Mitteilung 116

ENTWURF EINER RELATIONALEN DATENBANK ZUR UNTERSTÜTZUNG DER ANALYSE VON BILDFOLGEN

Wolfgang Benn und **Bernd Radig**

IFI-HH-M-116

Dezember 1983

Universität Hamburg

Schlüterstraße 70

D-2000 Hamburg 13

1.0

Zusammenfassung

Es wird das Konzept einer Datenbank beschrieben, die unter Verwendung von PASCAL- und ADA-Sprachkonzepten Relationengebilde zur Beschreibung von Bildobjekten verwaltet und Strukturvergleiche durch RS-Morphismen unterstützt. Relationengebilde sind Darstellungsformen von Bildsymbolen, die bei der automatischen Szenenanalyse zur Beschreibung von Bildobjekten verwendet werden. Das besondere Gewicht dieses Berichtes liegt auf drei Gebieten:

1. Die Integration der Datenbank in ein vorhandenes Rechnernetz zur automatischen Analyse von Bildfolgen aus Realweltszenen und die aus der Spezialisierung eines der Netzwerkrechner zum Datenbankrechner entstehenden Verarbeitungsvorteile.

2. Die Art der Anfrage an die Datenbank, die die Grund-Konzeption des "Query-by-Example" wesentlich erweitert , jedoch so an das die Datenbank umgebende Bildverarbeitungssystem angepaßt ist, daß ein schneller Datenaustausch in einem lokalen Rechnernetz erfolgt.

3. Die Integration der Datenbank in ein ADA-Bildverarbeitungssystem, welches parallel zur Datenbank entsteht, und die Verwendung von ADA-Sprachkonstrukten zur Darstellung von Relationen, bis zur Übernahme von DIANA-Strukturen in das Datenverzeichnis der Datenbank.

Weiterhin werden ein Überblick über den aktuellen Stand der Arbeiten gegeben und die Perspektiven der Weiterentwicklung aufgezeigt.

Das Projekt wird von der Deutschen Forschungsgemeinschaft gefördert.

2.0

Einleitung

Die automatische Interpretation von Bildern und Bildfolgen setzt voraus, daß aus Grauton- oder Farbbildern eine symbolische Beschreibung gewonnen wird, die Grundlage des nachfolgenden Abstraktionsprozesses ist. Symbolische Beschreibungen können durch Relationen formalisiert werden, die Bildsymbolen Eigenschaften zuweisen oder Beziehungen zwischen Bildsymbolen darstellen. Zwei Prozesse sind wesentlich bei der Interpretation von Bildfolgen: Die Gruppierung von Bildsymbolen zur Beschreibung von Objekten und die Herstellung von Korrespondenzen zwischen den symbolischen Beschreibungen verschiedener Bilder einer Folge. Die bei der Auswertung von Bildfolgen anfallende Datenmenge ist so groß, daß sie systematisch auf einem Hintergrundspeicher verwaltet werden muß.

2.1

Zielsetzung des Projektes

Ziel der Entwicklung ist es, aus der Analyse von Realweltszenen gewonnene Bildobjektbeschreibungen durch eine Datenbank verwalten zu lassen. Durch eine neue Anfragesprache werden Strukturvergleiche zwischen einzelnen Bildobjekten aufeinanderfolgender Bilder einer Szene sowie zwischen Bildobjekten und vorgegebenen Prototypen erleichtert, indem wesentliche, dazu notwendige Operationen in diese integriert werden.

Bilddatenbanken werden im allgemeinen zur Handhabung großer, bei der Bildverarbeitung - zumeist bei der Auswertung von Luft- und Satellitenbildern (LANDSAT) - in schneller Folge anfallender Datenmengen eingesetzt. Sie verfügen zumeist über spezielle Anpassungen von Anfragesprachen sowie von Datenpräsentationsformen an das sie umgebenden Bildverarbeitungssystem.

Beschreibungen typischer Bildsymbole, die etwa Flächen, Linien oder Punkte repräsentieren lassen sich formal als Relationen definieren, denen Eigenschaften und semantische Beziehungen untereinander, beispielsweise

Nachbarschaftsverhältnisse von Orten, zuzuordnen sind. Durch solche Beziehungen können Bildsymbole hierarchisch zu komplexeren Gebilden, etwa der Beschreibung eines im Bild vorhandenen Objektes (Bildobjekt), gruppiert werden. Derart geordnete Mengen von Relationen werden als Relationengebilde bezeichnet. Beziehungen zwischen Bildobjekten lassen sich durch Morphismen zwischen Relationengebilden - RS-Morphismen - darstellen [RADIG82].

Die verwendeten Programme zur automatischen Szenenanalyse bedienen sich im allgemeinen nur in geringem Umfange eines Dialogs mit dem Bediener, weshalb die Schnittstelle zur Datenbank nicht dialogorientiert sein muß und auf schnellen Datenaustausch mit einem Anwendungsprogramm optimiert sein kann. Zu diesem Zweck ist die Datenbank in ein Prozeßrechnernetz integriert und verfügt über direkten Zugriff zum Speicher des Anwendungsprozesses.

Zur Erzielung weitgehender Parallelität zwischen Anwendungsprogramm und Datenbank residiert diese in einem dedizierten Kleinrechner mit ausreichendem Primär- und Sekundärspeicher zur Vor- und Nachbearbeitung angeforderter Daten nach dem Prinzip eines BACK-END-Rechners.

Parallel zur Entwicklung der Datenbank wird in einem gesonderten Projekt die Programmiersprache ADA auf ihre Eignung für den Einsatz in der automatischen Bildanalyse untersucht, was einerseits die Unterstützung von in ADA geschriebenen Anwendungsprogrammen erforderlich macht, andererseits die Einbeziehung moderner Sprachkonzepte in die Datenbankentwicklung ermöglicht.

2.2

Forderungen an die Datenbank

Ein modularer Aufbau der Datenbank, bei dem komplexe Dienstleistungen auf einfachen Grundfunktionen aufgebaut werden, unterstützt die rasche Entwicklung des Systems bei gleichzeitiger schrittweiser Eliminierung konzeptioneller Fehler. Um verschiedene, parallel verfolgte Ansätze der Bildanalyse unterstützen zu können, muß außer raschem Datenaustausch zwischen Datenbank und Anwendung auch eine benutzerzugängliche Informationsmöglichkeit vorhanden sein,

durch die sich ein Anwender bequem und zuverlässig über den gespeicherten Datenbestand und dessen Strukturen Auskunft verschaffen kann.

Das für die Bildanalyse verwendete Prozeßrechnernetz ist zur Zeit für PASCAL-Programme verfügbar und gestattet Datenverkehr auf der Basis sequentiellen Datenflusses über im Programm deklarierte Dateien. In dieses Rechnernetz ist die Datenbank über ein zum Datenbank-Kern gehörendes Kommunikationsmodul integriert. Zur Zeit existieren ein Kommandokanal, über den das Anwendungsprogramm Transaktionsbefehle senden, sowie ein Statuskanal, über den es Status- und Fehlermeldungen empfangen kann.

Anwendungsdaten werden nicht segmentiert, sondern als Bitfolge in den Datenbestand aufgenommen. Referenzen auf Programmobjekte, die nicht in der Datenbank enthalten sind, werden als virtuelle Felder geführt. Eine Inhaltsrelation enthält die Angaben über Art, Struktur und Bezeichnungen aller in der Datenbank gespeicherten Relationen. Sie ermöglicht eine Kompatibilitätsprüfung der zu bearbeitenden Relationen mit den im Datenbestand enthaltenen. Ebenso bietet sie dem Anwendungsprogrammierer die Möglichkeit, sich über den aktuellen Datenbestand zu informieren.

Die Datenbank gibt einen systemdefinierten Identifikationsschlüssel für jedes Tupel vor, dessen Gültigkeit sich über alle Relationen erstreckt. Die Anforderung von Tupeln kann über diesen Primärschlüssel erfolgen, was einen besonders effektiven Zugriff gestattet, oder über die Angabe von Attributen mit benutzerdefinierter Schlüsselfunktion. Schließlich kann die Anforderung von Tupeln selektiv oder sequentiell nach einer vom Datenbanksystem vorgegebenen Ordnung erfolgen.

2.3

Überblick der aktuellen Forschung

International werden verschiedene Ansätze erforscht, Bilddaten mit Hilfe von meist relationalen Datenbanksystemen organisiert zu erfassen und anwendungsbezogen zu verarbeiten. Die Skala reicht hier vom Expertensystem für

geographische und militärische Zwecke [AKERSTEN80] bis hin zur vereinheitlichten Datendarstellung in Bildverarbeitungsprogramm und Datenbank sowie Integration anwendungsspezifischer Operationen in die Anfragesprache [CHANG81]. Ebenso reichhaltig sind die verschiedensten Forderungskataloge zur Handhabung und Kontrolle von Bilddaten [BILLINGSLEY80] [ENCARNACAO+NEUMANN80] [LORIE81] [WOLFE80].

Generell treten viele Detailprobleme bei der Organisation flächiger Daten auf, die beispielsweise im Bereich der Datenverwaltung und -darstellung [FENG81] [FRIDELL+80] oder der Klassifizierung von Anfragen in relationalen Systemen allgemein [ROUSSOPOULOS82], zu beschreiben und zu lösen versucht werden. Gute Erfahrungen wurden auch mit dem Back-End-Prinzip gemacht, bei dem ein oder mehrere Rechner die komplette Datenaufbereitung zwischen Anwendung und Sekundärspeicher, also die Übergänge zwischen internem und externem Schema, vornehmen [CHANG+77] [FENG81]. Besondere Beachtung findet auch die Erweiterung von Anfragesprachen um graphische oder konstruierende Elemente.

Erste Versuche mit Bilddatenbanken wurden schon 1974 in Japan von T.L. Kunii und Mitarbeitern [KUNII+74] zur Organisation von Bildern und Objektbeschreibungen durchgeführt, die heute zu Untersuchungen über Analyse und Synthese von Texturen mit Datenbankunterstützung geführt haben [CHANG+KUNII81].

1980/81 stellten N.S. Chang und K.S. Fu in mehreren Artikeln ein relationales Datenbanksystem (IMAID) vor, welches in eine LANDSAT-Bildverarbeitung integriert ist. Die Daten werden in graphartiger Form beschrieben (GARG = Generalized Attributed Relational Graph) und von Verarbeitungseinheiten (processing sets) erzeugt und verarbeitet [CHANG+FU80a] [CHANG+FU81] [CHANG+FU81a] [CHANG81]. Insbesondere wurden zweidimensionale Operationen in die Anfragesprache "Query-by-Example" eingefügt, worauf im Abschnitt 3.3.1 weiter eingegangen wird [CHANG+FU80b] [CHANG+FU80c]. Eine recht umfangreiche Übersicht verschiedener Bilddatenbanken ist in [CHOCK82] nachzulesen; einen kurzen Überblick verschaffen die Artikel [CHOCK+81] [CHANG+KUNII81] [ZOBRIST+NAGY81].

Graphikorientierte Anfragesprachen und vektorielle Programmiersprachen, wie APL, sind offenbar des öfteren Untersuchungsobjekt im Bereich der Bilddatenbanken, was gerade APL eine Art Renaissance beschert hat. So wird von IBM-Deutschland ein auf APL/VMS basierendes System, IDAMS (Integrated Data Analysis and

Management System), vertrieben, welches QBE in verschiedene Anwendungsprogrammiersprachen einbettet und als EQBE, embedded QBE, angeboten wird [BLASER+81]. Gleichfalls mit APL arbeitet ein geographisches und kartographisches Datenbanksystem der mexikanischen Regierung [BURGUENO80] und auch aus Italien, von der dortigen IBM-Tochter, sind Bemühungen um auf APL aufbauende Datenbanksysteme bekannt.

Diese Bemerkungen sollen nicht darüber hinwegtäuschen, daß gerade Sprachneuentwicklungen wie ADA [ADA80] [ADA83] im Blickpunkt des Interesses von Programmentwicklern stehen [DRUFFEL82] [ABBOTT83].

Der Einfluß neuer Sprachkonzepte auf Datenbanken und speziell auf Bilddatenbanken bringt neue Lösungen bekannter Probleme ebenso wie zukkunftweisende Ausblicke. ADA bietet eine Reihe solcher Konzepte, die im Abschnitt 3.4 in Bezug zu diesem Projekt gesetzt werden.

Auch andere Ansätze werden in diesem Bereich verfolgt. Einer ist beispielsweise, ADA um eine Anfragesprache, DAPLEX, zu ADAPLEX zu erweitern [CHAN+83]. Ein Preprozessor sorgt für die Umsetzung der ADAPLEX-Befehle in ADA-übersetzbare Konstrukte bei gleichzeitiger Optimierung. Die Erweiterung der Sprache selbst, die vom Initiator nicht erwünscht ist, kann so trotz gleichzeitiger Funktionsausweitung umgangen werden. Ein weiterer Ansatz verfolgt den Anschluß bestehender Datenbanksysteme an ADA durch Emulation der Systemfunktionen. Zu jeder Operation einer Datenmanipulationssprache existierte dann ein ADA-Paket, welches die erwartete Funktion nachbildet [BEVER+82]. Weitere interessante Ansätze enthält ein Artikel von P. Hall, die in der Vorstellung einer permanenten Datenbank ohne Sekundärspeicher gipfeln [HALL83].

3.0

Das Datenbankkonzept

Die Betrachtung in der Literatur beschriebener Lösungen, eine genaue Analyse der Anwendungsbedürfnisse und die Einbeziehung der technischen Gegebenheiten brachten mehrere interne Arbeitsberichte [BENN81-83] hervor. Quintessenz der Problemformulierungen sowie intensiver Diskussionen über den Einsatz vorhandener und die Einschätzung zukünftiger Resourcen ist das vorliegende Datenbankkonzept.

Schwerpunkt dieses Abschnittes sind die Absätze 3.3 über Anfragesprachen und 3.4 über ADA-Sprachkonzepte und das konzeptuelle Datenbankschema.

3.1

Komplexe Datenvorverarbeitung

Die Verwaltung virtueller Felder sowie die Durchführung von Prüfungen auf Kompatibilität zwischen externem und konzeptuellem Schema bilden den Standardrahmen datenbanktypischer Vorverarbeitungen. Die Analyse und Erstellung von Relationengebilden und der Referenzaustausch innerhalb dieser Strukturen stellen konzeptions- und implementationsbedingte Besonderheiten dar. Um den Begriff Datenvorverarbeitung zu präzisieren sei folgende Definition gegeben:

> Zur DATENVORVERARBEITUNG ist jeder Arbeitsschritt der Datenbank zu rechnen, der zur Analyse gegebener und zur Erstellung angeforderter Relationengebilde im Rechner der Datenbank ausgeführt werden muß. Transferoperationen, wie sie im Abschnitt 3.2 beschrieben werden, bilden den Übergangsrahmen zu bzw. von der Vorverarbeitung. Die Art der Vorverarbeitung richtet sich nach den gegebenen Transaktionsbefehlen.

Die Datenvorverarbeitung muß danach als in zwei Richtungen ablaufend angesehen werden. Die eine Richtung ist mit den Transaktionsbefehlen der

Verarbeitungsrichtung Datenbank - Anwendung, beispielsweise "GET", verbunden und beinhaltet alle Vorgänge des Übergangs zwischen internem und externem Schema. Die Gegenrichtung ist demnach mit Befehlen der Richtung Anwendung - Datenbank verbunden und kennzeichnet den umgekehrten Weg.

Zum Verständnis der folgenden Sachverhalte wird die Menge der zugelassenen *Speicherreferenzen* definiert als

$$REF = \mathbf{NIL}\ \cup\ \{ref \mid ref \in [\mathbf{Adressbereich}]\}.$$

und die Menge der zugelassenen systemvergebenen *Tupelidentifikatoren* als

$$TID = \{tid \mid tid \in [\mathbf{Schlüsselbereich}]\}$$

wobei die Intervalle *Adressbereich* und *Schlüsselbereich* implementationsabhängige Größen sind.

REF wird in zwei *Basismengen* (domains) für Attribute aufgeteilt: zum einen in die Menge der Tupel referenzierenden Werte D_{RT} und zum anderen in die Menge der nicht Tupel referenzierenden Werte D_R. Hierbei gelten die Beziehungen:

$$D_{RT} \cup D_R = REF \text{ und } D_{RT} \cap D_R = \mathbf{NIL}.$$

Zur Kennzeichnung der Referenzmengen in den verschiedenen Rechnern wird eine hochgestellte Abkürzung der Prozessorbezeichnung verwendet. Die Menge D_{RT}^{AP} bezeichnet so die Menge D_{RT} im Prozessor des Anwendungsprogrammes, die Abkürzung D_{RT}^{DB} die Basismenge im Prozessor der Datenbank.

3.1.1 Expansion virtueller Felder

Attribute der Basismenge D_R^{AP} werden im internen Schema unterdrückt, d.h. virtualisiert. Dieser Vorgang wird durch einen Eintrag im Datenverzeichnis in der Attributstruktur der betreffenden Relation vermerkt und ist damit reversibel. Die Anforderung eines Tupels mit virtuellen Attributen im Rahmen der Anforderung eines Relationengebildes erfordert die Expansion genau dieser

virtuellen Attribute, bzw. virtuellen Felder. Da die Attributwerte generell zu keinem Zeitpunkt nach der ersten Ablage der Tupel in der Datenbank gültig sind, geschieht die Expansion der Tupel durch Einfügen des Attributwertes für eine nicht definierte Referenz, nämlich NIL.

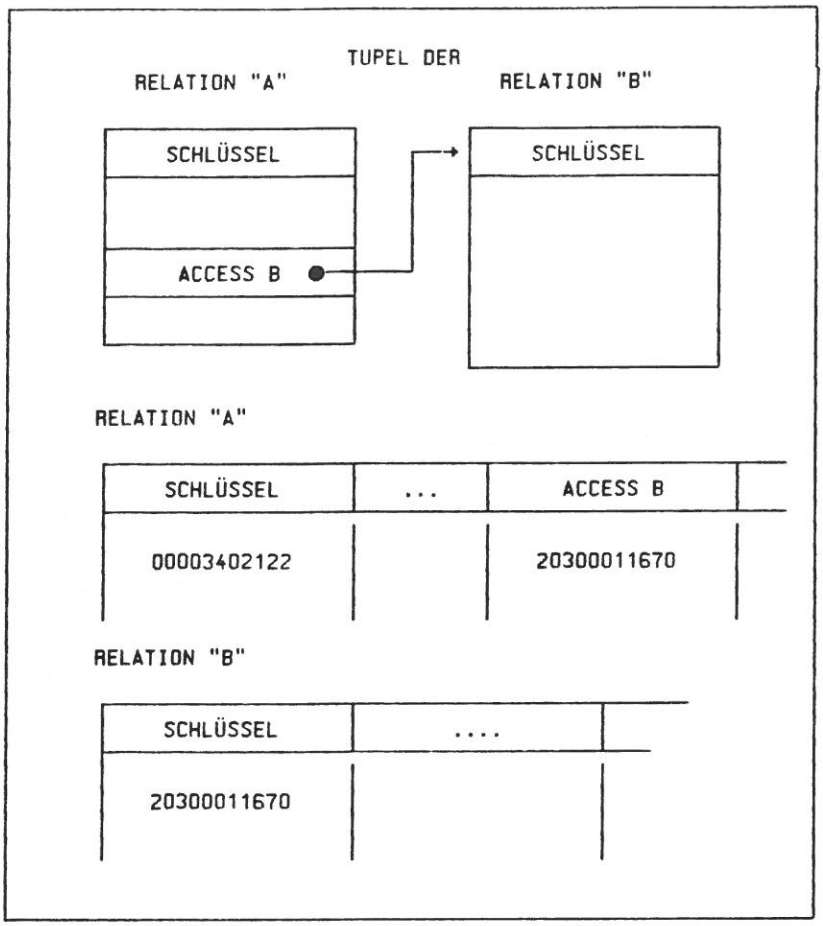

Abb 1: Austausch von D_{RT}^{AP} gegen TID

3.1.2 Schemaübergänge

Werte der Basismenge D_{RT}^{AP}, welche im externen Datenbankschema ein Relationengebilde konstruieren, werden im internen Datenbankschema durch Werte der Basismenge TID dargestellt. Hierfür ist eine Abbildung ψ notwendig, die wegen implementationsabhängiger Basismengen generell nicht klassifizierbar, zum Zeitpunkt der Ausführung jedoch bijektiv ist.

$$\psi: \; D_{RT}^{AP} \rightarrow TID.$$

Die Anforderung von Relationengebilden erfordert die inverse Abbildung ψ^{-1}, indem die Schlüssel wieder auf Speicherreferenzen abgebildet werden. Da ein angefordertes Relationengebilde, soweit es der Speicherplatz im Datenbankrechner zuläßt, dort zusammengesetzt wird, stammen die ersetzten Referenzen in diesem Stadium der Verarbeitung nicht aus der Basismenge D_{RT}^{AP}, die nur im Speicher des anfordernden Programmes definiert ist, sondern aus der Menge D_{RT}^{DB}. Die Abbildung ψ ist folglich in zwei Teilabbildungen,

$$\psi_R : D_{RT}^{AP} \rightarrow D_{RT}^{DB} \quad \text{und} \quad \psi_K : D_{RT}^{DB} \rightarrow TID$$

aufteilbar. Die Abbildung ψ_R geschieht während der Übertragung des Relationengebildes aus dem Speicher des Anwendungsprogrammes in den Bereich der Datenbank sie ist zum Transaktionszeitpunkt als bijektiv anzusehen. Die Repräsentation der Abbildung ist eine interne Relation Ψ_R. ψ_K ist zum Transaktionszeitpunkt ebenfalls bijektiv und wird durch eine zweite Relation Ψ_K dargestellt. Der *natürliche Verbund* (natural join) über D_{RT}^{DB} der Relationen Ψ_R und Ψ_K ergibt eine dreistellige Relation Ψ, über deren Tupel die Abbildungen

$$\psi(adr) = tid \text{ und } \psi^{-1}(tid) = adr \text{ mit } adr \in D_{RT}^{AP} \text{ und } tid \in TID$$

vorgenommen werden können.

3.1.3 Typ- und Zugriffskontrollen

Relationen werden im Anwendungsprogramm durch die Typdeklaration eines Tupels der Relation in der Form eines Datenverbundes deklariert. Zur Begriffsbestimmung sei folgende Definition gegeben:

> Als TUPELSTRUKTUR wird die vom Übersetzer des Anwendungsprogrammes bei der Übersetzung erzeugte abstrakte Beschreibung eines namens einer Relation deklarierten Datenverbundes bezeichnet.

Ein Kommunikationsverbund, der beim Kommandoverkehr über das Rechnernetz der Datenbank übergeben wird, enthält zugleich die Benutzersicht. Die Analyse des

Verbundes gestattet den Zugriff auf die Tupelstrukturen der zu verwendenden Relationen. Diese Strukturen liegen linearisiert als attributierte Teilbäume einer intermediären Programmbeschreibung vor und können wie Relationengebilde behandelt werden und durch Prüfung auf einen RS-morphismus zwischen deklarierter und im Datenbestand enthaltener Relation auf Kompatibilität untersucht werden.

Neu aufzunehmende Relationen werden durch die Anlage eines neuen Tupels der Inhaltsrelation und Aufnahme der aus dem Anwendungsprogramm extrahierten Tupelstruktur als Attribut initialisiert. Zugriffskontrollen erfolgen ebenfalls über die Inhaltsrelation, die Informationen wie Operationsbeschränkungen, etwa Lese-/Schreibschutz, und Verfalldatum einer Relation enthalten kann.

3.2

Schneller Datenaustausch

Datenbanksysteme, die mit Anwendungen in einem Prozessor ablaufen, müssen möglichst geschickt sequentiell oder, bei Verwendung verschiedener Prozesse, quasiparallel mit den Anwendungsprogrammen ablaufen. In diesem Fall stellt sich kein Datentransportproblem ein, da lediglich von der Platte über eventuell vorhandene Datenpuffer kommuniziert wird. In einem verteilten System, bei dem Anwendung und Datenbank auf verschiedenen Prozessoren residieren, sind die Probleme weitaus größer, da jetzt Transportaufgaben zwischen verschiedenen Rechnern entstehen.

In beiden Fällen ist die Abnahme der von der Datenbank gelieferten Daten aus einem vereinbarten Pufferbereich üblich, sofern die Programmiersprache des Anwendungsprogrammes dieses gestattet - beispielsweise durch eine Erweiterung um Teile einer Datenanfragesprache. In der hier beschriebenen Datenbank wird aus zwei Gründen ein anderer Weg verfolgt:

1. soll keine Erweiterung der Anwendungsprogrammiersprache vorgenommen werden, eine pragmatische Entscheidung, und

2. steht die Möglichkeit des direkten Speicherzugriffes zu allen Einzelrechnern zur Verfügung, welche eine hohe Geschwindigkeit gestattet, jedoch eine aufwendigere Adressverwaltung zwischen Datenbank und Anwendungsrechner erfordert.

Letzteres Argument konnte bezüglich der Geschwindigkeit durch eine Vernetzung der Rechner auf der Basis des PASCAL-Dateisystems und deren zuverlässiger Funktion über einen Zeitraum von fast zwei Jahren erhärtet werden [BENN+FAASCH81]. Mittlerweile steht eine Neukonzeption der Vernetzung, auch unter Einbeziehung eines ETHERNET-Anschlusses an die Zentralrechner des Fachbereiches an, die für die Erstellung weiterer Datenbankversionen von grundlegender Bedeutung sein wird.

3.2.1 Technische Ausrüstung

Das Labor des Arbeitsbereiches Kognitive Systeme (KOGS) des FB Informatik verfügt zur Analyse von Realweltszenen über vier Prozessrechner der Firma H. DIETZ, Mülheim. Diese Kleinrechner besitzen jeweils unterschiedliche Speicherausbauten zwischen 256 KByte und 0.5 MByte. Drei der Rechner sind über VT52-kompatible Bildschirmsichtgeräte bedienbar und haben Anschlüsse an Plattenlaufwerke für 60 MByte- bzw. an eine 10 MByte Platte. Abbildung 2 gibt den Anlagenaufbau schematisch wieder. Der schraffierte Kanal stellt den die Rechner und Platten verbindenden Querbus, eine Verlängerung des externen Busanschlusses jeden Rechners dar. Weiterhin bestehen Verbindungen zu den Rechnern des Fachbereiches, einer VAX 780 und einer DEC10-Anlage.

Zur schnellen Verarbeitung und Darstellung von Farbbildern dient ein Farbspeicher mit einer Kapazität von 1,5 MByte, aufgeteilt in drei kontinuierlich adressierbare und aneinandergrenzende Bereiche von jeweils 0,5 MByte zur Aufnahme je eines Farbauszuges. Die optische Darstellung geschieht mit verschiedenen Farb- und Schwarz-weiß-Monitoren, unter anderem einem Rastersichtgerät der Firma COMTAL und einem modernen Bildverarbeitungssystem der Firma VTE.

Die längerfristige Speicherung von Analogdaten ist mit zwei AMPEX-Analogplatten

zur Aufnahme von 600 Schwarz-weiß-Bildern pro Kanal und einer resultierenden Szenenlänge von maximal 24 Sekunden möglich, sowie einem semiprofessionellen Videobandgerät der Firma GRUNDIG zur Aufzeichnung von Farbszenen. Zur Aufnahme der Szenen werden eine Schwarz-weiß- und eine Farbkamera verwendet.

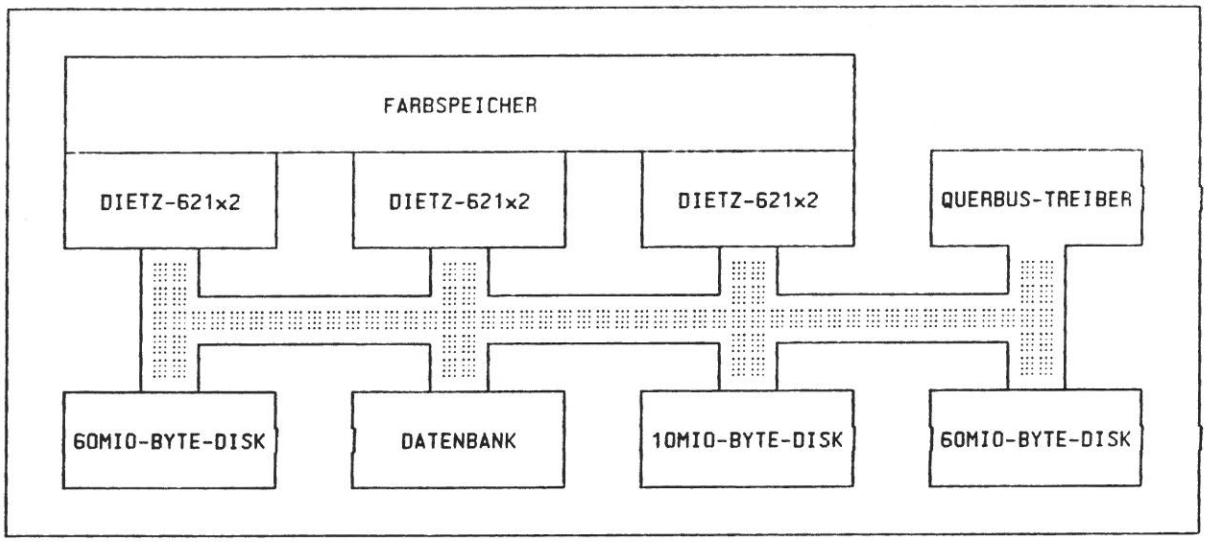

Abb 2: Das Rechnernetz der Gruppe KOGS

3.2.2 Datentransport durch Speicherzugriff

Alle vier zum Rechnernetz zusammengeschlossenen Prozessrechner verkehren miteinander über den 16-bit parallelen Adreß- und Datenbus. Da dieser nicht mit Maschinenbefehlen angesprochen werden kann, bedarf es einer Querbus-Steuerung, die, über Registerbelegungen bedient, direkte Speicherzugriffe ausführt. Der Treiber ist von jedem Rechner erreichbar und bildet das Rückgrat der Netzkommunikation. Alle Transaktionen der Datenbank und des Anwendungsprogrammes werden durch diesen Treiber ausgeführt.

Die Übertragung von Relationengebilden als Abschluß des Überganges vom internen zum externen Schema erfolgt durch die Abbildung ψ_R^{-1} . Bei heterogener Verteilung des Relationengebildes im Speicher wird die Abbildung tupelweise

vorgenommen und ist damit ineffizienter als bei homogener Anlage, bei der blockweise und entsprechend schneller vorgegangen werden kann.

3.3

Anfragen durch Strukturbeispiele

Der Anfrageverkehr herkömmlicher Systeme besteht in der Regel aus der Konstruktion einer Antwortrelation durch die Datenbank nach Angabe von Anfragerelation(en) und Verarbeitungsvorschrift durch den Anwender. Dieses Verfahren ist für die vorliegenden Anfrageanforderungen nicht ausreichend, da nicht eine Antwortrelation gefordert wird, sondern eine geordnete Menge durch Selektion entstandener Teilrelationen. Dennoch bietet sich aus noch näher zu beschreibenden Gründen der Zuschnitt einer bekannten Anfragesprache auf die besonderen Bedürfnisse an.

Spracherweiterungen der Anwendungsprogrammiersprachen scheiden aus, da

1. der bislang verwendete PASCAL-Übersetzer hierdurch an die Grenze seiner Wartbarkeit gelänge und

2. die hauptsächlich zu unterstützende Sprache ADA prinzipiell nicht erweitert werden soll.

Folglich kommt nur eine selbständige Anfragesprache in Betracht. Gute Erfahrungen wurden bei Bilddatenbanken mit Query-by Example (QBE) und davon abgeleiteten, angepaßten Versionen gemacht.

3.3.1 Anfragesprachen für Datenbanken

Anfrage- oder Datenmanipulationssprachen (DML) sind Werkzeuge zum Umgang mit einer Datenbank, die es einem Anwender erlauben, Daten abzufragen und zu verändern. Zu unterscheiden sind hier freiformulierte von standardisierten Anfragen. Während letztere lediglich die Auswahl vorgegebener Anfragen und deren spezielle Parametrisierung vorsehen, bieten freiformulierte Anfragen die Möglichkeit der problembezogenen Fragekonstruktion auf der Basis der Kenntnis des konzeptuellen Schemas. Für den wissenschaftlichen Einsatz mit stets wechselnden Anforderungen eignen sich daher die standardisierten Anfragen weniger.

Eine weitere Klassifizierung wird zwischen selbständigen und in eine Wirtssprache eingebetteten sowie zwischen prozeduralen und deskriptiven Anfragesprachen vorgenommen. Im Gegensatz zu selbständigen Sprachen, die alle zur Anfrage notwendigen Sprachkonstrukte anbieten, sind eingebettete Sprachen stets von den Konstrukten ihrer Wirtssprache abhängig, was die Vor- aber auch die Nachteile einer engen Sprachbindung bedingt. Ein Beispiel extremer Integration in eine Wirtssprache ist PASCAL/R [SCHMIDT+MALL80].

Die weitere Unterscheidung zwischen deskriptiven und prozeduralen Anfragesprachen ist ebenso einfach: Prozedurale Sprachen verwenden Transaktionsbefehle, deren Zweck es ist, die auf den Daten auszuführenden Operationen zu beschreiben, deskriptive Sprachen dagegen beschreiben die Transaktionen durch Wahrheitswertkonstrukte über der Menge der Attribute aller verwendeter Relationen. Hier ist besonders die von Codd eingeführte Relationenalgebra, basierend auf dem Prädikatenkalkül erster Ordnung, zu nennen.

3.3.1.1 Query by Example (QBE)

QBE, 1975 von M. Zloof vorgestellt, ist eine graphik-orientierte Sprache, die in zweidimensionaler Form einem Datenbankbenutzer Anfragen in Dialogform anbietet. Nach benutzerseitiger Spezifikation der zu verwendenden Relationen stellt das System eine leere Relationsschablone auf dem Bildschirm dar. Das Eintragen des Relationsnamens in das leere Schema veranlaßt das System, die

Attributspalten mit den zugehörigen Attributbezeichnern auszufüllen. Anschließend trägt der Benutzer in die Attributspalten Beispielwerte und Operatoren ein, z.B. ein vorangestelltes "P." für "print". Die Abbildung 3 demonstriert den Ablauf einer Anfrage, deren Formulierung im Prädikatenkalkül wie folgt lautet:

$$\{ Nr : \exists (Nr,Start,End,Length) \in LINE \wedge Length > 20 \}$$

Eine Klassifizierung nach den zuvor genannten Prinzipien ordnet QBE in die selbständig deskriptiven Sprachen mit freiformulierbaren Anfragen ein.

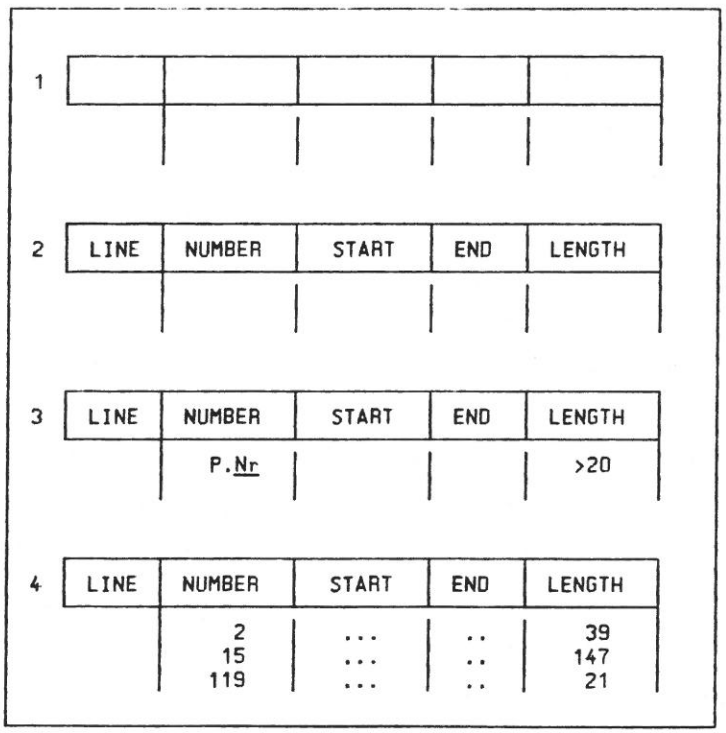

Abb 3: Anfrage in QBE

3.3.1.2 Query by Pictorial Example (QPE)

Aufbauend auf QBE wurde QPE von Chang und Fu für das von ihnen verwendete Bildverarbeitungssystem IMAID und die zugrundeliegende Datenbank weiterentwickelt [CHANG+FU80c]. QPE ist mit einigen Erweiterungen zur Bezeichnung und Anfrage nach zweidimensionalen Daten versehen, die es gestatten,

Bilder und Bildausschnitte zu bearbeiten. Die Dateneingabe ist nicht mehr auf den das Relationsschema darstellenden Bildschirm begrenzt, sondern kann über interaktive Graphiksichtgeräte erfolgen - z.B. Rollkugeleingabe, Maus, etc.

Weiterhin sind in die Sprache Operatoren integriert, die Berechnungen zwischen graphischen Daten ausführen [CHANG81]. Stellvertretend sollen hier die folgenden genannt werden:

1. Die Berechnung des nächsten Nachbarpunktes zu einem gegebenen,

2. die Erzeugung einer Verbindungsgeraden zwischen zwei bezeichneten Punkten,

3. die Berechnung der Entfernung zwischen zwei Punkten, bzw. der Länge einer Verbindungslinie zwischen zwei Punkten.

Die Kennzeichnung von Bildbeispielen im Schema von QPE geschieht durch spezielle Symbole. So kennzeichnet "*" in der Spalte der Bildnummer das gerade dargestellte Bild und "∂" die Eingabe von Werten über ein interaktives Graphikgerät.

Die Abfrage aus dem vorangegangenen Abschnitt sähe unter Bezugnahme auf ein aktuell dargestelltes Bild wie in Abbildung 4 wiedergegeben aus.

LINE	FRAME	NUMBER	START	END	LENGTH
	*	P.Nr	∂	∂	>20

Abb 4: Anfrage in QPE

3.3.2 Query by Structural Example (QSE)

QSE stellt eine Erweiterung des Prinzips der Beispielgebung von Attributwerten dar, das Basis für QBE und QPE ist. Während letztgenannte auf den Dialog mit einem menschlichen Datenbankanwender ausgerichtet sind, ist QSE auf den Datenverkehr zwischen Maschinen spezialisiert. Deshalb wird (zumindest in erster Ausbaustufe) auf jede optische Darstellung verzichtet.

Gleichfalls entfällt die Vorgabe eines Relationsschemas, da der Datenbank zum Zeitpunkt vor der Anfragestellung nicht bekannt ist, ob eine oder mehrere Relationen bearbeitet werden müssen. Weiterhin ist nicht bekannt, in welcher Ordnung die zu bearbeitenden Relationen zueinander stehen. Es gibt also kein standardisiertes Schema eines leeren Relationengebildes, welches einem Anfrageprogramm vorgelegt werden könnte.

Zum Verständnis der Detailbeschreibung sei noch folgende Definition gegeben:

> Ein STARTTUPEL ist ein Tupel, von dem ausgehend, ein Relationengebilde ganz oder zum Teil in seiner Struktur analysiert werden kann. Relationengebilde können mehrere Starttupel haben.

In der Praxis bedeutet dies, daß, ausgehend von den Starttupeln die gesamte Struktur des Relationengebildes durch Verfolgung von Referenzen und automatische Assoziation von Referenztypen mit Tupelstrukturen erkennbar ist. Ein Relationengebilde ohne Starttupel kann nicht als Beispielschema, wie es im nächsten Abschnitt beschrieben wird dienen.

3.3.2.1 Erzeugung von Beispielrelationen

Die Erweiterung der Beispielgebung besteht in der Vorgabe eines leeren, nicht notwendigerweise komplett im Datenbestand enthaltenen Relationengebildes durch das anfragende Anwendungsprogramm. Dieses erzeugt in seinem Speicherbereich Leertupel der zum Relationengebilde gehörenden Relationen in der benötigten Anzahl. Diese Leertupel werden dann nach der dem Gebilde zugrunde liegenden Ordnung miteinander verknüpft - d.h. Attribute, deren Basismenge D_{RT}^{AP} ist, werden mit Werten besetzt. Ein oder mehrere Tupel werden als Starttupeln

bestimmt und deren Adressen der Datenbank mit dem Transaktionsbefehl übermittelt.

3.3.2.2 Auswertung der Beispielrelationen

Die Datenbank erhält vom Anwendungsprogramm den Transaktionsbefehl und die Adresse der Starttupel. Mit Hilfe des direkten Speicherzugriffes und unter Beachtung der Beispielwerte in den besetzten Attributfeldern wird das Relationengebilde in den Speicher der Datenbank eingelesen. Hierbei wird die Relation Ψ_R angelegt. Die Identifikation der gelesenen Leertupel ist durch die Assoziation von Referenzattributen und Tupelstrukturen möglich.

Nachdem das ganze Gebilde bekannt ist, werden die einzelnen Teilrelationen durch Selektion aus den Relationen des Datenbestandes gebildet, wobei die interne Einzelanfrage (stark vereinfacht) etwa so formuliert werden kann:

```
select DREIECK where a3 = reference L1 and
                     a4 = reference L2 and
                     a5 = reference L3 and
                     aK > ....;
```

Hierzu ist die Abbildung ψ_K^{-1} notwendig, da die mit *reference Lj* (j,Kε 1..N) bezeichneten Referenzen im Datenbestand aus der Basismenge *TID* stammen.

Ist das Relationengebilde vollständig, kann es durch die Abbildung ψ_R^{-1} zum Anwendungsprogramm transferiert werden. Für Transaktionen der Richtung Anwendung - Datenbank erfolgen die Transfers und Abbildungen lediglich in umgekehrter Reihenfolge.

3.4

Anwendung hochsprachlicher Konzepte (ADA/DIANA)

Die Verwendung von Sprachkonzepten höherer Programmiersprachen bietet konzeptionelle Vereinfachungen im Bereich der Datenbankentwicklung bei gleichzeitiger Verbesserung der Systemübersicht im konzeptuellen und externen Bereich. Vereinfachungen entstehen zum Beispiel durch den Fortfall expliziter Datendefinitionssprachen bei Verwendung gut strukturierter Anwendungsprogrammiersprachen oder durch starke Integration von Anfragemöglichkeiten in eine Wirtssprache - wie etwa in PASCAL/R.

Als letzte breitangelegte Neuentwicklung höherer Programmiersprachen bietet ADA eine Reihe von modernen Konzepten, die zur Entwicklung und zum Betrieb einer Datenbank verwendet werden können. Strenge Kompatibilitätsregeln, modularer Aufbau von Programmen und exakte Strukturierungsregeln zur Datendeklaration verlagern einen Teil herkömmlicher Datenbankaufgaben in den Bereich der Sprachübersetzung.

Neu ist ebenfalls das Konzept einer einheitlichen ADA-Programmierumgebung, APSE (ADA-Program-Support-Environment), und der standardisierten intermediären Darstellung eines jeden ADA-Programmes in der speziell hierfür entwickelten Zwischensprache DIANA [GOOS+WULF81].

3.4.1 ADA-Sprachkonzepte und Datenbanken

Datenbanken an ADA anzuschließen ohne die Sprache zu erweitern, was vom DOD (Department of Defense), dem Initiator der Sprache, abgelehnt wird, ist auf verschiedene auch emulative Weisen untersucht worden (siehe 2.3). Der bereits erwähnte Artikel von P. Hall versucht dagegen herkömmliche Denkschemata bei der Konzeption von Datenbanken derart zu erweitern, daß futuristische Permanent-Systeme mit Zugriff zu gemeinsamen Primärspeichern durch ADA-Sprachkonzepte realisierbar erscheinen und die bislang verwendeten langsamen Sekundärspeicher ersetzen.

Die folgenden Abschnitte bescheiden sich jedoch auf eine Darstellung der Verwendung findenden Konzepte bei der Erstellung und dem Betrieb unserer Datenbank.

3.4.1.1 ADA als Datendefinitionssprache (DDL)

Datendefinitionssprachen (DDL) herkömmlicher Systeme ermöglichen die Beschreibung von Datenstrukturen gegenüber einer Datenbank auch für Programme, deren Quellsprache keine oder nur geringfügige Typbeschreibungsmöglichkeiten bieten, bzw. deren typbeschreibenden Konstrukte Implementationsabhängigkeiten enthalten. Moderne Programmiersprachen, wie PASCAL oder ADA, bieten allerdings wohldefinierte Möglichkeiten zur Datenstrukturierung unabhängig von deren Implementation. Hierdurch wird der Gebrauch einer gesonderten DDL überflüssig. Datendefinitions- und Programmiersprache der Anwendungen sind identisch.

Vordergründig mag diese enge Bindung an eine Programmiersprache nachteilig erscheinen. Im folgenden wird jedoch ersichtlich werden, daß es sich um eine Bindung handelt, deren Allgemeinheit auch die Einbeziehung anderer gut strukturierter Sprachen gestattet.

3.4.1.2 "Strong-Data-Typing"

Die strenge Inkompatibilität nicht explizit als kompatibel deklarierter Datentypen, das "strong-data-typing", ermöglichen eindeutige Kompatibilitätsprüfungen deklarierter Relationen. Gleichzeitig ermöglicht dieses Prinzip eine Reduzierung in der Darstellung von Tupelstrukturen auf wenige charakterisierende Merkmale. Demgegenüber steht die Erschwerung von Normalisierungen, etwa bei Indextypen, da der einmal festgelegte Indextyp und dessen Ausprägung im typerzeugenden Programm eine dauernde Festlegung bedeuten.

3.4.1.3 Das Modulkonzept

Die Rekonstruktionsmöglichkeit der Tupelstruktur aus der intermediären Darstellung, z.B. aus dem Datenverzeichnis, erlaubt die Bereitstellung aller in der Datenbank enthaltenen Relationen in einem Programmpaket. Dieses Paket kann in Anwendungsprogrammen ganz oder teilweise importiert werden, untersteht aber nicht dem Einflußbereich eines solchen Programmes. So wird eine Klasse von Datentypen, Relationen, geschaffen, deren vollständige Kontrolle der Datenbank obliegt. Die Vervollständigung des Klassenkonzeptes wäre die Bereitstellung reservierter Operationen auf den Relationen als Implementation einer Anfragesprache.

3.4.1.4 Descriptive Intermediate Attributed Notation for ADA (DIANA)

ADA-Programme in allgemeiner zwischensprachlicher Form darzustellen ist die Aufgabe von DIANA. Speziell als Bindeglied zwischen Übersetzungs-Front-End, d.h. zwischen Syntax- und Semantikprüfung, und der Codeerzeugung, dem Übersetzer-Back-End, entwickelt, basiert DIANA auf der formalen Definition von ADA und stellt eine Zusammenfassung der konzeptionell besten Aspekte zweier Vorgängersprachen, AIDA und TCOL, dar. Mathematische Grundlage der intermediären Darstellung sind attributierte Bäume. Die Notation folgt einer speziellen Definitionssprache, IDL, die, angelehnt an eine BNF-Notation, gestattet, Knotenklassen als Nichtterminale und Knoten mit ihren Attributen als Terminale zu spezifizieren. Wichtigste Prinzipien bei der Erstellung von DIANA waren beispielsweise:

* Eine Unabhängigkeit von der Repräsentation der Zwischensprache zu bieten,
* einfache Implementation zu gewährleisten und
* Regeltreue in Notation und Beschreibung einzuhalten [GOOS+WULF81].

Damit bildet DIANA die Basis für Testhilfesysteme (Debug), Formatierungs- und Editierprogramme, d.h. für alle in einem ADA-Programmsystem denkbaren Programmwerkzeuge. Als solches versteht sich auch die Datenbank, die in DIANA alle notwendigen Informationen findet, die Tupelstruktur herauszuarbeiten und in

einem Datenverzeichnis abzulegen.

Dem Grundgedanken einer abstrakten Programmbeschreibung liefe es zuwider, anwendungsspezifische Informationen durch einen Übersetzer zu generieren und in die DIANA-Darstellung einzufügen. Darum steht es frei, spezielle Informationen, für die Datenbank wären das beispielsweise Vermerke über virtuelle Felder, durch Erweiterungen darzustellen.

3.4.2 Leistungen eines Datenverzeichnisses (DVZ)

Ein Datenverzeichnis, auch Datenkatalog, soll in Form einer Relation ein stets zur Verfügung stehendes Auskunftsmittel sein, das es erlaubt, Informationen über Relationen, ihren Aufbau, Schutzbestimmungen und temporäre Gültigkeitsbereiche zu erhalten [CODD82]. Die automatische Auswertung und Führung des Verzeichnisses bietet Vorteile wie die permanent aktualisierende Führung des konzeptuellen Schemas und die Haltung anwendungsverdeckter Vermerke über Speicherung, interne Vorverarbeitungen und Daten zur Überprüfung der Datenbankkonsistenz.

3.4.2.1 Informationsübersicht

Das Datenverzeichnis soll einem menschlichen Datenbankanwender Übersicht über alle verfügbaren Relationen bieten. Zweckmäßig geschieht das in der zur Datendefinition verwendeten Definitionssprache, die hier mit der Programmiersprache der Anwendungsprogramme identisch ist. Die DIANA-Struktur ermöglicht die vollständige Recodierung des Ursprungsprogrammes, weshalb von der Datenbank ein übersetzungsfähiges Deklarationspaket mit allen verfügbaren Relationen erstellt wird. Damit ist eine Darstellung im Dialog am Sichtgerät und ein Import in Anwendungsprogramme möglich. Dem Anwendungsprogrammierer steht es frei, die vordefinierten Relationen zu verwenden oder neu zu definieren.

3.4.2.2 Datenbeschreibung

Die Tupel zur Ablage im Datenbestand werden bis auf die Kompression virtueller Felder nicht verändert. D.h. es wird auf die üblicherweise vorgenommene Segmentation verzichtet und die Tupel unter Voranstellung interner Organisationsfelder als Bitkette eines Datenverbundes abgelegt. Hierbei geht die Tupelstruktur verloren, was nicht generell hinzunehmen ist. Zur Interpretation von Attributen im Vergleich mit Beispielwerten angeforderter Tupel wird die Kenntnis der Tupelstruktur durch die Interpretation der im DVZ enthaltenen Tupelbeschreibung vorgenommen.

3.4.2.3 Konzeptuelles Datenbankschema

Während das konzeptuelle Schema im Entwurfsstadium einer Datenbank als Strukturierungshilfe und Kommunikationsbasis mit den späteren Anwendern dient, stellt es späterhin eine Orientierungsmöglichkeit für neue Anwendungen, eine Informationsübersicht und die Basis für den Übergang zwischen internem und externem Schema dar. Alle drei Kriterien sind durch ein maschineninterpretierbares Datenverzeichnis erfüllbar, weshalb hier das konzeptuelle Schema der Datenbank mit dem Datenverzeichnis identisch ist.

> **Das KONZEPTUELLE SCHEMA der Datenbank besteht aus der Gesamtheit aller im Datenverzeichnis enthaltenen Tupelstrukturen in maschinenlesbarer Form. Es ist mit dem Datenverzeichnis identisch.**

3.4.3 Automatische Erzeugung des Konzeptuellen Schemas

Betrachtet man das konzeptuelle Schema der Datenbank ***(KS)*** als Menge aller im Datenbestand enthaltener Relationen in ADA-deklarationskompatibler Form:

$$KS = \{ Rel \mid Rel \in Datenbestand \}$$

und ein externes Schema *(ES)* , eine Benutzersicht, als Menge von in ADA deklarierten Relationen zur Bearbeitung in einem Anwendungsprogramm:

$$ES = \{ Rel \mid Rel \in KS \vee Rel \notin KS \},$$

lassen sich die zwei Ausgangssituationen einer automatischen Verarbeitung leicht klassifizieren.

Im Falle $ES \subseteq KS$ sind alle notwendigen Prüfungen durch den Übersetzer vorgenommen. Ist $ES \not\subseteq KS$, erzeugt die Datenbank eine neue Menge

$$KS' = KS \cup \{ Rel \mid Rel \notin KS \}.$$

Durch die maschinenlesbare Form intermediärer Deklarationsdarstellungen sind diese Arbeitsschritte automatisierbar. Eine detaillierte Beschreibung folgt in den nächsten Abschnitten.

```
with DB_STANDARD_TYPES;
procedure USE_DB is use DB_STANDARD_TYPES;

  type RELATION1 is record
          KEY          : STANDARDKEY;
          VARI1, VARI2 : TYPE1;
       end record;

   type RELATION2 is record
          KEY          : STANDARDKEY;
          VARI1, VARI2 : TYPE2;
       end record;

   type ACC_REL1 is access RELATION1;
   type ACC_REL2 is access RELATION2;
   type REL      is (R1, R2);

   type COMREC (VAR : REL) is record
          COMMAND : DB_COMMANDS;
          case VAR is
             when R1 => AR1 : ACC_REL1;
             when R2 => AR2 : ACC_REL2;
          end case;
       end record;

begin null;
end USE_DB;
```

Abb 5: Beispieldeklaration COMREC

3.4.3.1 Deklaration der Benutzersicht (view)

Die Deklaration eines externen Datenbankschemas, also einer Benutzersicht, erfolgt durch einen Datenverbund, dessen variabler Verbundteil Verweise auf alle im Programm zu verwendenden Relationen enthält. Diese beziehen sich entweder auf aus dem Datenverzeichnis importierte ($ES \subseteq KS$) oder selbstdeklarierte, bzw. aus anderen Paketen importierte Relationen ($ES \not\subseteq KS$). Eine Mischung aus beidem ist aufteilbar und nach oben genannten Regeln zu bearbeiten.

Dieser Datenverbund hat einen festgelegten Namen (COMREC) und bildet außer der Benutzersichtdeklaration die Basis der Netzkommunikation. D.h. er enthält weiterhin Platz für die Befehls- und Adreßübermittlung an die Datenbank (siehe Abb 5).

3.4.3.2 Traversierung des DIANA-Baumes

Basis der DIANA-Konzeption ist es, relevante Information - Namen und Strukturinformation - in wenigen, meist drei bis vier Schritten durch den DIANA-Baum zu erreichen. Auf die Informationsgewinnung für ein Datenverzeichnis durch die Analyse des Kommunikationsverbundes bezogen, bedeutet das

* Aufsuchen des variablen Verbundteiles (zwei betrachtete Knoten im DIANA-Baum)
* weiter zur Beschreibung dieses Verbundteiles (DN-INNER-RECORD = drei betrachtete Knoten)
* und zu den jeweiligen Verweistypen im variablen Verbundteil (DN-ACCESS = drei betrachtete Knoten).
* Letztlich ist nach zwei weiteren Schritten im DIANA-Baum die Typbeschreibung der referenzierten Relationen erreicht.

Abbildung 6 verdeutlicht diesen Analysevorgang in dem das jeweils unterstrichene Attribut eines Knotens auf den darunterstehenden, bzw. auf den ersten in der zweiten Spalte weist.

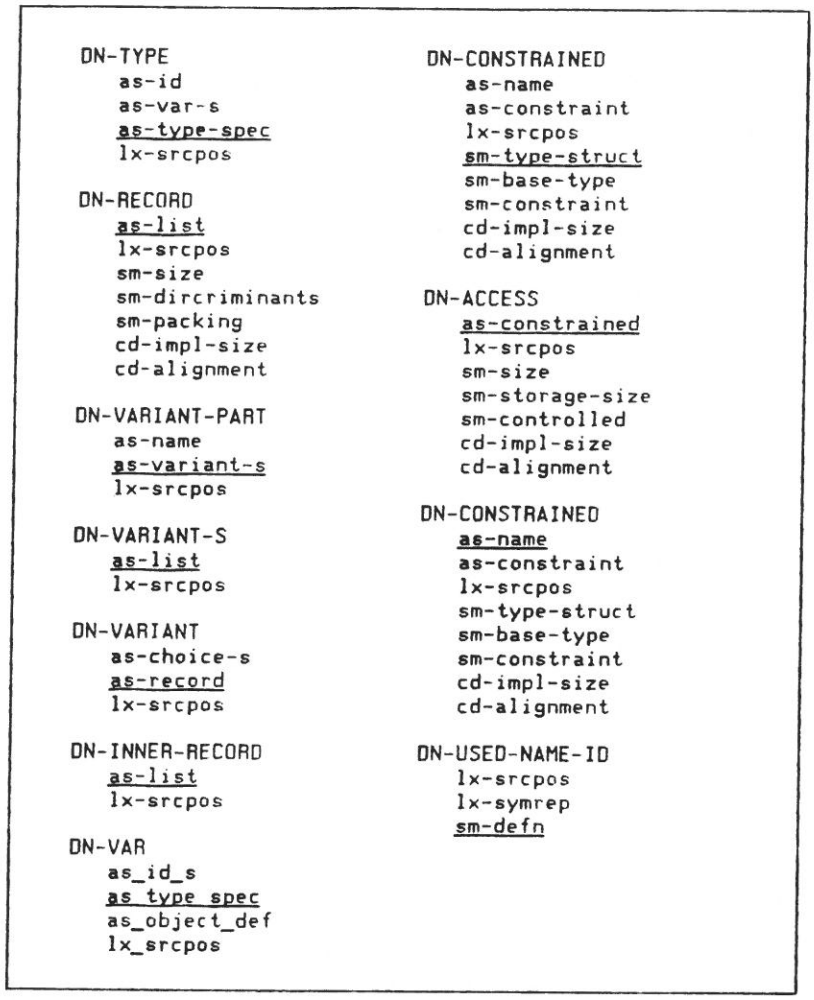

Abb 6: Spurverfolgung COMREC - RELATION in DIANA

3.4.3.3 DIANA-Darstellung im Datenverzeichnis

Das DIANA-Reference-Manual [GOOS+WULF81] schreibt keine Repräsentation der DIANA-Darstellung vor. Daher wäre es beispielsweise möglich, alle Information für ein DVZ in einen Datenverbund zu legen und diesen wie ein Tupel einer Relation zu behandeln. Bei Änderungen der Informationshaltung im DVZ sind allerdings beträchtliche Änderungen in der zugrunde liegenden Datenstruktur notwendig, die eine Änderung aller Beschreibungen im DVZ nach sich zöge.

Ein anderer Weg wäre die Speicherung der DIANA-Darstellung als lesbare ASCII-Repräsentation, so wie auch der ADA-Übersetzer solch eine Liste erstellt. Dieser Weg besitzt den Vorteil erhöhter Wartungsfreundlichkeit. Das Einfügen neuer Knoten und Attribute bei erweitertem Informationsbedürfnis der Datenbank beträfe nur die Beschreibungen der betroffenen Relationen und nicht, wie im vorigen Beispiel, alle Beschreibungen. Die Linearisierung des DIANA-Baumes würde jedoch eine Fülle von Verweisen enthalten, die möglicherweise die Anzahl der relavanten Attribute übersteigt.

Die dritte Lösung ist die Repräsentation des DVZ in Form eines Relationengebildes. Jeder DIANA-Knoten wird durch eine Relation dargestellt, Knotenattribute sind Tupelattribute. Die dem Relationengebilde zu grundeliegende Ordnung ist durch die Referenzierung der DIANA-Typbeschreibungen gegeben. Anfragen nach Kompatibilität von Relationen lassen sich durch QSE bewältigen, wobei die Beispielstruktur aus dem DIANA-Baum des Anwendungsprogrammes gewonnen wird.

3.4.3.4 Reduktion des ursprünglichen DIANA-Baumes

Die Übernahme der vollständigen, vom ADA-Übersetzer erzeugten DIANA-Struktur ist wegen des gewaltigen Umfanges, der nicht zuletzt durch die mannigfaltigen Redundanzen zur schnellen Informationsgewinnung entsteht, nicht möglich. Zudem enthält die Originaldarstellung Knoten und Attribute, welche der Codeerzeugung dienen, nicht aber in einem DVZ enthalten sein müssen. Es ist also notwendig, den ursprünglichen DIANA-Baum auf ein Minimum, d.h. auf alle typspezifizierenden Knoten und Attribute, zu reduzieren. Die Definition der benötigten Knotenklassen und der dazugehörigen Knoten mit ihren Attributen sind, der Notation von DIANA in IDL folgend, in den Abbildungen 7 und 8 dargestellt.

Darstellungsspezifische Informationen der Datenbank, wie Verweise auf virtuelle Felder und die daraus abweichende Speichergröße eines Tupels in der Datenbank gegenüber der Speichergröße im Hauptspeicher, sind durch Einfügen neuer, DIANA-kompatibler Knoten und Attribute darstellbar. Im Wesentlichen beziehen sich derartige Erweiterungen auf den in ADA nicht vorhandenen Datentyp "RELATION" und die damit verbundenen Referenzen. Diese Erweiterungen sind ebenfalls in den genannten Abbildungen ersichtlich.

```
COMP            ::= var | null_comp | variant_part

DB_ACCESS       ::= standardkey | void

DEF_OCCURRENCE::= DEF_ID

DEF_ID          ::= var_id | type_id | subtype_id |
                    private_type_id |
                    l_private_type_id

DESIGNATOR      ::= ID

DSCRT_RANGE     ::= constrained | range | index

DSCRT_RANGE_S ::= dscrt_range_s

EXP             ::= numeric_literal | string_literal

ID              ::= DEF_ID | USED_ID

INNER_RECORD    ::= inner_record

NAME            ::= DESIGNATOR

RANGE           ::= range

RANGE_VOID      ::= RANGE | void

RELATION        ::= relation | record

TYPE_SPEC       ::= integer | fixed | float |
                    enum_literal_s | record |
                    array | reference | constrained |
                    derived | void

USED_ID         ::= used_name_id

VAR             ::= var

VARIANT         ::= variant

VARIANT_S       ::= variant_s

VAR_S           ::= var_s | void
```

Abb 7: Definition der DIANA-Knotenklassen

```
array              =>  as_dscrt_range_s :  DSCRT_RANGE_S
                       as_constrained   :  constrained

constrained        =>  as_name          :  NAME
                       sm_base_type     :  TYPE_SPEC
                       sm_type_struct   :  TYPE_SPEC

comp_id            =>  lx_symrep        :  symbol_rep

derived            =>  as_constrained   :  constrained

dscrt_range_s      =>  as_list          :  seq. of DSCRT_RANGE

id_s               =>  as_list          :  seq. of ID

index              =>  as_name          :  NAME

inner_record       =>  as_list          :  seq. of COMP

integer            =>  as_range         :  RANGE

fixed              =>  as_range_void    :  RANGE_VOID
                       as_actual_delta  :  float

float              =>  as_range_void    :  RANGE_VOID

l_private_type_id  =>  lx_symrep        :  symbol_rep
                       sm_type_spec     :  TYPE_SPEC

private_type_id    =>  lx_symrep        :  symbol_rep
                       sm_type_spec     :  TYPE_SPEC

range              =>  as_exp1          :  EXP
                       as_exp2          :  EXP
range              =>  sm_base_type     :  TYPE_SPEC

record             =>  as_list          :  seq. of COMP
record             =>  sm_discriminants :  VAR_S

reference          =>  db_access        :  DB_ACCESS

relation           =>  db_rel_desc      :  record
                       db_virt_s        :  seq. of COMP
                       db_storage_size  :  integer

standardkey        =>  db_reference     :  integer

subtype_id         =>  lx_symrep        :  symbol_rep
                       sm_type_spec     :  CONSTRAINED

type_id            =>  lx_symrep        :  symbol_rep
                       sm_type_spec     :  TYPE_SPEC

used_name_id       =>  lx_symrep        :  symbol_rep

var                =>  as_id_seq        :  id_s
                       as_type_spec     :  TYPE_SPEC

variant            =>  as_record        :  INNER_RECORD

variant_part       =>  as_variant_s     :  VARIANT_S

variant_s          =>  as_list          :  seq. of VARIANT

var_id             =>  lx_symrep        :  symbol_rep

var_s              =>  as_list          :  seq. of VAR

void               =>  ;
```

Abb 8: Definition der DIANA-Knoten

4.0

Stand des Projektes

Die Realisierung des Datenbankkonzeptes stieß trotz gründlicher Vorbereitung auf erhebliche Schwierigkeiten im Bereich der Systemprogrammierung. Erst die Bereitstellung neuester Programmwerkzeuge durch den Rechnerhersteller ermöglichte die Erstellung eines Datenbankprototyps, wie er zur Zeit verwendet werden kann. Die bislang realisierten Datenbankfunktionen beschränken sich auf Tupeloperationen, wie GET, PUT, DELETE und MODIFY. Eine Anforderung von Relationengebilden nach QSE bedarf des vollständigen Datenverzeichnisses, welches erst in rudimentärer Form implementiert ist.

Der Version 0.0 genannte Prototyp ist in einem PASCAL-Dialekt realisiert, der es erlaubt, mehrere quasiparallele Prozesse auf einem Prozessor ablaufen zu lassen. Damit konnten der Datenbankkern in vier einzelnen Prozessen realisiert werden, deren Aufgaben wie folgt verteilt sind:

* Ein Prozess regelt den Verkehr mit dem Rechnernetz, d.h. er ist für die Annahme und Weiterleitung von Aufträgen sowie für die Übermittlung von Status- und Fehlermeldungen an das Anwendungsprogramm zuständig.

* Ein weiterer Prozess regelt den Verkehr mit dem Sekundärspeicher und realisiert die genannten Grundfunktionen. Er stellt sozusagen das interne Organisationszentrum dar.

* Zwei Prozesse sorgen für Ein- und Ausgabe zur Kontrolle der erstgenannten. Hierfür sind, bedingt durch die besondere technische Struktur der Maschinen und die Tatsache, daß nur Sichtgeräte im Halbduplexverkehr unterstützt werden, zwei Prozesse notwendig gewesen, da trotz laufender Kontrolleingaben aktuelle Fehlermeldungen sofort auf dem Bildschirm gemeldet werden sollen.

PASCAL wurde gewählt, weil zu damaliger Zeit der unter der Leitung von H.-H. Nagel entwickelte Codegenerator für das in Karlsruhe unter G. Goos erstellte Übersetzer-Front-End nicht verfügbar war und eine Umcodierung nach ADA als wenig arbeitsintensiv angesehen wurde.

5.0

Perspektiven der Weiterentwicklung

Aktuelle Ziele der Weiterentwicklung sind zur Zeit

* die UMCODIERUNG des Prototyps nach ADA, da der genannte ADA-Übersetzer in Kürze voll einsetzbar sein wird,
* die Weiterentwicklung des DATENVERZEICHNISSES nach den im Abschnitt 3.4 genannten Anforderungen,
* die Erweiterung des doch recht starren Prinzips systemvergebener Primärschlüssel in die Richtung FREIDEFINIERTER SCHLÜSSELATTRIBUTE und
* die Implementation von QSE.

Langfristiges Ziel der Entwicklung ist:

* Vergleiche zwischen Relationengebilden verschiedener Szenen,
* Vergleiche zwischen Relationengebilden in Szenen erkannter Bildobjekte und in der Datenbank enthaltener Prototypen zu ermöglichen sowie
* Ähnlichkeitsanfragen zwischen Relationengebilden und
* Anfragen nach unvollständigen, etwa durch Verdeckung von Objektteilen reduzierten Gebilden.

6.0

Literaturangaben

[ABBOTT83]
Russell J. Abbott
PROGRAM DESIGN BY INFORMAL ENGLISH DESCRIPTIONS
CACM, No. 11, November 1983, pp. 882 - 894

[ADA80]
REFERENCE MANUAL FOR THE ADA PROGRAMMING LANGUAGE
United States Department of Defense, July 1980

[ADA83]
American National Standards Institute, Inc.
MILITARY STANDARD
ADA PROGRAMMING LANGUAGE
ANSI/MIL-STD-1815A, January 1983

[AKERSTEN80]
Ingvar S. Akersten
PIXLIB ON PDP 11/34: SYSTEMS DOCUMENTATION
FOA Rapport C30191-E1, Linkoeping, April 1980

[BENN81-83]
Wolfgang Benn
INTERNE PROJEKTBERICHTE Nr. 1 bis 7
FB Informatik, Universität Hamburg
1981 - 1983

[BENN+FAASCH81]
Wolfgang Benn und Helmut Faasch
AUSWEITUNG EINER PASCAL-IMPLEMENTATION AUF EIN
LABORNETZWERK VON MINCALx2
Diplomarbeit am Fachbereich Informatik der Universitaet
Hamburg, Maerz 1981

[BEVER+82]
M. Bever, M. Dausmann, S. Drossopoulou, W. Kirchgaessner,
P.C. Lockemann, G. Persch and G. Winterstein
THE INTEGRATION OF EXISTING DATABASE SYSTEMS IN AN ADA ENVIRONMENT
ADATEC Tutorial and Conference on ADA,
Arlington, Virginia, October 4-8, 1982, pp. 162 - 170

[BILLINGSLEY80]
Fred C. Billingsley
DATA BASE SYSTEMS FOR REMOTE SENSING
in [BLASER80] pp. 299-318

[BLASER80]
A. Blaser (Editor)
DATABASE TECHNIQUES FOR PICTORIAL APPLICATIONS
Florence, June 20.-22. 1979
Lecture Notes in Computer Science 81
Springer Verlag, Heidelberg 1980

[BLASER+81]
A. Blaser, H. Eberle, R. Erbe, H. Lehmann, G. Mueller,
U. Schauer und H. Schmutz
INTEGRATED DATA ANALYSIS AND MANAGEMENT SYSTEM
FEATURE DESCRIPTION
IBM-Germany, TR 81.07.005, July 1981

[BURGUENO80]
Juan Fco. Corona Burgueno
A GEOGRAPHICAL DATA BASE
in [BLASER80] pp. 347-363

[CHAN+83]
Arvola Chan, Umeshwar Dayal, Stephen Fox, Nathan Goodman,
Daniel R. Ries and Dale Skeen
OVERVIEW OF AN ADA COMPATIBLE DISTRIBUTED DATABASE MANAGER
in [DEWITT+GARDARIN83], pp. 228 - 237

[CHANG81]
Ning-San Chang
IMAGE ANALYSIS AND IMAGE DATABASE MANAGEMENT
Computer Science, Nr.9
UMI RESEARCH PRESS, Ann Arbor, Michigan 48106, 1981

[CHANG+77]
Chang, Donato, B.H. McCormick, J.L. Reuss und Rochetti
A RELATIONAL DATA-BASE SYSTEM FOR PICTURES
Proc. IEEE Workshop on PDDM, April 1977, pp.142-149

[CHANG+FU80]
Shi-Kuo Chang und King-Sun Fu (ed.)
PICTORIAL INFORMATION SYSTEMS
Lecture Notes in Computer Science 80
Springer-Verlag, Heidelberg 1980

[CHANG+FU80a]
Ning-San Chang und King-Sun Fu
A RELATIONAL DATABASE SYSTEM FOR IMAGES
in [CHANG+FU80], pp. 288 ff

[CHANG+FU80b]
Ning-San Chang und King-Sun Fu
A QUERY LANGUAGE FOR RELATIONAL IMAGE DATABASE SYSTEMS
Proc. IEEE Workshop on PDDM, August 1980, pp. 67-73

[CHANG+FU80c]
Ning-San Chang und King-Sun Fu
QUERY-BY-PICTORIAL-EXAMPLE
IEEE Transact. on SE, November 1980, pp 519-524

[CHANG+FU81]
Ning-San Chang und King-Sun Fu
AN INTEGRATED IMAGE ANALYSIS AND IMAGE DATABASE
MANAGEMENT SYSTEM
Proc. COMPCON Fall 81, Washington D.C.,
September 1981

[CHANG+FU81a]
Ning-San Chang und King-Sun Fu
PICTURE QUERY LANGUAGES FOR PICTORIAL DATA-BASE SYSTEMS
COMPUTER, Nr.14, pp. 23-33, November 1981

[CHANG+KUNII81]
Shi-Kuo Chang und Tosiyasu L. Kunii
PICTORIAL DATABASE SYSTEMS
Computer, November 1981, pp. 13-21

[CHOCK82]
Margaret Irvine Chock
A DATA BASE MANAGEMENT SYSTEM FOR IMAGE PROCESSING
Dissertation an der University of California
Los Angeles, 1982

[CHOCK+81]
Margaret Chock, Alfonso F. Cardenas und Allen Klinger
MANIPULATING DATA STRUCTURES IN PICTORIAL
INFORMATION SYSTEMS
Computer, November 1981, pp. 43-50

[CODD82]
Edgar F. Codd
RELATIONAL DATABASE : A PRACTICAL FOUNDATION FOR PRODUCTIVITY
CACM, Vol.25, February 1982

[DEWITT+GARDARIN83]
David J. DeWitt and Georges Gardarin (Editors)
SIGMOD 83, PROCEEDINGS OF ANNUAL MEETING
Database Week, San Jose, 23.-26. May 1983
ACM-SIGMOD-Record, Vol. 13, No. 4

[DRUFFEL82]
Larry E. Druffel
THE POTENTIAL EFFECT OF ADA ON SOFTWARE ENGINEERING
IN THE 1980'S
ACM-Software Engineering Notes, July 1982, pp. 5-11

[ENCARNACAO+NEUMANN80]
J. Encarnacao und T. Neumann
A SURVEY OF DB REQUIREMENTS FOR GRAPHICAL
APPLICATIONS IN ENGINEERING
in [BLASER80] pp. 285-297

[FENG81]
Tse-Yun Feng
A VERY LARGE DATA BASE COMPUTER
IEEE Worksh. on Comp. Arch. for Pattern Analysis and
Image Database Management, Hot Springs,
November 1981, pp. 12-24

[FRIDELL+80]
M. Fridell, R. Carling, D.Kramlich und Ch.F. Herot
THE MANAGEMENT OF VERY LARGE TWO-DIMENSIONAL
RASTER GRAPHICS ENVIRONMENTS
Proc. IEEE Workshop on PDDM, August 1980,

[GOOS+WULF81]
G. Goos and W. A. Wulf (Editors)
DIANA REFERENCE MANUAL
Bericht 1/81, Institut fuer Informatik II
Universitaet Karlsruhe, Maerz 1981

[HALL83]
Patrick A.V. Hall
ADDING DATABASE MANAGEMENT TO ADA
ACM-SIGMOD-Record, Vol. 13, No. 3, April 1983, pp. 13 - 17

[KUNII+74]
Tosiyasu L. Kunii, Weyl und Tenenbaum
A RELATIONAL DATABASE SCHEME FOR DESCRIBING
COMPUTER PICTURES WITH COLOUR AND TEXTURE
Proc. of the Second IJCPR
Lyngby-Copenhagen, August 1974, pp. 310-316

[LORIE81]
Reymond A. Lorie
ISSUES IN DATABASE FOR DESIGN APPLICATIONS
Research Report (RJ3176)
IBM Res. Lab., San Jose, California 95193, October 1981

[RADIG82]
Bernd Radig
SYMBOLISCHE BESCHREIBUNG VON BILDFOLGEN I:
RELATIONENGEBILDE UND MORPHISMEN
Bericht des Fachbereichs Informatik der Universitaet
Hamburg, 1982

[ROUSSOPOULOS82]
Nicholas Roussopoulos
VIEW INDEXING IN RELATIONAL DATABASES
ACM-TODS, Vol. 7, No. 2, June 1982, pp. 258 - 290

[SCHMIDT+MALL80]
Joachim W. Schmidt und Manuel Mall
PASCAL/R REPORT
IFI-HH-B-66/80
Bericht Nr. 66 des Fachbereiches Informatik
der Universität Hamburg
Januar 1980

[WOLFE80]
Robert N. Wolfe
3D GEOMETRIC DATABASES FOR MECHANICAL ENGINEERING
in [BLASER80] pp.253-261

[ZOBRIST+NAGY81]
Albert N. Zobrist und George Nagy
PICTORIAL INFORMATION PROCESSING OF LANDSAT DATA
FOR GEOGRAPHIC ANALYSIS
Computer, November 1981, pp. 34-41