanismen gibt es auch für die Suche über Texturen (Muster) und/ oder Konturen. Für das Retrieval über Video-, Audio-, Text- und räumlichen Daten müssen ebenso Methoden vorhanden sein.

2.3 Multimediadatenbanken

Das Problem der effektive Verwaltung der MM-Daten lässt sich am elegantesten durch Datenbankmanagementsysteme lösen. Die Grundfunktionalität für Multimediadatenbankmanagementsysteme (MMDBMS) kann man am ehesten durch folgende Definition nach [KB95] festlegen.

Ein MMDBS bietet

1. Unterstützung für Bild- und Mediendatentypen (kontinuierliche Datentypen ermöglichen den kontinuierlichen Zugriff in Echtzeit; spezielle Caching-Strategien),
2. Fähigkeiten zur Verwaltung einer sehr großen Anzahl von Medien-Objekten,

3. eine effiziente Speicherverwaltung,
4. grundlegende Datenbankfunktionalitäten sowie
5. Information-Retrieval-Funktionalität.

MMDBS müssen geometrische Datentypen für geometrische und räumliche Zusammenhänge bieten, deren Funktionalität auch in der Anfragesprache genutzt werden kann. Dies impliziert auch die Unterstützung sowohl eindimensionaler als auch mehrdimensionaler Indexierung. In einem interaktiven Anfragemodus müssen die Antworten nach Relevanz und Feedback bewertet werden können. Ähnlichkeitsanfragen werden durch Feature-Extraktion und –Indexierung (Content-Retrieval Indexing) geboten. Wegen der anfallenden großen Datenmengen bei der Arbeit mit MM-Daten muss ein MMDBS die Speicherung von großen Binärobjekten als BLOBs (Binary Large Objects) oder CLOBs (Character Large Objects) und das Clustern bzw. Declustern komplexer Objekte ermöglichen.

Weiterführende Konzepte von MMDBMS sind [KB95] und [Sub98] zu entnehmen.

2.4 Datenbankmanagementsysteme

2.4.1 Klassifikation von DBMS

DBMS, die im Rahmen von Client/Server-Architekturen das sogenannte „Back-End“ bilden, das heißt sie sind für die persistente Datenhaltung verantwortlich, sind aus modernen IT-Umgebungen schon lange nicht mehr wegzudenken. Eine von Stonebraker und Moore eingeführte Klassifikationsmatrix, die die DBMS nach ihrer Anfrage- und Datenkomplexität unterteilt, zeigt Abbildung 2.3 .

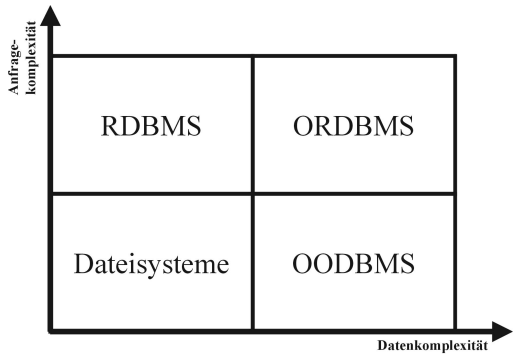

Abbildung 2.3 Klassifikation von DBMS

Anhand dieser Klassifikation zeigt sich, dass für die komplexen Datenstrukturen, die im Rahmen eines Produktinformationssystems verwendet werden sollen, die relationalen Datenbankmanagementsysteme (RDBMS) sowie die reinen Dateisysteme nicht in Frage kommen. RDBMS sind zwar weit verbreitet und werden erfolgreich eingesetzt, haben jedoch grundlegende Nachteile in den Bereichen Datenbankentwurf, Anwendungsmodellierung und Anfrageformulierung:

- Die SQL-Datentypen sind begrenzt und nicht erweiterbar.
- Die Darstellung komplexer Objekte und ihrer Beziehungen mittels Tabellen ist schwierig (Semantikverlust).

- Das Navigieren mit SQL ist schwierig und es sind sehr komplexe SQL-Konstrukte notwendig, um die Struktur der Daten wieder sichtbar zu machen.
- Die Operationen zur Datenanalyse sind in SQL sehr eingeschränkt. [Ver97]

Objektorientierte Konzepte (siehe Tabelle 2.1 nach [Heu97a],[Heu98]) sollen diese Probleme nun beheben. In den beiden folgenden Abschnitten möchte ich deshalb die objekt-relationalen DBMS (ORDBMS) und die objektorientierten DBMS (OO-DBMS) vorstellen und hinsichtlich der Umsetzung dieser Konzepte kurz erläutern.

Kriterien		Kategorien	
Bestandteile	Konzepte	ORDBMS	OODBMS
Strukturteil	Typkonstruktoren	♦	♦
	Objektidentität	♦	♦
	Klassen	◊	◊
	Beziehungen	◊	
	Strukturvererbung	♦	♦
	Integritätsbeding.	♦	
Operationenteil	generische	♦	♦
	relationale	♦	
	objekterzeugende		
	objekterhaltende		
Höhere Konzepte	Metaklassen		
	Methoden	♦	♦
	Vererbung	♦	♦
	Overriding	◊	♦
	Einkapselung	◊	◊
	Mehrfachvererbung		◊

Tabelle 2.1 Merkmale von objekt-relationalen und objektorientierten Datenbanksystemen im Vergleich (♦ = K.O. Kriterium, ◊ = optionales Kriterium)

2.4.2 Das objekt-relationale Datenbanksystem

Die objekt-relationalen Datenbanksysteme (ORDBS) bauen auf die Technologie relationaler Konzepte auf und haben die Fähigkeit, komplexe Objekte durch objektorientierte Konzepte zu verwalten. Die Objektorientierung steckt eigentlich in den benutzerdefinierten Datentypen, weil hier die Methoden und die Vererbung wirksam werden. So lässt sich ein ORDBMS folgendermaßen charakterisieren:

(1) *Benutzerdefinierter Typen* (UDT für „**U**ser **D**efined **T**ype“) können aus Standarddatentypen und bereits vorhandenen UDT unter Benutzung von Typkonstruktoren (`SET OF, TUPLE OF, LIST OF`) gebildet werden. An solche UDT können Methoden (UDR für „**U**ser **D**efined **R**outines“) gebunden werden, die nur auf diesem Typ ausführbar sind. Der Benutzer sollte nicht von außerhalb auf die interne Struktur der komplexen Typen zugreifen können. (*Einkapselung*).

Bei der Tabellendeklaration können benutzerdefinierte- und Standarddatentypen gleichermaßen verwendet werden.

(2) *Vererbung*: UDT können Strukturinformationen und Methoden von andern benutzerdefinierten Datentypen erben und somit eine Typhierarchie aufbauen. Eine Relation (*Klasse*) kann als Unterklasse einer anderen Relation (in diesem Fall Oberklasse) definiert werden; die Objekte der Unterklasse sind dann eine Teilmenge der Oberklasse. Dadurch wird eine Klassenhierarchie aufgebaut.

(3) *Objektidentität* (OID) sichert die Trennung von Schema und Werten.

(4) Die *Anfragesprache* sollte einerseits den Standard von SQL erfüllen und andererseits die Möglichkeiten der neu hinzugekommenen komplexen Objekte ausnutzen: Navigation über Referenzen, Zugriff auf geschachtelte Objekte / Relationen durch „Pfadausdrücke“.

(5) *Overriding* sollte es ermöglichen, ererbte Methoden zu redefinieren oder zu überschreiben, so dass diese Methoden dem neuen Datentyp angepasst bzw. verfeinert werden können.

Abschließend ist noch zu erwähnen, dass man wie bei den Programmiersprachen auch bei den objekt-relationalen Datenbanksystemen dazu übergeht, vordefinierte Typ-Bibliotheken für die verschiedensten Anwendungsbereiche anzubieten, die dann einfach in das System integriert werden können. Informix bietet zum Beispiel sogenannte *DataBlades* an, die das Datenbanksystem um z.B. Multimediadatentypen für Text-, Bild-, Videodaten u.a. erweitern und zahlreiche spezielle Funktionen auf diesen Datentypen bereitstellen. Diese Erweiterungen gibt es nicht nur vom Hersteller sondern auch von Drittanbietern.

2.4.3 Das objektorientierte Datenbanksystem

Im Gegensatz zum ORDBS, das relationale um objektorientierte Konzepte erweitert, ist ein OODBS ein System, das

(1) ein objektorientiertes Datenbankmodel mit Strukturteil, Operationen und höheren Konstrukten unterstützt (siehe Tabelle 2.1),

(2) Erweiterbarkeit zumindest auf der konzeptuellen Ebene bietet,

(3) Persistenz, Dateiorganisationsformen und Zugriffspfade, Transaktionen, Concurrency Control und Recovery realisiert, sowie

(4) neben einer interaktiv nutzbaren Anfragesprache auch eine komplette Programmierumgebung enthält. [Heu97a]

Betrachten wir Punkt 1 betreffend das objektorientierte Datenbankmodell (OODM) so arbeiten wir jetzt nicht mehr auf Relationen, sondern modellieren die Zusammenhänge durch Klassen. Diese Klassen dienen als Behälter für Objekte, d.h. wir strukturieren eine Klasse und bestimmen somit das Verhalten der Objekte (siehe objektorientierte Programmiersprachen – OOPL für „**o**bject **o**riented **p**rogramming **l**anguage“).

Integritätsbedingungen (wie Fremdschlüssel, Einschränkung von Domänen, Primärschlüssel) und Beziehungen (Kardinalitäten) sind explizit nicht notwendig bzw. optional, werden sie doch durch inhärente Konzepte objektorientierter Datenbankmodelle wie Objektidentität, Klasse-Komponentenklasse, Klasse-Unterklasse, Untertypen von Standard-Datentypen abgebildet. Metaklassen und Mehrfachvererbung haben hier eine höhere Bedeutung als bei den ORDBS. So kann man in Metaklassen (Grundprinzip: Klassen werden als Instanzen von Metaklassen aufgefasst) statische

Attribute (gelten für alle Instanzen einer Klasse) und die Methoden-Implementierungen einer Klasse unterbringen.

Erweiterbarkeit hinsichtlich der konzeptuellen Ebene (siehe Drei-Ebenen-Architektur in [Heu97b]) bedeutet die Erweiterung um neue Klassen und ihre Methoden, Typen (z.B. abstrakte Datentypen) und ihre Funktionen, Sichten und abgeleitete Klassen, für die nicht unbedingt neue Speicherstrukturen auf der internen Ebene realisiert werden müssen: Die Typkonstruktoren und Standard-Datentypen bestimmen dann die Dateistrukturen und Zugriffspfade.

Punkt 3 bestimmt die generellen Anforderungen an ein Datenbanksystem, muss aber auf die komplexen Datentypen und Methoden bei den OODBMS angepasst werden.

Neben einer interaktiven auf dem Operationenteil basierenden Anfragesprache soll natürlich ein Werkzeug zur Anwendungsprogrammierung bereitgestellt werden. Im Gegensatz zu einer eher schlechten Realisierung bei den relationalen Datenbanken („impedance-mismatch-Problem“) soll die Anbindung einer Anfragesprache an die Programmierumgebung homogener erfolgen.

Ausführlichere Betrachtung und Bewertung sind [Heu97a], dem Manifesto [ABD+89] und [Cat97] zu entnehmen.

Beispiele und Umsetzung der in diesem Abschnitt vorgestellten Konzepte anhand von JASMINE sind Kapitel 3.2 zu entnehmen.

2.5 Fazit

In diesem Kapitel habe ich ein mögliches Schema für Produktinformationssysteme vorgestellt. Tabelle 2.2 fasst diese Kriterien noch einmal zusammen.

Kategorie	Kriterium	
Daten	komplexe Strukturen	♦
	Beziehungen	♦
	Multimediafähigkeit	♦
	Erweiterbarkeit	♦
	Einbindung der Altdaten	◊
Zugriffskontrolle	Transaktionen	♦
	Mehrbenutzerbetrieb	♦
	Sessions	◊
Navigation	Suchfunktionalität	♦
	Information – Retrieval	♦
Präsentation	WWW-Anbindung	♦
	Portierbarkeit	◊
Kooperation	Mehrwertdienste	◊

Tabelle 2.2 Kriterien für PISs (♦ = notwendig, ◊ = optional)

Wesentliche Kriterien sind dabei

- die Unterstützung von komplexen Datenstrukturen, Multimedia, Beziehungen und Hierarchien,
- die Erweiterbarkeit des Systems um neue Datentypen, Methoden und Geschäftsvorgänge,

- Transaktionskontrolle und Mehrbenutzerbetrieb,
- die Möglichkeit über die vorhandenen Datenbestände und Multimediadaten, Suchanfragen zu stellen sowie
- die Anbindung an das WWW.

Durch den Einsatz von DBMSs haben wir eine Grundlage für die effiziente Datenverwaltung und -haltung. Die Unterstützung komplexer Datenstrukturen und Multimedia-Datentypen nebst Funktionen auf diesen Datentypen erreichen wir durch die Umsetzung objektorientierte Konzepte.

Je nach Anwendungsfall ist es wünschenswert bestehende Altdaten in das PIS zu übernehmen bzw. auf sie zugreifen zu können. Diese Möglichkeit kann entweder durch das DBMS oder durch eine Programmierkomponente (z.B. CGI-Anbindung der Altdaten) realisiert werden.

Auch das Session-Prinzip, d.h. die Aktionen sind in einer benutzspezifischen Sitzung gekapselt, ist bei einem reinen Präsentationssystem nicht notwendig. Bei Bedarf kann es auch über die programmiertechnische Seite umgesetzt werden. Gleiches gilt auch für das Angebot von Mehrwertdiensten wie Finanzierung etc., das nicht unbedingt vom PIS sondern von weiteren Drittanbietern (z.B. Banken) abhängt.

Portierbarkeit auf externe Medien ist nicht immer zu gewährleisten, da man schlecht das komplette Datenbanksystem vertreiben will, könnte aber entweder über das Internet (Client-Anwendung greift auf den Datenbankserver zu) oder über exportierte Datenstrukturen erreicht werden.

3. ORDBMS und OODBMS im Vergleich

In diesem Kapitel möchte ich nun als Vertreter der objekt-relationalen DBMS den „Universal Server v9.14“ von Informix und für die objektorientierten DBMS „Jasmine“ von Computer Associates (CA) und Fujitsu vorstellen und hinsichtlich der im vorhergehenden Kapitel erläuterten Kriterien (siehe Tabelle 2.1) bewerten.

3.1 Informix Universal Server v9.14

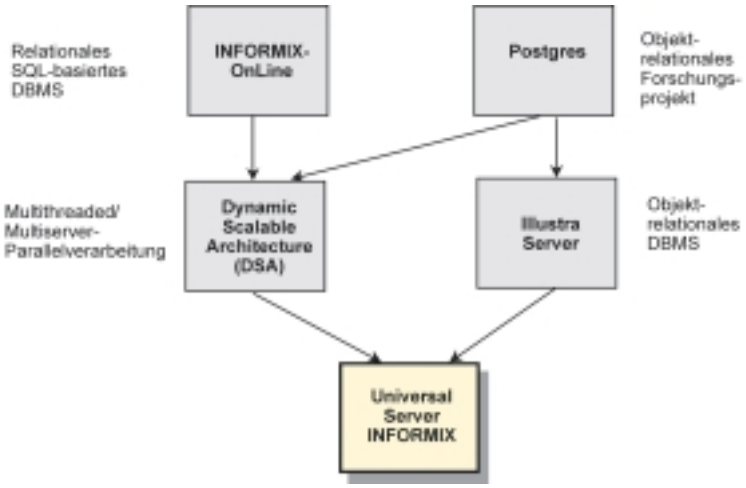

Abbildung 3.1 Entwicklungshierarchie des INFORMIX Universal Server

Informix Software Inc. wurde 1980 gegründet und war eine der ersten Firmen, die sich mit der Entwicklung relationaler Datenbanksysteme beschäftigten. Auf der Basis des relationalen *INFORMIX-OnLine* und des objekt-relationalen *Illustra* wurde der *Universal Server* entwickelt, der dann in den *Dynamic Server* mit *Universal Data Option* überging (siehe Abbildung 3.1). In diesem Abschnitt beschäftigen wir uns mit dem *Universal Server v9.14*, der im folgenden nur noch mit *Informix* bezeichnet werden soll. Der Unterschied zwischen dem DBMS und der Firma soll herbei durch *Kursivschrift* hervorgehoben werden.

Bei den relationalen Datenbankmanagementsystemen war die Einschränkung auf Standarddatentypen einer der größten Nachteile. Durch die Erweiterung relationaler Systeme um benutzerdefinierte Datentypen und Funktionen soll dieses Manko nun behoben werden.

3.1.1 Strukturteil

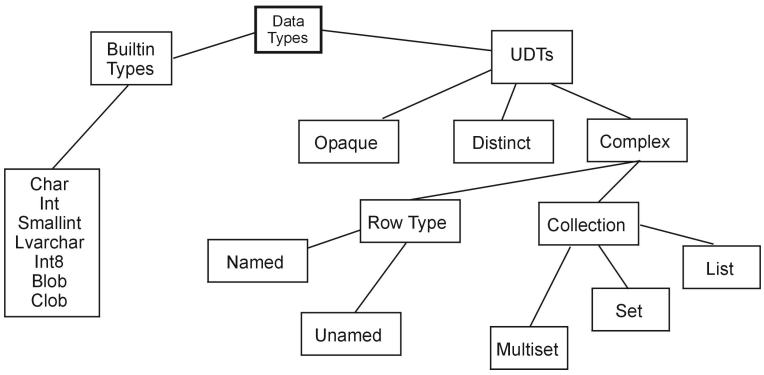

Abbildung 3.2 Neue Datentypen bei Informix

Typen und *Tabellen* (auch als *Klassen* bezeichnet) können bei *Informix* getrennt voneinander definiert werden. Wie Abbildung 3.2 zeigt, enthält der „Universal Server" eine Reihe von neuen Datentypen, die in

- Standarddatentypen: INT8, SERIAL8(n), BOOLEAN, LVARCHAR, CLOB und BLOB

und

- benutzerdefinierte Datentypen:
  - **opaque-Datentyp**
    - beschreibt abstrakten Datentyp (ADT), d.h. es erfolgt eine Kapselung der internen Struktur und der Operationen.
    - benötigt öffentliche Funktionen zum Zugriff auf den ADT
    - enthält Strukturinformationen zur Datenorganisation und
    - Umwandlungsfunktionen vom internen Format in das Benutzerformat und zurück
  - **distinct-Datentyp**
    - basiert auf Standard- oder anderen benutzerdefinierten Datentypen
    - interne Struktur wird durch Quelldatentyp bestimmt
    - Funktionen des Quelldatentyps werden übernommen
    - erlaubt Definition von neuen Operationen und Umwandlungsfunktionen (*cast*-Funktionen)
  - **komplexe Datentypen**
    - Reihentyp (row type)
      - Sammlung von Feldern (jedes Feld hat eigenen Namen und Datentyp)
      - muss mindestens ein Feld enthalten
    - Kollektionsdatentypen
      - Menge (vom Typ SET)
      - Multimenge (vom Typ MULTISET) und
      - Liste (vom Typ LIST)

unterteilt werden können.

Eine Typ- bzw. Tabellenhierarchie wird dabei durch die **UNDER**-Klausel erzeugt. Dabei sind Typ- und Tabellenhierarchie streng miteinander verknüpft und können

nicht voneinander getrennt werden. So lassen sich benutzerdefinierte Typen und Untertypen sich wie folgt definieren:

Beispiel 3.1 Anlegen von Reihentypen und typisierten Relationen

```
CREATE ROW TYPE Adresse_t ( Strasse VARCHAR(50),Ort VARCHAR(30));

CREATE ROW TYPE Person_t (
     Name VARCHAR(30),
     Alter INT,
     Adresse Adresse_t,
     Hobbies SET(VARCHAR(30) NOT NULL)
);

CREATE TABLE Personen OF TYPE Person_t;

CREATE ROW TYPE Student_t ( Matrnr VARCHAR(20)) UNDER Person_t;
CREATE TABLE Studenten OF TYPE Student_t UNDER Personen;
```

Hier haben wir Reihentypen für Personen, Adressen und Studenten angelegt, miteinander kombiniert und die Relationen *Personen* und *Studenten* angelegt. Ein möglicher Zustand der Datenbank könnte nach Einfügen von Studenten und (anderen) Personen so aussehen:

RowID	Name	Alter	Adresse		Hobbies
			Strasse	Ort	Hobby
257	Otto Ohnmacht	62	Am Weg 23	Bad Doberan	Wandern Briefmarken
258	Bernhard Meier	60	Unter dem Baum 4	Bad Doberan	Wandern Radfahren
259	Anna Meier	60	Unter dem Baum 4	Bad Doberan	Wandern Musik
257	Anton Meier	22	Mahlstrom 6	Rostock	Computer Schwimmen Musik
258	Karl Schmidt	24	Mahlstrom 6	Rostock	Schwimmen Lesen

Tabelle 3.1 Relation Personen

RowID	Name	Alter	Adresse		Hobbies	Matrnr
			Strasse	Ort	Hobby	
257	Anton Meier	22	Mahlstrom 6	Rostock	Computer Schwimmen Musik	0997808098
258	Karl Schmidt	24	Mahlstrom 6	Rostock	Schwimmen Lesen	0997808123

Tabelle 3.2 Relation Studenten

Das Konzept *Objektidentität* (OID) wird bei *Informix* durch die **RowID** simuliert, wobei die in *Personen* auftauchenden Studenten-Tupel automatisch bei den *Personen* auftauchen und mit ihrer OID bzgl. der Studenten-Relation dargestellt werden. Leider wird die RowID nicht für jedes Tupel neu ermittelt, d.h. wenn die Tupel aus Personen gelöscht und wieder eingefügt werden, bekommen sie wieder eine RowID beginnend mit „257“ aufsteigend zugewiesen.

Um uns z.B. durch eine Selektion auf Hobbies den Inhalt der Kollektion anzeigen zu lassen, bedarf es eines „Work-Arounds“, da *Informix* diese mehrzeilige Tupelausga-

be nicht zulässt. Folgende an `Person_t` gebundene SPL-Routine demonstriert eine mögliche Vorgehensweise:

Beispiel 3.2 Arbeiten mit Mengen

```
CREATE FUNCTION p_hobbies(param person_t)
RETURNING VARCHAR(200);
     DEFINE hobbies_s SET ( VARCHAR(30) NOT NULL);
     DEFINE hobby VARCHAR(30);
     DEFINE sammel VARCHAR(200);
     LET SAMMEL = "";
     SELECT hobbies INTO hobbies_s
        FROM Personen WHERE param.name = name;
     FOREACH k1 FOR
        SELECT * INTO hobby FROM TABLE(hobbies_s)
        LET sammel = sammel || hobby || ' ';
     END FOREACH;
     RETURN sammel;
END FUNCTION;

SELECT ROWID, name, alter, adresse, p_hobbies(e) AS hobbies
FROM Personen e;     -- ergibt dann Tabelle 3.1
```

Wir können an den Typ `Person_t` auch die folgende Funktion binden, die den Wert `true`[2] zurückgibt, falls die als Parameter übergebene Instanz des Typs `Person_t` älter als 60 Jahre ist.

Beispiel 3.3 Binden und Verwendung von UDRs

```
CREATE FUNCTION istRentner (param Person_t) RETURNS BOOLEAN;
     DEFINE ret BOOLEAN ;
     LET ret = 'f';
     IF (param.alter >= 60) THEN LET ret = 't'; END IF;
     RETURN ret ;
END FUNCTION;

SELECT e.Name FROM Personen e WHERE istRentner(e);
```

Durch die *Vererbung* kann diese Funktion auch für Studenten angewendet werden, die ja auch Personen sind. Durch obige Anfrage werden dann für Studenten (normalerweise) keine Ergebnisse zurückgeliefert. Durch *Overloading* können wir diese Methode auf den Fall „Studenten-Rentner“ (z.B. Alter über 30 Jahre) anpassen und folgendermaßen implementieren:

```
CREATE FUNCTION istRentner (param Student_t) RETURNS BOOLEAN;
     DEFINE ret BOOLEAN;
     LET ret = 'f';
     IF (param.alter >= 30) THEN LET ret = 't'; END IF;
     RETURN ret;
END FUNCTION;

SELECT e.Name FROM Personen e WHERE istRentner(e);
```

Jetzt liefert die Select-Anweisung über *Personen* auch die Studenten, deren Alter bzgl. Studenten schon etwas weiter „fortgeschritten“ ist („*Late-Binding*“).

[2] Die boolschen Werte `true/false` werden bei *Informix* durch die Literale `t/f` gekennzeichnet bzw. zugewiesen.

Klasse-Komponentenklasse-Beziehungen in Komponentenobjekten wie bei *Illustra* über **REF** und **DEREF** umgesetzt, sind beim „Universal Server“ nicht „mehr“ vorhanden.

3.1.2 Operationenteil

Die Anfragesprache heißt Informix-SQL und ist kompatibel zu Standard-SQL, enthält aber Erweiterungen, die optional genutzt werden können. Für Benutzer die reines Standard-SQL benutzen wollen, kann man den *Informix* SQL Prozessor anweisen, die verwendeten SQL Ausdrücke auf Kompatibilität zu untersuchen; es werden dann Warnungen bei der Verwendung von *Informix* spezifischen Erweiterungen ausgegeben.

Eine Besonderheit besteht zum Beispiel bei der Verwendung von Oberklassen bzgl. der Typ-Tabellenhierarchie in Anfragen. Wie in den vorherigen Beispielen zu sehen war, enthält die Select-Anweisung **`SELECT * FROM Personen`** auch die Tupel aus der Studenten-Relation, welche in der Hierarchie unter *Personen* angesiedelt ist. Möchte man die Ergebnis-Tupel auf *Personen* beschränken bzw. auf ungebundene Tupel der Oberklasse[3], muss bei der Anfragestellung das Schlüsselwort **ONLY** mit angegeben werden; die Anfrage müsste dann **`SELECT * FROM ONLY (Personen)`** lauten. In diesem Fall werden dann nur die Personen, die nicht Studenten sind, als Ergebnis-Tupel zurückgeliefert.

Diese Besonderheit des Hierarchie-Konzepts muss auch bei der Verwendung von UPDATE- und DELETE-Anweisungen berücksichtigt werden.

3.1.3 Höhere Konzepte

In der aktuellen Version 9.14 des „INFORMIX Universal Server“ wird nur Einfachvererbung unterstützt. *Methoden* können in den abstrakten Datentypen gekapselt werden, sind aber auch für Reihentypen möglich.

Durch CREATE FUNCTION werden die Funktionen deklariert und können/müssen in SPL oder C implementiert werden. Die interne Struktur und die notwendigen Support-Funktionen werden generell durch C-Kode definiert und implementiert, müssen aber in der Datenbank über SQL registriert werden. Außerdem müssen für die Support-Funktionen entsprechende Typkonvertierungs-Anweisungen (*Casts*) in der Datenbank registriert werden, die dann explizit oder implizit[4] die Konvertierung von dem ADT zu der internen Struktur und umgekehrt vornehmen.

Weiterführende Hinweise und Informationen sind [IdocA] Kapitel 5 zu entnehmen.

3.1.4 Multimedia

Zum Speichern und Bearbeiten großer Datenmengen, wie sie im Bereich Multimedia anfallen, gibt es bei *Informix* einerseits die Datentypen TEXT und BYTE für einfache große Objekte und die Typen BLOB und CLOB (zusammengefasst als LOB für „**L**arge **OB**jekts“) für sogenannte smarte große Objekte; siehe Abbildung 3.2.

---

[3] Ungebundene Tupel der Oberklasse bezeichnen Tupel, die in keiner der Unterklassen vorkommen.

[4] Ob ein explizites oder implizites Casting erfolgt ist vom Typ der Support-Funktion fest vorgegeben.

Im Unterschied zu den einfachen Objekten kann auf die smarten Objekte direkt zugegriffen werden; d.h. jeder Teil eines LOB kann gelesen oder überschrieben werden. Im Gegensatz dazu können Objekte vom Typ Text oder BYTE nur gelesen oder gelöscht, aber nicht verändert werden.

Weitere Vorteile der smarten Objekte sind unter anderem auch:

- das Recovery, weil sie im Transaktionsprotokoll gespeichert werden,
- die geringere Zugriffszeit und
- die Vergleichsmöglichkeiten für smarte Objekte desselben Types.

Beim Zugriff auf smarte Objekte ist zu beachten, dass dieser indirekt mittels eines vom Datenbank-Server gelieferten Zeigers erfolgt. Folgendes Beispiel aus dem „Image-Retrieval-Projekt“ verdeutlicht das Holen eines solchen Zeigers:

Beispiel 3.4 Arbeiten mit LOBs

```
select img_big.location.locator.lo_pointer from imagetable
where img_id=$REFPIC[5]

/* Hier erfolgt die Zuweisung des durch obiges Statement gehol-
ten Zeigers an die Variable $input */

select img_id, rank from imagetable where resembles(img_big,
     row('IpdColorImageFE(RETINA_WIDTH=32, RETINA_HEIGHT=32)',
     row(row(row('IFX_BLOB','$input',NULL::lvarchar),
     NULL::lvarchar),'','', 0, 0, '')),rank #REAL);

/* Der in die Variable $input gefetchte BLOB wurde hier als Pa-
rameter für eine Methode zur Farbähnlichkeitssuche verwendet. */
```

Zusammenfassend kann man sagen, dass die LOB-Technik die Grundlage für die Speicherung der MM-Datentypen bildet. Was jetzt noch fehlt, sind die eigentlichen MM-Datentypen mit ihrer Struktur und ihren Methoden.

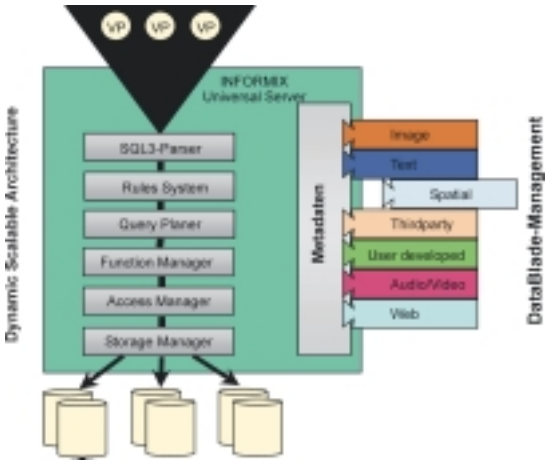

Abbildung 3.3 Architektur des Informix Universal Servers

[5] $REFPIC bezeichnet hier einen numerischen Parameter und könnte bspw. mit 22 initialisiert sein, was dann soviel wie Bild 22 bedeuten würde.

Abbildung 3.3 (*Quelle: Informix*) zeigt die Einbindung von sogenannten DataBlades in *Informix*. Hierbei wird das Datenbanksystem um in DataBlades gekapselte benutzerdefinierte Datentypen mit ihren entsprechenden Methoden zur Konvertierung, Speicherung und Zugriff (siehe opaque-Datentyp) erweitert. Die DataBlades werden von Informix oder Partnerfirmen angeboten, als versierter Benutzer hat man sogar die Möglichkeit eigene DataBlades zu programmieren und einzubinden. So wird von Excalibur ein DataBlade für den Datentyp Image angeboten, welches das Speichern und den Zugriff auf Bilddaten unterschiedlicher Formate und die Suche nach „passenden“ bzw. ähnlichen Bildern bezüglich Farben, Texturen oder Konturen unterstützt.

Durch anderer DataBlades weiterer Anbieter werden Multimedia-Datentypen wie Text, Video, Audio oder auch für räumliche Daten in das Datenbanksystem eingebunden, bspw. für:

- geospatiale Daten
  - Geocoding (MapInfo)
  - Geodetic (Informix)
  - Global/Interval (TelContar)
  - Spatial Database Engine (ESRI)
- Text- & Dokument-Management
  - Text DataBlade Module (Excalibur)
  - Real-Time Profiling Module (Excalibur)
  - COCOON (Dimedis)
- World Wide Web & E-Commerce
  - Message DataBlade (Informix)
  - Web DataBlade (Informix)

Im Rahmen dieser Studienarbeit wurden verschiedene Prototypen zur Untersuchung der Multimediafähigkeit entwickelt. Im Kapitel 4 werde ich diese Prototypen genauer vorstellen und noch einmal intensiver auf das Image-Datablade und auf das Text-Datablade von Excalibur eingehen.

3.1.5 WWW-Anbindung

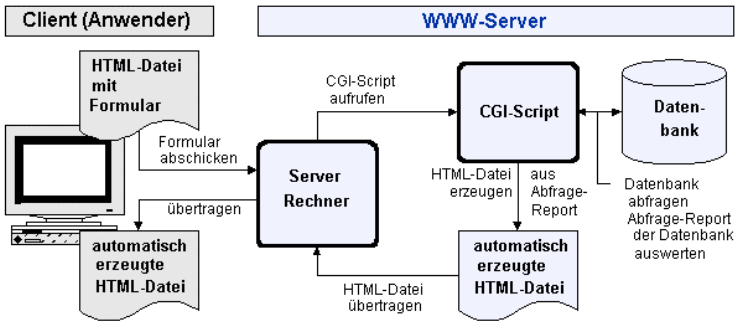

Abbildung 3.4 Herkömmliche WWW-Anbindung über CGI

Wie aus Abbildung 3.4 zu erkennen ist, wird normalerweise eine Datenbank an das WWW mittels CGI-Programmierung[6] angebunden. Die CGI-Programme werden in einem bestimmten Verzeichnis des Webservers (defaultmäßig „/cgi-bin/“) getrennt von der Datenbank gespeichert. Als Programmiersprache für diese CGI-Anwendungen wird meistens Perl, C oder Tcl genutzt.

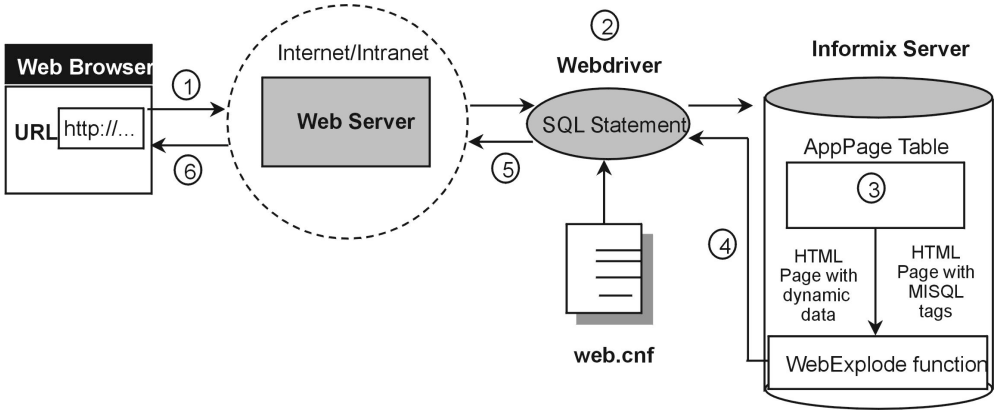

Abbildung 3.5 Datenbankanbindung über das Web-DataBlade

Bei *Informix* besteht diese Möglichkeit auch weiterhin; man hat allerdings die Möglichkeit, mit dem Web-DataBlade von Informix den Server so zu erweitern, dass man direkt auf das DBMS ohne Umweg über die CGI-Programmierung zugreifen kann; siehe Abbildung 3.5.

Die Datenbank wird dabei um den Datentyp *HTML* und die Funktion *WebExplode*() erweitert. Die HTML-Objekte enthalten zusätzlich zu den Standard-HTML-Tags noch spezielle Web-DataBlade-Tags, die durch WebExplode() dynamisch ausgewertet werden und so den Zugriff auf die Datenbank ermöglichen. Durch die DataBlade-Tags **`MIVAR`**, **`MIBLOCK`**, **`MISQL`**, **`MIERROR`** und diverse zusätzliche Funktionen zur Auswertung von booleschen und arithmetischen Ausdrücken wird HTML um datenbankspezifische Programmierkonstrukte wie Variablen, Verzweigungen, Fehlerbehandlung u.ä. erweitert. Über das **`MISQL`**-Tag werden die SQL-Anweisungen ausgeführt und die Ergebnisse dynamisch in die zu erzeugende Web-Seite einbebettet. Eine Möglichkeit zur Programmierung von bedingten oder iterativen Schleifen ist leider nicht vorhanden. Ein „Work-Around“ für dieses Manko wäre die rekursive Anwendung der WebExplode-Funktion, wobei es allerdings bei fehlerhafter Abbruchbedingung schon mal zum Absturz des Datenbankservers kommen kann. Zusammenfassend kann man sagen, dass durch das Web-DataBlade im Gegensatz zur herkömmlichen Datenbankanbindung nun die Programmierlogik persistent in der Datenbank gehalten wird.

Als dritte Komponente enthält das Web-DataBlade noch den *Webdriver*. Er ist das Kernstück des DataBlades und steuert den Zugriff auf die Datenbank. Dabei handelt es sich um ein Programm, das ähnlich zu CGI-Programmen bei jedem Aufruf der Form „`http://host:port/cgi-bin/webdriver?Parameter=Wert&...`“ durch den Web-Server gestartet wird, über die Datei „web.cnf“ die Konfigurationseinstellungen ein-

[6] CGI bedeutet **C**ommon **G**ateway **I**nterface und dient als Programmier-Schnittstelle zur dynamischen Erzeugung von WWW-Seiten.

liest und dann die WebExplode-Funktion aufruft, über die es dann die dynamisch erzeugten HTML-Seiten an den Web-Browser zurück gibt.

3.2 JASMINE v1.21

Das Datenbanksystem JASMINE, ein Gemeinschaftsprodukt der Firmen Computer Associates und Fujitsu, gehört zu den Neuentwicklungen, d.h. es baut nicht auf einer objektorientierten Programmiersprache oder dem relationalen Datenbankmodell auf. Ungeachtet der Tatsache, dass die komplexen Objekte intern in geschachtelten Relationen gespeichert werden, ist JASMINE ein „echtes“ objektorientiertes Datenbankmanagementsystem, da das Datenmodell und die Datenbanksprache ODQL („Object Definition & Query Language“) voll objektorientiert sind. [HFP+98]

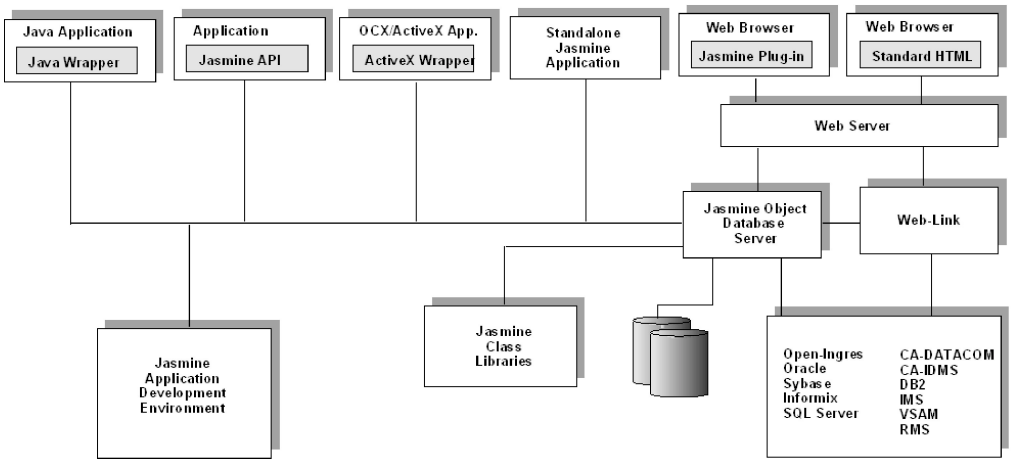

Abbildung 3.6 Die Architektur von JASMINE

Das ODBMS Jasmine beinhaltet folgende Komponenten (siehe Abbildung 3.6):

- JASMINE Object Database Server, zu dem ODB-II die Basis bildet.
- Die interne Datenbankebene unter Weiterverwendung von Komponenten von CA-OpenIngres.
- Viele Klassenbibliotheken für verschiedene Anwendungsbereiche wie die Anbindung von SQL-Datenbanksystemen und Multimedia-Applikationen.
- Das JASMINE Application Development Studio (JADS) als Entwurfswerkzeug für Applikationen.
- Das JASMINE WebLink Modul, durch das die Anbindung an das WWW erfolgt.
- Unterstützung verschiedener APIs und Sprachanbindungen für C, C++, ActiveX und OLE.

Wie schon angeführt, ist JASMINE ein voll objektorientiertes Datenbankmanagementsystem. Hinsichtlich der für OODBMS geltenden Kriterien (siehe Tabelle 2.1) möchte ich es im folgenden nun näher untersuchen.

3.2.1 Strukturteil

Mit Hilfe der *Typkonstruktoren* Menge (Set), Multimenge (Bag), Liste (List) und Feld (Array) können aus den Standarddatentypen komplexe Typen als Mengen gleichen Typs (Collection) erzeugt werden. Allerdings ist es nicht gestattet, Mengen von Mengen wie z.B. eine Liste von Integer-Sets zu definieren. Als Work-Around kann man aber eine Liste von Tupeln definieren, die dann wiederum ein Set repräsentieren können. In JASMINE gibt es zwar einen Tupelkonstruktor; man kann ihn allerdings nicht für die Klassendefinition nutzen, sondern nur um die Anfrageergebnisse in ODQL zu strukturieren. Sollen Komponenten unterschiedlichen Typs als Attribut einer Klasse zugeordnet werden, muss dafür eine eigene Klasse definiert werden und diese über eine Klasse-Komponentenklasse-Beziehung verbunden werden. Aus Performanzgründen ist es aber günstiger, auf diese Strukturierung zu verzichten und die unstrukturierten Attribute in der Klasse zu verwenden.[JasDocB] Vom objektorientierten Ansatz aus betrachtet ist diese Umsetzung zwar nicht wünschenswert, ist aber bei einer Anfrage über alle Personen die zum Beispiel in Rostock leben effizienter, da wir erstens die Attribute Ort und Postleitzahl besser durch Indexe unterstützen könnten und zweitens nicht für jedes Objekt die Klasse-Komponenklasse-Beziehung zur Adresse verfolgen müssen, was bei sehr vielen Personen schon zu Performanzeinbußen führen kann. Eine Übersicht über die Datentypen gibt Abbildung 3.7.

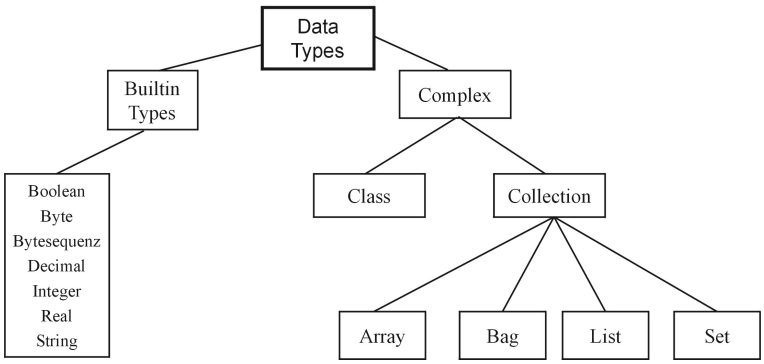

Abbildung 3.7 Datentypen in JASMINE

Beispiel 3.5 Die Klasse Person und das „Erzeugen“ einer Instanz

```
defineClass Person
  super: Composite
  description: "describes attributes and methods of persons"
{
  instance:
    String name unique: mandatory: default: NIL;
    Integer age default: 0;
    String street mandatory: default: NIL;
    String town mandatory: default: NIL;
    mediaCF::JpgFile image default: NIL;
    Bag[7]<String> hobbies default: NIL;
};
Person.new ( name := "Otto Ohnmacht", age := 45,
  street  := "Am Weg 23",town := "Bad Doberan",
```

[7] Eine Unterscheidung zwischen Menge und Multimenge habe ich bei den Beispielen generell nicht vorgenommen.

```
  hobbies := Bag{"Wandern", "Briefmarken"}
);
```

Analog zum *Informix*-Beispiel[8] habe ich eine Klasse angelegt, die als Behälter für die zu erzeugenden Personen dient, hinzugekommen ist hier noch das Attribut **`image`**, das später im Laufe einer Applikation ein Passfoto der jeweiligen Person aufnehmen könnte. Außerdem gilt die Annahme, dass eine Person über den Namen eindeutig identifiziert werden kann.

Im folgenden werde ich diese Klasse um zusätzliche Methoden erweitern. Eine ODQL-Anfrage nach allen Personen, deren Alter über/gleich 60 Jahre ist, sähe beispielsweise so aus:

Beispiel 3.6 ODQL Demo

```
Bag<Person> results;
results = Person from Person where Person.age >= "60";
results.print();
```

Mit dem Kommando **`addProcedure`** kann man obige Anfrage in eine Methode verpacken und an die Instanzen der Klasse binden (ähnlich zu *Informix*):

Beispiel 3.7 Instanzgebundene Methode

```
addProcedure Boolean Person::instance:istRentner()
{
  $defaultCF 'demoCF';
  $Boolean ret; $ret = FALSE;
  $if (self.age >= 60) { ret = TRUE; };
  $return( ret );
}
/* ODQL Zugriff: */
results = Person from Person where Person.istRentner() == TRUE;
results.print();
```

Auf den ersten Blick bringt diese Methode keinen Effizienzgewinn, ist aber nun in die Klasse eingebunden und somit auch durch die Vererbung in Unterklassen verfügbar.

In JASMINE können Methoden und Attribute nicht nur an Instanzen sondern auch an die Klasse selbst gebunden werden. Aufbauend auf obige Methode füge ich im folgenden der Person-Klasse eine Methode **`alleRentner()`** hinzu, die den ODQL-Rest (Definition des Multisets, Füllen über die Anfrage, Anzeigen der Ergebnisse) auf einen Aufruf beschränkt.

Beispiel 3.8 Klassengebundene Methode

```
addProcedure Bag<Person> Person::class:alleRentner()
{
  $defaultCF 'demoCF';
  $Bag<Person> b;
  $b = p from Person p where p.istRentner()==TRUE;
  $return(b);
}
/* ODQL Zugriff: */
Person.alleRentner().print();
```

[8] Aufgrund in JASMINE reservierter Wörter wie z.B. „alter“ habe ich die Attribute der Klassen im Gegensatz zu *Informix* in Englisch betitelt.

In JASMINE hat jedes Objekt eine auf logischer Ebene vergebene eindeutige *Objektidentität* (OID). Dieser Identifikator wird über den Namen der Klassenbibliothek, die Nummer der Klasse und die laufende Objektnummer in dieser Klasse gebildet. Demzufolge kann ein Objekt in nur einer Klasse[9] enthalten sein und auch ein Wechsel des Objekts in eine andere Klasse (Objektmigration) unter Beibehaltung der OID ist nicht möglich. Über ein in JASMINE vordefiniertes Makro ist es möglich, die OID auszulesen und so zwischen der Datenbanksprache ODQL und den Hostsprachen C bzw. C++ auszutauschen.

Durch eine *Klasse* wird einerseits die Menge der zu ihr gehörenden Instanzen bestimmt und andererseits auch deren Typ festgelegt. Das Schema ist also Klassen-Typ-basiert. Im Gegensatz zu *Informix* sind in die Klassendefinition auch die Methoden eingebunden.

Die Klassen können in JASMINE in einer Klasse-Komponentenklasse-Beziehung stehen. Wie die meisten OODBMS unterstützt JASMINE nur Referenzen, d.h. gemeinsame, unabhängige und nicht eingekapselte Komponentenobjekte. Spezielle Ausprägungen dieser *Beziehung* sind leider nicht spezifizierbar. Auch gibt es keine Möglichkeit, inverse Beziehungen[10] anzugeben. Kardinalitäten (1:1, 1:m und m:n) von Beziehungen werden durch inhärenten Konzepte (z.B. mengenwertige Attribute bei der Klassendefinition) dargestellt. Beispiel 3.9 zeigt das Erstellen von Unterklassen und die Umsetzung der Klasse-Komponentenklasse-Beziehung anhand der Attribute **`father`** und **`mother`**, die bei *Informix* weggelassen wurden, da diese Art der Beziehung dort über Fremdschlüssel umgesetzt werden muss.

Beispiel 3.9 Vererbung und Klasse-Komponentenklasse-Beziehungen

```
defineClass Student
  super: Person
  description: "describes student objects"
{
  instance:
    String matrno mandatory: default: NIL unique:;
    Person mother default: NIL;
    Person father default: NIL;
};
```

Durch das Schlüsselwort **`super`** wird dabei Person als Oberklasse von Student definiert, wobei Student dann alle Eigenschaften (d.h. alle Methoden und Attribute auf Klassen- und Instanzebene) von Person erbt.

Das Einfügen der Studenten müsste folgendermaßen vonstatten gehen:

1. Hole Referenz auf die Person der Mutter
2. Hole Referenz auf die Person des Vaters
3. Erzeuge den Studenten und weise die Referenzen zu

Um diese drei Schritte zusammenzufassen, könnte man an die Klasse Person eine Methode **`getPerson(String name)`**binden, die das entsprechende Personen-Objekt aus der Datenbank zurückgibt.

---

[9] Durch das Schlüsselwort „**alone**“ innerhalb der FROM-Klausel lässt sich steuern, ob sich eine Anfrage nur auf die ungebundenen Instanzen der Oberklasse bezieht (siehe „ungebundene“ Tupel in Fußnote 3/ Seite 21).

[10] Mit Methoden, die automatisch im Konstruktor ausgeführt werden, kann man die Funktionalität inverser Beziehungen umsetzen. Ein Automatismus ist durch JASMINE nicht gegeben.

Beispiel 3.10 „Erzeugen“ eines Studenten

```
Student.new ( name := "Anton Meier", age :=22,
  street  := "Mahlstrom 6", town := "Rostock",
  hobbies := Bag{"Computer","Schwimmen","Musik"},
  matrno  := "0997808098",
  father  := Person.getPerson("Bernhard Meier"),
  mother  := Person.getPerson("Anna Meier")
);
```

Die *Strukturhierarchie* in JASMINE ist eine Is-a-Hierarchie, die Klassen- und Typhierarchie miteinander koppelt. Dabei wird diese durch Spezialisierung um nutzerdefinierte Klassen erweitert, die dann Unterklassen der Klasse *Composite* sind. In den systemdefinierten Klassen (siehe Abbildung 3.8) sind die Objekte entweder Literale oder Entitäten, wobei Literale keine OID haben. Sie gelten als bereits zur Installationszeit erzeugt, so muss die Integer-Zahl „2“ nicht erst erzeugt werden, sondern kann sofort vom Anwender benutzt werden. Auch darf bzw. kann sie nicht gelöscht werden: es wäre schon problematisch wenn „1+1“ plötzlich keinen Wert mehr ergeben würde.

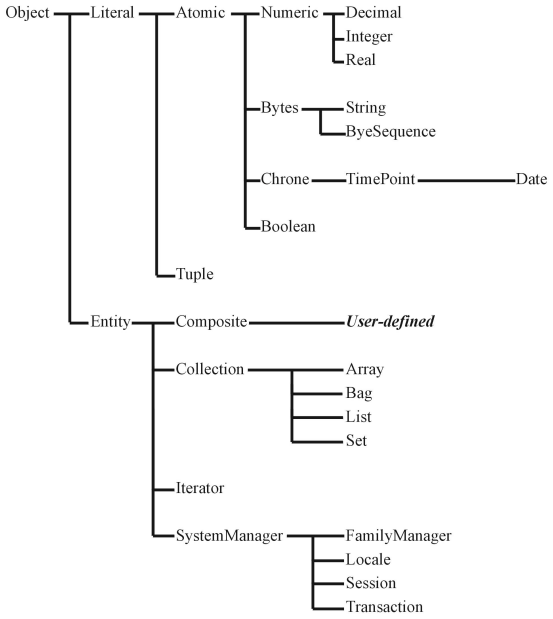

Abbildung 3.8 Systemdefinierte Klassen in Jasmine

Bei JASMINE erbt eine Klasse nicht nur die Struktur und das Verhalten einer anderen Klasse, sondern auch deren Werte der Klassenattribute und Klassenmethoden. Eine Methode wird durch das **`defineProcedure`**-Kommando redefiniert, wobei die Implementation entweder vollständig verändert oder angepasst werden kann. Allerdings ist darauf zu achten, dass Rückgabewert sowie Parameter in Typ und Anzahl übereinstimmen. In JASMINE ist es nicht möglich, gleiche Methoden mit unterschiedlichen Rückgabewerten und/oder Parametern zu implementieren, d.h. es gibt im Gegensatz zu *Informix* kein Overloading. Da Mehrfachvererbung erlaubt ist, werden auftretende Konflikte durch Überschreiben oder, wenn das nicht möglich ist, durch manuelles Umbenennen in einer der Oberklassen vermieden.

Bei der Klassendefinition können zusätzlich Integritätsbedingungen für Attribute mitangegeben werden. So können Attributen die Eigenschaften *unique* und *mandatory* [11] sowie Default-Werte zugewiesen werden. Innerhalb von Transaktionen kann die Überprüfung dieser Bedingungen bis ans Ende der Transaktion verzögert werden. Referentielle Integrität ist optional, da es in JASMINE erlaubt ist, Referenzen auf nicht mehr vorhandene Objekte zu halten; erst wenn über diese Referenz auf ein fehlendes Objekt zugegriffen wird, tritt ein Fehler auf.

3.2.2 Operationenteil

JASMINE beinhaltet einen eigene Datenbanksprache mit dem Namen ODQL (für ***O**bject **D**efinition and **Q**uery **L**anguage*). ODQL hat alle Fähigkeiten einer vollständigen Programmiersprache wie Schleifen und Bedingung, ist also eine Datenbankprogrammiersprache und bietet nicht nur Fähigkeiten zu Anfragen und Schemadefinition.

In folgenden vier Formen ist ODQL in JASMINE verfügbar:

- **Interpreted ODQL**: ODQL-Anweisungen können direkt oder indirekt (über eine Datei) in einer vom Interpreter bereitgestellten interaktiven Umgebung ausgeführt werden.
- **Embedded ODQL**: ODQL-Anweisungen werden in C/C++ eingebettet. In dieser Form werden Methoden implementiert. Dabei muss ODQL benutzt werden, um auf die Datenbank zuzugreifen, die Parameter der Methode abzufragen und die Ergebnisse zurückzuliefern. C/C++ kann verwendet werden, um komplexere Berechnungen oder Systemaufrufe durchzuführen. Jeder ODQL-Ausdruck muss durch ein vorangestelltes $-Zeichen gekennzeichnet werden. Das Embedded-ODQL wird dann in den Kode der Hostsprache (i.A. C/C++) übersetzt und dann mit dem restlichen Kode kombiniert.
- **Dynamic ODQL**: Anwendungen können Embedded-ODQL-Anweisungen dynamisch generieren und ausführen.
- **C-API**: ODQL-Anweisungen und –Methoden können über die Funktionen *odbExecODQL()* und *odbExecMethod()* in C- und C++-Programmen ausgeführt werden.

Die Programmiersprache Java erfährt durch persistent Java, Java Proxies, Java Beans und dem Java Media Framework eine besondere Unterstützung:

- Durch die „Java Proxies Facility“ werden aus Klassen Java-Proxy-Klassen und aus Objekten „Java Beans“ erzeugt. Java-Anwendungen können über das J-API die Daten abfragen und manipulieren.
- Mit Hilfe des „Java Persistence Processor“ wird JASMINE zu einem Server für persistente Java-Klassen und Java zu einer Datenbankprogrammiersprache. So wird durch den Generator der Quellcode der Java-Klassen erweitert und für die JASMINE-Datenbank Schemadefinitionen erzeugt. Durch „Persistent Java“ wird ODMG –OQL unterstützt.

[11] *unique* impliziert die Eindeutigkeit für dieses Attribut. Gilt ein Attribut als *mandatory* und ist kein Default-Wert mitangegeben, so sind Nullwerte nicht erlaubt.

- Über das „Java Media Framework“ , das ein J-API zum Abspielen unterschiedlicher Multimedia-Formate darstellt, kann das System als Multimedia-Datenbank für Java-Applikationen benutzt werden.

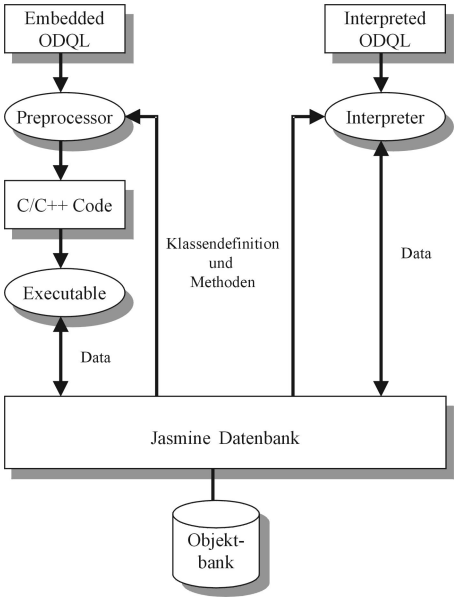

Abbildung 3.9 Architektur der Interpreter- und Präprozessorumgebung

Der Q-Teil von ODQL realisiert die Anfragesprache in JASMINE. Durch die Select-From-Where-Syntax sieht ODQL zunächst wie SQL aus, weist aber doch einige Unterschiede auf. So kann man in ODQL ähnlich wie bei *Informix*, Methoden in den Anfragen aufrufen. ODQL unterstützt relationale und sowohl objekterhaltende (siehe ODQL-Zugriff in Beispiel 3.7) als auch –erzeugende Anfragen:

Beispiel 3.11 Objekterzeugende Anfragestellung

```
Bag<[String n1, String n2]> paare
paare = [p.name, q.name] from Person p, Person q
        where p.town == q.town and
        p.street == q.street and
        p != q;
paare.print();
```

Die Schlüsselwort SELECT ist bei ODQL optional; eine Anfrage kann allein durch die WHERE-Klausel (angewendet auf eine Kollektion) realisiert werden. Wie bei *Informix* können Pfadausdrücke in Anfragen verwendet werden, um von einer Objektmenge direkt auf die jeweiligen Komponenten und Methoden zugreifen zu können.

Leider ist ODQL nicht kompatibel zum ODMG-Standard, obwohl mit Pfadausdrükken und Methodenaufrufen ähnliche Merkmale zur Verfügung stehen. ODQL ist ebenfalls nicht kompatibel zum relationalen SQL, da eine Schachtelung von Anfragen in der WHERE-Klausel nicht möglich ist. Dies wirkt sich jedoch nicht nachteilig aus, da man sich mit ODQL Mengen von Objekten generieren lassen kann, die man dann anstelle der Schachtelung nutzt.

3.2.3 Höhere Konzepte

Einige Konzepte wie Vererbung, Mehrfachvererbung und Overriding habe ich in den vorhergehenden Abschnitten schon angesprochen.

Einkapselung wird leider nicht gewährleistet: Attribute und Methoden sind von außen zugreifbar.

Metadaten sind Informationen über das Objektmodell selbst. In JASMINE sind die Klassendefinitionen selbst Objekte, wodurch es möglich wird, mit ODQL die Struktur des Objektmodells abzufragen und zu verändern. Dies ist vor allem für Entwickler von Zusatzprodukten oder Systemen interessant, die JASMINE als Basis verwenden wollen.

Durch die Möglichkeit bestehende Klassen zu ändern, wird die *Schemaevolution* gewährleistet, ohne die Anwendungen neu übersetzen oder die Datenbank neu organisieren zu müssen. So lassen sich zur Laufzeit:

- zu einer Klasse Attribute und Methoden hinzufügen, ändern und wieder löschen[12]
  - `addProperty(), removeProperty(),`
  - `addProcedure, defineProcedure, removeMethod()`
- zu einem Objekt den Namen der Klasse abfragen
  - `getClass(), getClassName()`
- einer Klasse weitere Oberklassen hinzufügen
  - `addSuper()`

und

- alle Klassen einer Klassenfamilie abfragen
  - `getAllClasses().`

Weiterhin hat man die Möglichkeit, sich den Quellcode und die Beschreibung des Interface einer Methode mit `getMethodSource()` und `getMethodInfo()` ausgeben zu lassen. Die Klassendefinition lässt sich sehr einfach z.B. mit **`Person.print()`** abfragen

3.2.4 Weitere Datenbankkonzepte

Die ***Persistenz*** der Daten wird dadurch gewährleistet, dass erstens die Daten nur mit ODQL manipuliert und zweitens die Objekte nur mit dem *Konstruktor* new() erzeugt und mit dem *Destruktor* delete() gelöscht werden können.

Durch ***Zugriffspfade*** können einzelne Attribute einer Klassen indiziert werden. Dabei können die Attribute entweder Literale (Werte) oder andere Entitäten (Beziehungen) darstellen. Soll ein Index auch für Attribute untergeordneter Klassen gelten, muss er separat mit der **`createIndex`**-Anweisung angelegt werden. Ob und wie der Index zur Steigerung der Performanz zu nutzen ist, wird zur Laufzeit durch das System bestimmt. Dies ist dann auch schon die einzige Art der Optimierung, die durch JASMINE unterstützt wird. Algebraische Optimierungsregeln wie etwa die Ausführung von Selektionen über Konstanten, die vor der eigentlichen Anfrageabarbeitung benutzt werden können, um die Effizienz zu steigern, muss der Benutzer manuell anwenden. Beispiel 3.12 zeigt zwei Anfragen, die von der Semantik her äquivalent

[12] Nach dem Ändern oder Hinzufügen von Methoden, ist allerdings eine Neuübersetzung der Klassenmethoden mit Hilfe von **`compileProcedure`** notwendig.

sind, sich jedoch von der Bearbeitungszeit auf z.B. 10.000 Personen bezogen, stark unterscheiden.

Beispiel 3.12 Manuelle Anfrageoptimierung

```
Anfrage 1:
  Person from Person
         where Person.image.hatMerkmalXYZ() and
               Person.name = "Anton Meier";
besser Anfrage 2:
  Person from Person
         where Person.name = "Anton Meier" and
               Person.image.hatMerkmalXYZ();
```

Transaktionen (TAs) werden in JASMINE durch die Systemklasse *Transaction* (siehe Abbildung 3.8) realisiert. Diese Klasse stellt die Methoden für Start (**`Transaction.start()`**), Ende (**`Transaction.end()`** entspricht dem commit der TA) und Roll-Back (**`Transaction.rollback()`**) der Transaktionen zur Verfügung. Wie aus den Methodenaufrufen schon zu sehen ist, wird vom System keine Instanz der Klasse *Transaction* benötigt, die Steuerung erfolgt strikt durch die Klassenmethoden. Die TAs folgen dabei dem klassischem ACID-Prinzip, sind auf eine Session beschränkt und können leider nicht geschachtelt werden. Lange Transaktionen werden ebenfalls nicht unterstützt. Im Gegenteil, es wird geraten, die Transaktionen so kurz wie möglich zu halten. Neben Lese-Schreib-Transaktionen gibt es noch Nur-Lese-Transaktionen und es wird geraten, wenn möglich erstere Art zu verwenden, da hier keine Sperren gehalten und somit andere TAs nicht blockiert werden.

3.2.5 Klassenbibliotheken

Vordefinierte Klassenbibliotheken für verschiedene Anwendungsbereiche waren in den objektorientierten Datenbanksystemen der ersten Generation kaum verfügbar. Die Anwender wurden damit vertröstet, dass sie sich ja die gewünschten Fähigkeiten selbst schaffen können. Im Gegensatz dazu bietet JASMINE schon seit der ersten Version Klassenbibliotheken, die verschiedene Funktionalitäten implementieren. So beinhaltet JASMINE die umfangreiche Klassenbibliothek *systemCF* für die Basisdatentypen mit einer Vielzahl von Methoden für Standardaufgaben, die SQL-Klassenbibliothek *sqlCF* für den Zugriff auf SQL-Datenbanken und die Klassenbibliothek *mediaCF* für den Multimedia-Bereich.

3.2.5.1 Anbindung an relationale Systeme

Über die SQL-Klassenbibliothek *sqlCF* ist es möglich, auf verschiedene relationale Datenbanken zuzugreifen. Vorraussetzung dafür ist ein relationales System mit einem Enterprise Access Gateway, wie etwa bei OpenIngres, Oracle, Informix oder SQLServer. Der Zugriff auf die aus diesen relationalen Datenbanken stammenden Altdaten erfolgt dann über das OpenIngres Enterprise Access Gateway. Da die SQL-Ausdrücke, die durch diese Klassenbibliothek generiert werden, der OpenSQL-Syntax entsprechen, ist somit eine Portabilität garantiert.

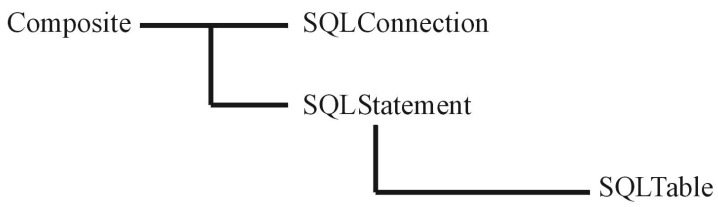

Abbildung 3.10 Die Klassenbibliothek *sqlCF*

Diese drei Klassen der *sqlCF* dienen zum Zugriff auf die Altdaten. Die *sqlCF* benutzt dabei durch Integer-Werte repräsentierte relationale Kontexte, um die Datenbank, die Relationen der Datenbank oder spezielle Tupel der Relationen eindeutig zu referenzieren. Diese Kontexte werden dann als Parameter an die jeweiligen Methoden der *sqlCF* übergeben, um auf die externen Daten zuzugreifen. Dabei wird über die Klasse **SQLConnection** der Verbindungsaufbau zu der migrierenden Datenbank und die Fehlerkontrolle gehandhabt. Die Klasse **SQLStatement** ermöglicht das Ausführen der Anfragen und Abfangen von dort auftretenden Fehlern. Als deren Unterklasse bietet **SQLTable** Methoden für Anfragen, Zugriff und Updates auf entsprechende Attribute der Relationen. Letztere ist dabei besonders interessant, da von ihr abgeleitete Unterklassen die jeweiligen Methoden für den direkten Zugriff auf vorhandene Tabellen realisieren können.

3.2.5.2 Multimedia-Klassenbibliothek

Besondere Aufmerksamkeit hat CA den Multimedia-Aspekten geschenkt. Von der Fachpresse wird JASMINE oft als Multimedia System bezeichnet. So unterstützt JASMINE von vornherein eine große Anzahl von Multimedia-Datentypen wie Text, Bilder, Animationen, verschiedene Audio- und Videoformate, OLE-Objekte und VR-Techniken. In der Klassenbibliothek *mediaCF* (siehe Abbildung 3.11) sind die für den Umgang mit diesen Datentypen benötigten Klassen und Methoden abgelegt.

Die oberen Klassen dieser Hierarchie - *MMedia*, *MMData*, *MMBody*, *MMStorageControl* und *MMFile* – bieten allgemeine Methoden, die für alle Arten von Multimediadaten wie Bild, Ton, Textdokumente und Video benötigt werden. So liegen in der Klasse *MMData* die allgemeinen Funktionen zum Verwalten von BLOBs, die zur Speicherung der großen Datenmengen benötigt werden. Die Speicherung der BLOBs kann nach drei Arten erfolgen:

- **intern:** Die Daten werden durch JASMINE vom externen in die Datenbank importiert. Dabei wird durch JASMINE der BLOB in Objekte der Klasse *MMBody* aufgeteilt und durch Methoden aus *MMData* wieder zusammengesetzt, wenn er durch die Applikation benötigt wird. Die Anwendung selbst bleibt davon unbeeinflusst.
- **kontrolliert:** Hier erfolgt die Speicherung des BLOBs in einer externen Datei, die aber durch JASMINE gesteuert/ kontrolliert wird. Der Datenbankadministrator legt mit dem Befehl `setmmarea` das Verzeichnis fest, in dem die Daten abgelegt werden. Der Dateiname wird jedoch durch JASMINE und über die Klasse *MMStorageControl* verwaltet. Die Vorteile der internen Speicherung wie Speicherung und Updates der Daten unter Transaktions- und Recovery-Kontrolle gelten auch hier, wobei der „lästige“ und unter Umständen zeitaufwendige Overhead des Kopierens der Daten in das System

entfällt. Diese Speicherungsweise bietet sich an, wenn viele große MM-Daten in die Datenbank importiert werden müssen.

- **extern:** Diese Art der Speicherung sollte dann angewendet werden, wenn die MM-Daten durch die Anwendung kontrolliert und verwaltet werden sollen. Hierbei erfolgt die Speicherung in einer externen Datei und das in der Datenbank wird nur eine Referenz auf die Datei abgelegt. Unterklassen von MMData enthalten dabei nur eine Referenz und die Methoden greifen direkt auf die Datei zu.

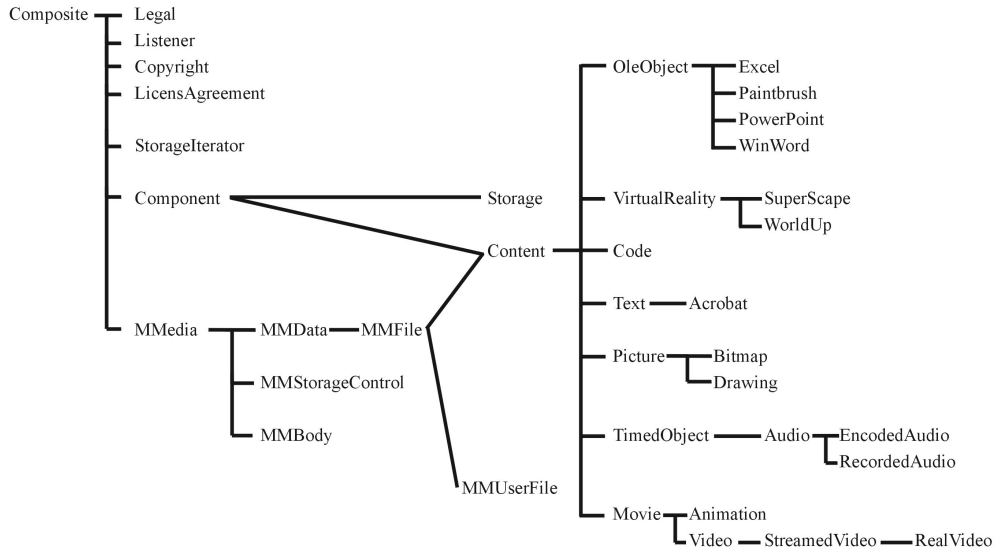

Abbildung 3.11 Die Multimedia-Klassenbibliothek *mediaCF*

Die abstrakte Klasse *MMFile* dient als Basis für alle Multimedia-Klassen, die mit Dateien arbeiten bietet. Sie enthält bspw. Methoden, die Import, Export, Komprimieren und Dekomprimieren realisieren. Diese Grundlagen werden dann z.B. durch die Klasse *Content* genutzt und um zusätzliche Eigenschaften erweitert. In den Unterklassen werden dann speziellere Typen für die MM-Daten bereitgestellt.

3.2.5.3 Partnerprodukte

Ähnlich wie bei *Informix* die DataBlades von Drittanbietern eingebunden werden können, besteht die Möglichkeit, JASMINE um neue Technologien anderer Firmen in Form von Klassenbibliotheken zu erweitern. So gibt es Klassenbibliotheken für

- Geographische Informationssysteme,
- Zeitreihenverarbeitung,
- Finanzfunktionen,
- Text- und Bildsuche,
- Audio- und Video-Streaming,
- Videokonferenzen und
- Virtuelle Realität

um nur einige Technologien zu nennen.

3.2.6 JADS

Der Multimediaschwerpunkt spiegelt sich vor allem in der Entwicklungsumgebung „JASMINE Application Development Studio“ (JADS), ehemals JADE, wieder, die auf der Multimedia-Klassenbibliothek aufbaut. JADS läuft im Gegensatz zum DBMS nur auf Windows-Rechnern (Windows 95 oder Windows NT). Das System ist mit netzwerkfähigen Werkzeugen ausgestattet, mit denen die Klassen und Objekte in der Datenbank durchsucht und editiert erzeugt sowie komplette Multimedia-Anwendungen per „Drag & Drop“ entworfen werden können.

Beim Entwickeln einer Anwendung erstellt der Programmierer sogenannte Szenen, die sich am ehesten mit HTML-Seiten assoziieren lassen. Jede Szene hat dabei bestimmte Eigenschaften und umfasst eine Menge von Objekten. Mögliche Objekte darin sind zum Beispiel Klassen, einzelne Objekte aus der Datenbank, Anfragen an die Datenbank und von JADS bereitgestellte Objekte, wie Buttons, Listboxen und Scrollbars etc.. Zusätzlich können auch ActiveX-Objekte in eine Szene eingefügt werden.

Alle Klassen, Objekte und Anfragen werden im Szenen-Editor-Fenster per Drag-and-Drop platziert und angezeigt. Das Verhalten eines Objekts, d.h. die Reaktion auf Benutzerinteraktion, Datenbankmeldungen und andere Ereignisse, wird anhand eines einfachen Ereignis-Aktion-Konzeptes definiert, für das keine Scripts erforderlich sind. Ein Ereignis kann alles sein, beispielsweise ein Mausklick, das Ablaufen eines Zeitintervalls, der Start oder das Ende von Video- und Animationssequenzen sowie das Empfangen von Nachrichten von anderen Objekten. Objekte können unterschiedliche Arten von Aktionen ausführen, z.B. Grafiken anzeigen, Animations-, Audio- oder Videosequenzen abspielen, die Szene wechseln oder Nachrichten an andere Objekte schicken. Aktionen wie Mausklicks lassen sich so direkt mit den Methoden zum Abspielen von Video- und Audiodaten verbinden.

Bei Objekten einer Klasse mit Methoden sind die Ausführung der serverseitigen Methode und die Reaktion auf das Ergebnis genauso einfach wie das Abspielen eines Videos oder der Start einer Animationssequenz. Aktionen können sich direkt auf Methoden und Eigenschaften des Objekts selbst oder anderer Objekte in der Datenbank beziehen.

Das System bietet außerdem Bewertung von Ausdrücken, bedingte Logik und Schleifen für komplexere Benutzerinteraktionen. Wirklich komplexe Logik wird jedoch nicht auf dem Multimedia-Frontend, sondern in Methoden auf Server-Seite ausgeführt, wobei die Leistungsstärke und Stabilität des objektorientierten Servers und seiner objektorientierten Sprache ODQL bzw. C oder C++ genutzt werden.

Die Ausführungsumgebung von Jasmine bildet die Rahmenstruktur, innerhalb derer die Client-Komponenten der Anwendungen laufen. Sie steuert Animation und Multimedia, koordiniert die Benutzerinteraktion mit internen Aktivitäten, sorgt dafür, dass Multimedia-Ressourcen rechtzeitig aus der Datenbank abgerufen werden, und kommuniziert mit dem Server zwecks Methodenausführung.

Durch die in JADS enthaltenen Werkzeuge wird die Datenbankdefinition und -wartung wesentlich vereinfacht. Auch die Entwicklung von kommerziellen Internet-Multimediaanwendungen (Stichwort: E-Commerce) ist mit JADS einfach, so dass man fast ohne eigentliche Programmierkenntnisse schnell gute Ergebnisse erzielen kann. An der Performance der mit JADS realisierten Anwendungen muss CA aber noch arbeiten.

JADS dient aber nicht nur zur Anwendungsentwicklung sondern auch zur Datenbankadministration. Mit dem **Klassen-Browser** werden Klassen und ihre Methoden definiert und editiert. Er stellt alle Klassen in der Datenbank sowie ihre Vererbungsbeziehungen, Eigenschaften und Methoden grafisch dar. Neben der Erstellung und Änderung von Klassen- und Objektdefinitionen können Methoden mit dem Methodeneditor implementiert, kompiliert und getestet werden. Der **Objekt-Browser** ermöglicht das Überprüfen, Definieren und Editieren von Objekten in der Datenbank. Er bildet alle Eigenschaften eines Objektes in einer grafischen Oberfläche ab, darunter auch Beziehungen zu anderen Objekten. Multimedia-Eigenschaften werden grafisch dargestellt. Der Objekt-Browser beinhaltet eine einfache Drag-and-Drop-Oberfläche zum Speichern von Multimedia-Ressourcen.

3.2.7 Web-Anbindung

Die Anbindung des Datenbanksystems kann bei JASMINE auf zwei Arten erfolgen: zum einen über ein Plug-In, mit dem eine mit JADS entwickelte Anwendung 1:1 in den Browser eingebunden wird (siehe Abbildung 3.6) und zum anderen über das Zusatzprodukt WebLink (siehe Abbildung 3.12), welches ich im folgenden näher vorstellen möchte.

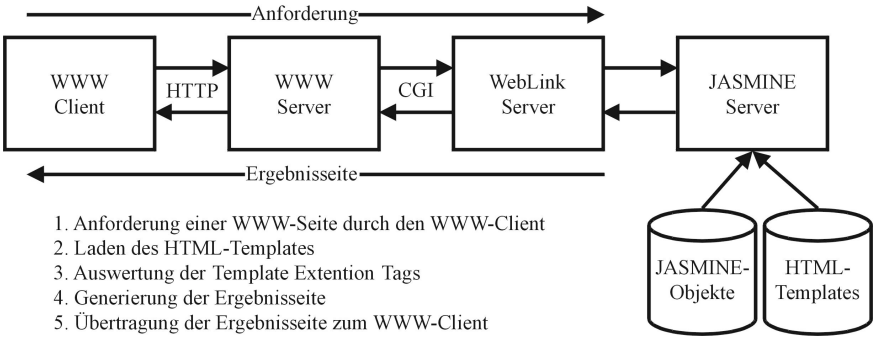

Abbildung 3.12 Datenbankanbindung von JASMINE über WebLink

Wird WebLink auf dem Datenbankserver installiert, so erhält man drei neue Komponenten, um die Datenbank an das Internet anzubinden:

- **Die Klassenbibliothek WebLink:** Sie enthält die Klasse *HTML_Template* und Methoden, Instanzen dieser Klasse anzulegen und zu verwalten. So muss für jede durch die Datenbank dynamisch zu generierende HTML-Seite ein Prototyp (auch als Template bezeichnet) in der Datenbank angelegt werden; dies kann durch die Klassenmethode **`createTemplate()`** oder durch den mitgelieferten *TemplateManager* erfolgen. Dabei enthält ein Template zusätzlich zu den Standard-HTML-Tags noch spezielle „*Template-Extension-Tags*“, durch welche ODQL-Anfragen und -Anweisungen (**`<!DO>,<!VAR>,<!REPLACE>`**), Schleifen (**`<!FOREACH>`**) und Verzweigungen (**`<!IF>,<!ELSE>`**), Fehlerbehandlung (**`<!CATCH>, <!THROW>`**) sowie Links (**`<!URL>`**) zu anderen Templates eingebunden werden.
- **Der WebLink-Server:** Er wird auf demselben Rechner wie der Web-Server gestartet und ermöglich über CGI-Kommandos oder eine spezielle APIs (ISAPI für Microsoft WWW-Server und NSAPI für Netscape Server) den Zugriff auf lokale oder auf entfernte Datenbanken; letztere

bezeichnen Datenbanken, die nicht auf derselben Maschine wie der WWW-Server laufen.

- **CGI Kommandos:** HTML-Templates werden durch das Kommando **`odb-get`** über den WebLink-Server aus der Datenbank geholt. Im Gegensatz zu *Informix* besteht bei JASMINE die Möglichkeit private und allgemeine (public) Sitzungen zu starten. Dazu ist es nötig eine Session mit **`odb-login`** zu starten und mit **`odb-logout`** zu beenden. Über „private sessions“ kann beispielsweise ein Online-Shop mit Login, Warenkorb öffnen, Warenkorb füllen, bezahlen, Logout implementiert werden. Dabei wird eine Session-Id generiert, die solange aktiv bleibt, bis ein Logout erfolgt. Alle Vorgänge und Variablen (auch aus vorhergehenden Templates) bleiben zugreifbar und können bei Bedarf abgefragt oder neu definiert werden. Im Gegensatz dazu „teilen“ sich bei den „public sessions“ mehrere Benutzer ein Template.

3.3 Fazit

Wie aus Kapitel 2 ersichtlich ist es unbedingt notwendig, für die Datenhaltung und -verwaltung DBMSs zu verwenden. Die Umsetzung der entsprechenden objektrelationalen bzw. objektorientierten Konzepte an konkreten Vertretern wie *Informix* und JASMINE habe ich in diesem Kapitel gezeigt. Darauf aufbauend möchte ich jetzt beide Systeme hinsichtlich der für Produktinformationssysteme aufgestellten Kriterien bewerten.

Kategorie	Kriterium	*Informix*	JASMINE
Daten	komplexe Strukturen	⊕	⊕
	Beziehungen	○	⊕
	Multimediafähigkeit	⊕	⊕⊕
	Erweiterbarkeit	⊕	⊕
	Einbindung der Altdaten	⊖	⊕⊕
Zugriffskontrolle	Transaktionen	⊕	⊖
	Mehrbenutzerbetrieb	⊕	⊖
	Sessions	⊖	⊕⊕
Navigation	Suchfunktionalität	○	⊕
	Information – Retrieval	○	○
Präsentation	WWW-Anbindung	⊕	⊕⊕
	Portierbarkeit	⊖⊖	⊖⊖

Tabelle 3.3 Eignung von *Informix* und JASMINE für PISs

Komplexe Strukturen lassen sich in beiden Systemen mit den entsprechenden Typkonstruktoren umsetzen. Bei JASMINE erwies sich die eingeschränkte Verwendbarkeit des Tupelkonstruktors sowie die mangelnde Schachtelungstiefe von Kollektionen als Nachteil. Klasse-Komponentenklasse-Beziehungen werden bei *Informix* (im Gegensatz zu Illustra) nicht mehr unterstützt und müssen durch Schlüssel-Fremdschlüssel simuliert werden.

Beide Systeme bieten für die Unterstützung von Multimediatypen und Funktionen DataBlades bzw. Klassenbibliotheken. JASMINE schneidet hier besser ab, weil es im Gegensatz zu herkömmlichen DBMSs schon im Lieferzustand eine umfangreiche Multimediaklassenbibliothek für verschiedenste Multimediadatentypen enthält. Allerdings darf man nicht übersehen, dass bei beiden nur die grundlegendste Funktionalität, d.h. zur Abspeicherung und Verwaltung, mitgeliefert wird: so enthält Informix das LOB-DataBlade und JASMINE die Multimediaklassenbibliothek, welche allerdings umfangreicher ist.

Etwas umständlich ist bei JASMINE die Erweiterung bestehender Klassen um neue Methoden, da eigentlich diese modifizierten Klassen neu compiliert werden müssten, was aber nicht machbar ist, wenn kein Quelltext mitgeliefert wird. Dadurch kann das Hinzufügen einer Methode, die auch in den Unterklassen verfügbar sein soll, eine wahre Klassenexplosion auslösen. So muss von der existierenden Klasse eine Unterklasse gebildet, dann die Methode hinzugefügt und ebenso mit den Unterklassen verfahren werden. Bei *Informix* wiederum ist das Erstellen von benutzerdefinierten Methoden recht unkomfortabel, da die mitgelieferte Datenbankprogrammiersprache sehr stark eingeschränkt ist und man dadurch bei komplizierteren Berechnungen auf C/C++ ausweichen muss. Umständliche oder fehlende Debug-Möglichkeiten der externen Funktionen machten dann eine Fehlersuche sehr schwer. Auch kam es öfters zu kompletten Serverabstürzen, die damit auch alle anderen Datenbanken des Systems betrafen, wenn z.B. die so erzeugte externe C-Funktion fehlerhaft arbeitete.

Das Transaktionskonzept beschränkt sich bei JASMINE auf **`commit`** und **`rollback`** während *Informix* auch geschachtelte Transaktionen zulässt.

Sehr interessant und für den E-Commerce-Bereich gedacht, ist das Session-Konzept von JASMINE. Bei Informix lässt sich dies nur über programmiertechnische Konstrukte bzw. zusätzliche CGI-Programme verwirklichen. Eine Möglichkeit den Zugriff auf die WWW-Seiten der Datenbank nur für autorisierte Nutzer zu gestatten, war in der untersuchten *Informix*-Version noch nicht vorhanden und musste deshalb durch den verwendeten Web-Server erfolgen.

Durch die Objekt-Navigation über die Klasse-Komponentenklasse-Beziehungen bietet JASMINE eine effiziente Möglichkeit, Beziehungen zwischen verschiedenen Objekten schnell zu verfolgen und ohne dafür Optimierer zu benötigen. Bei Informix muss die Navigation über herkömmliche Schlüssel-Fremdschlüssel dargestellt oder durch entsprechende programmiertechnische Mittel umgesetzt werden.

Bei beiden DBMSs müssen die Information-Retrieval-Techniken über Drittanbieter wie „Excalibur“ für Image- und Video-Retrieval umgesetzt werden. Sie können daher hinsichtlich der Retrieval-Fähigkeiten nicht weiter bewertet werden.

Durch die Möglichkeit, ODQL-Konstrukte direkt bei der Web-Anbindung zu nutzen und sich dadurch die Einarbeitung in neue Konzepte, wie beim Web-DataBlade von Informix, zu sparen, schneidet JASMINE besser ab. Mangelhaft ist aber, dass es für die HTML-Templates keine Projektverwaltung gibt und deshalb alle Templates der Datenbank ungeordnet abgelegt werden.

Portierbarkeit auf externe Medien ist bei beiden Systemen nicht möglich, es sei denn man vertreibt das komplette DBMS mit an die Benutzer.

4. Realisierung und Umsetzung

Im Rahmen dieser Studienarbeit habe ich am Zentrum für Graphische Datenverarbeitung Rostock e.V. (ZGDV) einen Prototyp zur Realisierung eines Produktinformationssystems entwickelt. In diesem Kapitel möchte ich nun erläutern, wie die in Kapitel 2 vorgestellten Konzepte in diesem Prototyp umgesetzt wurden und welche Schwierigkeiten auftraten.

4.1 ProInfo-MV

Abbildung 4.1 Startbildschirm von „ProInfo-MV“

4.1.1 Anwendungsszenario

Ursprünglich war die Realisierung eines Informationssystems für ein imaginäres Autohaus geplant. Leider war es mir nicht möglich, entsprechendes Datenmaterial sowohl über verschiedenen Autotypen als auch multimediale Inhalte zu bekommen. Auch Anfragen an Autokonzerne wie Volkswagen führten zu keinem brauchbaren Material. In Zusammenarbeit mit dem ZGDV und der Industrie- und Handelskammer Rostock (IHK) konnten im Rahmen des Verbundforschungsprojektes „Business-MV“ entsprechende Daten über Firmen und Institutionen aus dem Raum Mecklenburg-Vorpommern gewonnen werden, so dass ich diese in den Prototyp eines Produktinformationssystem integrieren konnte. Inhaltlich, das heißt vom Datenbestand her, ist „ProInfo-MV“[13] also auf die IHK ausgerichtet. Je nach Anwendungsfall ist es aber von der Grundfunktionalität her ohne Schwierigkeiten bzw. großen Programmieraufwand auch auf andere Datenbestände anpassbar.

[13] „ProInfo-MV“ wurde gemeinsam im Projektteam „Business-MV“ konzipiert und entwikkelt.

4.1.2 Realisierung

Der Schwerpunkt bei der Entwicklung des Informationssystems lag vor allem auf der Datenbankanbindung an das WWW. Einerseits kann der Benutzer immer auf den aktuellen Datenbestand zugreifen und andererseits wird der Administrations- und Wartungsaufwand erheblich reduziert.

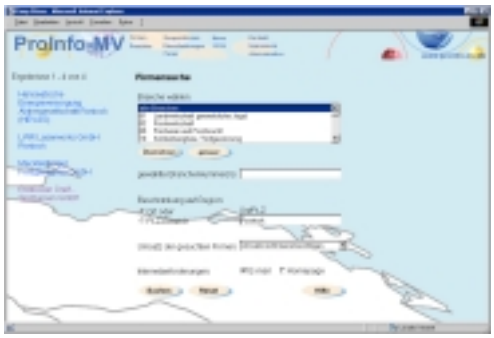

Abbildung 4.2 Beispielanfrage für alle Firmen aus Rostock und Umgebung mit E-Mail

Abbildung 4.3 Darstellung der Firmeninformationen

Dem Anwender stehen zwei Möglichkeiten zur Verfügung, auf den Datenbestand zuzugreifen. Er kann sowohl über die Firmeninformationen (siehe Abbildung 4.2 und Abbildung 4.3) als auch über die Produktdaten suchen. Realisiert wurde also letztendlich der im Kapitel 2 vorgestellte selektive Zugriff über die Formulierung von Suchanfragen. Als Ansatz für den explorativen Zugriff könnte man den Zuständigkeitsbereich entweder in Kreise (Raum Rostock, Raum Schwerin, Raum Greifswald etc.) oder direkt in die Branchenbereiche und –unterbereiche aufteilen und diese Aufteilung als Ausgangspunkt für die Navigation benutzen. Da aber ausgehend vom Anwendungsszenario dieses Anwenderverhalten nicht zu erwarten ist[14] und außerdem durch den selektiven Zugriff dieselben Ergebnisse effizienter erlangt werden können, wurde diese Navigationsart nicht implementiert.

Die Suchmaschine wurde mit dem Excalibur Text DataBlade realisiert, welches die entsprechenden Datentypen und Methoden bereitstellt, um effizient in Dokumenten zu suchen. Durch dieses DataBlade wird neben exakter und unscharfer Suche auch die Suche über Phrasen sowie Synonyme unterstützt. Der Suchprozess läuft dabei in einem vom Datenbankserver kontrollierten virtuellen Prozess. Der Aufruf des Text DataBlades erfolgt dabei über die Methode **`etx_contains()`**, die dann über einen speziellen Textindex den übergebenen Text sucht, ein Ranking vornimmt und die beispielsweise über eine Selekt-Anweisung referenzierten Attribute für die gefundenen Dokumente zurückliefert.

Beispiel 4.1 Erzeugen des Suchindex

```
create index ftext_idx on firma (f_text etx_lvarc_ops)
     using etx (WORD_SUPPORT = 'EXACT',
     PHRASE_SUPPORT = 'MAXIMUM', CHAR_SET = 'ISO')
     in sbspace1;
```

[14] Eine Firma, die Kooperationspartner sucht, wird nicht erst durch den Katalog „blättern“, sondern hat bestimmte Vorstellungen und Kriterien, die zu erfüllen sind.

Beispiel 4.1 zeigt das Anlegen des Index. Dabei wird der Index **ftext_idx** auf dem Attribut **f_text** der Relation **firma** erzeugt. Als Indexparameter wurden folgende Parameter übergeben:

- PHRASE_SUPPORT mit maximaler Größe, um eine exakte bzw. hohe Trefferwahrscheinlichkeit bei der Suche nach Phrasen bzw. Teilsätzen zu gewährleisten,
- WORD_SUPPORT mit der Einstellung für exakte Schlüsselwortsuche

und

- als verwendeter Zeichensatz wird ISO-konform gewählt.

Mögliche Suchanfragen für boolesche (A) -, Phrasen- (B) oder auch Umgebungssuche (C) zeigt Beispiel 4.2.

Beispiel 4.2 mögliche Suchanfragen

```
(A) select f_name, f_text from firma
        where etx_contains(f_text,row('Entwicklung & Vertrieb',
          'SEARCH_TYPE = BOOLEAN_SEARCH'));

(B) select f_name, f_text from firma
        where etx_contains(f_text,row('Entwicklung und Vertrieb',
          'SEARCH_TYPE = PHRASE_EXACT'));

(C) select f_name, f_text from firma
        where etx_contains(f_text,row('Entwicklung Vertrieb',
          'SEARCH_TYPE=PROX_SEARCH(5)'));
```

Im Abschnitt zum Text-Retrieval werde ich diese Einstellungen und auch den genauen Ablauf des Retrieval-Prozesses ausführlicher vorstellen.

Als Mehrwertdienst wurde eine Art Forum für Kooperationsgesuche im In- und Ausland umgesetzt, durch das Firmen die Möglichkeit erhalten, ihre Kooperationsbereitschaft anderen Firmen bekannt zu machen bzw. sich darüber informieren können, welche Firmen in welcher Art und Weise als Kooperationspartner zur Verfügung stehen. „ProInfo-MV“ ist so aufgebaut, dass Integrationsmöglichkeiten für weitere Mehrwertdienste wie zum Beispiel ein „News“-Dienst oder auch eine Dienstleistungsbörse gegeben sind.

Als Datenbankmanagementsystem wurde von Informix der Universal Server Version 9.14 gewählt. Weiter wurden folgende Datablades benutzt:

- Informix Web DataBlade Module Version 3.31
- Informix LOB DataBlade Module Version 1.20
- Excalibur Text Search DataBlade Module Version 1.10

Mit dem Applikation Page Builder (APB) des Web-Datablades wurden die einzelnen HTML-Seiten, die neben JavaScript-Bereichen auch die Datablade-Erweiterungen enthalten, in die Datenbank eingebracht. Unschön war dabei die Tatsache, dass der APB nur über einen Browser wie Netscape oder Internet Explorer verwendet werden konnte. Es gab zwar auch eine Windows NT Variante, den „Data Director for Web“, der zum Realisierungszeitpunkt aber noch zu instabil lief und deshalb nicht weiter in Betracht gezogen wurde. Positiv ist vor allem die Unterteilungsmöglichkeit in verschiedene Projekte und verschiedene Autorisationslevel, wobei der Autorisierungsmechanismus durch den verwendeten WWW-Server umgesetzt werden muss. Dies gilt jedenfalls für den WWW-Server von Apache. Für kommerzielle Produkte wie den WWW-Server von Netscape gibt es die Schnittstelle NSAPI, die eine ähnliche

Funktionalität wie der WebDriver bereitstellt, allerdings speziell auf den Netscape-Server zugeschnitten ist.

Gründe, die letztendlich für die Wahl von *Informix* als DBMS für die Realisierung sprachen, waren die Robustheit und Performance, welche bei JASMINE leider nicht so gegeben waren, da ich zum Zeitpunkt des Entwurfs nur auf eine Betaversion, die unter Windows NT sehr instabil lief, zurückgreifen konnte. Nachteilig für JASMINE ist auch das unzureichende Transaktionskonzept, womit ich die fehlende Unterstützung geschachtelter Transaktionen wie in Kapitel 3 beschrieben meine. Letztendlich war auch die Server-Plattform entscheidend, da der WWW-Server auf einer Unix-Maschine lief und dementsprechend auch das Datenbanksystem unter Unix laufen sollte.

4.2 Content-Based-Information-Retrieval

Ursprünglich hatte ich im Rahmen dieser Studienarbeit geplant, die Information-Retrieval-Fähigkeiten innerhalb des Produktinformationssystems umzusetzen. Da das aufgrund der oben beschriebenen fehlenden Datengrundlage nicht möglich war, habe ich zwei Prototypen realisiert, die der Evaluierung des Text- bzw. Image-Retrievals dienen.

4.2.1 Image-Retrieval

Abbildung 4.4 Prototyp für die Evaluierung der Image-Retrieval-Fähigkeit

Die Retrieval-Fähigkeiten wurden in diesem Prototyp durch das Excalibur Image Datablade Module Version 1.1 bereitgestellt. In Abbildung 4.5 werden die wesentlichen Charakteristiken des Image-Retrieval-Prozesses dargestellt. Zunächst werden für die in der Informix-Datenbank abgelegten Bilder[15] über die System-Funktion *feature extractor* die Charakteristiken der Bilder binärcodiert in Form eines *feature vector* extrahiert. Diese *feature*-Vektoren werden indiziert (*frnet*). Beim Start des Such-Prozesses wurden für das gegebene Bild (*Search Object*) der *feature vector*

[15] In der Test-Datenbank sind ca. 4000 unterschiedliche Farbbilder abgelegt

und über den *frnet*-Index alle ähnlichen Vektoren (Pattern Matching) ermittelt, über die dann der Zugriff auf die Tabellen-Spalten mit den enthaltenen, ähnlichen Bildern erfolgt. Der Zugriff kann direkt auf der SQL-Ebene über die *Resembles*-Funktion innerhalb der *where*-Klausel erfolgen. Zusätzlich besteht die Möglichkeit, einen Rank-Faktor als SLV-Variable (*Statement Local Variable*) anzugeben.

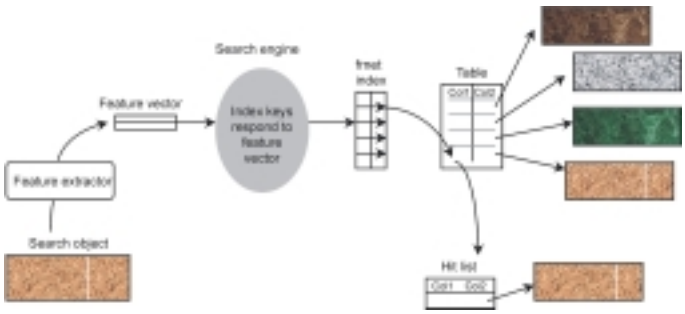

Abbildung 4.5 Image-Retrieval-Prozess

Beispiel 4.3 SQL-Syntax (A - Schema, B - Beispielanfrage)

```
(A) SELECT *, rank FROM table_name,
      WHERE RESEMBLES (indexed_column_name,
      row('feature_extractor(feature_extractor_params)'),
          search_object, search_options),
      rank #REAL);

(B) SELECT img_id, rank FROM imagetable
      WHERE RESEMBLES (img_big,
      row( 'IpdColorImageFE(RETINA_WIDTH=32,RETINA_HEIGHT=32)',
      row(row(row('IFX_BLOB','$input[16]',NULL::lvarchar),
        NULL::lvarchar),'','',0,0,')),
      rank #REAL);
```

Abbildung 4.6 gerankte Retrieval-Ergebnisse

[16] Durch die Variable **`$input`** wird der entsprechende LOB-Handle auf das zu suchende Bild, in diesem Fall das zweite Bild von links oben ausgehend, übergeben. Die Ergebnisse sind der Reihe nach von links oben nach rechts unten gerankt.

4.3 Text-Retrieval

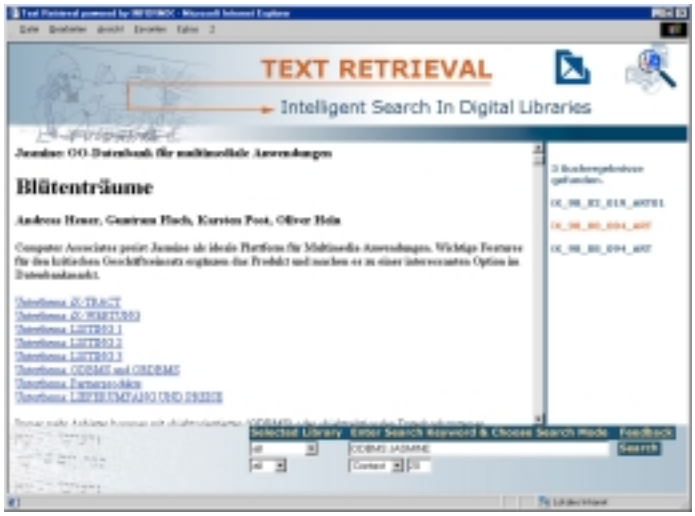

Abbildung 4.7 Web-Interface des Image-Retrieval-Prototypen

In einem gesonderten Prototyp, der im Rahmen des MoVi-Projektes am ZGDV entstand, habe ich in einem digitalen Archiv zunächst die Artikel-Sammlungen zweier Zeitschriften (ca. 20000 Dokumente) verwendet. Mit dem entwickelten Web-Interface (siehe Abbildung 4.7) kann interaktiv die Volltextsuche über die im Datenbanksystem vorliegenden Dokumente erfolgen.

Das verwendete Excalibur Text DataBlade ist ein System für die Volltextsuche, das aus einem Textsuch-Server besteht, der durch den Operator *etx_contains* gestartet wird. Bevor die Suche eingeleitet werden kann, muss analog zum Image-DataBlade ein Index für die Spalte, die den zu durchsuchenden Text enthält, erstellt werden. Mit diesem Index werden alle Wörter des Dokuments, die nicht explizit durch sogenannte Stopwortlisten von der Suche ausgeschlossen sind, indiziert, damit die Suche performant durchgeführt werden kann.

Das DataBlade-Modul besteht aus der *etx*-Zugriffsmethode, dem *etx_contains*-Operator, entsprechenden Filtern für ASCII, HTML, Word, Excel, PowerPoint, oder auch PDF und enthält außerdem noch spezielle Systemroutinen (Definition von Synonym-, Stoppwort-Listen, oder eigener Zeichensätze).

Durch die CREATE INDEX-Anweisung wird der etx-Index erstellt. Dabei kann während der Index-Erstellung die unterstützte Suchfunktionalität über zusätzliche Parameter wie *WORD_SUPPORT, PHRASE_SUPPORT, STOPWORD_LIST* und *CHAR_SET* festgelegt werden. Der *etx_contains*-Operator ermöglicht das Durchsuchen der Werte einer so indizierten Spalte und muss in der WHERE-Klausel explizit mit angegeben werden, um die Suche zu starten (siehe Beispiel 4.2). Durch den *SEARCH_TYPE*-Parameter wird dabei dem *etx_contains*-Operator die entsprechende Suchart[17] zugewiesen: Stichwortsuche (**`WORD`**), Logische Suche (**`BOOLEAN_SEARCH`**), Teilsatzsuche (**`PHRASE_EXACT`**) bzw. Näherungssuche[18] (**`PROX_SEARCH`**).

[17] In den Klammern ist der der Suchart entsprechende Wert angegeben.

[18] Bei der Näherungssuche muss zusätzlich noch die maximale Wortanzahl der Suchbegriffe im Suchtext mit angegeben werden: z.B. **`PROX_SEARCH(5)`**.

Bezüglich der Sucharten sind allerdings einige Einschränkungen feststellbar, da eine Stammformreduktion oder die Verwendung von Wildcards fehlt. Des weiteren beziehen sich sämtliche Tuning-Parameter einer Retrieval-Anfrage stets auf das gesamte Suchkriterium, so dass eine spezifische Formulierung für einzelne Terme nur durch eine getrennte Anfrage realisierbar ist.

Performance-Untersuchungen wurden im Rahmen dieser Arbeit nicht durchgeführt. Signifikant in diesem Kontext wären insbesondere Untersuchungen zur Indexerstellung, die Indexgröße (Speicheranforderung), die Indexaktualisierung und das Verhalten bei Updates. Eine umfassende, adäquate Untersuchung erscheint äußerst schwierig, da die jeweiligen Systeme eine Vielzahl von Tuning-Möglichkeiten auf logischer, physikalischer oder Systemparameter-Ebene aufweisen (z.B. hinsichtlich der Parallelisierung von Anfragen, Indexfragmentierung), so dass nur mit Einschränkungen vergleichbare Messergebnisse zu erzielen sind.

4.4 Fazit

In Bezug auf die Erweiterungsmöglichkeiten hinsichtlich neuer Datentypen, insbesondere Multimediadatentypen, gibt es keine signifikanten Unterschiede zwischen *Informix* und JASMINE. Beide verwenden Partnerprodukte, um entsprechende Such-Funktionalität in das DBMS zu integrieren. Bei beiden Systemen bleibt der Suchmechanismus eine „Blackbox“, d.h. man hat von außen keine Eingriffs- bzw. Änderungsmöglichkeiten. Durch die objektorientierte Kapselung bei JASMINE, kann die Erweiterung bestehender Klassen um neue Methoden und speziellere Retrieval-Funktionalität eine wahre „Klassenexplosion“ (siehe Kapitel 3) auslösen, während bei Informix dem Datenbanksystem letztendlich „nur“ eine neue UDR hinzugefügt wird und die Bindung an den entsprechenden Datentyp über den Parameter der Funktion realisiert wird. Andererseits ist es durch die Objektorientierung bei JASMINE einfacher, komplexe Geschäftsvorgänge und Objekte der realen Umwelt (siehe Kapitel 3, Modellierung von Personen und Studenten) umzusetzen. Ein wesentlicher Pluspunkt ist hier auch die enge Koppelung von ODQL und C. Bei Informix erwies es sich mitunter doch als sehr kompliziert, Methoden in C zu implementieren. Ursache dafür waren unter anderem die unzureichenden Debug-Möglichkeiten und die ungenügende Ausnahmebehandlung.

5. Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, verschiedene Strategien zur Entwicklung von datenbankbasierten Multimediaanwendungen zu untersuchen. Dabei wurden sowohl objekt-relationale als auch objektorientierte Ansätze evaluiert.

Zu Beginn dieser Studienarbeit habe ich grundlegende bzw. allgemeingültige Kriterien für Produktinformationssysteme aufgestellt. Dabei hat sich gezeigt, dass ein entsprechendes objekt-relationales bzw. objektorientiertes Datenbankmanagementsystem unbedingt notwendig ist, um den zugrundeliegenden Datenbestand effizient zu verwalten. Ferner muss das verwendete DBMS und damit auch das Produktinformationssystem so ausgelegt sein, dass komplexe Datenstrukturen und Beziehungen zwischen den einzelnen Daten verwaltet können. Ein wesentlicher Schwerpunkt liegt dabei auch auf der Einbettung von Multimediadaten, deren Speicherung und die damit verbundenen Zugriffs- und Retrieval-Möglichkeiten. Weiterhin muss ein PIS erweiterbar für neue Datenstrukturen, d.h. neue Typen und Methoden, bleiben und sollte neben einer entsprechenden Transaktionskontrolle auch einen stabilen Mehrbenutzerbetrieb sowie die Möglichkeit der WWW-Anbindung garantieren. Die WWW-Anbindung kann einerseits durch externe HTML/ CGI- Anwendungen als auch durch interne Speicherung, Verwaltung und dynamische Generierung der HTML-Inhalte erfolgen.

Hinsichtlich dieser Gesichtspunkte habe ich zwei Datenbankmanagementsysteme, *Informix* als Vertreter der ORDBMSs und JASMINE für die OODBMSs, betrachtet und näher vorgestellt. Dabei hat sich gezeigt, dass sich sowohl die ORDBMSs als auch die OODBMSs gleichermaßen gut für die Realisierung von Produktinformationssystemen mit speziellem Multimediaschwerpunkt eignen. Infolge der reinen Objektorientierung schneiden die Objektorientierten Systeme jedoch etwas „besser“ ab, da es mit ihnen einfacher ist, komplexe Modelle und Geschäftsvorfälle umzusetzen. Leider fristen sie zur Zeit noch eine Art „Nischendasein“. Vor allen Dingen in Sachen Performance und Stabilität muss noch einiges getan werden.

Durch die Zusammenarbeit mit Partnerfirmen wie Excalibur, die entsprechende DataBlades bzw. Klassenbibliotheken für die Verwaltung, die Speicherung, die Indexierung und sowie den Zugriff auf die verschiedensten multimedialen Datentypen bieten, ist es ohne Probleme möglich, Multimediainformationen effizient durch das entsprechende DBMS zu verwalten und in das Produktinformationssystem einzubinden. Allerdings besteht noch ein großer Bedarf im Bereich Content-Based-Information-Retrieval über diese „neuen“ Datentypen. Gerade bei der Realisierung des Prototypen, hat sich gezeigt, dass hier noch nicht alle Möglichkeiten ausgeschöpft sind. So reicht es heutzutage nicht aus, die Suchfunktionalität etwa der Art „gib mir alle Texte/Bilder die Merkmal XYZ enthalten“ zu gewährleisten, sondern hier sind tiefergehende neue Ansätze wie „Relevance-Feedback“ gefragt, d.h. der Benutzer kennzeichnet die für ihn wichtigen Suchergebnisse und das System gestaltet dahingehend die Anfrage um, so dass möglichst viele dieser Ergebnisse zurückgeliefert werden. Auch der Bereich XML könnte durch die Strukturierung der Informationen neue Ansätze in Sachen Retrieval-Fähigkeit geben.

6. Literaturverzeichnis

[ABD+89] M. P. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik. The Object-Oriented Database System Manifesto. in: W. Kim, J.-M. Nicolas, S. Nishio (Hrsg.): Proc. of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD'98), Kyoto Japan. Elsevier Science Publishers, Dezember 1989

[Cat97] R. G. G. Cattel. The Object Database Standard: ODMG 2.0. Morgan Kaufmann Publishers 1997

[Deu96] U. M. Deubl. Dokumentenverwaltung am Beispiel Elektronischer Produktkataloge. Diplomarbeit, Universität Erlangen-Nürnberg FB Informatik, 1996

[Hei98] J. Heinrich. Studie über das Einsatzpotential objekt-relationaler DBMS und Evaluation kommerzieller Produkte. Diplomarbeit, Universität Kaiserslautern FB Informatik, 1998

[Heu97a] A. Heuer. Objektorientierte Datenbanken – Konzepte, Modelle, Standards und Systeme. 2., aktualisierte Auflage, Addison-Wesley-Longman, 1997

[Heu97b] A. Heuer, G. Saake, Datenbanken – Konzepte und Sprachen, International Thomson Publishing, 1997 (erster korrigierter Nachdruck)

[Heu98] A. Heuer. Objektorientierte und objekt-relationale Datenbanksysteme – Kriterien und Vergleichsmerkmale. in OBJEKTspectrum 1/98 S.73

[HFP+98] A. Heuer, G. Flach, K. Post, O. Hein. „Blütenträume" - Jasmine: OO-Datenbank für multimediale Anwendungen. in: iX 8/98 S.84

[IdocA] Produktdokumentation. Extending INFORMIX - Universal Server: Data Types, Informix Press

[IdocB] Produktdokumentation. Informix Web DataBlade Module, Informix Press

[IdocC] Produktdokumentation. Excalibur Text Search DataBlade Module, Informix Press

[IdocD] Produktdokumentation. Excalibur Image DataBlade Module, Informix Press

[JasDocA] Technische Übersicht Jasmine. Computer Associates, 1997

[JasDocB] Jasmine Online Hilfe, Produktdokumentation

[KB95] S. Khoshafian, A. B. Baker. Multimedia and Imaging Databases. Morgan Kaufmann Publishers, CA, 1996

[LF98a] O. Linssen, M. Flygare. Das OODBMS Jasmine (1): Grundlagen und Architektur. in: it FOKUS 10/98 S.52

[LF98b] O. Linssen, M. Flygare. Das OODBMS Jasmine (2): Verwaltung von Objekt-Datenbanken. in: it FOKUS 11/98 S.56

[LF98c] O. Linssen, M. Flygare. Das OODBMS Jasmine (3): Die Abfragesprache ODQL. in: it FOKUS 12/98 S.58

[LF99] O. Linssen, M. Flygare. Das OODBMS Jasmine (4): Anwendungsentwicklung mit Jasmine. in: it FOKUS 1/99 S.32

[Nan98] B. Nance. Computer Associates zieht in den Blumenkrieg. in: BYTE Mai/98 S.70

[Post98] K. Post. Nutzung von Legacy-Anwendungen in objektorientierten Entwicklungsumgebungen am Beispiel von Oracle und Jasmine. Diplomarbeit, Universität Rostock, FB Informatik, 1998

[SM96] M. Stonebraker, D. Moore. Object-Relational DBMSs – The Next Great Wave. Morgan Kaufmann Publishers, CA, 1996

[Sub98] V. S. Subrahmanian. Principles of Multimedia Database Systems. Morgan Kaufmann Publishers, CA, 1998

[Ver97] R. Vermeulen. Objekt-relationale Datenbanken: Oracle und Informix stellen sich der Herausforderung. in: OBJEKTspektrum 6/97 S.68

[Vos00] G. Vossen. Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. 4., korrigierte und ergänzte Auflage, Oldenbourg Verlag, 2000

7. Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis