Medienspezifische Datentypen für objekt-relationale DBMS: Abstraktionen und Konzepte*

Ulrich Marder

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
67653 Kaiserslautern
marder@informatik.uni-kl.de

Kurzfassung. Medienspezifische Daten bestehen aus zwei Arten von Daten: den (unformatierten) Rohdaten und den Metadaten, welche benötigt werden, um die Rohdaten zu interpretieren und um nach Medienobjekten suchen zu können. Die aktuelle Multimedia-Datenbank-Forschung beschäftigt sich sehr intensiv mit Modellen und Anfragesprachen für die Metadaten. Teilweise haben diese Entwicklungen auch schon den Weg in die Praxis gefunden, wie ein Blick auf eine Auswahl von Multimedia-Erweiterungen für heutige objekt-relationale DBMS (ORDBMS) zeigt. Schwerpunkte dieses Aufsatzes sind hingegen Abstraktionen und Konzepte für die Verwaltung und das Arbeiten mit Medienobjekt*rohdaten*. Neben bekannten Abstraktionen wird eine neue, die sog. *Transformationsunabhängigkeit*, eingeführt und erläutert. Diese Abstraktionen werden durch ein neues Konzept für medienspezifische Datentypen realisiert, das es mittels eines speziellen Verarbeitungsmodells und einer darauf abgestimmten Anfragesprache den Anwendungen erlaubt, mit sog. *virtuellen Medienobjekten* zu arbeiten. Die Präsentation wird durch einen Vorschlag, wie dieses Konzept in ein heutiges ORDBMS zu integrieren ist, abgerundet.

1 Einführung

Traditionelle relationale Datenbankverwaltungssysteme (RDBMS) sind auf die Verwaltung formatierter Daten spezialisiert. Sie bieten für diesen Zweck geeignete numerische und alphanumerische Datentypen und darauf abgestimmte Operationen an. Die Verwaltung von Mediendaten wie Bilder oder Videos war hingegen nie eine Stärke solcher Systeme. Viele bieten zwar für unformatierte Daten den generischen Datentyp *Binary Large Object* (BLOB) an, jedoch keine medienspezifischen Operationen, die (integriert in die Anfragesprache SQL) auf derartige Objekte anwendbar sind. Die Anwendungen müssen das dem DBMS unbekannte Datenformat eines BLOBs interpretieren können, um über-

* Diese Arbeit wurde von der Deutschen Forschungsgemeinschaft (DFG) als Teil des Sonderforschungsbereichs (SFB) 501 „Entwicklung großer Systeme mit generischen Methoden“ unterstützt.

haupt etwas damit anfangen zu können. Kurz gesagt: BLOBs sind im Prinzip vom DBMS verwaltete Dateien. Infolgedessen wird dieses Konzept auch nur wenig genutzt, d. h., der weitaus größte Teil multimedialer Daten wird bis heute in gewöhnlichen Dateisystemen und folglich nicht in DBMS verwaltet.

Betrachtet man heutige Multimediasysteme und –anwendungen, so drängt sich schnell der Eindruck auf, dass man sich meist recht bereitwillig mit der Datei- (bzw. BLOB-) Abstraktion arrangiert hat. Beispielsweise verhalten sich die meisten Medienserver entweder wie DBMS mit BLOBs, d. h., sie kümmern sich nicht um die Datenformate und bieten keinerlei medienspezifische Funktionen an[1], oder sie spezialisieren sich auf bestimmte Datenformate (z. B. MPEG), um darauf spezielle Operationen bereitzustellen, z. B. typische VCR-Funktionen wie Wiedergabe oder schneller Vorlauf[2]. Im ersten Fall müssen die Anwendungen jegliche medienspezifische Funktionalität selbst realisieren, im zweiten müssen sie sich auf das vom Server verwendete, meist nur für einen ganz bestimmten Anwendungsfall optimale Datenformat einstellen.

Durch die rasante Entwicklung des Internets in den letzten Jahren sind verteilte Multimedia-Anwendungen in zunehmend größere Dimensionen vorgestoßen. Beispiele dafür sind die zahlreichen Projekte zur Errichtung digitaler Bibliotheken sowie multimedialer Teleteaching-Systeme. Charakteristisch für solche Systeme ist, dass eine große Zahl von Nutzern mit teils sehr unterschiedlichen Interessen und (technischen) Möglichkeiten auf einen globalen multimedialen Datenbestand zugreift und damit arbeitet. Je nach Interesse des Anwenders müssen dabei sehr verschiedenartige Anwendungsprogramme, z. B. Browser, Viewer, Editoren, Indexierer oder Analysierer, eingesetzt werden, woraus folgt, dass die zur Verwaltung der Mediendaten eingesetzten Server möglichst anwendungsneutral sein sollten.

Anwendungsneutralität wird i. d. R. durch Datenabstraktion erzielt. Da dies schon immer eine Domäne von DBMS war, wird für Multimedia-Systeme (wie die oben genannten) der Einsatz von DBMS mit erweiterbaren Typsystemen zunehmend interessant. Diese bieten Datenabstraktion nicht nur für „eingebaute“ Datentypen, sondern auch für beliebige benutzerdefinierte Datentypen. Es stellt sich jedoch die Frage, wie weit man mit diesen Mitteln die Entwicklung medienspezifischer abstrakter Datentypen treiben kann und welche DBMS dafür besonders gut geeignet sind. Sowohl objektorientierte [23] als auch objekt-relationale DBMS [24] sind denkbare Kandidaten.

Dieses Papier konzentriert sich auf den ersten Teil der Frage, d. h. den Entwurf medienspezifischer abstrakter Datentypen (MADT) mit Hilfe heutiger erweiterbarer DBMS. Ohne die Entscheidung, welche DBMS-Technologie hierfür am besten geeignet ist, vorwegzunehmen, wird dabei ein objekt-relationales DBMS (ORDBMS) zugrunde gelegt. Kapitel 2 führt daher die wichtigsten Eigenschaften von ORDBMS kurz ein und beschreibt, was diese Systeme für die Verwaltung multimedialer Daten attraktiv macht und wie sie bereits heute in Praxis und Forschung dafür genutzt werden. In Kapitel 3 werden dann die von MADTs zu erfüllenden Abstraktionen technologieunabhängig spezifiziert. Kapitel 4 führt das VirtualMedia-Konzept ein, nach dem MADTs mit den gewünschten Eigenschaften in ein ORDBMS integriert werden können, wie anschließend in Kapitel 5 gezeigt wird. Kapitel 6 beschließt den Aufsatz mit einer Zusammenfassung und einem Ausblick auf zukünftige Forschungsschwerpunkte.

---

[1] In diese Kategorie fallen z. B. Datei-, Ftp- und Http-Server.

[2] Z. B. Video-on-Demand (VoD) Server

2 ORDBMS und Multimedia

2.1 Motivation

Die Hersteller von ORDBMS sehen in ihren Systemen die jeweiligen Stärken von relationalen und objektorientierten DBMS miteinander vereint. Als ein ganz wesentlicher Beitrag aus der ODBMS-Welt wird hierbei häufig die (verglichen mit relationalen DBMS) gute Unterstützung der sog. VITA-Datentypen (Video, Image, Text, Audio) hervorgehoben [10, 19].

Dieser Schritt hin zu einer integrierten Verwaltung „klassischer" formatierter *und* unformatierter Daten erleichtert insbesondere die Entwicklung multimedialer Informationssysteme. Hier müssen oftmals eine Vielzahl unterschiedlicher Daten gemeinsam verwaltet und miteinander verknüpft werden. In einer Teleteaching-Anwendung sind das z. B. Lehrtexte (üblicherweise als Hypertext organisiert), Bilder, Lehrfilme, interaktive Fragebögen, Übungen, Klausuren, Schülerdaten (Personaldaten, belegte Kurse, Noten), Nachschlagewerke, Diskussionsbretter usw. Weitere Anwendungsbeispiele sind digitale Bibliotheken, Workflow-Management-Systeme, Dokumentenmanagementsysteme, Erfahrungsdatenbanken (*Experience Factory*) und Datenbanken für die Verbrechensbekämpfung, um nur einige zu nennen.

Die entscheidenden Merkmale, die ORDBMS als Grundlage solcher Anwendungen attraktiv machen, sind:

- **Solide relationale Basis:** Praktisch alle kommerziellen ORDBMS sind Weiterentwicklungen bewährter relationaler DBMS und verfügen über deren Eigenschaften, z. B. Robustheit, Skalierbarkeit, Performanz, Fehlertoleranz, Anfragesprache SQL und Mehrbenutzerfähigkeit. Daraus folgt auch, dass bereits bestehende Anwendungen für RDBMS leicht auf ORDBMS migriert werden können, um sie beispielsweise mit multimedialen Aspekten zu erweitern.
- **Benutzerdefinierte Datentypen (UDT):** Es können neue Datentypen aufbauend auf schon existierenden Typen (z. B. Integer, String, BLOB) gebildet werden. Ein solcher Datentyp kann auch von einem anderen abgeleitet werden, so dass Typhierarchien entstehen. Mittels Typkonstruktoren lassen sich Kollektionen von Objekten eines Typs erzeugen und Referenzen ermöglichen direkte Assoziationen zwischen Objekten (ohne Primär-Fremdschlüssel-Beziehung).
- **Benutzerdefinierte Funktionen (UDF):** Um die interne Struktur der benutzerdefinierten Datentypen vor den Anwendungen verbergen zu können, wird mit benutzerdefinierten Funktionen eine Schnittstelle geschaffen. Auf diese Weise entstehen „echte" abstrakte Datentypen (ADTs) mit gekapselten Daten und Operationen, die allerdings nur auf Serverseite ausgeführt werden können. Diese UDFs erweitern die Anfragesprache SQL, wobei einige Systeme (z. B. Informix) auch eine Erweiterung des Optimierers erlauben, um beispielsweise die Auswertung von UDFs in der WHERE-Klausel durch Indexstrukturen zu unterstützen.
- **Verwaltung großer Objekte:** Von den RDBMS haben ORDBMS auch den BLOB-Datentyp geerbt. Dieser kann zusammen mit dem UDT/UDF-Mechanismus genutzt werden, um semantisch reiche Medienobjekte vollständig innerhalb des DBMS zu verwalten. (Allerdings gibt es immer noch keine Unterstützung für kontinuierliche Medienobjekte.)

- **Erweiterungsmodule:** In der Regel sind zur Unterstützung eines Medientyps mehr als nur *ein* UDT zu definieren und dementsprechend viele UDFs. Daneben wird häufig auch noch ein Schema (mit Tabellen, Assoziationen etc.) für die Metadaten vorgegeben. All diese zu einem Medientyp (z. B. Video) gehörenden Komponenten können zu einem Paket (je nach DBMS-Hersteller z. B. DataBlade, Extender oder Data Cartridge genannt) geschnürt werden. Dies erleichtert die Entwicklung und Weitergabe solcher ORDBMS-Erweiterungen durch Drittanbieter.

2.2 Existierende medienspezifische Datentypen und ihre Möglichkeiten

ORDBMS verfügen nicht nur theoretisch über das Potential, VITA-Datentypen zu integrieren. Es sind auch schon etliche solche Datentypen entwickelt worden. Es ist indes nicht das Ziel dieser Abhandlung, eine umfassende Übersicht über diese Entwicklungen zu präsentieren. Um dem mit ORDBMS nicht so vertrauten Leser dennoch einen Eindruck von der Mächtigkeit dieser Datentypen zu vermitteln, sollen im folgenden zumindest einige aus Industrie (Informix, Excalibur) und Forschung (GMD) kurz vorgestellt werden.

2.2.1 Excalibur Text Search DataBlade

Dies ist ein Erweiterungsmodul [5] für den IDS/UDO[3] von Informix. Es bietet im wesentlichen Funktionen für die Volltextsuche an. Die Texte können sowohl innerhalb des DBMS verwaltet werden (alle CHAR- und LOB-Datentypen) als auch extern in verschiedenen proprietären Formaten (z. B. MS Word; die benötigten Zugriffsfunktionen stellt Informix bereit). Die zentrale Rolle bei der Textsuche spielt die UDF `etx_contains`, die z. B. wie folgt zur Formulierung einer unscharfen Anfrage mit Ergebnis-Ranking einsetzbar ist:

```
SELECT rc.score, id, description FROM videos
WHERE etx_contains(description,
      Row('multimedia', 'PATTERN_TRANS & PATTERN_SUBS'),
            rc # etx_ReturnType)
ORDER BY 1;
```

Der erste Parameter der `etx_contains`-Funktion ist der zu durchsuchende Text (hier eine Spalte der Tabelle `videos`. Der zweite definiert das zu suchende Textmuster, und der dritte (optionale) Parameter ist ein OUT-Parameter, der einen *Retrieval Status Value (Score)* für jeden durchsuchten Text zurückliefert. Dieser ist um so höher, je besser der Text die Anfrage erfüllt, und kann daher zur Sortierung der Texte nach Relevanz herangezogen werden. Um die Auswertung der `etx_contains`-Prädikate effizient durchführen zu können, erstellt und verwaltet das DataBlade spezielle Volltextindexstrukturen, die mit einer entsprechenden "Create index"-Anweisung angelegt werden müssen.

2.2.2 Excalibur Image DataBlade

Wie das Text Search DataBlade konzentriert sich auch Excalibur's Image Datablade [4] auf die Unterstützung der unscharfen Suche, in diesem Fall nach Bildern. Wenn ein Bild in der Datenbank gespeichert wird, extrahiert eine sogenannte *feature extractor function*

[3] Informix Dynamic Server with Universal Data Option

charakteristische Merkmale aus dem Bild und bildet daraus einen Merkmalsvektor, welcher dann anstelle des eigentlichen Bildes indexiert wird.

Für eine Anfrage muss ein Suchbild bereitgestellt werden (i. e. *Query by Example*), für das ebenfalls ein Merkmalsvektor berechnet wird. Dieser wird mit den gespeicherten Merkmalsvektoren verglichen, was im Ergebnis immer eine Rangfolge der ähnlichsten Bilder ergibt. Syntaktisch ähnelt eine solche Anfrage dem Beispiel aus dem vorigen Abschnitt 2.2.1.

Das DataBlade verwendet ein eigenes Rasterbildformat, das auch den Anwendungen als externes Speicherformat angeboten wird. Für einige andere Formate wie GIF, JPEG, TIFF etc. sind Konverter vorhanden. Außer zum Schreiben, Lesen und Konvertieren gibt es nur wenige Funktionen (Abfrage einiger Parameter wie Bildhöhe, -breite, Speicherformat etc. sowie Skalierung der Bildgröße).

2.2.3 Informix Video Foundation DataBlade (VFDB)

Dieses DataBlade [11] stellt einige Basisdatentypen und –funktionen für den Medientyp Video zur Verfügung. Obwohl es auch direkt durch Anwendungen genutzt werden kann, ist es in erster Linie als Ausgangspunkt für Entwickler von Video-DataBlades mit weitergehender Funktionalität gedacht. Beispiele hierfür folgen in den nächsten beiden Abschnitten. Die vom VFDB bereitgestellte Funktionalität umfasst im wesentlichen folgendes:

- **Formatunabhängige Adressierung:** Punkte in der *Timeline* eines Videos können unabhängig vom Speicherungsformat (z. B. MPEG oder AVI) beschrieben werden, beispielsweise im SMPTE-Format oder in Form von Frame-Nummern. Die Verwendung solcher *Timecodes* erleichtert die Verknüpfung von beschreibenden Metadaten (z. B. Szenenbeschreibungen oder Szenenwechselindexierung) mit den Videorohdaten, vor allem, wenn die Rohdaten in verschiedenen Speicherformaten vorliegen (z. B. einmal als MPEG-Datei und einmal auf VHS-Band).
- **Geräteunabhängige Speicherungsschnittstelle:** Die Videorohdaten können auf den verschiedensten Speichermedien abgelegt werden, z. B. Dateisysteme, Videorecorder oder Video-Streaming-Server. Das VFDB definiert für die Nutzung dieser Speichermedien eine einheitliche, Gerätedetails verbergende Schnittstelle, das sog. *Virtual Storage Interface* (VSI).
- **Metadatenschema:** Es werden einige Tabellen vordefiniert, die Metadaten zu Videos aufnehmen können, darunter die wichtigsten physischen Attribute (Länge, Format usw.) sowie das Video oder einzelne Szenen beschreibende Daten, die verschiedenen Typs sein können (Text, Grafik, Audio, Video) und zu Gruppen mit semantischen Gemeinsamkeiten gebündelt werden können (Stratifikation). Das Schema ist nach Bedarf erweiterbar.

2.2.4 GMD IPSI Continuous Long Field DataBlade

Dieses DataBlade erweitert das VFDB im wesentlichen um die Fähigkeit, Präsentationen in Echtzeit zu unterstützen [9]. Dazu wird ein neuer UDT *Continuous Long Field* (CLF) eingeführt sowie – integriert in die Client-API – spezielle Puffer- und Qualitätsanpassungsmechanismen. Allerdings können die CLF-Rohdaten nicht auf beliebigen über das VSI nutzbaren Speichern abgelegt werden, sondern nur auf einem CLF-Server, der ein bestimmtes, proprietäres Streaming-Protokoll verwendet.

CLF ist ein generischer Datentyp, der beliebige kontinuierliche Medienströme (Video, Audio, Animation etc.) und Speicherformate unterstützt. Er erlaubt Adressierung und Zugriff auf die Rohdaten in logischen Einheiten (z. B. Frames bei Videos oder Samples bei Audio) und bietet auf dieser Granularitätsstufe auch Edit-Funktionen wie Einfügen, Löschen und Anhängen. Physisch zusammengesetzte Medienströme (z. B. ein Video mit einer oder mehreren Tonspuren) werden logisch dekomponiert, d. h., jeder Einzelstrom wird durch ein eigenes CLF-Objekt modelliert und ist somit separat zugreif- und manipulierbar.

2.2.5 GMD IPSI MPEG DataBlade

Dieses DataBlade erweitert das VFDB um spezielle Datentypen und Funktionen für Videos, die im MPEG-Format gespeichert sind [7]. Gegenwärtig werden die Standards MPEG1 und MPEG2 angeboten, Unterstützung der noch in der Spezifikationsphase befindlichen Standards MPEG4 und MPEG7 ist ebenso wie die Nutzung des CLF-DataBlades für die Zukunft geplant.

Die bereitgestellten Funktionen erlauben den Zugriff auf interne Strukturen eines MPEG-Datenstroms, das direkte Dekodieren einzelner Frames sowie die Extraktion von Metadaten und deren Speicherung in den vom VFDB-Metadatenschema definierten Tabellen.

3 Abstraktionen für medienspezifische Datentypen

Das Konzept der medienspezifischen abstrakten Datentypen (MADT) geht auf [14] zurück. Das Ziel ist hierbei, neue Datentypen für Medienobjekte einzuführen, welche die gleichen Abstraktionen wie „eingebaute“ Datentypen erfüllen. Dazu genügt es nicht, die Daten nur zu kapseln, auch ihre interne Struktur muss durch eine logische Struktur überlagert werden, auf der dann die Operationen für den Datentyp spezifiziert werden. Die logische Struktur für den Datentyp Text könnte z. B. eine hierarchische Struktur sein mit Wörtern, die Zeilen bilden, die wiederum Absätze bilden usw. Beim Datentyp Image könnte die logische Struktur eine Pixelmatrix sein und beim Datentyp Video eine Sequenz von Frames, die ihrerseits wieder als Pixelmatrix aufgefasst werden.

Der Vorteil dieses Konzepts ist, dass die Semantik der Datentypen durch die zugrundegelegte logische Struktur und die darauf erlaubten Operationen eindeutig und explizit festgelegt ist und nicht durch eine anwendungsgetriebene Wahl von Speicherungsformaten und –geräten in unvorhersehbarer Weise manipulierbar ist. Dies ist eine wichtige Voraussetzung für Anwendungsneutralität bei gleichzeitiger Anreicherung des DBMS um mächtige medienspezifische Funktionen.

Die im vorigen Kapitel beschriebenen Datentypen sind keine MADTs im obigen Sinne. Um dies noch etwas deutlicher werden zu lassen, werden die wichtigsten durch das MADT-Konzept zu realisierenden Abstraktionen in den folgenden Abschnitten dargestellt. Es ist jedoch zu beachten, dass es sich hier um Abstraktionen bezüglich *einzelner* Medienobjekte und nicht bezüglich (möglicherweise heterogener) *Mengen* von Medienobjekten, wie sie z. B. für die Anfrageverarbeitung und Synchronisation/Präsentation *multi*medialer Daten entwickelt wurden [8, 17].

3.1 Ortstransparenz und Geräteunabhängigkeit

Es gibt gute Gründe, Mediendaten nicht unter allen Umständen im direkt vom DBMS verwalteten Speicher (z. B. BLOBs) abzulegen, sondern auf (für das DBMS) externen Speichermedien. Die beiden wichtigsten sind:

- **Präsentationsunterstützung:** Dies bedeutet bei kontinuierlichen Medien, dass bestimmte Echtzeitanforderungen zu erfüllen sind, was universelle DBMS nicht leisten (können). Die hierfür benötigten sog. *Continuous Media Server* müssen am DBMS „vorbei" auf die Medienrohdaten zugreifen können.
- **Geräteunterstützung:** Viele für Mediendaten gebräuchliche Speichergeräte werden von universellen DBMS nicht unterstützt (z. B. Bildplatten).

Den Speicherort und das –gerät nicht zu verbergen, würde die Anwendungen dazu verleiten oder sogar zwingen, solche extern gespeicherten Daten direkt zu manipulieren und Annahmen über die Charakteristika der verwendeten Geräte zu treffen. Folglich würde das DBMS nur eine sehr geringe Kontrolle über diese Daten ausüben, also normalerweise „klassische" DBMS-Aufgaben gar nicht wahrnehmen.

Um Ortstransparenz und Geräteunabhängigkeit zu gewährleisten, dürfen die externen Identifikatoren der Medienobjekte keine Rückschlüsse auf die Speicherorte und –geräte zulassen, so dass Zugriffe nur über Operationen des MADTs möglich sind. Mit Ausnahme des CLF DataBlades wird bei den in Kap. 2.2 vorgestellten Mediendatentypen (und vielen anderen) exakt gegenteilig vorgegangen, mit dem Ziel, möglichst einfach viele verschiedene externe Speichermedien einbinden zu können, wobei jedoch die Last, sich auf deren unterschiedliche Schnittstellen einzustellen, den Anwendungen aufgebürdet wird. Daneben entsteht auch noch das Problem, die Integrität der externen Referenzen zu wahren, da Anwendungen ohne Kenntnis des DBMS Medienobjekte z. B. löschen oder verschieben können. Eine Lösung ist hierfür von IBM präsentiert worden (Datalink-Konzept [20]).

3.2 Datenunabhängigkeit

Mit Datenunabhängigkeit ist normalerweise die Eigenschaft der Datentypen eines DBMS gemeint, ein ausschließlich an ihrer Semantik (d. h. den auf ihnen definierten Operationen) ausgerichtetes Bild ihrer inneren Struktur nach außen (d. h. zu den Anwendungen) hin zu präsentieren. Die tatsächlichen physischen Speicherungsstrukturen bleiben den Anwendungen verborgen und sind daher auch ohne Auswirkungen auf die Anwendungen veränderbar. Zur Anwendung zu übertragende Datenobjekte werden in ein der Anwendung bekanntes externes Speicherformat konvertiert.

Für medienspezifische Datentypen gibt es keine so eindeutige Definition der Datenunabhängigkeit. Die Ursache ist, dass zu solchen Datentypen sowohl eine (Roh-)Datenebene als auch eine Metadatenebene gehört. Häufig wird daher schon von Datenunabhängigkeit gesprochen, wenn dies eigentlich nur für die Metaebene gilt. Möglich wird dies etwa durch die Einführung logischer Adressierung, wie z. B. beim VFDB (Kap. 2.2.3), wodurch eine vom Speicherungsformat der Rohdaten vollkommen unabhängige Modellierung der Metadaten erreicht wird. Solange die Anwendung nur mit den Metadaten arbeitet (z. B. Suchanfragen stellt) ist i. d. R. Datenunabhängigkeit gewährleistet. Sobald jedoch mit den Rohdaten gearbeitet werden soll, wird deren internes Format sichtbar, was bedeutet, dass die Anwendung erst zur Laufzeit feststellen kann, ob und, wenn ja, was sie damit

anstellen kann (außer, sie arbeitet nur mit selbst generierten Daten, was bei den in der Einführung aufgezählten Anwendungsszenarien aber nur ein Ausnahmefall sein kann).

Beim MADT-Konzept wird folglich von Datenunabhängigkeit nur unter Einbeziehung der Rohdaten gesprochen. Dieser spezielle Aspekt wird als *Formatunabhängigkeit* bezeichnet und ist Thema des nächsten Abschnitts.

3.3 Formatunabhängigkeit

Das MADT-Konzept unterscheidet bei den Rohdaten zwischen dem *internen*, dem *externen* und dem *logischen* (oder auch konzeptionellen) Datenformat. Letzteres ist bereits weiter oben erläutert worden. Das interne Format wird vom DBMS für die Speicherung und interne Verarbeitung der Rohdaten verwendet. Eine wichtige Anforderung ist hier, dass das interne Format *alle* von den Anwendungen erzeugten Daten bewahren kann. Daraus folgt: das DBMS darf intern z. B. keine verlustbehafteten Komprimierungsverfahren einsetzen, da damit auch ein Verlust an Anwendungsneutralität einherginge.

Wenn zwischen Anwendungen und DBMS Rohdaten übertragen werden, z. B. bei der Erzeugung von Medienobjekten oder bei der Ausgabe zwecks Präsentation oder Weiterverarbeitung auf dem Client, werden von der jeweiligen Anwendung gewählte externe Datenformate verwendet. Bei der Ausgabe sollen hierbei nicht allein die Art der Kodierung (z. B. JPEG oder TIFF bei Bilddaten), sondern auch zahlreiche Qualitätsparameter bestimmt werden können. Das heißt, es müssen auch nur soviel Daten zur Anwendung geschickt werden, wie zur Erfüllung der gegebenen Qualitätsansprüche nötig sind. Möchte eine Anwendung dem Benutzer z. B. eine Vorschau einer Menge von Bildern präsentieren, so kann sie die Bilder genau in der geringen Qualität anfordern, die hierfür angemessen ist. Später könnte der Benutzer eines der Bilder auf einen Laserdrucker ausgeben wollen. Dafür würde die Anwendung dieses eine Bild dann noch mal anfordern, dieses Mal aber vermutlich als hochauflösendes Graustufenbild im Postscript-Format.

Formatunabhängigkeit impliziert, dass das DBMS über Konvertierungsfähigkeiten verfügen muss, um interne in externe Formate umzuwandeln (und umgekehrt), und diese für die Anwendungen transparent einzusetzen vermag. Man kann sich leicht vorstellen, dass dies bei Mediendaten (insbesondere kontinuierlichen) mit erheblich größerem Aufwand verbunden ist als bei Standarddatentypen wie Integer oder Character. Andererseits ist das Konvertieren von Mediendaten ein (nicht selten lästiges) Bedürfnis, das viele Multimediaanwendungen gemeinsam haben, so dass es durchaus lohnenswert ist, dieses Problem zentral und elegant aus der Welt zu schaffen. Architekturen für Multimedia-DBMS, die Formatunabhängigkeit unterstützen, sind z. B. in [21] und [16] beschrieben.

3.4 Transformationsunabhängigkeit

Das MADT-Konzept sieht auch Operationen vor, mit denen Medienobjekte verändert (editiert) oder in ein anderes Medium umgeformt werden können. Typische Beispiele für den ersteren Fall sind Effektfilter (z. B. Weichzeichner, Solarisationsfilter oder Verzerrer für Bilder). Eine Medienumformung wäre z. B. die Transkription einer Sprachaufzeichnung (Audio → Text).

Die Ausführung solcher Operationen ist oft sehr rechenintensiv und erzeugt einen hohen Speicherbedarf. Aus Anwendungssicht ist jedoch nur entscheidend, dass die Operationen mit der verlangten Dienstgüte (dazu zählen bei kontinuierlichen Medien meist auch

zeitliche Restriktionen) ausgeführt werden. Wenn man also beabsichtigt, derartige Operationen zu DBMS-Datentypen anzubieten, dann sollte das DBMS zu deren Optimierung befähigt werden, wozu auch eine sorgfältige Planung und Überwachung der benötigten Betriebsmittel gehört. Ein weiteres durch das DBMS zu lösendes Problem ist, dass eine Transformation ein Medienobjekt für eine andere Anwendung unbrauchbar machen kann (signifikanter Qualitätsverlust), was dem Prinzip der Anwendungsneutralität zuwiderliefe. Dies sollte sehr ernst genommen werden, denn Medienobjekte in globalen Datenbanken werden oftmals als „media assets“ angesehen, bei denen entsprechend vorsichtig mit Änderungsoperationen umzugehen ist, weil viele davon irreversibel sind.

Um die Komplexität sowohl der Optimierung von Transformationen als auch des Schutzes konkurrierender Anwendungen vor deren möglicherweise unerwünschten Auswirkungen von der Anwendungsschnittstelle fernzuhalten, wird das Konzept der Transformationsunabhängigkeit eingeführt, das im wesentlichen Abstraktionen bezüglich folgender drei Aspekte beinhaltet:

- **Ausführungsort:** Das DBMS entscheidet erst zum Zeitpunkt des Aufrufs, ob eine Operation auf dem DBMS-Server, einem anderen Server oder der Client-Maschine auszuführen ist. Da die Erfüllbarkeit der Dienstgüteanforderungen von der Verfügbarkeit bestimmter Betriebsmittel zur Ausführungszeit abhängt, ist zu einem früheren Zeitpunkt (z. B. statisch) keine optimale Wahl des Ausführungsortes möglich.
- **Ausführungsreihenfolge:** Wenn mehrere Operationen auf einem Medienobjekt hintereinander ausgeführt werden sollen, ist deren Reihenfolge oftmals ohne Bedeutung für das Endergebnis (z. B. ist es egal, ob ein Bild zuerst gespiegelt und dann der Kontrast erhöht wird oder umgekehrt). Damit dies vom DBMS für die Optimierung ausgenutzt werden kann, ist es den Anwendungen zu ermöglichen, beliebig viele Operationen *auf einmal* aufzurufen, wobei deren Reihenfolge nur, soweit sie von semantischer Bedeutung ist, zu spezifizieren ist. Dies wird im folgenden als Transformationsanfrage bezeichnet.
- **Dauerhaftigkeit von Transformationen:** Um das Problem irreversibler Transformationen in den Griff zu bekommen, ist es sinnvoll, die Originale am besten immer unangetastet zu lassen. Um dennoch dauerhafte Transformationen zulassen zu können, liegt es nahe, ein Versionierungskonzept einzusetzen. Da Mediendaten sehr großvolumig sind, kann das allerdings leicht zu einem explodierenden Speicherbedarf führen. Je nachdem, wie man die Kosten für Festspeicher- und Prozessornutzung gegeneinander abwägt, ist es also u. U. günstiger, nicht die transformierten Daten, sondern nur die erzeugende Transformationsanfrage zu speichern und die Daten bei erneuter Anforderung „on the fly“ wieder aus den Originaldaten zu generieren (z. B. sinnvoll bei Skalierungen). Wenn es explizit gemacht wird, ähnelt dieses Vorgehen dem Sichtenkonzept bei relationalen DBMS (statt von einer Version könnte man daher auch von einer materialisierten Sicht sprechen). Bei Transformationsunabhängigkeit wird indessen angestrebt, den Einsatz von Versionierungs- und Sichtenkonzepten vor den Anwendungen so zu verbergen, dass das DBMS die Entscheidung für die eine oder die andere Methode allein aufgrund interner Kostenfunktionen und Policies treffen kann. Selbst ein Wechsel, also Umwandlung einer Version in eine Sicht oder umgekehrt, sollte von den Anwendungen unbemerkt bleiben. Prinzipiell kann man die Entscheidung, ob ein transformiertes Medienobjekt *überhaupt* dauerhaft gespeichert wird, sogar gänzlich dem DBMS überlassen: die Anwendung könnte sich die entsprechende Transformationsanfrage selbst merken (anstelle einer Medienobjekt-ID), später erneut an das DBMS sen-

den und genau dasselbe Ergebnis erhalten – unabhängig davon, ob das DBMS die Anfrage komplett neu auswertet oder dabei auf eine vorsorglich erstellte Version oder Sicht zurückgreift.

Zusammenfassend lässt sich Transformationsunabhängigkeit wie folgt charakterisieren: Anwendungen können Transformationsanfragen stellen, die entweder persistent (d. h. eine neue extern sichtbare Medienobjektversion erzeugend) oder transient sind. Eine solche Anfrage spezifiziert beliebig viele einzelne, semantisch bedeutsame Operationen in deskriptiver Form. Das DBMS übernimmt die Aufgabe, aus der Transformationsanfrage zunächst eine Transformationsvorschrift zu generieren, indem es eine gültige Operationssequenz, eventuell unter Hinzufügung von Hilfsoperationen (z. B. Formatkonvertierungen), berechnet. Schließlich wird die Transformationsvorschrift zu einem (nur aktuell gültigen) Transformationsplan expandiert, der eine Zuordnung der einzelnen Operationen zu einem oder mehreren Servern, die über die erforderlichen Betriebsmittel verfügen, festlegt.

Die Anwendungen sind damit in der Lage, Transformationen von Medienobjekten sowohl kurzfristig als auch dauerhaft durchzuführen, ohne sich darum kümmern zu müssen, welche Maßnahmen zur Sicherstellung bzw. Maximierung der Dienstgüte (→ Optimierung) und zur Erhaltung der Anwendungsneutralität (→ Isolation) hierbei notwendig sind. Es ist wohl fast unnötig zu erwähnen, dass dies weit über das hinausgeht, was heutige medienspezifische Datentypen diesbezüglich leisten (vgl. Kap. 2.2). Nur das Konzept der *Enhanced Abstract Data Types (E-ADT)* des ORDBMS-Prototypen *Predator* basiert z. T. auf ähnlichen Überlegungen zur Optimierung von Transformationen [22]. Insbesondere kommt man auch dort zu dem Schluss, dass Transformationsanfragen deskriptiv sein müssen, um ihre Optimierung durch das DBMS zu ermöglichen. Das E-ADT-Konzept behandelt jedoch nicht das Problem der irreversiblen Transformationen und bietet folglich noch keine Transformationsunabhängigkeit.

Transformationsunabhängigkeit ist eine substantielle Erweiterung des ursprünglichen MADT-Konzepts. Das daraus resultierende neue MADT-Konzept wird in den folgenden beiden Kapiteln als *VirtualMedia-Konzept* weiter verfeinert, insbesondere unter Betrachtung wichtiger Realisierungsaspekte wie der Wahl eines mit kontinuierlichen Medien verträglichen Verarbeitungsmodells und einer Sprache für Transformationsanfragen sowie der Frage, wie ein solches Konzept in heutige ORDBMS einbettbar ist.

4 Das VirtualMedia-Konzept

4.1 Medienspezifische Abstrakte Datentypen: Erfahrungen aus früheren Projekten

Das (ursprüngliche) MADT-Konzept wurde bereits mehrfach prototypisch umgesetzt. Eine Entwicklungslinie konzentrierte sich dabei auf die Realisierung von MADTs als Erweiterung eines kommerziellen DBMS. Konkret kam hier Ingres mit Object Management Extension (INGRES/OME) zum Einsatz [25]. Da die Möglichkeiten von Ingres-OME begrenzt waren, vor allem in Bezug auf Echtzeitunterstützung, wurden parallel dazu das Medienobjektspeicherungssystem MOSS [13] und sein Nachfolger KANGAROO[4] [16]

[4] KANGAROO steht für *Kernel Architecture for Next Generation Archives of Realtime-Operable Objects*.

entwickelt. Diese sollten die notwendige Kernfunktionalität bereitstellen (z. B. Speicher- und Pufferverwaltung, Echtzeit-Scheduling, Betriebsmittelreservierung, Admission Control,...), die nötig ist, um Dienstgütegarantien, Daten- und Formatunabhängigkeit usw. zu ermöglichen.

Während für den MOSS/KANGAROO-Kern durchaus tragfähige Konzepte entwickelt werden konnten, z. B. [15], stellte sich die Gestaltung der MOSS-MADT-Schnittstelle, die möglichst generisch sein sollte, zunehmend als das konzeptionell schwierigste Problem heraus. Der Grund dafür war das vom klassischen ADT-Konzept übernommene (und in heutigen OO-Konzepten weiterhin dominierende) Verarbeitungsmodell, das auf einem synchronen Request-Response-Protokoll aufbaut. Dieses Verarbeitungsmodell weicht nämlich erheblich von dem intern verwendeten ab, das auf für Mediendatenströme wesentlich besser geeigneten asynchronen Nachrichten- bzw. Datenaustauschprotokollen basiert. Der Vorteil des internen Verarbeitungsmodells liegt in seinem Potential zur Modularisierung, Parallelisierung und Verteilung (auf mehrere Verarbeitungsstationen) von Operationen – vergleichbar dem von super-skalaren Prozessoren bekannten Pipelining (das allerdings i. allg. synchron arbeitet).

Das Request-Response-Modell besticht hingegen durch seine Klarheit und Einfachheit und ist daher für die Anwendungsschnittstelle vorzuziehen. Es gelang jedoch nicht, eine Abbildung auf das interne Verarbeitungsmodell zu finden, die dessen Möglichkeiten und Vorteile *im Verborgenen* nutzt, d. h., ohne die mögliche und gewünschte Einfachheit der MADT-Schnittstellen zu zerstören. Folgendes Beispiel soll dies verdeutlichen (vgl. auch das motivierende Beispiel in [22]):

`Image` sei eine Instanz des MADTs IMAGE. Damit die Anwendung das Bild auf dem Monitor des Anwenders anzeigen kann, sollte es eine entsprechende Funktion geben:

```
Image.display( screen ) // zeige Image auf dem Device screen an
```

Angenommen, `Image` ist in einer Auflösung von 2048×2048 Bildpunkten gespeichert, um die Ausgabe auf hochauflösende Drucker optimal zu unterstützen. Dann sollte das Bild vor der Ausgabe besser etwas in der Größe reduziert werden:

```
Image.scale( 0.5 )      // verkleinere Image auf 50%
```

Die Anwendung erreicht somit alles, was sie will, indem sie zuerst `scale` und dann `display` aufruft. Das DBMS versäumt hier jedoch eine Möglichkeit zur Optimierung (in diesem Fall Skalierung „on the fly“ während die Bilddaten zum Client geschickt werden), weil es *nicht weiß*, dass die Anwendung die Skalierung nur für das anschließende Anzeigen verlangt. Abhilfe brächte hier nur ein Zusammenziehen der beiden Funktionen, wofür es verschiedene Möglichkeiten gibt, die jedoch allesamt die Komplexität der MADT-Schnittstelle explodieren lassen, z. B.:

```
Image.scale_and_display( 0.5, screen )
// zeige Image 50% verkleinert auf dem Device screen an
```

Noch deutlicher wird das, wenn noch eine dritte Funktion hinzukommt. Wenn `Image` z. B. ein Satellitenbild ist, könnte eine Farbverfälschung die Interpretation am Bildschirm wesentlich erleichtern. Auch dies müsste nun theoretisch der `display`-Funktion hinzugefügt werden:

```
Image.scale_and_color_distortion_and_display( ... )
```

Man kann sich leicht vorstellen, dass ein derartiger Entwurf nicht gerade zu einer gut verständlichen und handhabbaren MADT-Schnittstelle führt. Die Alternative besteht darin, das interne Verarbeitungsmodell des DBMS in einer abstrakten Form auf die MADT-Ebene „hochzuziehen“, um eine handhabbare, durch die Semantik und nicht durch Optimierungsprobleme geprägte Anwendungsschnittstelle zu erhalten, die dem DBMS dennoch eine maximale Anzahl an Freiheitsgraden zur Optimierung von Operationen auf und mit Medienobjekten eröffnet. Dieses sog. VirtualMedia-Verarbeitungsmodell, das im übrigen alle in Kapitel 3 beschriebenen Abstraktionskonzepte unterstützt, wird im folgenden Abschnitt vorgestellt.

4.2 Das VirtualMedia-Verarbeitungsmodell

4.2.1 Filtergraphen

Das VirtualMedia-Verarbeitungsmodell basiert auf dem Filtergraph-Konzept, das ein bereits etabliertes Verarbeitungsmodell für Medienobjektdaten ist, vgl. z. B. [2, 3, 18]. Ein Filtergraph ist ein gerichteter azyklischer Graph bestehend aus mindestens einer Datenquelle, beliebig vielen Medienfiltern mit jeweils mindestens einem Datenein- und –ausgang und mindestens einer Datensenke. Ein möglicher Filtergraph für das obige IMAGE-Beispiel ist in Fig. 1 dargestellt. Hier werden die Bilddaten zunächst vom DBMS-Speicher geholt, zum Skalierer geschickt, der die Bildgröße verändert, dann zum Farbverzerrer, der die Falschfarbendarstellung erzeugt, und schließlich zum Renderer, der die Daten für die Darstellung auf dem Bildschirm des Anwenders aufbereitet.

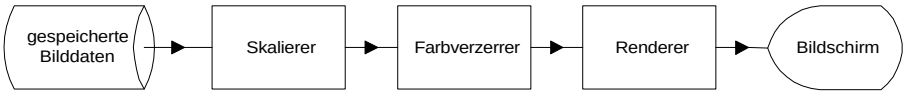

Fig. 1. Filtergraph für das IMAGE-Beispiel aus Kap. 4.1

Dieses erste Beispiel zeigt noch eine relativ simple Filterkette. Um zu demonstrieren, wie leicht daraus ein „richtiger“ Filtergraph werden kann, soll das Beispiel noch etwas erweitert werden. Es ist z. B. gut vorstellbar, dass das Bild zu Vergleichszwecken neben der Falschfarbendarstellung auch noch in den Originalfarben angezeigt werden soll. Auch diese zweite Version sollte skaliert werden, beispielsweise auf ein Viertel der Originalgröße. Den resultierenden (genauer: einen möglichen) Filtergraphen zeigt die folgende Fig. 2.

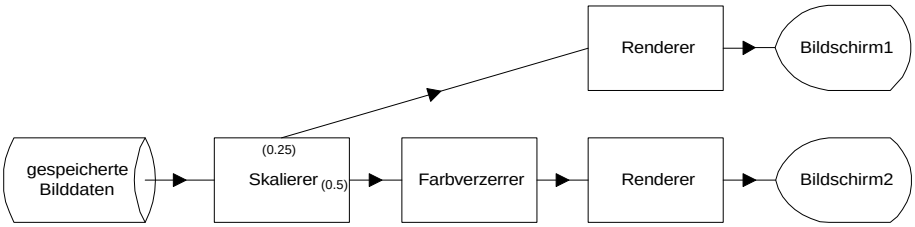

Fig. 2. Filtergraph für das erweiterte IMAGE-Beispiel

Die beiden Filtergraphen stellen die Verarbeitungsfolge gemäß der abstrakten Sicht des Anwendungsentwicklers dar. Aus der Perspektive des DBMS sieht der Filtergraph für das

IMAGE-Beispiel etwas anders aus, da die Darstellung der Bilder auf dem Bildschirm nicht in den Zuständigkeitsbereich des DBMS fällt. Andererseits könnte das DBMS gezwungen sein, weitere Filter hinzuzufügen, die für die Anwendung transparent sind. Dies wäre beispielsweise notwendig, um die Bilddaten in ein für das Rendering geeignetes Format zu konvertieren. Fig. 3 zeigt einen solchen Filtergraphen.

Der Filtergraph in Fig 3 ist eine Transformationsvorschrift im Sinne von Kapitel 3.4, wobei auf einige Details (z. B. Operationsparameter) in der Darstellung verzichtet wird. Im nächsten Abschnitt soll nun skizziert werden, wie eine Transformationsanfrage aufgebaut sein muss, aus der das DBMS einen solchen Filtergraphen als Transformationsvorschrift ableiten kann.

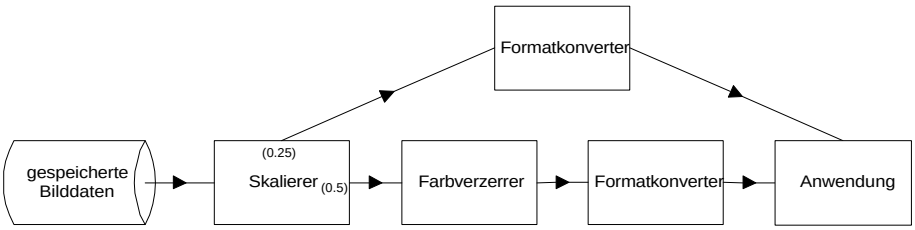

Fig. 3. DBMS-Filtergraph für das erweiterte IMAGE-Beispiel

4.2.2 Die Anwendungsschnittstelle: Deskriptoren beschreiben virtuelle Medienobjekte

Wie in Kapitel 4.1 demonstriert wurde, ist es mit einer operationalen Schnittstelle kaum möglich, von Optimierungsproblemen, deren Lösung man dem DBMS überlassen möchte, zu abstrahieren. Daher wird als Anwendungsschnittstelle eine Medienbeschreibungssprache eingeführt, die als *VirtualMedia Language*, kurz VML, bezeichnet wird. Mit Hilfe dieser Sprache muss es den Anwendungen möglich sein, jedes Transformationsziel zu beschreiben, ohne zu wissen oder gar vorwegzunehmen, welches der optimale Weg zu diesem Ziel ist. Das Transformationsziel, genannt *VirtualMedia Descriptor* (VMD), ist dann identisch mit der Transformationsanfrage, die an das DBMS zu senden ist.

Es würde den Rahmen dieses Aufsatzes sprengen, die VML hier im Detail vorzustellen und zu erläutern. Stattdessen sollen die wesentlichen Eigenschaften wiederum anhand des IMAGE-Beispiels beschrieben werden. Der VirtualMedia Descriptor für dieses Beispiel ist in Fig. 4 dargestellt (da die Notation in XML [1] syntaktisch bekannten Markup Languages wie HTML ähnelt, wird auf nähere Erläuterungen zur Syntax verzichtet).

VML unterscheidet zwischen Quellobjekten (SOURCE) und virtuellen Objekten (VIRTUAL). Ein Quellobjekt ist ein reales Medienobjekt, das in einer Datenbank gespeichert und durch einen eindeutigen Schlüssel identifizierbar ist. Ein virtuelles Medienobjekt entsteht durch Transformation eines oder Komposition mehrerer Medienobjekte, welche sowohl real als auch virtuell sein dürfen. Der Beispiel-VMD beschreibt ein Quellobjekt (das in der DB gespeicherte Original-Satellitenbild) und zwei virtuelle Objekte, die beide durch unterschiedliche Transformationen des Quellobjekts erzeugt werden (sollen). Die Spezifikation eines Quellobjekts besteht im wesentlichen aus der Angabe des Medienobjektidentifikators (MOID) und der Zuordnung eines Alias, über den das Objekt innerhalb des VMD referenziert wird. Die Spezifikation eines virtuellen Medienobjekts erfordert hingegen wesentlich mehr Angaben:

```
<?XML version="1.0"
 RMD="NONE"?>
<VMDESC>
<SOURCE>
   <MOID ALIAS="DBImg"
     REF="SatelliteImages/4711">
   </MOID>
</SOURCE>
<VIRTUAL NAME="DistortedImage"
   MAINTYPE="IMAGE"
   SUBTYPE="RASTER"
   ENCODING="BMP">
   <QUALITY>
      <SIZE SPECTYPE="Relative"
        DIMENSION="Both"
        VALUE="50%"/>
      <COLOR SPECTYPE="Depth"
        MODE="C" VALUE="8"/>
   </QUALITY>
   <TRANSFORMATION
     NAME="Distort">
      <OPERATION
        NAME="ColorDistortion">
          <INPUT NAME="DBImg"/>
          <PARAM NAME="Dist"
            VALUE="10"/>
      </OPERATION>
   </TRANSFORMATION>
</VIRTUAL>
<VIRTUAL NAME="SmallImage"
 MAINTYPE="IMAGE"
 SUBTYPE="RASTER"
 ENCODING="BMP">
   <QUALITY>
      <SIZE SPECTYPE="Relative"
        DIMENSION="Both"
        VALUE="25%"/>
      <COLOR SPECTYPE="Depth"
        MODE="C" VALUE="24"/>
   </QUALITY>
   <TRANSFORMATION>
      <OPERATION NAME="Null">
         <INPUT NAME="DBImg"/>
      </OPERATION>
   </TRANSFORMATION>
</VIRTUAL>
</VMDESC>
```

Fig. 4. VirtualMedia Descriptor für das erweiterte IMAGE-Beispiel

- **NAME:** Über seinen Namen, der innerhalb des VMD eindeutig sein muss, wird das virtuelle Objekt referenziert. Der Name findet sowohl innerhalb des VMD Verwendung (wenn das virtuelle Objekt gleichzeitig Quellobjekt eines anderen virtuellen Objekts ist) als auch nach Verarbeitung des VMDs durch ein DBMS zur Referenzierung des materialisierten virtuellen Medienobjekts auf dem Client.
- **MAINTYPE:** Gibt den Medientyp an, z. B. IMAGE, AUDIO, VIDEO oder MULTIPART (für komposite Medienobjekte).
- **SUBTYPE:** Gibt einen Untertyp zum Medientyp an. Bilder (Medientyp IMAGE) können z. B. Rasterbilder (Untertyp RASTER) oder Vektorgrafiken (Untertyp GRAPHIC) sein.
- **ENCODING:** Gibt das (externe) Datenformat an.
- **QUALITY:** Hier können beliebig viele Dienstgüteparameter angegeben werden. Dies können einfache Qualitätsparameter sein wie Bildgröße oder Samplingfrequenz, aber auch Parameter, die das Echtzeitverhalten beschreiben, z. B. ob es deterministisch oder statistisch garantiert sein soll oder welche Formen der Interaktivität zu unterstützen sind (z. B. VCR-Funktionalität).
- **TRANSFORMATION:** Hier sind schließlich die Quellobjekte (*INPUT*) und die Operationen (*OPERATION*) samt Steuerparametern (*PARAM*) zu spezifizieren. Es kann mehrere Transformationsabschnitte in einer virtuellen Objektbeschreibung geben. Dadurch kann, falls erforderlich, eine bestimmte Operationssequenz erzwungen werden, da benannte Transformationen Zwischenobjekte

repräsentieren, die wiederum als Input für andere Transformationen dienen können.

Anhand des IMAGE-Beispiels konnten bis hierhin nur die grundlegenden Konzepte des VirtualMedia-Verarbeitungsmodells und der VML dargelegt werden. Erweiterte Konzepte, die hier aus Platzgründen nicht mehr behandelt werden können, befassen sich insbesondere mit der Beschreibung und Behandlung virtueller *kompositer* Medienobjekte (Medientyp MULTIPART). Ebenfalls nicht weiter erörtert wird die Frage, *wie* das DBMS aus dem VMD einen passenden Filtergraphen berechnen kann, da hierzu im wesentlichen bekannte Algorithmen aus dem Bereich der regelbasierten Systeme die Grundlage bilden. Stattdessen soll im folgenden letzten Abschnitt dieses Kapitels das durch VirtualMedia eröffnete Optimierungspotential beleuchtet werden.

4.2.3 Optimierung hinter den Kulissen

VirtualMedia erlaubt den transparenten Einsatz verschiedener Optimierungsstrategien, wie schon in Kapitel 3.4 „Transformationsunabhängigkeit“ in Aussicht gestellt wurde. Eine naheliegende Strategie ist die Variation der Filtergraphen. Dass es selbst bei einfachen Filtergraphen mehrere Varianten geben kann, die zum selben Ziel führen, lässt sich leicht anhand des IMAGE-Beispiels demonstrieren: Fig. 5 zeigt eine gültige Alternative für den Filtergraphen aus Fig. 3.

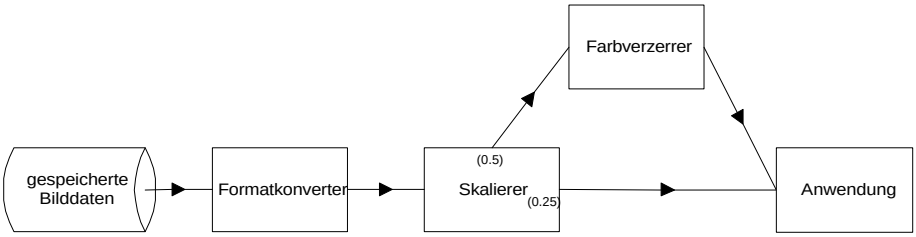

Fig. 5. Alternativer DBMS-Filtergraph für das IMAGE-Beispiel

Jeder Filter und jede Datenübertragung kostet Betriebsmittel, sobald das DBMS den Filtergraphen instanziiert. Ziel der Optimierung muss daher sein, die von der Anwendung gewünschte Dienstgüte mit einem minimalen Einsatz von Betriebsmitteln zu gewährleisten. Da Betriebsmittel nicht unbegrenzt verfügbar sind, spielt deren Auslastung zum Zeitpunkt der Anfrage eine große Rolle bei der Optimierung. Bei der Realisierung von VirtualMedia als ein verteiltes System (s. Kap. 5), ist es jedoch unrealistisch, ein globales Optimum berechnen zu wollen. Stattdessen wird vorgeschlagen, die Optimierung in zwei Phasen, die Filtergraph-Optimierung und die Betriebsmitteloptimierung, zu unterteilen. Die Kostenfunktion für die Filtergraph-Optimierung, die die optimale Filtergraph-Variante unabhängig von der Betriebsmittelauslastung bestimmt, muss im wesentlichen folgende Kostenarten berücksichtigen:

- **Speicherungskosten:** Jeder Filter muss die Mediendaten, die er verarbeiten soll, zwischenspeichern (puffern). Die Größe dieses Puffers hängt u. a. vom Datenformat ab, das durch Variation des Filtergraphen beeinflussbar ist. Beispielsweise arbeitet der Skalierer in Fig. 5 mit einem anderen Datenformat als der in Fig. 3, da die Konvertierung vorgezogen wurde.

- **Verarbeitungskosten:** Die Verarbeitungskosten hängen ebenfalls vom Datenformat ab und zusätzlich noch von der Datenmenge. Oftmals ist es daher günstiger, Filter, welche die Datenmenge verringern, möglichst weit vorne im Filtergraphen zu positionieren, also im IMAGE-Beispiel etwa den Skalierer vor dem Farbverzerrer.
- **Verzögerungskosten:** Jeder Filter verzögert wegen der Zwischenpufferung die Antwortzeit des DBMS (Anfangsverzögerung) und bei interaktiv gesteuerten kontinuierlichen Medien auch die Reaktionszeit. Daher sollte der optimale Filtergraph eine möglichst minimale Gesamtverzögerung besitzen.
- **Übertragungskosten:** Die Mediendaten müssen natürlich von einem Filter zum nächsten übertragen werden. Es sollte also auch hier daraufhin optimiert werden, immer möglichst geringe Datenmengen zu übertragen. Da jedoch in dieser Optimierungsphase noch nicht festgelegt werden soll, ob die Übergabe direkt über den Hauptspeicher (→ sehr geringe Kosten) oder über ein Netzwerk (→ relativ hohe Kosten) erfolgt, fließt dieser Kostenfaktor nur mit geringem Gewicht in die Filtergraph-Optimierung ein. Die endgültige Optimierung bezüglich der Übertragungskosten erfolgt erst im Zuge der Betriebsmitteloptimierung.

Zusätzlich sind auch langfristig wirkende Optimierungsstrategien möglich, bei denen man bestimmte Zwischen- oder Endresultate von Transformationen aufhebt, anstatt sie immer wieder neu zu berechnen. Da dies viel zusätzlichen Speicher kostet, ist eine effiziente Auswahl der aufzuhebenden Resultate erforderlich, um möglichst nur solche mit einer hohen Wiederverwendungswahrscheinlichkeit zu erwischen. Wenn im voraus nur wenig darüber bekannt ist, wie die Anwendungen mit den Medienobjekten arbeiten, können diese Wahrscheinlichkeiten nur mittels statistischer Methoden geschätzt werden. Anderenfalls kommen auch Policy-getriebene Verfahren in Betracht. In unserem IMAGE-Beispiel könnte z. B. angenommen werden, dass die Satellitenbilder zur Betrachtung bzw. Auswertung am Monitor meistens im Bitmap-Format und dann auch nur in bestimmten Auflösungen angefordert werden. In eine entsprechende Richtlinie umgesetzt, hieße das beispielsweise, dass von jedem Bild bei erstmaliger Anforderung eine dauerhafte Kopie (materialisierte Sicht) im Bitmap-Format in 50%-iger Originalauflösung zu erstellen ist. Das hätte zur Folge, dass das DBMS den in Fig. 6 dargestellten Filtergraphen erzeugen würde, wenn es den VMD aus Fig. 4 ein *zweites Mal* als Transformationsanfrage geschickt bekommt.

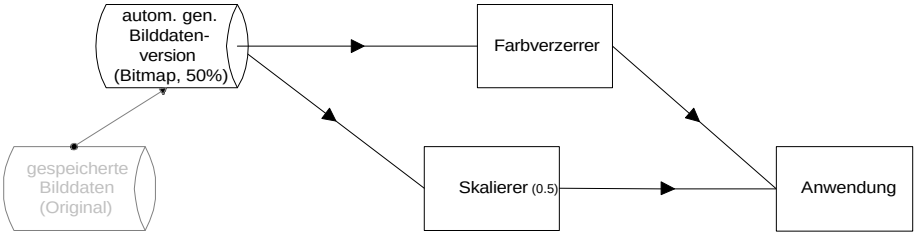

Fig. 6. DBMS-Filtergraph für das IMAGE-Beispiel bei Policy-getriebener Optimierung

Nachdem die Filtergraph-Optimierung abgeschlossen ist, muss das DBMS die Betriebsmittel für den Filtergraphen reservieren. Welche Optimalitätskriterien und Kostenarten bei einer in dieser Phase betriebenen Betriebsmitteloptimierung eine Rolle spielen, ist stark implementierungsabhängig. Wenn z. B. Netzwerkprotokolle verwendet werden, die keine

Betriebsmittelreservierung erlauben, dann wird ein Optimierungsziel sicherlich sein, die Netzwerkauslastung nicht über einen bestimmten Grenzwert steigen zu lassen, um bereits zugesicherte Dienstgütegarantien nicht zu gefährden. Wie bereits oben beim Punkt „Übertragungskosten“ angedeutet, steht dieses Optimierungsziel möglicherweise in Konflikt mit der Filtergraph-Optimierung, d. h., dies ist ein Beispiel, bei dem die zweiphasige Optimierung i. allg. nur ein suboptimales Ergebnis liefern würde.

Alle drei Filtergraphen in den Fig. 3, 5 und 6 sind vom selben VMD abgeleitet (und es gibt noch einige weitere Variationen, die hier nicht dargestellt sind). Man kann sich deshalb leicht vorstellen, dass bei wesentlich komplexeren VMDs als dem in Fig. 4 die Zahl der zu berechnenden und anschließend zu bewertenden Filtergraph-Varianten wegen der vielen Kombinationsmöglichkeiten der Filter sehr groß werden kann. Daher wird ein "Brute Force"-Ansatz, im Stile einer Breitensuche alle kombinatorisch möglichen Varianten durchzurechnen (d. h., erst mal zu bestimmen und anschließend mit der Kostenfunktion zu bewerten), vermutlich schlecht skalieren. Die gleiche Überlegung gilt in noch höherem Maße für die Betriebsmitteloptimierung, wenn man die im folgenden Kapitel beschriebene verteilte Architektur zugrunde legt. Diesem Problem wird man durch Einschränkung der Breitensuche (basierend auf Heuristiken und Statistiken) begegnen müssen, wodurch deutlich wird, dass das durch VirtualMedia geschaffene Optimierungspotential tendenziell eher zu groß ist, um es in einem reichhaltig mit Medienfiltern ausgestatteten DBMS überhaupt vollständig ausnutzen zu können.

5 Erweiterung eines ORDBMS um VirtualMedia

Ein ORDBMS bietet die Möglichkeit der Erweiterung um medienspezifische Datentypen durch Dritte, wie in Kap. 2 dargelegt wurde. In diesem Kapitel wird gezeigt, dass dieses Potential prinzipiell (da noch keine Implementierung existiert) ausreicht, um das VirtualMedia-Konzept als Erweiterung eines ORDBMS zu realisieren.

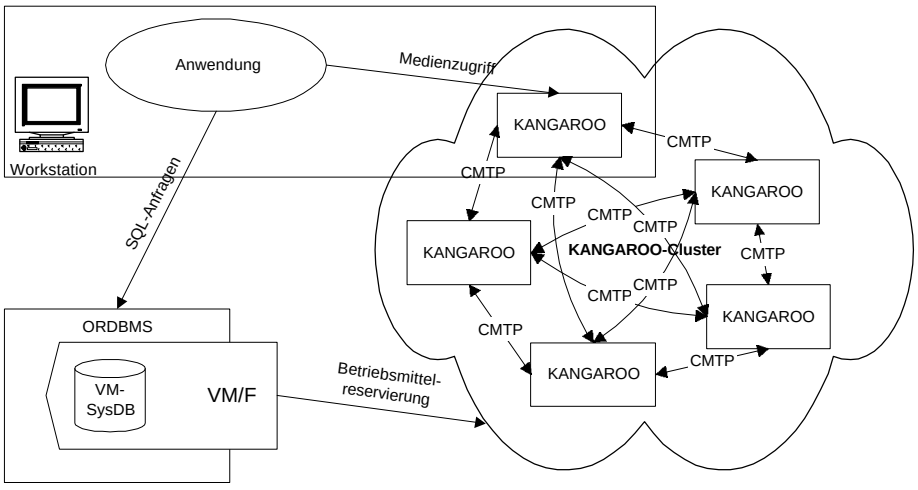

Fig. 7. Systemarchitektur der ORDBMS-VirtualMedia-Erweiterung

5.1 Systemarchitektur

Die Architektur der VirtualMedia-Erweiterung, die in Fig. 7 im Überblick dargestellt ist, unterscheidet sich grob betrachtet nicht wesentlich von der Systemarchitektur einiger in Kap. 2.2 beschriebener ORDBMS-Erweiterungen, insbesondere dem CLF-DataBlade, bei dem ebenfalls eine sehr enge Kopplung zwischen dem ORDBMS und dem externen Mediendatenserver hergestellt wird. Die Erweiterung besteht zum einen aus einem direkt in das ORDBMS integrierten Modul, genannt VM/F (VirtualMedia Foundation), und zum anderen aus einem verteilten Mediendaten- und Transformationsserver, dem KANGAROO-Cluster. Die Funktionalität dieser beiden Module wird im folgenden näher beschrieben.

5.1.1 VM/F

Das VM/F-Modul ermöglicht es, Medienobjekte in einem ORDBMS zu verwalten und mittels VirtualMedia-Deskriptoren zu manipulieren. Dazu stellt es folgende Funktionalität zur Verfügung:

- **Ein Metadatenschema** zur Verwaltung der Registrierungsdaten zu den Medienobjekten.
- **Funktionen** für das Stellen von Transformationsanfragen in Form von VMDs. Näheres dazu folgt im Abschnitt über die Anwendungsschnittstelle (Kap. 5.2).
- **Einen VML-Übersetzer**, der die VMDs in Filtergraphen übersetzt, diese optimiert und die notwendigen Betriebsmittel auf der KANGAROO-Seite reserviert.
- **Administrationsfunktionen und ein Datenschema für eine Systemdatenbank (VM-SysDB)** zur Verwaltung der KANGAROO-Cluster-Konfiguration und zur Unterstützung des Optimierungsprozesses (Optimierungs-Policies, Versionenverwaltung, KANGAROO-Laststatistik, etc.).

5.1.2 KANGAROO-Cluster

Das bereits in Kap. 4.1 erwähnte KANGAROO-Projekt liefert die Funktionalität für den Mediendaten- und Transformationsserver. Der Cluster kann aus beliebig vielen KANGAROO-Modulen bestehen, die sowohl auf dedizierten Servern als auch auf den Arbeitsstationen der Anwender residieren. Dadurch ist es möglich, Verarbeitungskapazität auf Clientseite auszunutzen, die im Gegensatz zu den Kapazitäten auf den Servern i. d. R. einer Anwendung exklusiv zur Verfügung steht. Ein KANGAROO-Modul kann im einzelnen folgende Komponenten besitzen:

- **Continuous Media Transport Protocol (CMTP):** Das CMTP sorgt dafür, dass KANGAROO-Module untereinander und mit den Anwendungen kommunizieren können. Dies ist die einzige obligatorische Komponente für KANGAROO-Module. Solche minimalen Module sind für „leichtgewichtige“ Clients gedacht, die keine registrierten Teile des KANGAROO-Clusters sein müssen (da keine Betriebsmittelreservierung möglich ist).
- **Zugangskontrolle (Admission Control) und Betriebsmittelverwaltung (Resource Management):** Diese Komponente erlaubt Betriebsmittelreservierung und –zusicherung für Filtergraphen (bzw. die auf dem kontrollierten KANGAROO-Modul instanziierten Teile eines Filtergraphen).
- **Medienfilter:** Jedes KANGAROO-Modul kann beliebig viele Medienfilter zur Ausführung unterschiedlicher Transformationen beinhalten. Dabei ist es z. B. denkbar, dass

sich einzelne Module nur auf ganz bestimmte Filtertypen spezialisieren, die durch besondere Hardware in dem Server, auf dem sie installiert sind, unterstützt werden.

- **Speicherserver:** Der Speicherserver ist für die persistente Speicherung der Mediendaten auf sekundären und eventuell auch tertiären Speichermedien verantwortlich. Mindestens ein KANGAROO-Modul im Cluster muss diesen Service zur Verfügung stellen.

5.1.3 Eignung gegenwärtiger ORDBMS

Grundsätzlich reichen die in Kap. 2.1 beschriebenen Möglichkeiten zur Definition neuer Datentypen (UDT) und Funktionen (UDF) aus, um das VM/F-Modul mit heutiger ORDBMS-Technologie zu realisieren. Auf den ersten Blick unkritisch scheint dabei auch die VM-SysDB zu sein. Experimente mit verschiedenen ORDBMS an der Universität Kaiserslautern [6] haben jedoch gezeigt, dass die untersuchten Produkte (Oracle V8.0.3, Informix Universal Server V9.12, IBM DB2 Universal Database V5.0) z. T. erhebliche Schwächen in Bezug auf die benutzerdefinierten Funktionen aufweisen. Besonders hervorzuheben ist hierbei, dass bei DB2 gar kein und bei Oracle 8 nur lesender Zugriff auf die VM-SysDB *innerhalb* einer UDF möglich wäre. Viele Funktionen des VM/F-Moduls müssten daher als Stored Procedures, die im Gegensatz zu UDFs nicht innerhalb von SQL-SELECT-Anweisungen aufgerufen werden dürfen, implementiert werden. Dies ist zwar kein generelles Hindernis, erschwert jedoch den Entwurf und die spätere Nutzung der Anwendungsschnittstelle.

Des weiteren wurde in der bereits erwähnten Untersuchung [6] deutlich, dass keines der genannten Systeme derzeit mit einer guten Unterstützung für Entwickler von Erweiterungsmodulen aufwartet. Besonders kritisch für die Implementierung der zum Teil sehr komplexen VM/F-Funktionen ist hierbei das Fehlen von Debugging-Werkzeugen. Auch das Laufzeitverhalten der benutzerdefinierten Erweiterungen lässt häufig noch zu wünschen übrig (z. B. wurden während der Tests eine vergleichsweise geringe Performanz bei Oracle sowie Server-Abstürze bei Informix und DB2 beobachtet).

5.2 Anwendungsschnittstelle

Zur exemplarischen Erläuterung der Anwendungsschnittstelle wird nochmals auf das IMAGE-Beispiel aus Kap. 4.2 zurückgegriffen. Der Zugriff auf das Satellitenbild beginnt hier sicherlich damit, dass das anzuzeigende Bild zuerst einmal gefunden werden muss. Multimedia-Retrieval ist indes nicht Sache des VM/F, weshalb das Suchprädikat in der folgenden Beispielanfrage offen bleibt[5]:

```
SELECT moid(Img), size_info(Img), color_info(Img)
FROM SatelliteImages
WHERE <any_suitable_search_predicate>
```

Das Attribut `Img` der Tabelle `SatelliteImages` ist hierbei vom Typ IMAGE, der vom VM/F definiert wird. Dazu gehören auch die drei UDFs in der SELECT-Klausel, die der Anwendung die notwendigen Informationen zur Generierung des VMDs aus Fig. 4 liefern. Der VMD wird dann mit folgender Prozedur verarbeitet:

---

[5] Freilich kann VirtualMedia Multimedia-Retrieval-Methoden wirksam unterstützen, insbesondere die Indexgenerierung. Diese Option wird jedoch in diesem Aufsatz nicht weiter vertieft.

```
EXECUTE PROCEDURE process_vmd(VMD,
            result # VMD_ResultType)
```

Diese Prozedur übersetzt den VMD in einen optimierten Filtergraphen, führt aber noch keine Betriebsmittelreservierung durch. Stattdessen wird – falls die Anfrage erfüllbar ist – eine Kostenabschätzung zurückgeliefert. Nimmt die Anwendung (bzw. der Anwender) dieses Angebot an, initiiert sie mit folgender Prozedur, der nur das Ergebnis der vorherigen wieder übergeben werden muss, den Zugriff auf die (virtuellen) Mediendaten:

```
EXECUTE PROCEDURE schedule_vm_access(
            process_vmd_result, result # K_KeyType)
```

Hiermit werden die Betriebsmittel für den zuvor berechneten Filtergraphen reserviert und als Ergebnis ein Zugriffsschlüssel zurückgeliefert. Mit diesem wendet sich die Anwendung anschließend an das lokale KANGAROO-Modul, um auf die Daten zuzugreifen.

6 Zusammenfassung und Ausblick

In diesem Aufsatz werden Abstraktionen und Konzepte für medienspezifische Datentypen vorgestellt. Neben bekannte Abstraktionen wie Geräte- und Datenunabhängigkeit tritt hierbei die neue, speziell für medienspezifische Datentypen entwickelte Abstraktion der Transformationsunabhängigkeit, welche im wesentliche zwei Vorteile sichert:

1. *Isolation der Anwendungen bzgl. Transformationen*, d. h., die Wirkung von Änderungsoperationen auf Mediendaten sieht nur die Anwendung, die sie sehen will. Jede Anwendung kann sich also ungestört die Sicht auf die Mediendaten schaffen, die sie benötigt.
2. *Globale Optimierbarkeit von Medienobjekttransformationen durch das DBMS*, dadurch weitgehende Entlastung der Anwendungsentwickler von derartigen Optimierungsproblemen.

Mit VirtualMedia wird ein Konzept für die Realisierung medienspezifischer Datentypen in einem DBMS präsentiert, das Transformationsunabhängigkeit ermöglicht. Grundlage von VirtualMedia sind das auf dem Filtergraph-Prinzip beruhende Verarbeitungsmodell sowie die Anfragesprache VML. Die VML erlaubt es, virtuelle Medienobjekte zu beschreiben, die das DBMS durch geeignete Transformation realer Medienobjekte (ohne deren Integrität zu verletzen) der Anwendung bereitstellen muss. Eine Möglichkeit, wie ein objektrelationales DBMS unter Nutzung dessen Erweiterungsmechanismen um VirtualMedia erweitert werden kann, wird ebenfalls skizziert.

Die aktuellen und für die nähere Zukunft geplanten Arbeiten an VirtualMedia konzentrieren sich weiterhin auf die Verfeinerung und Präzisierung der konzeptionellen Aspekte. Schwerpunkte sind hierbei u. a. die Komplettierung der VML, die Auswahl und anschließende Ausarbeitung geeigneter Algorithmen für den VML-Übersetzer/Optimierer, die Spezifikation der Schnittstelle zwischen VM/F und KANGAROO sowie die Spezifikation der Anwendungsprogrammierschnittstellen von KANGAROO und VM/F. Neben solchen naheliegenden Aufgaben gibt es noch zahlreiche weitere zu untersuchende Fragestellungen, z. B.:

- Wie generisch ist das Konzept tatsächlich, d. h., welche (Klassen von) Medienarten und Operationen werden unterstützt? Das Potential ist bei weitem noch nicht ausgelotet,

z. B. ist die Frage, ob ein kompletter *virtueller* Videofilm – basierend auf Rohmaterial wie Videoclips und Tonaufnahmen sowie Schnittlisten, Effekt- und Überblendfiltern etc. – generierbar ist, noch unbeantwortet.

- Wie kann VirtualMedia eingesetzt werden, um bestmögliche Dienstgüte-Adaptivität bei Nutzung nicht exklusiv reservierbarer Betriebsmittel (z. B. Internetverbindungen) zu erzielen?
- Sind interaktive Filtergraphen sinnvoll? Ein solcher Filtergraph würde einen Filter enthalten, der mit einem Benutzer interagiert. Vorstellbar wäre das z. B. bei semiautomatischen Inhaltsanalyseverfahren zur Unterstützung von Multimedia-Information-Retrieval.

Natürlich besteht auch das Ziel, die in Kap. 5 präsentierte MM-DBMS-Architektur zu implementieren. Der geplante Prototyp soll zunächst als Demonstrator für das VirtualMedia-Konzept dienen, ist aber nicht zuletzt auch die Voraussetzung für eine intensivere Erforschung der Optimierungsproblematik.

Danksagung
Der Autor dankt den Herren Prof. Dr. Theo Härder, Dr. Norbert Ritter, Günter Robbert sowie den anonymen Gutachtern für ihre wertvollen Hinweise zur Verbesserung dieses Aufsatzes.

7 Literatur

1. Bray, T., Paoli, J., Sperberg-McQueen, C. M., (ed.): Extensible Markup Language (XML) 1.0. W3C Recommandation REC-xml-19980210, World Wide Web Consortium, 10. Feb. 1998, URL: http://www.w3.org/TR/1998/REC-xml-19980210.
2. Candan, K. S., Subrahmanian, V. S., Venkat Rangan, P.: Towards a Theory of Collaborative Multimedia. In: Proc. IEEE International Conference on Multimedia Computing and Systems (Hiroshima, Japan, Juni 96), 1996.
3. Dingeldein, D.: Multimedia interactions and how they can be realized. In: Proc. Int. Conf. on Multimedia Computing and Networking, 1995.
4. Excalibur Image DataBlade Module. User's Guide Version 1.1. Informix Press, Juli 1997.
5. Excalibur Text Search DataBlade Module. User's Guide Version 1.1. Informix Press, Juli 1997.
6. Heinrich, J.: Studie über das Einsatzpotential Objekt-Relationaler DBMS und Evaluation kommerzieller Produkte. Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, AG Datenbanken und Informationssysteme, Feb. 1998.
7. Hemmje, M.: MPEG DataBlade Module – Module Structured Video Stream and Meta Data Management. Handout, GMD IPSI, Darmstadt, Okt. 1997.
8. Hollfelder, S., Lee, H.-J.: Data Abstractions for Multimedia Database Systems. GMD Technical Report No. 1075, GMD IPSI, Darmstadt, Mai 1997.
9. Hollfelder, S., Schmidt, F., Hemmje, M., Aberer, K., Steinmetz, A.: Transparent Integration of Continuous Media Support into a Multimedia DBMS. In: Proc. Int. Workshop on Issues and Applications of Database Technology (Berlin, 6.-9. Juli), 1998.
10. Informix Digital Media Solutions: The Emerging Industry Standard for Information Management. Informix White Paper, Informix Software, Inc., 1997.
11. Informix Video Foundation DataBlade Module. User's Guide Version 1.1. Informix Press, Juni 1997.

12. Käckenhoff, R., Merten, D., Meyer-Wegener, K.: Eine vergleichende Untersuchung der Speicherungsformen für multimediale Datenobjekte**.** In: Stucky, W., Oberweis, A., (Hrsg.): Datenbanken in Büro, Technik und Wissenschaft, Proc. GI-Fachtagung (Braunschweig, März 1993), Berlin: Springer-Verlag, 1993, S. 164–180.
13. Käckenhoff, R., Merten, D., Meyer-Wegener, K.: MOSS as Multimedia Object Server – Extended Summary. In: Steinmetz, R., (ed.): Multimedia: Advanced Teleservices and High Speed Communication Architectures, Proc. 2nd Int. Workshop IWACA '94, (Heidelberg, 26.-28. Sept.), Lecture Notes in Computer Science vol. 868, Berlin: Springer-Verlag, 1994, S. 413–425.
14. Meyer-Wegener, K.: Multimedia-Datenbanken. Stuttgart: B. G. Teubner, 1991.
15. Marder, U., Merten, D.: Systempufferverwaltung in Multimedia-Datenbankverwaltungssystemen. In: Lausen, G., (Hrsg.): Datenbanken in Büro, Technik und Wissenschaft, Proc. GI-Fachtagung (Dresden, 22.-24. März 1995), Berlin: Springer-Verlag, 1995, S. 179–193.
16. Marder, U., Robbert, G.: The KANGAROO Project. In: Proc. 3rd Int. Workshop on Multimedia Information Systems (Como, Italien, 25.-27. Sept.), 1997, S. 154–158.
17. Marcus, S., Subrahmanian, V. S.: Towards a Theory of Multimedia Database Systems. In: Subrahmanian, V. S., Jajodia, S., (eds.): Multimedia Database Systems: Issues and Research Directions. Berlin, Heidelberg: Springer-Verlag, 1996, S. 1–35.
18. Microsoft ActiveMovie 1.0 SDK. Online-Dokumentation, Microsoft Corporation, 1996, URL: http://www.microsoft.com/developer/tech/amov1doc.
19. Introduction to Teradata® Multimedia Services. Doc. No. B035-1002-097A, NCR Corporation, 1997.
20. Narang, I., Mohan, C., Brannon, K.: Coordinated Backup and Recovery between DBMS and File Systems. IBM Research Report, IBM Almaden Research Center, Okt. 1996.
21. Prückler, T., Schrefl, M.: An Architecture of a Hypermedia DBMS Supporting Physical Data Independence. In: Proc. 9th ERCIM Database Research Group Workshop on Multimedia Database Systems (Darmstadt, 18.-19. März), 1996.
22. Seshadri, P.: Enhanced abstract data types in object-relational databases. In: The VLDB Journal Vol. 7 No. 3, Berlin, Heidelberg: Springer-Verlag, Aug. 1998, S. 130–140.
23. Saake, G., Schmitt, I., Türker, C.: Objektdatenbanken: Konzepte, Sprachen, Architekturen. Int. Thomson Publishing, 1997.
24. Stonebraker, M.: Object-Relational DBMSs - The Next Great Wave. Morgan Kaufman, 1998.
25. Tannhäuser, D.: Realisierung eines Multimedia-Datenbanksystems mit INGRES/OME unter OSF. Diplomarbeit, Universität Erlangen-Nürnberg, IMMD, Lehrstuhl für Datenbanksysteme, Jan. 1995.