Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
des Fachbereichs Wirtschaftswissenschaften
der Universität Osnabrück

OBJECT WAREHOUSE

Konzeption der Basis objektorientierter Management Support Systems am Beispiel von Smalltalk und dem ERP Baan

vorgelegt von

Dipl.-Kfm. Eitel von Maur

am 29. Mai 2000

Dekan des Fachbereichs:	Prof. Dr. B. Meyer
Referent:	Prof. Dr. B. Rieger
Korreferent:	Prof. Dr. U. Hoppe
Tag der mündlichen Prüfung:	27. Juli 2000

Für Rainer Mager und
meine Freunde von der Probebühne.

GLIEDERUNG

GLIEDERUNG ..... **V**

INHALTSVERZEICHNIS ..... **VII**

ABBILDUNGS- UND TABELLENVERZEICHNIS ..... **XI**

ABKÜRZUNGSVERZEICHNIS ..... **XV**

VORWORT ..... **XXI**

KAPITEL 1 **MOTIVATION UND ERKENNTNISZIEL** **1**

TEIL I **ENTWICKLUNG EINER OBJECT-WAREHOUSE-KONZEPTION** **11**

KAPITEL 2 **DATA WAREHOUSE** **13**

2.1 Management Support Systems 13
2.2 Entwicklungspfad des Data-Warehouse-Konzepts 36
2.3 Data-Warehouse-Konzept 47
2.4 Data-Warehouse-Architektur 58
2.5 Kritik an bisherigen Data-Warehouse-Ansätzen 68

KAPITEL 3 **OBJEKT-PARADIGMA** **79**

3.1 Begriff des Software-Paradigma 80
3.2 Software-Paradigma-Zielsetzung 82
3.3 Entwicklungs-Phasen 87
3.4 Applikations-Schichten 127
3.5 Weitere Objekt-Paradigmen 191

KAPITEL 4 OBJECT WAREHOUSE 197
4.1 Grundidee 198
4.2 Architektur 200
4.3 Quell-Schichten 203
4.4 Object-Warehouse-Kern 204
4.5 Anwendungs-orientierte Schichten 211

TEIL II ENTWICKLUNG EINES OBJECT-WAREHOUSE-PROTOTYPEN 217

KAPITEL 5 PROTOTYP-KONZEPTION 219
5.1 Praxisprojekte und praktische Erfahrungen 219
5.2 Informationsbedarfsanalyse 221
5.3 Datenquellenanalyse 225
5.4 Entwicklung des Domänen-Modells 233

KAPITEL 6 PROTOTYP-IMPLEMENTIERUNG 237
6.1 Vorgehen 237
6.2 Export der ASCII-Dateien aus TBase 239
6.3 Import der ASCII-Dateien in Oracle 243
6.4 Import der Oracle-Tabellen in VisualWorks 244
6.5 GemStone-Objektmodell 248
6.6 Objekt-Transformation 257

KAPITEL 7 KONKLUSION UND PERSPEKTIVEN 261

LITERATURVERZEICHNIS 265

INHALTSVERZEICHNIS

GLIEDERUNG ..... **V**

INHALTSVERZEICHNIS ..... **VII**

ABBILDUNGS- UND TABELLENVERZEICHNIS ..... **XI**

ABKÜRZUNGSVERZEICHNIS ..... **XV**

VORWORT ..... **XXI**

KAPITEL 1 **MOTIVATION UND ERKENNTNISZIEL** **1**

TEIL I **ENTWICKLUNG EINER OBJECT-WAREHOUSE-KONZEPTION** **11**

KAPITEL 2 **DATA WAREHOUSE** **13**

2.1 Management Support Systems 13
2.1.1 Anwenderklassen-spezifische Klassifikation 14
2.1.2 Problemklassen-spezifische Klassifikation 16
2.1.3 Produktarten-spezifische Klassifikation 20
2.1.4 Wissensbasierte Klassifikation 27
2.1.5 Neue Begriffe im MSS-Umfeld 27
2.1.6 MSS-Definition 33
2.2 Entwicklungspfad des Data-Warehouse-Konzepts 36
2.2.1 Informationstechnologie zur Prozessautomatisierung 37
2.2.2 Software-Engineering und Datenorientierung 38
2.2.3 Benutzer-Fokussierung und Dezentralisierung 39
2.2.4 Client/Server-Strukturierung 41
2.2.5 Wettbewerbsfaktor Information 42
2.2.6 Data Warehouse 46

2.3 Data-Warehouse-Konzept 47
2.3.1 Data-Warehouse-Daten 49
2.3.2 Data-Warehouse-Datenmodell 53
2.3.3 Data-Warehouse-Engineering 57
2.4 Data-Warehouse-Architektur 58
2.4.1 Quell-Schichten 58
2.4.2 Data-Warehouse-Kern 62
2.4.3 Anwendungs-orientierte Schichten 65
2.5 Kritik an bisherigen Data-Warehouse-Ansätzen 68
2.5.1 Quell-Schichten 69
2.5.2 Data-Warehouse-Kern 71
2.5.3 Anwendungs-orientierte Schichten 74
2.5.4 Data-Warehouse-Engineering 77

KAPITEL 3 OBJEKT-PARADIGMA 79

3.1 Begriff des Software-Paradigma 80
3.2 Software-Paradigma-Zielsetzung 82
3.2.1 Systems Engineering 83
3.2.2 Software-Paradigma-Qualität 84
3.2.3 Software-Qualität 85
3.3 Entwicklungs-Phasen 87
3.3.1 Bezugsrahmen 88
3.3.2 Objektorientierte Analyse 89
3.3.2.1 Gegenstand 89
3.3.2.2 Zielsetzung 90
3.3.2.3 Grundidee 91
3.3.2.4 Grundkonzepte 91
3.3.2.5 Vorgehensweise 96
3.3.2.6 Fazit 100
3.3.3 Objektorientiertes Design 101
3.3.3.1 Gegenstand 101
3.3.3.2 Design-Heuristiken 103
3.3.3.3 Design-Patterns 106
3.3.3.4 Abgrenzung 108
3.3.4 Objektorientierte Programmierung 111
3.3.4.1 Gegenstand 111
3.3.4.2 Entwicklungspfad 111
3.3.4.3 Grundkonzepte 112
3.3.4.4 Smalltalk-Konzepte 115
3.3.4.5 Typen, Klassen und Vererbung 117
3.3.4.6 Abgrenzung 122
3.3.5 Vorgehens-Methodik 124
3.4 Applikations-Schichten 127
3.4.1 Verteilung 127
3.4.2 Grafische Benutzeroberflächen 128
3.4.2.1 Konzeptionell objektorientiert 129
3.4.2.2 Objektorientiert realisiert 132
3.4.3 Objectbase Management Systems 135
3.4.3.1 OOP-Blickwinkel 136
3.4.3.2 DBMS-Blickwinkel 139
3.4.3.3 Evolution und Versionierung 148
3.4.3.4 Sicht-Konzept 156
3.4.3.5 Persistenzmodelle 159

3.4.3.6 Typen, Klassen und Vererbung 161
3.4.3.7 Object Query Language 166
3.4.3.8 Architektur 180
3.4.3.9 Begriff 188
3.4.4 Applikations-Schicht 189
3.5 Weitere Objekt-Paradigmen 191
3.5.1 Künstliche Intelligenz 191
3.5.2 Organisation 194

KAPITEL 4 OBJECT WAREHOUSE 197
4.1 Grundidee 198
4.2 Architektur 200
4.3 Quell-Schichten 203
4.4 Object-Warehouse-Kern 204
4.5 Anwendungs-orientierte Schichten 211

TEIL II ENTWICKLUNG EINES OBJECT-WAREHOUSE-PROTOTYPEN 217

KAPITEL 5 PROTOTYP-KONZEPTION 219
5.1 Praxisprojekte und praktische Erfahrungen 219
5.2 Informationsbedarfsanalyse 221
5.2.1 Vorgehen und Ergebnisse 221
5.2.2 Informationsbedarf der Drogeriemarktkette 222
5.2.3 Informationsbedarf des Landmaschinenherstellers 223
5.2.4 Fazit 224
5.3 Datenquellenanalyse 225
5.3.1 Vorgehen und Ergebnisse 225
5.3.2 Datenquellenanalyse am Beispiel der Baan-Daten 226
5.3.3 Datenquellenanalyse am Beispiel der Scannerdaten 230
5.4 Entwicklung des Domänen-Modells 233

KAPITEL 6 PROTOTYP-IMPLEMENTIERUNG 237
6.1 Vorgehen 237
6.2 Export der ASCII-Dateien aus TBase 239
6.3 Import der ASCII-Dateien in Oracle 243
6.4 Import der Oracle-Tabellen in VisualWorks 244
6.5 GemStone-Objektmodell 248
6.6 Objekt-Transformation 257

KAPITEL 7 KONKLUSION UND PERSPEKTIVEN 261

LITERATURVERZEICHNIS 265

ABBILDUNGS- UND TABELLENVERZEICHNIS

Abbildung 1: MSS-Klassifikation nach *Chamoni*, *Gabriel* und *Gluchowski* [ChGl99b, S. 9] 17

Abbildung 2: MSS-Klassifikation nach *Krallmann*, *Mertens* und *Schiemann* [KMS97, S. 150] 18

Abbildung 3: MSS-Klassifikation nach *Rieger* [Rieg93] 18

Abbildung 4: *Riegers* organisatorische DSS/EIS-Einordnung [Rieg93] 19

Abbildung 5: Einordnung neuer Begriffe im MSS-Umfeld 33

Abbildung 6: Definition des Begriffs Management Support Systems 34

Abbildung 7: Informationsobjekt im Kontext (Informations-Facette) [Rieg93] 52

Abbildung 8: Data-Warehouse-Architektur 60

Abbildung 9: Coad/Yourdon-Notation im Klassendiagramm [CNM97, Einband] 93

Abbildung 10: Klassifizierung von Objekt-Sprachen nach *Wegner* [Wegn87, S. 169] 113

Abbildung 11: Taxonomie der Schema-Evolution in OBMS nach *Bertino* und *Martino* [BeMa94, S. 129 ff.] 152

Abbildung 12: Programmiersprachen- und Datenbank-Einkapselung im Vergleich [Heue97, S. 286] 165

Abbildung 13: Drei-Ebenen-Schema-Architektur (ANSI/SPARC) für OBMS nach *Khoshafian* [Khos93, S. 111] 181

Abbildung 14: OBMS-Definition 188

Abbildung 15: Drei-Schichten-Architektur einer objektorientierten Applikation 190

Abbildung 16: Generisches Unternehmens-Objektmodell (GUOM) 199

Abbildung 17: Object-Warehouse-Architektur 202

Abbildung 18: Die MSS-relevanten Artikel-Attribute der Ursprungstabelle „tiitm001“ am Beispiel eines Artikel-Objekts im Object Warehouse des Landmaschinenherstellers 207

Abbildung 19: Klassendiagramm eines Object-OLAP-Marts [Ohle98, S. 232] 213

Abbildung 20: Darstellung eines möglichen Object-OLAP-Marts auf Instanz-Ebene [Ohle98, S. 233] 214

Abbildung 21: Überführung einer Objektstruktur des ROL in ein Relationen-Modell eines Data Mart am Beispiel eines einfachen Kunden-Objekts 215

Abbildung 22: Eine erste Analyse möglicher Attibutwertausprägungen und deren Häufigkeit am Beispiel der Installationen-Tabelle „tssma102“ mit Forest & Trees 228

Abbildung 23: Relevantes Datenmodell aus TBase von Baan/Triton 3.1b 229

Abbildung 24: Ausschnitt aus einer Journaldatei der Scannerkassen des Drogeriemarktprojekts 230

Abbildung 25: Grafische Auswertung der Satzartfolgen einer Journaldatei 233

Abbildung 26: Datenmodell in Oracle mit den relevanten Attributen 235

Abbildung 27: Grundmodell des Reconciled Object Layer mit den relevanten Attributen und Methoden 236

Abbildung 28: Alternative Transformations-Wege der TBase-Daten nach GemStone 238

Abbildung 29: Maintain Exchange Schemes 240

Abbildung 30: Maintain ASCII Files Teil 1 240

Abbildung 31: Maintain ASCII Files Teil 2 241

Abbildung 32: Maintain ASCII File Formats 241

Abbildung 33: Maintain Batches 242

Abbildung 34: Maintain Table Relations (Export) am Beispiel des Batch DIM 242

Abbildung 35: Generierung der Klassendefinition aus Oracle-Tabellen 245

Abbildung 36: Aus dem Oracle-Datenmodell generiertes Smalltalk-Objektmodell (Ausschnitt) 246

Abbildung 37: Mapping zwischen einer Oracle-Tabelle und der dazugeörigen Smalltalk-Klasse 246

Abbildung 38: Generierung einer Oracle-Query 247

Abbildung 39: Klassendefinition des Objektmodells in GemStone 249
Abbildung 40: Methode zur Generierung einer Zugriffsmethode für eine einwertige binäre Referenz..... 254
Abbildung 41: Vereinfachte Methode zur Generierung einer Zugriffsmethode..... 255
Abbildung 42: Symbol List Browser mit Dictionaries in GemStone 255
Abbildung 43: Segment Browser zur Definition der Objektrechte 256
Abbildung 44: GemBuilder Connector Browser 258
Abbildung 45: GemStone Inspector mit den transformierten GemStone-Objekten 259

Tabelle 1: Die Attribute der Artikel-Tabelle „tiitm001“ des Landmaschinenherstellers im ERP Baan..... 206
Tabelle 2: Stückanalyse im Projekt Drogeriemarkt..... 222
Tabelle 3: Artikelanfrage nach Artikelnummern im Projekt Drogeriemarkt ... 222
Tabelle 4: Bonwertanalyse im Projekt Drogeriemarkt..... 223
Tabelle 5: Relevante Baan-Tabellen des Vertriebsinformationssystems..... 227
Tabelle 6: Satzarthäufigkeiten in 1 bzw. 128 Journaldatei(en)..... 231
Tabelle 7: Auswertung der Satzartfolgen einer Journaldatei..... 232

ABKÜRZUNGSVERZEICHNIS

4GL	Fourth Generation Language
ADT	Abstract Data Type
AIS	analytisches Informationssystem
AKO	A-Kind-Of
ANSI	American National Standard Institute
APL	Application Programming Language
ASCII	American Standard Code of Information Interchange
ATM	Automatic Teller Machine
BDW	Business Data Warehouse
BIT	Business Intelligence Tools
BIW	Business Information Warehouse
BLOB	Binary Large Object
CAD	Computer-Aided Design
CASE	Computer-Aided Software (Systems) Engineering
CEO	Chief Executive Officer
CIF	Corporate Information Factory
CIM	Computer Integrated Manufacturing
CODASYL	Conference on Data Systems Languages, als Bezeichnung für prä-relationale DBMS
CORBA	Common Object Request Broker Architecture
CRC	Classes-Responsibilities-Collaborators

DASD	Direct Access Storage Device
DBA	Database Administrator
DBMS	Database Management System
DBS	Database System
DDL	Data Definition Language
DDL	Derived Data Layer
DFD	Datenfluss-Diagramm
DML	Data Manipulation Language
DOL	Derived Object Layer
DPL	Database Programming Language
DSS	Decision Support System
EAN	European Article Numbering
EBIS	EMEA Business Information System
ECA	Event-Condition-Action-(Regel)
EER	Enhanced Entity Relationship
EIS	Executive Information System
EMEA	IBM Europe, Middle East and Africa
eNF²	extended Non-First-Normal-Form
ERD	Entity Relationship Diagram
ERM	Entity Relationship Model
ERP	Enterprise Resource Planning (System)
ESS	Executive Support System
ETL	Extraction-Transformation-Load
EVA	Eingabe-Verarbeitung-Ausgabe
Gl	Gesellschaft für Informatik
GUDM	generisches Unternehmens-Datenmodell
GUI	Graphical User Interface
GUOM	generisches Unternehmens-Objektmodell
HCI	Human Computer Interaction
HTML	Hypertext Markup Language
ISAM	Index-Sequential Access Mode

ISS	Intelligent Support System
IT	Information Technology
IUK	Informations- und Kommunikations-(Systeme/Technologie)
JPEG	Joint Photographics Expert Group
KDD	Knowledge Discovery in Databases
KI	Künstliche Intelligenz
KPI	Key Performance Indicators
LAN	Local Area Network
MDBMS	Meta Database Management System
MFC	Microsoft Foundation Classes
MIS	Management Information System
MOLAP	Multi-Dimensional OLAP
MPEG	Motion Picture Expert Group
MQE	Managed Query Environment
MSS	Management Support System
MSS-UDM	MSS-Unternehmens-Datenmodell
MSS-UOM	MSS-Unternehmens-Objektmodell
MVC	Model-View-Controller
NF^2	Non-First-Normal-Form
NSD	Nassi-Shneiderman-Diagram
OB	Objectbase
OBMS	Objectbase Management System
OBS	Objectbase System
ODB	Object Database
ODBMS	Object Database Management System
ODL	Object Definition Language
ODMG	Object Data Management Group
ODS	Operational Data Store
OEM	objektorientierte Entwurfsmodelle
OID	Object Identification
OLAP	Online Analytical Processing

OLTP	Online Transaction Processing
OMG	Object Management Group
OML	Object Manipulation Language
OMSS	Object-Oriented Management Support System
OO	Object-Oriented
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OODB	Object-Oriented Database
OODBMS	Object-Oriented Database Management System
OODBS	Object-Oriented Database System
OODDL	Object-Oriented Data Definition Language
OODM	objektorientiertes Datenmodell
OOP	Object-Oriented Programming
OOPL	Object-Oriented Programming Language
OOS	Operational Object Store
OOUI	Object-Oriented User-Interface
OQL	Object Query Language
ORB	Object Request Broker
ORDBMS	Object-Relational Database Management System
OUDM	operatives Unternehmens-Datenmodell
OUOM	operatives Unternehmens-Objektmodell
PIMS	Personal Information Management System
PPS	Produktionsplanungs- und -steuerungssystem
QBE	Query by Example
RAP	rechnerunterstützter Arbeitsplatz
RDBMS	Relational Database Management System
RDL	Reconciled Data Layer
RMOL	Reconciled Meta-Object Layer
ROL	Reconciled Object Layer
ROLAP	Relational OLAP
SA	Structured Analysis

SD	Structured Design
SDL	Storage Definition Language
SDM	Semantic Data Model
SP	Structured Programming
SPARC	Systems Planning and Requirements Committee
SQL	Structured (Standard) Query Language
SSS	Spreadsheet System
TC-Shell	Enhanced Berkeley C-Shell (tcsh)
UDM	Unternehmens-Datenmodell
UML	Unified Modeling Language
UOM	Unternehmens-Objektmodell
VDL	View Definition Language
VSAM	Virtual Storage Access Method
WAN	Wide Area Network
WBS	wissensbasiertes System
WWS	Warenwirtschaftssystem
WWW	World Wide Web
XP	Extreme Programming
XPS	Expert System

VORWORT

Die vorliegende Arbeit entstand in den fünf Jahren meiner Lehr- und Forschungstätigkeit am Fachgebiet BWL/Wirtschaftsinformatik II, des Instituts für Informationsmanagement und Unternehmensführung (IMU) im Fachbereich Wirtschaftswissenschaften der Universität Osnabrück bei *Herrn Prof. Dr. Bodo Rieger*.

Wir wußten von Anfang an, was das Thema dieser Arbeit sein sollte. Der Schwerpunkt des Fachgebiets sind Management Support Systems. Einen weiteren Schwerpunkt sollte die Objektorientierung bilden. Unsere Vision stand fest: wir wollten objektorientierte Management Support Systems (OMSS) entwickeln.

Als anwendungs- und praxisorientiertes Fachgebiet war unser Ziel die unmittelbare Entwicklung von konkreten OMSS für unsere Praxispartner. Wie bereits in den bisherigen Projekten mit konventionellen MSS, sollte anhand von leistungsstarken Prototypen den Unternehmen das MSS-Potenzial verdeutlicht werden.

Wir wußten, dass es der richtige Weg sein würde. Doch es zeigte sich, dass etwas fehlte. Es reichte nicht wie sonst, die Daten aus den in der Regel proprietären Datenspeichern über ASCII-Export oder sogar Drucklisten in eine integrative relationale Datenbasis zu überführen, die damals noch als EIS-Datenbasis bezeichnet wurde, um uns dann über standardisierte Schnittstellen mithilfe von SQL und MSS-Front-Ends auf den eigentlichen Zweck der MSS zu konzentrieren.

Uns fehlte ein DBMS für die Verwaltung der Objekte, MSS-Werkzeuge, Schnittstellen, Standards, eine Abfragesprache wie SQL et cetera. Und es zeigte sich, dass uns nicht wirklich klar war, was konkret unter Objektorientierung verstanden wird und was damit alles machbar ist. Das Zitat von *King* bewies, dass wir mit dieser Erfahrung nicht alleine waren:

> „It's interesting to note, however, that there is considerable disagreement concerning the definition of 'object-oriented.' We know it's a good thing, but not everyone agrees on what it is." [King89, S. 23]

Doch das war kein Problem. Wir lasen unzählige Bücher und Artikel, implementierten verschiedene Prototypen mit Smalltalk, beteiligten uns aktiv in den einschlägigen Newsgroups und Mailinglisten, nahmen an unzähligen Tagungen teil und unterrichteten getreu dem „alten deutschen Sprichwort", das wir bei *Coad* fanden: „You learn the most when you teach!" [CNM97, S. XI].

Es zeigte sich überall dasselbe Phänomen:

- Einerseits konnten wir feststellen, wieviel Euphorie in der OO-Community herrscht, wieviel Spaß es machen kann objektorientierte Analyse zu betreiben und mit Smalltalk hoch-produktiv Qualitätssoftware zu implementieren.
- Andererseits mussten wir lernen, dass dieser Ansatz alles andere als „kinderleicht" ist, die meisten Bücher zwar viel versprechen und motivierend wirken können, für die konkrete Umsetzung allerdings nur wenig hilfreich sind (immer wieder dieselben nichtssagenden Beispiele, wie z. B. ATM), der Begriff Objektorientierung für völlig unterschiedliche Dinge verwendet wird und auch innerhalb der einzelnen Gebiete nur für wenige der Bereiche klare Konzepte oder gar einheitliche Begriffswelten existieren.

Dementsprechend meint auch *Heuer*:

> „Objektorientierung scheint in der Informatik zum großen Schlagwort der neunziger Jahre zu werden. Benutzeroberflächen werden objektorientiert, ebenso wie das 'Software Engineering', Programmiersprachen und natürlich auch Datenbanksysteme. Was sind nun aber objektorientierte Systeme? In Bezug auf Benutzeroberflächen scheint es schon zu reichen, dass man irgendwelche Objekte auf dem Bildschirm 'anklicken' kann und dann ein Menü mit zur Verfügung stehenden Funktionen aufklappt. Anbietern, die seit Jahren als Marktführer auf dem Gebiet relationaler Datenbanksysteme gelten, verkaufen die gleichen Systeme mit wenigen Erweiterungen plötzlich als objektorientiert. […]
>
> Durch die Vielzahl von Veröffentlichungen aus unterschiedlichsten Forschungsrichtungen gibt es im Bereich der objektorientierten Datenbanksysteme dann auch — zu allem Überfluss — eine babylonische Sprachverwirrung. Einige kleine Beispiele sollen diese Behauptung belegen. In objektorientierten Systemen geht es immer um Klassen und Typen: in einigen Veröffentlichungen werden nur Klassen erwähnt, in anderen nur Typen, die Definitionen stimmen dann teilweise überein, oder werden sogar über „Kreuz" vertauscht. Ähnliche Begriffsverwirrung gibt es mit der Definition von Klassen- oder Typ-Hierarchien und der Bedeutung von Extension, Instanz und Ausprägung. Teilweise entwickeln sich 'Religionskriege' zwischen verschiedenen Gruppierungen. […] Für die Systementwicklung gibt es meist völlig unterschiedliche Ansätze, die noch nicht einmal ohne weiteres vergleichbar sind." [Heue97]

Als Folge entschlossen wir uns, den Schwerpunkt dieser Arbeit auf die grundlegenden Konzepte einer OMSS-Konzeption zu legen, d. h. wie eine solche Architektur überhaupt aussehen und die Basis einer OMSS-Architektur, also die Objektbasis umgesetzt werden könnte.

Vor allen anderen danke ich *Herrn Prof. Dr. Bodo Rieger*, der mir zu jeder Zeit die Freiheiten und die notwendige Unterstützung gab, um meine Ideen verwirklichen zu können. Ohne seinen Einsatz hätte diese Arbeit nicht in dieser Form gelingen können. Er half mir in mancher Situation, sowohl fachlich wie auch persönlich, selbst wenn dazu der Einsatz von Hammer und Schraubenzieher notwendig wurde.

Dem Korreferenten, *Herrn Prof. Dr. Uwe Hoppe*, danke ich für die Erfahrungen, die ich durch ihn machen konnte und es mir ermöglichten neue Perspektiven für meinen Arbeits- und Lebensweg zu gewinnen. Sicherlich werde ich mich über die Jahre immer wieder daran erinnern. *Herrn Prof. Dr. Peter Machemer* danke ich vor allem für seine bewiesene Menschlichkeit, die mir half manche Erfahrung zu verarbeiten. Er hat mir durch die Nebenfachprüfung im Bereich Psychotherapie und Buddhismus nicht nur äusserst spannende und bewegende Inhalte vermittelt, sondern wird mir durch seine Verhaltensweise im Promotionsverfahren auch Vorbild für meine eigene akademische Laufbahn sein.

Weiterhin möchte ich mich bei den *Professoren Jörg Becker*, *Dieter Ehrenberg, Joachim Griese*, *Alexander Hars*, *Friedrich Roithmayr* und *August-Wilhelm Scheer* für die konkreten Hinweise und wertvollen Anregungen zur vorliegenden Arbeit im Rahmen des Doktorandenseminars zur WI'99 im Internationalen Begegnungs- und Forschungszentrum für Informatik (IBFI) auf Schloß Dagstuhl bedanken. Besonderen Einfluss auf die Kapitel zwei und drei hatte *Herr Scheer*, der uns mit seiner deutlichen Kritik an Kollegen mit mangelhaftem Grundlagenwissen dazu aufforderte, uns verstärkt den grundlegenden Konzepten zu widmen. Das erste Kapitel und die Gliederung wesentlich geprägt hat *Herr Becker* mit seinem engagierten Plädoyer für eine wissenschaftstheoretische Einordnung der Arbeit und seinen Vorschlägen für das Vorgehen bei der Untersuchung der Object-Warehouse-Konzeption.

Herrn Prof. Dr. Hermann Krallmann danke ich für die Möglichkeit der vierjährigen Lehrtätigkeit am Fachgebiet Systemanalyse und EDV der Technischen Universität Berlin, der mein wesentliches Wissens-Fundament im Bereich der Wirtschaftsinformatik entstammt. Dafür danke ich auch besonders meinen ehemaligen Kollegen, *Dipl.-Kfm. Lars Zimmermann* und *Dipl.-Inform. Oliver Kundt*, für die endlosen Diskussionen über Algorithmen, RDBMS et cetera und nicht selten über die Frage, was denn nun unter dem Objekt-Paradigma zu verstehen sei, sowie meinem ehemaligen Betreuer *Dipl.-Kfm. Dipl.-Hdl. Helmut Frank*.

Meinen Kollegen in Osnabrück, *Dipl.-Kffr. Anja Mentrup*, *Dipl.-Kfm. Stephan Postert*, *Dipl.-Kfm. Dietmar Krüger*, *Dipl.-Kfm. Karsten Brodmann*, *Frau Jutta Stelter* und *Frau Ursula Bertels*, bin ich zu ganz besonderem Dank verpflichtet. Sie alle haben mich in den Jahren fortwährend unterstützt, haben mich mit Literatur versorgt, haben sich immer wieder meine Ideen angehört, meine Artikel gelesen und Prototypen kritisiert und haben nicht selten dazu beigetragen, dass es mir gelang, Irrwege noch rechtzeitig erkennen zu können.

Für die letztmalige Überprüfung der Orthographie danke ich meinem lieben Patenonkel *Peter Emrich*, der einfach nicht verstehen kann, warum nicht auch die restlichen Worte dieser Arbeit in englischer Sprache verfasst sind und meiner Mutter *Hiltraud von Maur*.

Zuletzt möchte ich meinen Studenten danken, vor allem jenen engagierten, die bei den Praxisprojekten dabei waren. Gerade die Rückkopplung durch die Studierenden und die Motivation durch die Lehrveranstaltungen haben das Object-Warehouse-Konzept vorangebracht.

Eitel von Maur, St. Gallen im Oktober 2000

KAPITEL 1

MOTIVATION UND ERKENNTNISZIEL

Das Thema dieser Arbeit ist das Object Warehouse. Ein Object Warehouse fungiert als integrative Datenbasis objektorientierter Management Support Systems und kann insofern als ein objektorientiertes Analogon des Data Warehouse verstanden werden. Die Idee besteht darin, das objektorientierte Paradigma auch auf Management Support Systems im Allgemeinen und das Data Warehouse im Speziellen zu übertragen. Diese Arbeit untersucht, ob es sich dabei um einen sinnvollen Ansatz handelt.

Am Anfang einer wissenschaftlichen Arbeit stellt sich die Frage, was eine solche Arbeit leisten kann, und vor allem, was diese leisten sollte. Diese Frage tatsächlich am Anfang einer Arbeit zu stellen und auch zu beantworten, ist nach *Becker* eine grundlegende Aufgabe, auf deren Erfüllung oftmals kein Wert gelegt wird.[1] Es handelt sich dabei um eine äußerst schwierige Aufgabe, denn eine einfache Antwort kann darauf nicht gegeben werden. Die Behandlung dieser Fragestellung ist selbst Forschungsgegenstand. Die diesbezüglichen Auffassungen weichen zum Teil erheblich von einander ab. Der Diskurs wird in manchen Fällen mit mehr Polemik, als mit einer sachlich logischen Argumentation geführt, was die These stützt, dass darauf zumindest keine eindeutige Antwort gefunden werden kann. [FiHo98, S. 163]

Einzelne Fachvertreter sehen dies allerdings anders. Teilweise gehen Wissenschaftler sogar so weit, anderen Wissenschaftsgebieten vorzuwerfen, keine „echte“ Wissenschaft zu betreiben. Dieser Vorwurf wird gelegentlich auch gegenüber dem jungen Wissenschaftsgebiet der Wirtschaftsinformatik und sogar der inzwischen längst etablierten Betriebswirtschaftslehre geäußert. Eine in der „Scienti-

[1] *Prof. Dr. Jörg Becker* beim Doktorandenseminar auf Schloß Dagstuhl anlässlich der Tagung Wirtschaftsinformatik ‘99 in Saarbrücken.

fic Community“ durchgängig bzw. allseits akzeptierte Erkenntnistheorie ist jedoch nicht vorhanden. In diesem Sinne diagnostiziert auch *Behrens*, dass zwar in den Formalwissenschaften weitgehend Konsens über die Methodik herrscht, dass aber in den Realwissenschaften keine Einigkeit über eine präferable Wissenschaftstheorie besteht. [Behr91, S. 4763 ff.] Dieses Problem führt zum Teil dazu, dass ein möglicher Erkenntnisgewinn generell negiert wird, zum Teil dazu, dass die Beschäftigung mit diesem Thema mit der Begründung abgelehnt wird, dass dies alleiniger Gegenstand der Philosophie bzw. Wissenschaftstheorie sei. Beide Lösungen sind unbefriedigend, da im ersten Fall keine Bewertung von wissenschaftlichem Fortschritt stattfindet, und im zweiten Fall keine Bewertung stattfinden kann, da das diesbezügliche Instrumentarium fehlt. Dies hat auch die deutsche Wirtschaftsinformatik erkannt, die sich deshalb und in den letzten Jahren zunehmend mit dem Problem einer Wissenschaftstheorie für die Wirtschaftsinformatik beschäftigt. [Wiss99]

Fitzgerald und *Howcroft* zeigen, wie umfangreich die diesbezügliche Kontroverse in der Wirtschaftsinformatik auf internationaler Ebene ist und welches die Probleme der einzelnen Ansätze sind. [FiHo98, S. 155 ff.] *Lehner* betont die besondere Bedeutung der empirischen Forschung in der Wirtschaftsinformatik, deren Intensivierung er fordert. [Lehn97, S. 438 f.] *Popper* verweist allerdings auf das Induktionsproblem und lehnt damit jene Formen des Empirismus ab, die im Logischen Positivismus des Wiener Kreises in den zwanziger und dreissiger Jahren des 20. Jahrhunderts hervorgebracht wurden. Statt dessen entwickelt er den Kritischen Rationalismus. Dessen Kernthese ist, dass sich empirische Hypothesen bzw. Aussagen über die Verhältnisse in der realen Welt nicht positiv verifizieren, sondern vielmehr nur falsifizieren lassen. Die Wahrheit einer Hypothese lässt sich nicht positiv beweisen, wohl aber ihre Falschheit. Ist eine Hypothese als falsch erwiesen, dann kann und muß sie — im Sinne der *darwinschen* Evolutionstheorie — durch eine bessere Hypothese ersetzt werden. [Popp94, S. 243 ff., Popp95, S. 127 ff.]

Diese Wissenschaftsheorien, sowohl der Logische Positivismus wie *Poppers* Kritischer Rationalismus, gehen alle von einer kontinuierlichen Entwicklung, von einem stetigen Zuwachs, von einem linearen Fortschritt der Erkenntnis aus. *Kuhn* dagegen spricht von „revolutionären“ Erkenntnissprüngen und führt dafür den Begriff des Paradigmenwechsels ein. Paradigmata lösen andere ab, wenn und weil sie bei der Lösung einiger Probleme erfolgreicher sind als die ursprünglichen, mit den neuen zunächst konkurrierenden. Einzelne Misserfolge in der korrekten Vorhersage des Geschehens stellen dabei, in der sogenannten „Normalwissenschaft“ zunächst nur einzelne Hypothesen, nicht aber das Paradigma an sich infrage (siehe Abschnitt 3.2). [Behr91, S. 4766 f.] Ein Wechsel des Paradigmas erfolgt dann, wenn sich eine Mehrzahl der Forscher von einem alten Paradigma ab- und dem neuen zuwendet.

Kuhns Theorie des Paradigmenwechsels gibt allerdings, so wenig wie der Logische Positivismus und *Poppers* Kritischer Rationalismus, keinen hinreichenden Aufschluss über das Theorie-Praxis-Problem. *Behrens* meint deshalb: „Für die Verknüpfung von Theorien mit praktischen Problemen bietet die Wissenschaftstheorie keine überzeugende Lösung." [Behr91, S. 4769] Daher wird das Problem als Hilfskonstruktion in die logisch nicht verknüpften Bereiche Entdeckungs-, Begründungs- und Verwendungszusammenhang zerlegt. Der Entdeckungszusammenhang bezieht sich auf das Entstehen von Theorien und ist ein spontaner oder systematischer psychischer Prozess. Gegenstand der Wissenschaftslogik ist der Begründungszusammenhang, der bei der Überprüfung von Theorien ausschlaggebend ist. Im Verwendungszusammenhang schließlich geht es um die Anwendung der Theorien auf praktische Probleme. [Behr91, S. 4770]

Um die mit dem Problem des Verwendungszusammenhangs entstehenden Fragen zu lösen, also auch Arbeiten einzubeziehen, die nicht noch durch logisch bewährte Theorien gesichert sind, sich jedoch in der Praxis bewähren, hat insbesondere in den USA der Pragmatismus Verbreitung gefunden. Über die Wahrheit einer Theorie entscheidet nicht zunächst und nicht allein die Frage ihrer empirischen Überprüfung, sondern zuletzt ihr praktischer Nutzen. [Behr91, S. 4770] Dabei bleibt allerdings umstritten, inwieweit ein solcher praktischer Nutzen gesichert beurteilt werden kann — insbesondere bei der Betrachtung einer langfristigen Perspektive.

Eine weitere Lösung besteht darin, die wissenschaftliche Disziplin von praktischen Empfehlungen zu trennen. Beispielsweise gibt es in der Betriebswirtschaftslehre eine bedeutende Kontroverse zwischen *Schmalenbach* und *Rieger* bzw. zwischen *Gutenberg* und *Mellerowicz* darüber, ob die Orientierung der Betriebswirtschaftslehre theoretisch oder anwendungsorientiert zu sein habe. [Behr91, S. 4768 f.] Die theoretisch orientierte Betriebswirtschaftslehre wurde von *Schmalenbach* als „Kunstlehre" bezeichnet, da sie als reine Wissenschaft praktische Empfehlungen ablehnt. *Albert* nennt dies „Modell-Platonismus". Statt dessen wird das Erkenntnisziel und das Interesse an möglichst werturteilsfreien abstrakten Theorien betont. Anwendungsorientierte Wissenschaften hingegen suchen nach normativen Problemlösungen. [Behr91, S. 4768 f.]

Dies zeigt, wie schwierig es tatsächlich ist, das Ziel und den Weg einer wissenschaftlichen Arbeit zu definieren. Als Ziel einer Wissenschaft gilt die Erforschung der Wahrheit, die Gewinnung eines sachlich geordneten Zusammenhangs von wahren und in ihrer Wahrheit gesicherten Urteilen. [Wöhe91, S. 23] Doch bereits diese Aussage ist nach *Wöhe* erkenntnismäßig nicht zu sichern. Vielmehr ist es zunächst nur ein Bekenntnis, das allgemeine Anerkennung gefunden hat. Somit gründet sich die Wissenschaft selbst auf ein Werturteil, dessen Wahrheit mit logischen Mitteln nicht gesichert werden kann. [Wöhe91, S. 23] Auch gibt es wissenschaftliche Disziplinen, die die betrachteten Formen des Erkenntnisgewinns

generell anzweifeln und damit bereits das Fundament einer solchen Wissenschaftstheorie infrage stellen. [Tulk99]

Das Ergebnis dieser Überlegungen muss es dann sein, die zur Verfügung stehenden Möglichkeiten von Erkenntnisgewinn zu nutzen, also innerhalb der betrachtbaren Grenzen Erkenntnisfortschritte zu beurteilen und damit im *popperschen* Sinne „besseres Wissen" von „weniger gutem Wissen" zu unterscheiden. Das führt zurück zur Problematik der vorliegenden Arbeit.

Wesentlicher Gegenstand der Wirtschaftsinformatik sind Entwurf, Gestaltung, Entwicklung, Implementierung und Wartung von Informations- und Kommunikationssystemen in Wirtschaft und Verwaltung. Besondere Bedeutung kommt hierbei der Information als wirtschaftlichem Gut zu. Erkenntnisgegenstand ist die Analyse und Verbesserung von diesbezüglichen Methoden und Werkzeugen. Entwickelt werden Konzepte, Modelle und Prototypen. Dabei wird verlangt, dass bei der Betrachtung der soziotechnischen Systeme nicht nur technische Effizienz, sondern auch die ökonomische und soziale Einsetzbarkeit untersucht werden. [Lehn97, S. 438 f.]

Zwei Teilgebiete der Wirtschaftsinformatik, die in den letzten Jahren Forschungsschwerpunkte waren und starke Verbreitung in der Praxis gefunden haben, sind das objektorientierte Paradigma und der Bereich der Management Support Systems (MSS).

MSS sind spezielle Informations- und Kommunikationssysteme zur Unterstützung von Personen bei der Versorgung mit und Analyse von Daten bzw. Informationen, um im Sinne des Unternehmenszieles bessere Entscheidungen zu treffen und Probleme effizienter lösen zu können. Während in früheren Jahren in der Wirtschaftsinformatik der Schwerpunkt des Interesses auf operative Systeme (Administrations- und Dispositionssysteme) gerichtet war, hat sich dieser seit geraumer Zeit auf die produktive Nutzung von Daten, Informationen und Wissen für unternehmerische Entscheidungen verlagert. [Rieg85, S. 1 ff.]

Das objektorientierte Paradigma dagegen ist ein Ansatz für den gesamten Systems-Engineering-Prozess. Während in der Praxis immer noch hauptsächlich strukturierte Techniken wie strukturierte Analyse, strukturiertes Design, strukturierte Programmierung (4GL, RDBMS) und das Wasserfallmodell Verwendung finden, konnten in Forschung und Praxis mithilfe objektorientierter Techniken bessere Lösungen für die Probleme der „Softwarekrise" erreicht werden. [Your93, S. 92 ff.]

Diese beiden Forschungsschwerpunkte der Wirtschaftsinformatik zu verbinden, um objektorientierte Management Support Systems (OMSS) zu entwickeln, ist eine logische Konsequenz — stellt mithin einen folgerichtigen Ansatz dar. Können die Potenziale des Objekt-Paradigmas auch für den MSS-Bereich nutzbar ge-

macht werden, dann bedeutet dies einen weiteren Schritt zur Verbesserung unternehmerischer Entscheidungen. Damit erklärt sich der Entdeckungszusammenhang, dem die vorliegende Arbeit zugrundeliegt, also ihre Idee. Sie beschränkt sich auf die Untersuchung einer Object-Warehouse-Konzeption und ordnet sich damit in die Gesamtaufgabe des Forschungsschwerpunktes zu objektorientierten Management Support Systems des Lehrstuhls[2] ein.

Die Forschungsfrage, die dabei gestellt ist, lautet: Wie kann die Konzeption eines Object Warehouse überhaupt aussehen? Die hier vertretene Hypothese für die Beantwortung dieser Frage behauptet, dass die beiden vielversprechenden Ansätze Data Warehouse und Objekt-Paradigma in ihrer Kombination wiederum zu einem sinnvollen Ansatz führen. Wie zu zeigen sein wird, ist diese Antwort durchaus nicht trivial.

Daraus ergibt sich zunächst die Frage nach der Qualität der Object-Warehouse-Konzeption, also zum einen der Software-Qualität und zum anderen der Software-Prozess-Qualität. Der Bewertungsmaßstab wird dabei letztlich von der konventionellen Data-Warehouse-Konzeption vorgegeben. Das Data Warehouse kann insofern im *popperschen* Sinne als beste bestehende Hypothese aufgefasst werden. Die Object-Warehouse-Konzeption muss dann eine bessere Lösungsadäquanz bieten.

Dazu müssen zwei weitere Forschungsfragen bzw. Problemkomplexe beanwortet werden:

1. Wie muss die Data-Warehouse-Konzeption aussehen? Welches sind die Anforderungen an ein Data Warehouse und wie können diese bestmöglich erfüllt werden?
2. Wie muss das Objekt-Paradigma aussehen? Was wird überhaupt unter dem Objekt-Paradigma verstanden? Welche unterschiedlichen Objekt-Paradigmata gibt es? Wie kann die Qualität eines Objekt-Paradigmas beurteilt werden? Wie muss das Objekt-Paradigma für ein Object Warehouse aussehen?

Für die Konzeption eines Object Warehouse ist mithin die detaillierte Analyse des konventionellen Data Warehouse (siehe Kapitel 2) und des Objekt-Paradigmas (siehe Kapitel 3) notwendig. Die Analyse dieser beiden Gebiete stellt damit eine notwendige Bedingung dar, um die Leitfrage der vorliegenden Arbeit zu beantworten. Hier liegt deshalb auch ihr Schwerpunkt. Wie sich zeigt, können die gefundenen Antworten zu einer möglichen Konzeption eines Object Warehouse verbunden werden (siehe Kapitel 4). Dennoch erhebt die Arbeit keinen Anspruch darauf„die bestmögliche“ Object-Warehouse-Konzeption zu entwickeln. Hierfür

[2] Fachgebiet BWL/Wirschaftsinformatik II bei *Prof. Dr. Bodo Rieger* der Universität Osnabrück.

sind die verschiedenen Objekt-Paradigmata (im *kuhnschen* Sinne verstanden) zu unterschiedlich. Allerdings liesse sich ein Vergleich zu anderen konkreten Konzeptionen durchaus bewerkstelligen und ist erklärtes Forschungsbestreben zukünftiger Arbeiten.

Der Begründungszusammenhang an dieser Stelle ist tatsächlich in vielen Fällen recht unproblematisch (siehe Kapitel 2 und 3). Umfangreiche statistische Untersuchungen, wie im amerikanischen Sprachraum üblich [FiHo98, S. 158 ff.], oder modelltheoretische Überlegungen, wie in der klassischen Nationalökonomie [Behr91, S. 4768 ff.], sind deshalb unangebracht. Logische Argumentation, Expertenwissen und empirische Erfahrungen bilden die Grundlage für die Object-Warehouse-Konzeption (siehe Teil I, der sich aus Kapitel 2 bis 4 zusammensetzt).

Das Expertenwissen ergibt sich aus einem umfangreichen Literaturstudium. Die empirischen Erfahrungen stammen aus drei größeren Projekten mit mittelständischen Unternehmen [BBB+97, DHJ+96, Nieb99, Weus99], die für diese Arbeit durchgeführt worden sind. Außerdem sind gewonnene Erkenntnisse unmittelbar in die Lehre des Fachgebiets[2] eingeflossen. [Lehr00] Insbesondere durch die praktischen Übungen und Rückkopplungen von seiten der Studierenden konnten wertvolle zusätzliche Erfahrungen gesammelt werden. Wie sich zeigt, befindet man sich dabei immer auf einer Gratwanderung zwischen einer primär konzeptionellen Vorgehensweise und einem Versinken in technischen Details. Es ist nicht Aufgabe des Wirtschaftsinformatikers, technische Lösungen zu entwickeln oder Werkzeuge und Anwendungen zu implementieren — dies wird dem Gebiet der Kerninformatik überlassen. Vielmehr ist die Analyse von Werkzeugen und Methoden sowie die Entwicklung von Konzeptionen das primäre Interesse des Wirtschaftsinformatikers. Allerdings ist die klare Trennung zwischen dem Wirtschaftsinformatiker, der etwa die Systemanalyse bis zum Pflichtenheft durchführt, und dem klassischen „Implementierer", der lediglich die Spezifikation umsetzt, nicht mehr in dieser Form möglich. Software-Entwicklung ist zu einem hochkomplexen Gebiet geworden, das ohne große Teams[3] und CASE-Umgebungen nicht mehr zu bewerkstelligen ist.

Dies hat zur Folge, dass auch die Analysen und Konzeptionen von Systemen, Methoden und Werkzeugen nicht mehr isoliert betrachtet werden können. So ergibt sich ein Dilemma. Es besteht die Gefahr, **entweder** zu wenig konzeptionell vorzugehen, also Lösungen zu technisch zu betrachten und dadurch die praktischen Anforderungen und die Gesamtsicht zu vernachlässigen, **oder** sich gegenüber technischen Gegebenheiten zu ignorant zu verhalten, womit die Konzeption

---

[3] *Burleson* fordert als Team-Mitglieder für die Entwicklung eines Data Warehouse mindestens: „Project Sponsor, Project Manager, Warehouse Architect, Systems Administrator, Legacy Database Experts, Data Warehouse DBA, Operations Manager, Metadata Specialist, End-User Interface Specialist, Network Administrator, Oracle Developers, and a Statistician." [Burl97, S. 73]

schlicht unbrauchbar würde, weil sie nicht umsetzbar ist. Den Anforderungen aller skizzierten Ebenen gleichermaßen gerecht werden zu wollen, ist durch eine Einzelperson[3] nicht zu erreichen und würde deshalb den Rahmen der vorliegenden Arbeit sprengen.

Die Erfahrungen zeigen, wie gravierend die Auswirkungen einer Vernachlässigung von Konzeption und/oder technischen Details sein können, insbesondere, wenn bedacht wird, in welcher Höhe Fehlinvestitionen in Unternehmen aufgrund solcher mangelhaften Lösungen getätigt werden (siehe insbesondere Abschnitt 2.5). Deshalb muss der Detaillierungsgrad der grundlegenden Konzepte Data Warehouse und Objekt-Paradigma teilweise sehr hoch sein, was auch durch die Analyse bestätigt wird. Im Vordergrund steht aber die Gesamtkonzeption. (Siehe Kapitel 2 und 3.)

Da die Wirtschaftsinformatik als Teilgebiet der Betriebswirtschaftslehre entstanden ist und zumindest teilweise immer noch als deren Teilgebiet angesehen wird [Kurb97], ist die Frage nach deren Orientierung auch für die Wirtschaftsinformatik zu stellen. Die Wirtschaftsinformatik wird als anwendungsorientierte Wissenschaft verstanden [FHS00] und verfolgt mithin (neben Erklärungszielen) primär das Ziel, Gestaltungsempfehlungen geben zu können, d. h. normative Problemlösungen anzubieten. Deshalb ist auch diese Arbeit primär problemlösungsorientiert, verfolgt mithin ein Gestaltungsziel. Insofern ist auch der Verwendungszusammenhang von hohem Interesse.

Die Bewertung des Verwendungszusammenhangs stützt sich auf die drei Praxisprojekte und die Umsetzungen in der Lehre. Der praktische Test ist nicht unproblematisch. [RMK98, S. 247 ff.] Es hat sich gezeigt, wie unterschiedlich die konkreten Zielsetzungen von Forschung und Praxis, also konkret die Zielsetzungen des Fachgebiets auf der einen und den Unternehmen auf der anderen Seite sind. Unter anderem verbieten kurzfristige pragmatische Zielsetzungen der Unternehmen wie auch ökonomische Zwänge, ferner Risikovermeidungsstrategien und schließlich psycho-soziale Faktoren den Einsatz von Konzepten, die sich nicht bereits an anderer Stelle bewährt haben. Dies ist verständlich. Und tatsächlich muss auch das Object-Warehouse-Konzept als eine Technologie betrachtet und eingeordnet werden, die eine unternehmerisch vertretbare Praxistauglichkeit noch nicht erreicht hat. Doch selbst wenn dieser letzte Punkt erfüllt wäre, bedingt allein der Tatbestand, dass ein erforderliches Know-how noch überhaupt nicht im Unternehmen vorhanden sein kann, das Scheitern eines echten Praxistests.[4]

---

[4] Dieses Dilemma zeigt auch, wie kritisch deshalb unreflektierte Forderungen nach mehr Praxiskontakten beurteilt werden müssen. Praxiskontakte bilden eine notwendige Bedingung für die Überprüfung von Konzepten

Ein weiterer wichtiger Ansatz für den Verwendungszusammenhang eines Konzepts ist ein Prototyp. Ein Prototyp bietet die Möglichkeit, die Realisierbarkeit einer Konzeption aufzuzeigen. Dabei kann zwar keine generelle Machbarkeit bewiesen werden, doch immerhin kann gezeigt werden, dass die Konzeption in mindestens einem Fall umsetzbar ist, mithin eine prinzipielle Umsetzbarkeit gegeben ist. Prototypen bilden deshalb ein wichtiges Instrumentarium in der Wirtschaftsinformatik. Folglich wird auch in dieser Arbeit ein Prototyp realisiert (siehe Teil II, der sich aus Kapitel 5 und 6 zusammensetzt).

Doch auch die Bildung eines Prototypen ist nur begrenzt aussagefähig. Dies rührt zum einen aus der Diskrepanz zwischen Konzepten und Werkzeugen, die dem Object Warehouse zugrunde liegen, und zum zweiten aus der Diskrepanz zwischen den einem Unternehmen zugrunde liegenden Anforderungen und der praktischen Realität.

Konkret bedeutet dies, dass **erstens** essenzielle Objekt-Paradigma-Konzepte noch nicht in Werkzeugen umgesetzt worden sind. Dies liegt zum einen in der bisher noch ausstehenden Umsetzung der Konzepte in die Werkzeuge durch die jeweiligen Hersteller, zum anderen daran, dass für bedeutende Objekt-Paradigma-Konzepte noch keine geeigneten Lösungen und Standards gefunden worden sind und diese folglich auch nicht in kommerzielle Werkzeuge umgesetzt werden können, was zur Forderung nach einer weiteren Intensivierung der diesbezüglichen Forschungsaktivitäten führt. Da der Aufgabenbereich eines Wirtschaftsinformatikers in der Anwendung von Werkzeugen besteht und nicht in deren Implementierung, kann nur die nötige Analyse der Werkzeuge erfolgen (in diesem Fall GemStone und VisualWorks). Solange die Hersteller (in diesem Fall also Cincom und GemStone) die Verbesserungsvorschläge nicht umsetzen, können die geforderten Konzepte keine Verwendung in den Prototypen finden.

Zweitens hat sich in allen durchgeführten Projekten gezeigt, dass die vorhandenen Quellsysteme und -daten in hohem Maße unbrauchbar sind, so dass selbst die Implementierung eines klassischen Data Warehouse nicht sinnvoll zu realisieren ist — die Integration von beispielsweise semistrukturierten Daten oder einem Meta-Object-Layer deshalb in diesem Stadium nicht zu verwirklichen ist. So konnte etwa mithilfe eines Prototypen in einem der Projekte zwar ein Teil des Datenbestands für die gewünschten Analysen verwendet werden, doch konnte die jeweilige Korrektheit der Daten nicht überprüft werden, da die Speicherung der Daten durch das operative System fehlerhaft war [DHJ+96], so dass in einem ersten Schritt das Warenwirtschafts- und das Kassensystem ausgewechselt wurden.

---

im Verwendungszusammenhang, sind aber von geringem Nutzen, werden dort lediglich Konzepte umgesetzt die sich bereits bewährt haben.

Diese Probleme mit Praxispartnern und der Prototypentwicklung dürfen jedoch keinesfalls dazu führen, dass das universitäre Interesse auf solche Gebiete verlagert wird, die erstens mit vorhandenen Werkzeugen zu realisieren und zweitens bei Praxispartnern umzusetzen sind. Der Zielsetzung einer Forschungseinrichtung wäre damit nicht gedient. Die vorliegende Arbeit legt deshalb, in Anbetracht des zu behandelnden Themas, ihren Schwerpunkt auf den konzeptionellen Teil (siehe Teil I).

TEIL I

ENTWICKLUNG EINER OBJECT-WAREHOUSE-KONZEPTION

KAPITEL 2

DATA WAREHOUSE

Das Data-Warehouse-Konzept stellt einen wesentlichen Bestandteil des Management-Support-Systems-Ansatzes dar. Es bildet die Grundlage der Anwendungssysteme zur Informationsgewinnung und Entscheidungsvorbereitung, von *Chamoni* und *Gluchowski* als Analytische Informationssysteme bezeichnet (z. B. EIS, DSS, Query Tools, OLAP, Data Mining), indem es diesen eine rekonziliierte Datenbasis zur Verfügung stellt.

Zunächst erfolgt eine Klassifikation von Management Support Systems und eine diesbezügliche Einordnung des Data Warehouse in die IUK-Gesamtarchitektur. Dem schließt sich die Begründung für die Notwendigkeit zur Entstehung des Data-Warehouse-Konzepts und dessen Zielsetzung an. Auf dieser Grundlage wird eine idealtypische Data-Warehouse-Architektur entwickelt. Es werden Aspekte genannt, unter welchen Gegebenheiten die jeweiligen Schichten explizit notwendig sind bzw. funktional integriert werden können. Abschließend wird die entwickelte Architektur den bisher üblichen Implementierungen und den in der Literatur vorherrschenden Konzepten kritisch gegenüber gestellt.

2.1 Management Support Systems

Management Support Systems beinhalten vielerlei Komponenten, Methoden und Prinzipien. Zum Verständnis der MSS-Intention ist es daher wichtig, überhaupt Klarheit darüber zu erlangen, was Management Support Systems sind, welche einzelnen Bestandteile dazu gehören und welche konkreten Zielsetzungen MSS haben bzw. Problemklassen damit zu lösen versucht werden. Hierzu lassen sich MSS nach verschiedenen Klassifikationskriterien einteilen.

MSS werden nach Anwenderklasse, Problemklasse, Produktart und Wissensbasierung differenziert. Die Aufzählung wird um neuere Begriffe aus dem MSS-Umfeld ergänzt. Es erfolgt jeweils eine Bewertung des Klassifikationsansatzes

anhand dessen praktischer Nützlichkeit. Abschließend wird eine zusammenfassende Darstellung und Definition für diese Arbeit gegeben.

Primäres Ziel ist die Einordnung des Data Warehouse in die MSS-Landschaft sowie die Herausarbeitung der MSS-Anforderungen an das Data-Warehouse-Konzept. Insbesondere der vielschichtige Unterstützungsbedarf durch MSS bzw. das breite MSS-Spektrum geben die Anforderungen an das Data Warehouse vor.

2.1.1 Anwenderklassen-spezifische Klassifikation

Unter Management Support Systems wird die integrierte Nutzung von Informations- und Kommunikationstechnologie (IUK) zur Unterstützung des Managements verstanden. [Scot83] Damit definiert *Scott Morton* den Begriff der MSS anhand der Anwenderklasse. Dies entspricht auch der Intention des Vorläuferbegriffs, dem Management Information System (MIS), bei welchem „Führungskräfte der höheren Führungsebenen“ [Stah97, S. 247] unterstützt werden sollen. Auch *Rieger* entwickelt mit seinem Rechnerunterstützten Arbeitsplatz (RAP) ein Konzept, das sich über die Bedürfnisse der Anwender, den Führungskräften, definiert. [Rieg93] Dieser Klassifikationsansatz ist üblich [GGC97, S. 1 ff., HoWh-96, S. 138 ff., KMS97, S. 150, RoDe88, S. 14 ff., Stah97, S. 247 f., Turb95, S. 3 ff.] und schlägt sich auch in den verschiedenen Begrifflichkeiten der MSS-Komponenten wie Executive Information System [WHR97, S. 3 ff.], Executive Support System [RoDe88, S. 14], Management Information System [LaLa96, S. 24 ff.], Chefinformationssystem [ChGl99b, S. 8] oder Vorstandsinformationssystem [St-Ha97, S. 426] nieder.

Eine These über den Ursprung der Konzentration auf „Führungskräfte der höchsten Ebene“ [BaSe97c, S. 157 f.] (Top Executive [BaSe97a, S. 9], Top Manager [KMS97, S. 150], CEO [RoDe88, S. 16]) und damit einer anwenderkreisbezogenen Klassifikation gründet sich auf den Bedarf zur Erschließung neuer Anwendungsfelder für IUK-Anbieter und das besondere ökonomische Potenzial der zu unterstützenden Klientel. [Kell96, S. 1 ff.] In der Regel wird diese Ausrichtung allerdings mit dem besonderen Unterstützungsbedarf von Führungskräften begründet. Fehlendes Know-how im Umgang mit IUK-Technologie und ein Mangel an Zeitressource verlangen hauptsächlich eine einfache Bedienbarkeit und hochaggregierte Daten in angemessener Zeit. [Vets95, S. 82 ff.] *Rieger* fordert die: „Direkte Versorgung von Führungskräften mit führungsrelevanten Informationen in führungsadäquater Form.“ [Rieg93]

Anwender- bzw. Arbeitsplatzbedürfnisse als Gegenstand des Forschungsinteresses zu wählen ist eine legitime und sinnvolle Vorgehensweise. Dies gilt um so mehr, wird während der Analyse ein besonderes Unterstützungspotenzial der Zielgruppe aufgezeigt, was für das Management zutreffend ist. [Mint91, S. 21 ff.]

Die auf diesem Forschungsweg gewonnenen Erkenntnisse sind jedoch auch auf andere Zielgruppen übertragbar.

Der Informations- und Entscheidungs-Unterstützungsbedarf besteht durchaus nicht nur für Top Executives, sondern auch für die mittlere und untere Managementebene, Controller, verschiedene Sachbearbeiter et cetera. Verstärkend wirkt der zunehmende Stellenwert des Produktionsfaktors Information, die wachsende Umweltkomplexität und -dynamik, die Globalisierung, die Entwicklung der IUK-Technologie und die resultierende Umverteilung von Entscheidungskompetenzen in den Unternehmen (Lean Management, Kaizen, Gruppenorientierung et cetera). [GGC97, S. 37, Tayl92, S. 5 ff.] Auch ist nicht einsichtig, warum ein Sachbearbeiter ein weniger benutzerfreundliches Werkzeug zur Informationsgewinnung verwenden sollte als ein Top Manager. Die Beschränkung auf Detaildaten und damit der Verzicht auf aggregierte Informationen für die nicht den höchsten Führungsebenen angehörigen Mitarbeiter kann ebenfalls nur schwerlich begründet werden. Und letztlich wirkt sich eine zeitliche Verzögerung bei der Informationsbereitstellung für Entscheidungsprozesse gerade auf Sachbearbeiterebene multiplikativ aus.

Außerdem hat die Praxis gezeigt, dass bisher ausgerechnet bei der angesprochenen Klientel der Top Manager erhebliche Akzeptanzbarrieren bestehen und diese folglich nur in seltenen Fällen direkt mit MSS-Applikationen arbeitet. [BaSe97a, S. 9] Deren fehlendes IUK-Know-how und die Komplexität der Anwendungen konnte durch Techniken zur Komplexitätsreduktion und geeignete Benutzerführung nicht ausreichend kompensiert werden. Die Verwendung der Management Support Systems durch die höheren Führungspersonen bleibt in der Regel auf eine passive Nutzung beschränkt. Einige wenige Kennzahlen und hochaggregierte Grafiken werden z. B. durch Assistenten vorgegeben und dann nur noch analog zu einer Zeitung oder einem herkömmlichen Bericht gelesen (z. B. Briefing Book).

Insgesamt ergibt sich daraus eine Verlagerung des MSS-Interesses auf alle Anwenderklassen und damit eine Ablösung des traditionellen Ansatzes zur Unterstützung des Top Managements.

Dem trägt *Rieger* durch die Übertragbarkeit seines Ansatzes auf alle Unternehmensebenen Rechnung und fordert die Ausstattung mit „Rechnerunterstützten Arbeitsplätzen“ auch beispielsweise für Sachbearbeiter und Controller. [Rieg93] Gerade die organisatorische Durchdringung mit MSS bzw. „Rechnerunterstützten Arbeitsplätzen“ und deren Integration zu einem Netzwerk eröffnen wichtige Potenziale zur Verbesserung unternehmerischer Entscheidungsprozesse.[5]

---

[5] Beispielsweise könnte an die Anbindung von Workflow-Systemen gedacht werden. Manuell oder durch Exception Reporting getriggert könnten Workflows angestoßen werden. Auch direkte Rückwirkungen auf Administrations- und Dispositionssysteme sind denkbar.

Watson, *Houdeshel* und *Rainer* sehen ebenfalls die Entwicklung zur organisatorischen Durchdringung mit MSS und verweisen auf Unternehmensbeispiele, bei welchen hunderte bis tausende von Arbeitsplätzen[6] mit EIS ausgestattet wurden. Folgerichtig empfehlen sie eine Umdeutung des Akronyms EIS und schlagen statt Executive Information System Everybodies Information System oder Enterprise Information System vor. [WHR97, S. 14] Dies empfiehlt auch *King*. [King96, S. 184]

Eine Begriffsuminterpretation wird dagegen von *Back* und *Seufert* durchgeführt, indem sie Management[7] im Sinne von „to manage" verstehen [BaSe97a, S. 1] und ebenfalls die Abkehr von der Beschränkung von MSS auf die obersten Führungsebenen fordern [BaSe97c, S. 157 f.]. Teilweise wird bezüglich der MSS-Anwender auch von „Entscheidungsträgern auf allen Ebenen" gesprochen. Dieser Nutzerkreis muß um die entscheidungsvorbereitenden Personen ergänzt werden. Je nach Unternehmen können dann einige bis alle Stellen diesem Anwenderkreis zugerechnet werden.

2.1.2 Problemklassen-spezifische Klassifikation

Neben der anwenderklassen-spezifischen Klassifikation gibt es bereits früh eine Unterteilung von MSS-Komponenten nach der Problemklasse.

Scott Morton stellt Data Support Systems, Decision Support Systems (DSS) und Executive Support Systems (ESS) neben einander. [Scot83] ESS definiert er als Mischform aus Data Support und Decision Support. Eine klare Abgrenzung zwischen ESS und Data Support ist bei *Scott Morton* allerdings nur schwer ersichtlich.

Chamoni, *Gabriel* und *Gluchowski* teilen MSS in die vier Kategorien Communication Support, Data Support, Decision Support und Basissysteme ein. Basissysteme werden u. a. als Textverarbeitung, Tabellenkalkulation und Grafikverarbeitung spezifiziert und dem Personal Information Management (PIM) zugeordnet; Communication Support wird z. B. durch E-Mail realisiert (siehe Abbildung 1). [ChGl99b, S. 9] Sie unterteilen damit Arbeitsplatz-bezogen nach MSS-Funktionen.

Unklar bleibt die Auswahl und Zweckmäßigkeit bezüglich der Basissysteme. Dazugerechnet werden müßten auch Internet Browser, Hilfsprogramme, evtl. das Betriebssystem und sicherlich Administrations- und Dispositionssysteme. Durch diese Definition wird das MSS-Konzept allerdings in einer Weise erweitert, die den Begriff des MSS nicht mehr sinnvoll zur Abgrenzung gegenüber anderen

---

[6] Beispielsweise hat das EIS bei Conoco über 7000 Benutzer. [WHR97]

[7] Der Duden dagegen übersetzt Manager mit „Leiter eines großen Unternehmens" [Dude96]

Softwarekonzepten verwenden läßt. Auch wird nicht erläutert, warum der Communication Support als Bestandteil des EIS neben dem Data Support gesehen wird (siehe Abbildung 1).

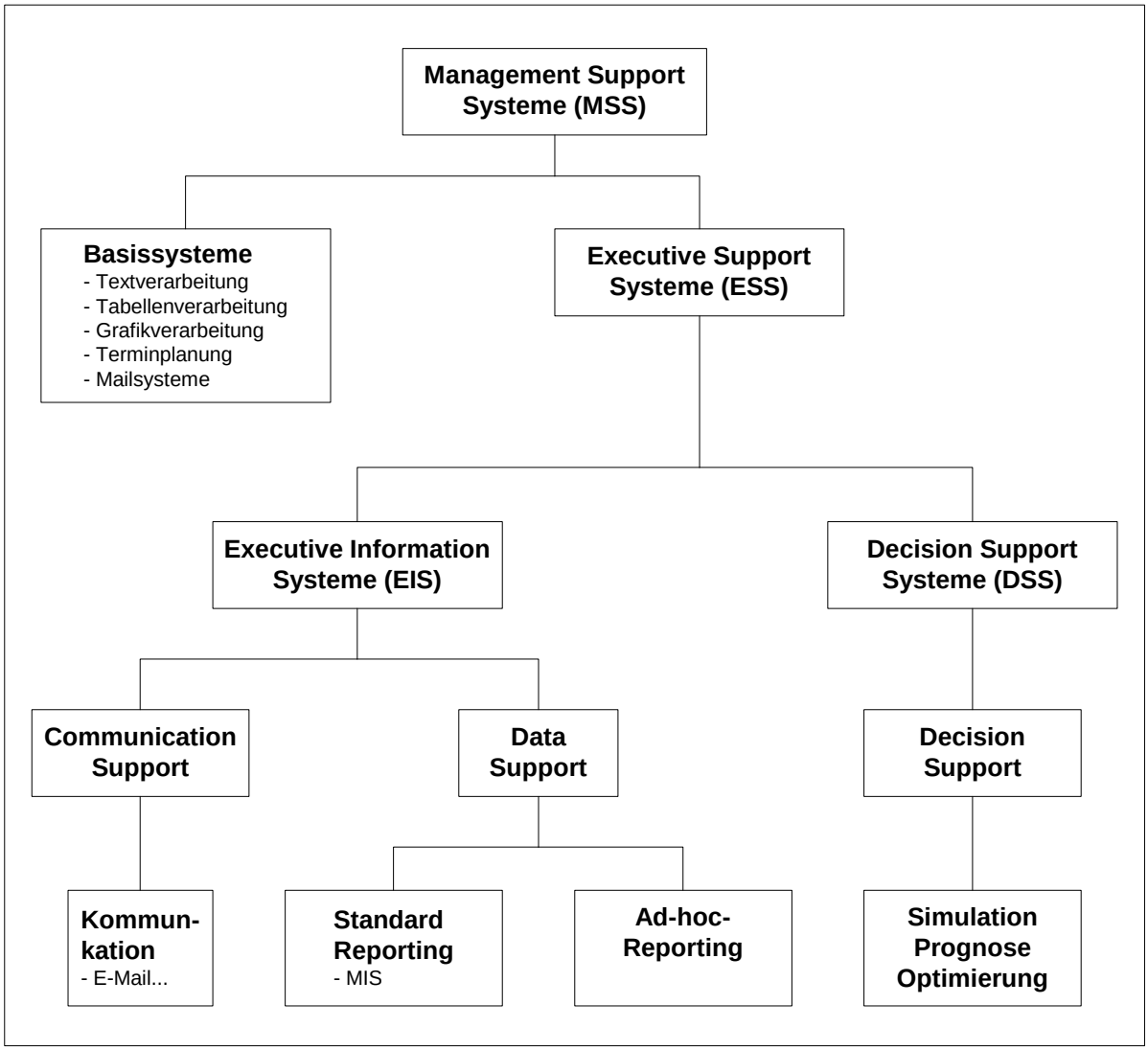


Abbildung 1: MSS-Klassifikation nach *Chamoni*, *Gabriel* und *Gluchowski* [ChGl99b, S. 9]

Krallmann, *Mertens* und *Schiemann* gliedern MSS nach den zwei Funktionsbereichen Data Support und Decision Support. Um einerseits den ursprünglichen Bezug zur anwenderklassen-spezifischen Klassifikation herzustellen und andererseits der organisatorischen Durchdringung mit MSS Rechnung zu tragen, unterteilen sie weiterhin nach der Unterstützung für das Management und für das Top-Management (siehe Abbildung 2). [KMS97, S. 150]

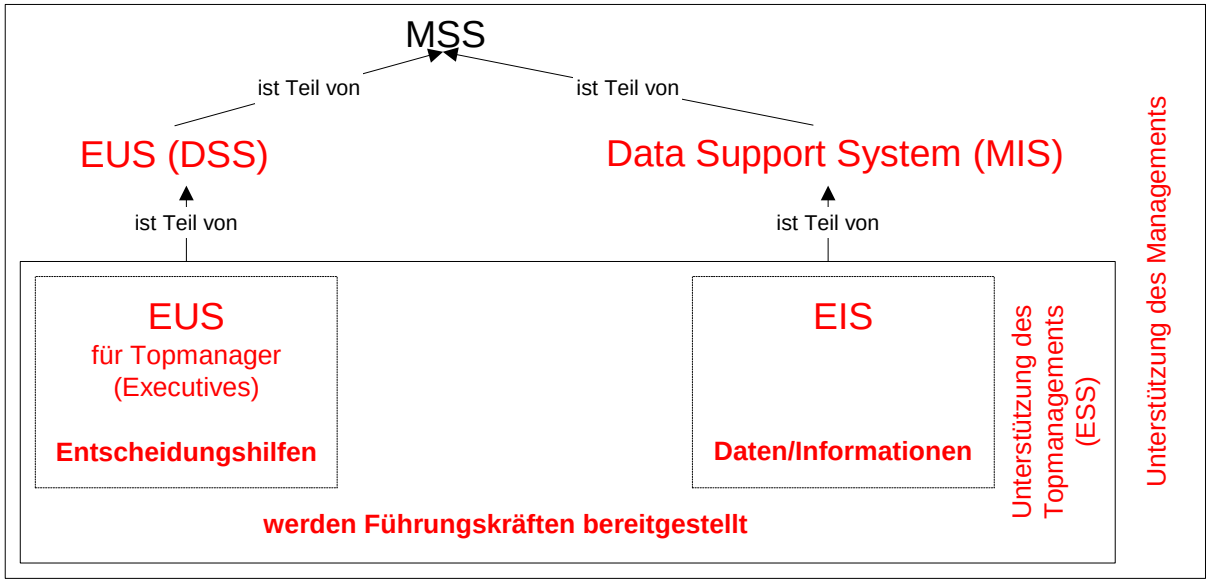


Abbildung 2: MSS-Klassifikation nach *Krallmann*, *Mertens* und *Schiemann* [KMS97, S. 150]

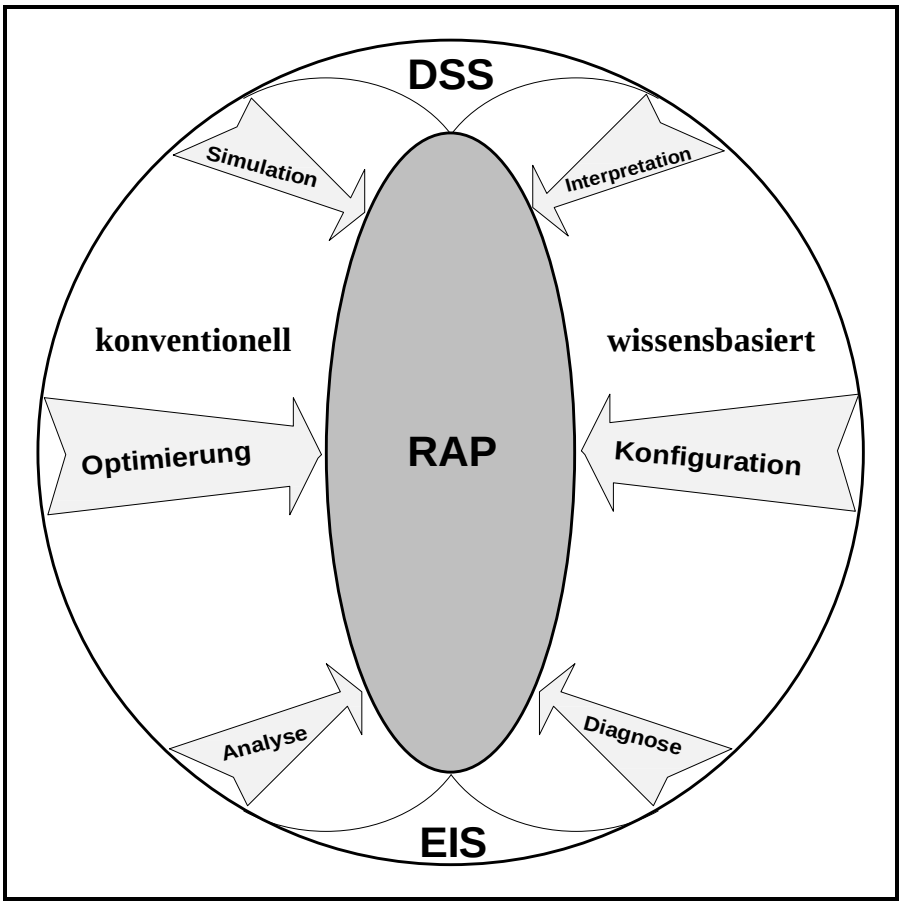


Abbildung 3: MSS-Klassifikation nach *Rieger* [Rieg93]

Welche Kriterien eine solche Unterteilung begründen, bleibt offen. Die Autoren können auch empirisch keine Bestätigung für ihre Unterteilung finden. Sie be-

merken, dass für Data Support im Management-Bereich keine „Drei-Buchstaben-Abkürzung“ gebräuchlich sei, jedoch eine Ähnlichkeit zur Begrifflichkeit des EIS und MIS bestehe. Für den Top-Management-Bereich wählen sie den Ausdruck EIS. Zur Unterscheidung von DSS für das Management bzw. Top-Management gäbe es auch keine Begrifflichkeit.

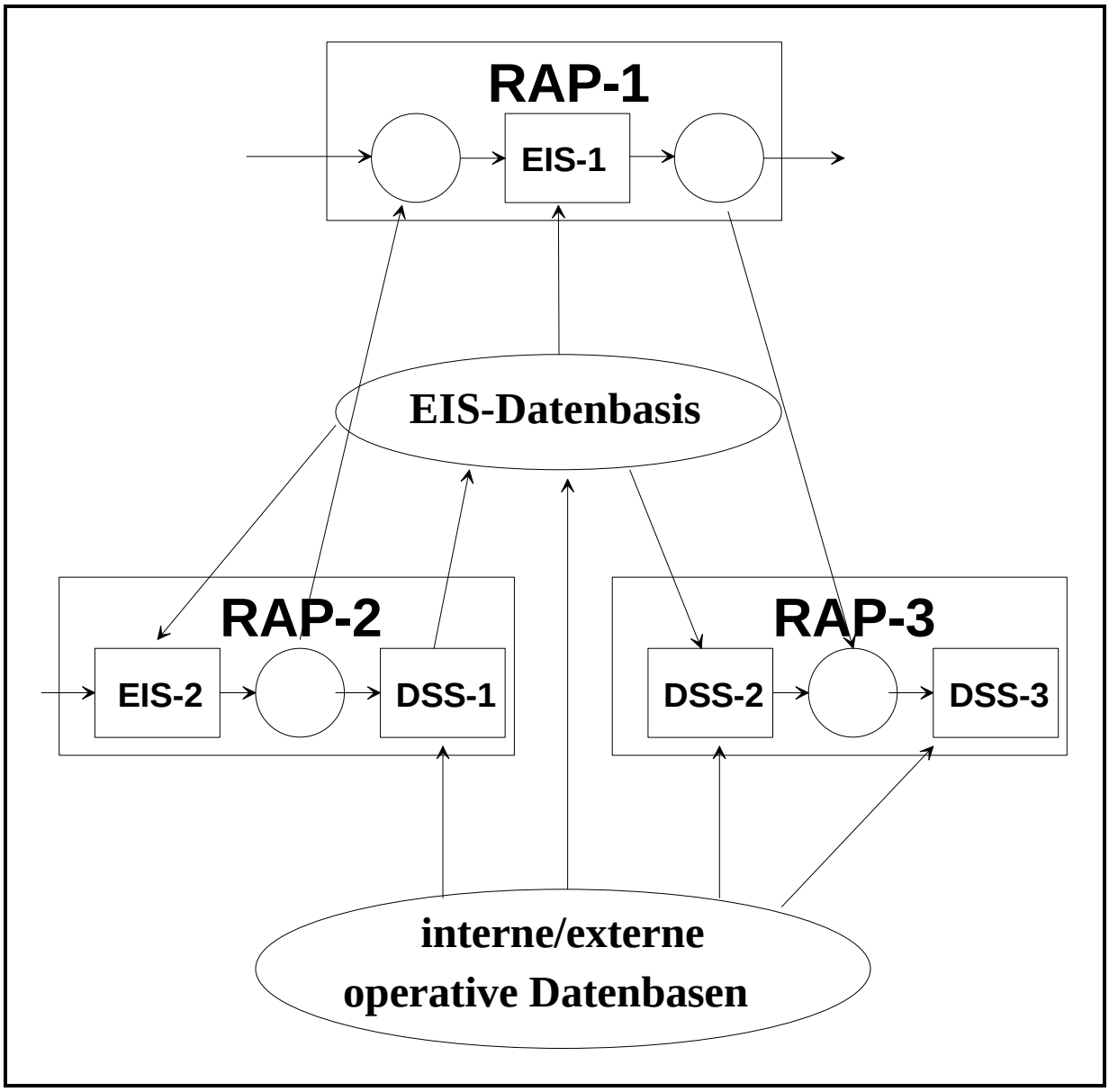


Abbildung 4: *Riegers* organisatorische DSS/EIS-Einordnung [Rieg93]

Tatsächlich fordern DSS in der Regel vertieftes Spezialwissen und sind daher für Führungskräfte oftmals weniger geeignet. Trotzdem hält *Rieger* es für notwendig, beide MSS-Komponenten auf allen Ebenen zu implementieren (siehe oben) und um weitere Unterstützungskomponenten zu erweitern. Dementsprechend wandelt sich auch der von *Rieger* ursprünglich verwendete Begriff des ESS [Kr-

Ri87] zum RAP [Rieg93]. Dies entspricht der Abkehr von der Konzentration auf Top Executives hin zu einer arbeitsplatz-orientierten Unterstützung. Den Data Support setzt er mit EIS gleich. Communication Support ist integraler Bestandteil des *riegerschen* RAP und muß für alle MSS-Funktionen vorhanden sein.

Bei *Rieger* bestehen MSS ebenfalls aus Decision Support Systems und Data Support Systems (siehe Abbildung 3). Danach differenziert er auch seine organisatorische Einordnung. Die höheren Führungsebenen (Vorstand) werden mit Information durch EIS versorgt (Data Support), um dadurch handlungsauslösend wirken zu können, während entscheidungsunterstützende bzw. -vorbereitende Systeme für die darunter liegenden organisatorischen Ebenen vorgesehen sind (siehe Abbildung 4). [Rieg93]

2.1.3 Produktarten-spezifische Klassifikation

MSS-Produkte lassen sich nicht uneingeschränkt (sinnvoll) bzw. nicht eindeutig anhand der Problemklasse klassifizieren. Während Decision Support Systems ihre Verschiedenartigkeit bereits durch deren spezielle Ausrichtungen — für unterschiedliche Phasen, Entscheidungsprobleme oder Methoden — zum Ausdruck bringen, besteht beim Data Support ein eher homogenes Aufgabenfeld. Dennoch ist auch das Funktionsspektrum bei Data Support Systems ein breitgefächertes. Executive Information Systems stellen diesbezüglich nur eine Produktvariante dar, die sich durch eine Kombination von Funktionsbündeln definiert. Weiterhin ist die Unterteilung am Markt erhältlicher MSS-Applikationen nach der Problemklasse deshalb nicht eindeutig zu treffen, weil die Trennlinie zwischen z. B. Data Support und Entscheidungsunterstützung oft nur schwer zu ziehen ist. [Rieg90, S. 103 ff.] Beispielsweise erfordert die Entscheidungsunterstützung in der Regel ein Mindestmaß an Data Support. Wird zusätzlich bedacht, dass bereits *Scott Morton* die Integration der unterschiedlichen Werkzeuge fordert (ESS-Ansatz) [Scot83], ergibt sich insgesamt der Bedarf einer hilfreicheren Klassifikation für MSS-Produkte.

MSS-Werkzeuge bestehen aus einer Reihe von Funktionsbündeln und weisen in der Regel eine Schwerpunktsetzung auf, wodurch die Einordnung in Kategorien möglich wird. Die Gründe für die Entstehung dieser spezifischen Werkzeuge liegen in der historischen Entwicklung der Nutzerbedürfnisse und Realisationspotenziale (z. B. ermöglichten erst DBMS, Netzwerke und Hardware-Fortschritte eine effiziente Datenhandhabung, welche notwendige Voraussetzung für das Data-Warehouse-Konzept ist; ohne die Entwicklungen im GUI-Bereich ist das EIS-Konzept nicht realisierbar) sowie vielfach in Marketing-getriebenen Modewel-

len[8]. Dabei gibt es zwischen den einzelnen Produkten oft vielfältige Überschneidungen, die eine Abgrenzung nur schwerlich erlauben. Die übliche Benennung der Werkzeuge nach Markt-Moden statt nach tatsächlich vorhandenen Funktionalitäten verstärkt das Begrifflichkeits-Chaos entsprechend.[9]

Die Intentionen bzw. Funktionsbündel/Funktionalitäten der wichtigsten MSS-Werkzeug-Begriffe werden folgend erläutert und jeweils Beispiele für konkrete Produkte gegeben. Dabei ist es prinzipiell möglich, dass Produkte einen größeren Leistungsumfang aufweisen, als dies die Kategorie vorgibt. Durch die starke Ähnlichkeit der Begriffsintentionen und der funktionalen Überlappungen fällt eine große Anzahl von Produkten in mehrere Kategorien.

Chamoni, *Gabriel* und *Gluchowski* unterscheiden MIS, DSS und EIS. [GGC97, S. XIII ff.] Dies entspricht der Einteilung von *Vetschera* [Vets95, S. 7 ff.] und zumindest im Kern der amerikanischen MSS-Literatur [HoWh96, S. 138 ff., SpWa-96, S. XIII, Turb95, S. 3 ff., WHR97, S. 10 ff.]. *Back* und *Seufert* erwähnen Query-Tools, Spreadsheet-Systeme, Systeme für Geschäftsgrafiken, Report-Generatoren, Statistik-Tools und OLAP-Tools. [BaSe97b, S. 6] *Burleson* ergänzt die Aufzählung um Data-Mining-Werkzeuge. [Burl97, S. 509 ff.] Dementsprechend werden Query-Tools, Report-Generatoren, MIS[10], EIS, DSS, OLAP, MQE, Spread-

---

[8] Dieser Effekt ist von nicht zu unterschätzender Bedeutung. Gerade in der Wirtschaftsinformatik kann durch den engen Kontakt mit der Praxis beobachtet werden, wie die Marketingabteilungen von Unternehmen wie z. B. Oracle, IBM oder Microsoft von Wissenschaftlern selbst dann Unterstützung erfahren, wenn dies aus wissenschaftlicher Sicht des Wirtschaftsinformatikers unangemessen zu sein scheint. Dieser Vorwurf wird z. B. gegen *Codd* erhoben [GGC97, S. 276 f.], der, in einem Artikel im Auftrag des MOLAP-Herstellers Arbor, RDBMS zur Speicherung von mehrdimensionalen Daten als ungeeignet bezeichnet, [CCS93] obwohl dies nachweislich unrichtig ist [Kimb96, S. 1 ff.]. Es sei an dieser Stelle bemerkt, dass dieses Vorgehen aus Sicht des Marketings durchaus wissenschaftlich korrekt und in der Praxis vielversprechend sein kann. [ScKa97, S. 266 ff.]

[9] Gedacht sei beispielsweise an die Begriffe „objektorientiert“, „visual“, „Management“ oder „Knowledge“. In einer bestimmten Periode gab es beispielsweise - laut jeweiligem Hersteller - fast ausschließlich objektorientierte Applikationen, obwohl die Produkte nur selten objektorientiert realisiert worden waren. *King* meint deshalb in seinem bekannten Artikel: „I have a cat named Trash. In the current political climate, it would seem that if I were trying to sell him (at least to a Computer Scientist), I would not stress that he is gentle to humans and is self-sufficient, living mostly on field mice. Rather, I would argue that he is object-oriented. What’s interesting is that this sort of false advertising is currently common in the database research community.“ [King-89, S. 24] *Holub* schreibt beispielsweise: „In the case of Microsoft, it has simply redefined object-oriented to mean something that fits with its product line. Visual Basic, for example, isn’t in the least bit object-oriented. Neither is Microsoft Foundation Classes (MFC) or most of the other Microsoft technology that claims to be object-oriented.“ [Holu99] Es sei wiederum bemerkt, dass dieses Vorgehen, betrachtet aus Sicht des Marketings, erfolgreich und deshalb positiv zu bewerten ist. [ScKa97, S. 266 ff.] Die Aufgabe des (wissenschaftlichen) Wirtschaftsinformatikers ist diesbezüglich Klarheit zu schaffen.

[10] Die Ausdrücke MIS, EIS und DSS tauchen bereits bei der Klassifikation nach der Problemklasse auf. MIS und EIS sind dem Data Support zugeordnet, DSS der Entscheidungsunterstützung. Diese Begrifflichkeiten werden jedoch **auch** für spezifische Applikationen verwendet und zeichnen sich durch ein bestimmtes Funktionsbündel aus. Es sei auf diesen Unterschied hingewiesen, um einer diesbezüglichen Verwechslung vorbeugen.

sheet Systems, Statistik-Werkzeuge und Data Mining Tools anhand ihrer wesentlichen Merkmale erläutert.

- Im einfachsten Fall werden sogenannte **Query Tools** oder **Datenbank-Front-Ends** verwendet, um damit auf Datenbestände von Datenbanken zugreifen zu können. Gewöhnlich handelt es sich bei den zum Einsatz gebrachten Datenbanksystemen um relationale DBMS. Die Zugriffe erfolgen dann mit SQL oder Query by Example (QBE). Im Vordergrund stehen einfache bis hoch komplexe Anfragen. Teilweise kann — sofern entsprechende Schnittstellen vorhanden sind — auch auf unterschiedliche DBMS-Kategorien bzw. -Produkte zugegriffen werden. Vorwiegende Zielsetzung dieser Werkzeuge ist es, wiederkehrende Anfragen an RDBMS zu richten und effizient abarbeiten zu lassen. Beispiele sind MS-Access und Oracles SQL*Plus.
- Da der Umgang mit Informationstechnologie vielfach noch als ungewohnt und umständlich empfunden wird, werden von den Anwendern nach wie vor Berichte in Papierform bevorzugt. Papierberichte sind einfach zu handhaben und leicht portabel. Dies erklärt den hohen Stellenwert von **Report-Generatoren** (Berichtsgeneratoren) in den Unternehmen. Im Unterschied zu Query Tools ist der Schwerpunkt tendenziell auf die ansprechende Gestaltung von Reports gerichtet. Entscheidend dabei ist die leichte Bedienbarkeit. Beispiele sind Oracles SQL*ReportWriter oder der Safari ReportWriter von ISSI.
- Der Begriff **Management Information System** wird im deutschsprachigen Raum oft als „Sammelbegriff für alle Kategorien von Informationssystemen für das Management“ [BaSe97a, S. 2] definiert. Ansonsten werden unter MIS einfache und komfortable Anfragewerkzeuge für operative Systeme verstanden. [BaSe97a, S. 2] Es handelt sich im wesentlichen um komfortablere Report-Generatoren bzw. um weniger komfortable EIS. Anders als beim EIS wird für MIS das mittlere Management als Zielgruppe angesehen. [Vets95, S. 82 ff.] Vielfach werden als Zielgruppe auch Controller oder Vorstandsassistenten genannt.

  Bestandteile sind Kennzahlenbäume, Key Performance Indicators (KPI) und andere repetitive meist fest programmierte Anfragen — im Gegensatz zu Ad-hoc-Queries — an den Datenbestand. [WHR97, S. 10 ff.] Dies ist ursächlich darin zu sehen, dass MIS als Vorläufer von EIS betrachtet werden können und zum damaligen Zeitpunkt des Aufkommens von MIS noch keine der Benutzerfreundlichkeit des EIS-Gedankens entsprechenden Applikationen bzw. GUI verfügbar waren. Damit wird klar, dass der Gedanke der EIS, welche als primäre Zielsetzung gerade einfache Bedienbarkeit und Ad-hoc-Anfragemöglichkeiten voraussetzen, nicht erfüllt ist. Dies läßt den Schluss zu, dass MIS einzig aus historischen bzw. technisch-historischen Gründen einer anderen Intention als EIS folgen. Dies widerspricht allerdings der Sichtweise von *Whinston* und

Holsapple. Sie ordnen MIS unter MSS ein, während EIS einen Unterpunkt von DSS darstellen. [HoWh96, S. 138 ff.]

Da MIS und EIS in gleicher Weise primär für den Data Support ausgelegt sind [GGC97, S. 237 ff.], scheint diese Einteilung weniger geeignet zu sein. *Vetschera* sieht unter Verwendung der obig erläuterten Kriterien deutliche Unterschiede zwischen MIS und EIS. Auch *Chamoni*, *Gabriel* und *Gluchowski* teilen MSS in die drei Komponenten MIS, DSS und EIS ein. Sie unterteilen den MSS-Abschnitt ihres MSS-Buches [GGC97] in diese drei Teile und widmen diesen jeweils ein eigenes Kapitel. Doch trotz dieser ausführlichen getrennten Darstellung resümieren sie letztlich, dass es außer der Benutzerfreundlichkeit keine bedeutenden Unterschiede zwischen EIS und MIS gäbe und ihnen für die Bevorzugung einer solch geringeren Benutzerfreundlichkeit keine sinnvollen Gründe einfielen. Ein Beispiel für ein MIS ist MIKs TZ-Info [Vets95, S. 96 ff., Mik99].

- **Executive Information Systems** bieten außer den bereits von MIS und Report-Generatoren angebotenen festen Berichten die Möglichkeit ad-hoc vertiefte Analysen durchzuführen. Den Ausgangspunkt bilden in der Regel hochaggregierte Kennzahlen, die nach verschiedenen Kriterien herunter gebrochen werden können. Zielsetzung ist dabei die direkte Online-Versorgung der Endbenutzer, um der Analyse dadurch den notwendigen „Spielcharakter“ zu verleihen und die ansonsten notwendigen Vorselektionen durch Dritte zu verringern. Gerade der „Spielcharakter“, welcher die Benutzer zu einem anderen Umgang mit den Daten ermutigt, schafft dadurch neue Potenziale zur Informationsgewinnung.

  Zur Verwirklichung dessen sind eine hohe Benutzerfreundlichkeit durch u. a. graphische Benutzeroberflächen, Ad-hoc-Anfragen und Drill-Down-Funktionalität wichtige Eigenschaften. [WRK91, S. 13 ff.] Verwendung findet auch das Exception Reporting, durch welches der Benutzer nach vordefinierten Regeln auf Abweichungen hingewiesen wird. Dies ist effizient und ermöglicht insbesondere bei ansonsten nur schwer identifizierbaren Abweichungen strukturell aufhebende Effekte zu erkennen, wie dies beispielsweise durch Aggregationen der Fall sein kann.

  Eine besondere Form von EIS sind Briefing Books, die in Analogie zu einer Zeitung aufgebaut sind. Die Bedienung ist dadurch besonders einfach und kann als Versuch verstanden werden, Manager überhaupt an die Benutzung von Informationstechnologie zu gewöhnen. Obwohl die Inhalte der Briefing Books auch dynamisch sein können, bleibt das Konzept doch weitgehend auf das einer Zeitung beschränkt. Ein Beispiel für einen EIS-Generator ist Forest & Trees von Computer Associates (CA).

- Einzelne **Decision Support Systems** unterscheiden sich anhand ihres problemklassen-spezifischen Charakters ganz erheblich [KMS97, S. 149 f.] und bilden insofern eine Ausnahme in dieser Klassifikation. Während beispielsweise EIS in der Regel sehr ähnliche Zielsetzungen verfolgen, existieren DSS für alle möglichen Arten von Entscheidungsunterstützung. Beispiele sind Werkzeuge zur Bewertung nicht quantifizierter Lösungsalternativen (Nutzwertanalyse), zur Implementierung und Simulation von System Dynamics Modellen (z. B. Vensim von Ventana), zur Analyse von Finanzdaten (z. B. Oracles Financial Analyzer) oder für Alternativenrechnungen von Steuererklärungen (z. B. WISO-Steuerprogramm von Buhl).

  Es gibt allerdings auch eine Reihe von Eigenschaften, die DSS gemeinsam sind. Der Schwerpunkt von DSS liegt auf der Unterstützung der späteren *simonschen* Entscheidungsphasen Design und Choice sowie auf der Beantwortung semistrukturierter nicht-repetitiver Problemstellungen mit Ad-hoc-Charakter. [Turb95, S. 7 ff.] Den Kern bilden mithin analytische Modelle mit deren Hilfe Zusammenhänge beschrieben und unter Umständen simuliert oder optimiert werden können. [GGC97, S. 165 ff] Typische Fragestellungen, die mit Hilfe analytischer Modelle beantwortet werden sollen, sind „What-if"- und „How-to-achieve"-Analysen. [KMS97, S. 149] Systembestandteile sind Daten-, Modell- und Methodenbanken. [Bode97, S. 262 f.]

- **OLAP-Systeme (Online Analytical Processing)** zeichnen sich vor allem durch ihr konzeptionelles Modell in Form eines Würfels bzw. eines mehrdimensionalen Arrays aus. Die Darstellung erfolgt durch hintereinander angeordnete Kreuztabellen. Innerhalb der Würfelzellen werden die Faktwerte (Measures) abgebildet, während an den Kanten des Würfels die Dimensionen abgetragen werden. Die Werte des Würfels werden mithin anhand dreier expliziter Dimensionen dargestellt. Implizit können beliebig viele Dimensionen zur weiteren Spezifikation der Faktwerte ergänzt werden.

  Diese der „natürlichen" Sichtweise]Expr93, S. 9] — bzw. dem mentalen Modell [Thom97, S. XV] — des Benutzers analoge Darstellung der Daten, verbunden mit einer ansprechenden grafischen Oberfläche und intuitiv ausführbaren Mausoperationen („Point & Click"), vereinfacht die Bedienbarkeit erheblich. Eine einheitliche leicht verständliche Sicht auf die Daten und die Möglichkeit selbständig die gewünschten Informationen selektieren zu können, ohne dafür die Unterstützung der DV-Abteilung oder anderer Spezialisten zu benötigen bzw. Programmcode oder SQL-Skripte schreiben zu müssen, bilden das Potenzial dieser Systeme. [ChGl99c, S. 262 ff.]

  Wenn OLAP-Systeme auch im Gegensatz zu DSS oder Statistikpaketen im Schwerpunkt zur Beantwortung einfacherer Fragestellungen dienen, so sind es gerade diese Fragen, welche das Hauptinteresse der Benutzer darstellen. [WHR97, S. 146 ff.] Der besondere Ad-hoc-Charakter ergänzt durch Slice-&-

Dice-Techniken sowie Drill-Down- bzw. Drill-Across-Funktionalität verwirklicht so in besonderem Maße einen zentralen Bestandteil des EIS-Konzepts.[11]

Unterscheiden lassen sich OLAP-Front-Ends und OLAP-Datenhaltungskomponenten. Letztere wiederum werden nach der zugrundeliegenden Speichertechnologie unterschieden. Multidimensionale OLAP-Systeme (MOLAP) verfügen über proprietäre DBMS (z. B. Oracles Express Server), welche die Array-Strukturen möglichst effizient zu speichern versuchen. Relationale OLAP-Systeme (ROLAP) verwenden RDBMS zur Speicherung der Daten. [Cham97, S. 294 f.] Hierfür werden in der Regel spezielle Datenmodelle in einer Variante des Star-Schemas verwendet. RDBMS bieten hierfür inzwischen besondere Zugriffsmechanismen wie den Star-Join oder den Binary-Index (z. B. Oracles RDBMS ab Version 7.3).

Ein Beispiel für ein MOLAP-Front-End ist PowerPlay von Cognos; eines für ein ROLAP-Front-End ist der Discoverer von Oracle. BusinessObjects von der gleichnamigen Firma ist ein Vertreter eines OLAP-Front-Ends, welches gleichzeitig RDBMS und multidimensionale DBMS unterstützt.

- **Managed Query Environments (MQE)** sind Werkzeuge zur Unterstützung von Anfragen auf verschiedene Datenquellen, ohne dass die Benutzer nähere Kenntnisse über die Datenquellen, auf welche zugegriffen wird, und die zu verwendenden Anfragesprachen (meist SQL) benötigen. Um dieses Ziel zu erreichen, werden einfach bedienbare grafische Benutzeroberflächen und an der Domäne des Anwenders angelehnte Metaphern (z. B. Geschäftsbereich, Dimension) eingesetzt.

  MQE stellen damit eine Teilkomponente von EIS dar, sind jedoch auch als eigenständige Anwendungen erhältlich. Für den Management Support sind MQE insofern unzureichend. *Watson*, *Houdeshel* und *Rainer* vermuten das Aufkommen einer solchen Begrifflichkeit in dem Zwang zur Erfindung immer neuer Namen für MSS-Komponenten. [WHR97, S. 14] Ein Beispiel ist Impromptu von Cognos.

- **Spreadsheet Systems** (SSS, Tabellenkalkulationssoftware) stellen Daten in Form von ein- (oder zweidimensionalen Kreuz-) Tabellen[12] dar. Jede Zelle einer Tabelle bildet ein Datum ab und kann entweder eine Konstante oder eine Formel zur Generierung von variablen Ergebnisdaten enthalten. Eine Formel

---

[11] OLAP-Systeme entwickelten sich ursprünglich aus DSS-Werkzeugen. Die inzwischen vernachlässigten Analyse-Funktionalitäten und die Schwerpunktsetzung zugunsten des „ease of use“ [Thom97, S. XVIII ff.] bedingen eine Umklassifizierung zu den EIS-Werkzeugen.

[12] Inzwischen bietet eine gängige Tabellenkalkulationssoftware auch die Unterstützung für mehrdimensionale Daten durch Pivot-Tabellen an. Pivot-Tabellen stellen allerdings einen Bruch im Spreadsheet-Konzept dar. Beispielsweise müssen bei der Verwendung von MS-Excel die für eine Pivot-Tabelle benötigten Daten aus einem Arbeitsblatt importiert werden. [Thom97, S. 23]

bezieht sich auf andere Zellen, wodurch zwischen den einzelnen Zellen Bezüge hergestellt werden. [Nast97, S. 397 f.]

Dieses sehr einfache Konzept in Analogie zu einem konventionellen Arbeitsblatt erlaubt es einer großen Anzahl von Benutzern, Daten in gewünschter Weise zu verknüpfen, um dadurch ihren Informationsbedarf zu befriedigen. Komfortable Datenimport- und Grafikpräsentations-Features erhöhen die Nutzbarkeit dieser Systeme weiter. Diese SSS-Eigenschaften und deren Verfügbarkeit als Office-Paket-Bestandteil auf fast jedem PC führten zu einer starken Verbreitung dieser Systeme. In der Konsequenz sind Spreadsheet-Systeme die am häufigsten verwendeten MSS-Front-Ends.[13] [Thom97, S. 21] Das Potenzial von SSS liegt mithin in der Chance, kurzfristig große Teile der Informations-Anforderungen ohne besonderen Aufwand an Zeit und Know-how erfüllen zu können. Dieses bindet viele Endanwender an Tabellenkalkulationssoftware.[14]

Die wesentlichen Nachteile von SSS sind die nicht stattfindende Trennung von Daten, Formeln und Layout, die fehlende Datengetriebenheit bzw. dynamische Datenbankunterstützung, die fehlende Möglichkeit der Datensicht-Änderung in einfacher Form (slice und dice), mangelhafte Spreadsheet-Verknüpfungs-Eigenschaften [Thom97, S. 31] sowie insbesondere die lokale Ausrichtung solcher Systeme auf einzelne Anwender oder Anwendergruppen, wodurch die Integration der Einzellösungen in ein unternehmensweites DV-Konzept schwierig zu realisieren ist, weil der Aufbau und die Pflege der Modelle und Datenbestände weitgehend losgelöst von den anderen Systemen erfolgt, der Applikations-Wildwuchs zu inkonsistenten Ergebnissen führt und die Wartbarkeit aufgrund steigender Komplexität und üblicherweise mangelhafter Dokumentation immer schwieriger wird [ChGl99b, S. 4 ff.]. Ein Beispiel ist Excel von Microsoft.

- **Statistik-Werkzeuge** sind Programmpakete, deren Schwerpunkt auf der Bereitstellung und Anwendung statistischer Verfahren liegt. Insofern können sie als Methodenbanken mit statistischen Funktionen verstanden werden. Typische Verfahren, die unterstützt werden, sind Häufigkeitsverteilungen, Mittelwertvergleiche, Varianzanalysen, einfache und multiple Regression und Faktorenanalyse. [Humm97, S. 381 f.] Ein Beispiel ist das SPSS-Paket von SPSS.

---

[13] *Devlin* betont den Aspekt, dass es zum ersten Mal für die Anwender möglich wurde, Analysen durchzuführen, ohne auf die Unterstützung der zentralen DV-Abteilung angewiesen zu sein. Die sonst üblichen Kommunikationsprobleme und Zeitverzögerungen konnten dadurch deutlich reduziert werden. [Devl97, S. 9 f.]

[14] Dies führt im Extremfall sogar soweit, dass SSS selbst dann als Mittel zur Lösung angewendet werden, wenn dies offensichtlich inadäquat ist. Beispielsweise verwendet die Berliner Elektro Holding (BEH) MS-Excel um damit ihren Jahresabschluss zu erstellen.

- **Data Mining Tools** haben zum Ziel, in umfangreichen/großen vorhandenen Datenbeständen automatisch bedeutsame Muster zu erkennen und dem Anwender als Wissen zu präsentieren. [BiHa97, S. 104] Eine häufig verwendete und synonym gebrauchte Bezeichnung ist Knowledge Discovery in Databases (KDD).[15] Bestandteile sind zum einen Datenbanksysteme zur Speicherung und Verwaltung der Daten und zum anderen den Statistik-Werkzeugen vergleichbare Methodenbanken. Die Methoden bedienen sich dabei Verfahren aus der Statistik und der Künstlichen Intelligenz. Ein Beispiel ist Intelligent Miner von IBM.

2.1.4 Wissensbasierte Klassifikation

Als weiteres mögliches Unterscheidungskriterium lassen sich MSS in wissensbasierte und konventionell(e) (programmierte) Systeme unterscheiden. *Rieger* differenziert bezüglich EIS (im Sinne von Data Support) und DSS grundsätzlich zwischen konventionellen und wissensbasierten Systemen (siehe Abbildung 3 auf Seite 18). [Rieg93]

Dementsprechend ist auch die Bezeichnung der Intelligent Support Systems (ISS) folgerichtig. [King96, S. 182 ff., Scot83] *Chamoni*, *Gabriel* und *Gluchowski* beschreiben verschiedene Integrationsstufen von WBS/XPS und DSS. [GGC97, S. 254 ff.]

Insgesamt besteht heute der Trend wissensbasierte Komponenten in MSS zu integrieren. Statt der ursprünglichen Trennung von konventionellen Systemen und WBS wird damit versucht, WBS in den Teilbereichen einzusetzen, in denen sie ihre besonderen Stärken realisieren können. Bedingt durch diese Entwicklung ist die Unterscheidung in konventionelle und wissensbasierte MSS nicht weiter von hoher Priorität.

2.1.5 Neue Begriffe im MSS-Umfeld

Wie *Watson*, *Houdeshel* und *Rainer* betonen, besteht in der IT-Branche nach Verstreichen einer gewissen Zeitspanne der Zwang, auch bewährte Konzepte bzw. Produkte mit neuen Bezeichnungen zu versehen. Die gewünschte Aufmerksamkeit solle auf diese Weise nicht abflauen oder, noch besser, wieder neu darauf gelenkt werden. Dies gelte auch für MSS. Beispielsweise sei die Begrifflichkeit der Business Intelligence Tools (BIT) eine neue Wortschöpfung der Gartner Group für EIS. [WHR97, S. 14]

---

[15] Data Mining wird teilweise auch als Untermenge von KDD gesehen. [Düsi99, S. 346 ff.]

Viele dieser Begrifflichkeiten dominieren den Markt relativ schnell und werden in der Regel auch vom Wissenschaftsbereich verzögerungsfrei übernommen.[16] Dadurch ist es sinnvoll, eine Reihe neuer Begriffe aus dem MSS-Umfeld zu begutachten und auf evtl. neue Konzepte bzw. Intentionen hin zu untersuchen. Hierbei kann gelegentlich alleine die veränderte Betonung einzelner Aspekte durch Umbenennungen positive Entwicklungstendenzen auslösen. Das regelmäßig anzutreffende Argument des „alten Weines in neuen Schläuchen" ist insofern ex ante nicht haltbar.

Die wichtigsten „Buzzwords" für MSS-Konzepte sind momentan Business Intelligence Tools, Knowledge Management, Corporate Information Factory und Analytische Informationssysteme. Sie werden folgend behandelt.

- *White* definiert **Business Intelligence Tools (BIT)** als „… set of technologies and products for supplying users with the information they need to answer business questions, and make tactical and strategic business decisions". Er unterteilt MSS in BIT und Data Warehouse. [Whit99]

  Schinzer und *Bange* verstehen unter BIT Applikationen, welche die Informationsanfrage, -präsentation und -analyse ermöglichen und einfach bedienbar sind. [ScBa99, S. 47 ff.] Allerdings verstehen sie OLAP als Werkzeug zur Informationsmodellierung multidimensionaler Sichten und verlangen eine OLAP-Schicht als Zwischenschicht zwischen Data Warehouse und BIT. Dementsprechend formuliert *Schinzer* „BIT haben die Aufgabe, die in den Datenwürfeln enthaltenen Informationen zu analysieren und in einer ansprechenden Form aufzubereiten." [Schi97, S. 8] Die Beschränkung der BIT-Definition auf OLAP-Front-Ends ist zu eng. Die ausschließliche Datenversorgung in Form multidimensionaler Datenstrukturen ist nicht zu begründen.

  Der Aussage *Watson*, *Houdeshel* und *Rainer* im weitesten Sinne folgend [WHR97, S. 14 ff.] kann davon ausgegangen werden, dass der Begriff BIT in der Regel als Bezeichnung für alle möglichen Arten von MSS-Front-Ends Verwendung findet. EIS, DSS, Data Mining Tools, Query Tools, Reporting Tools und beliebige Kombinationen dieser Werkzeuge werden darunter zusammengefaßt.

  Der Begriff hat den Vorteil, im Gegensatz zu den vorherigen Ansätzen wie MSS, MIS oder EIS auf die Manager-Fokussierung zu verzichten. Statt dessen besteht die Chance, ähnlich wie beim Knowledge Management, den Blickwinkel auf das Potenzial des Unternehmens in Form von Information bzw. Wissens (Business Intelligence) zu lenken.

---

[16] Diese oft kritiklose Übernahme von Begriffen aktiver Marketingstrategen muß wiederum sehr kritisch beurteilt werden. Ab einer gewissen Begriffsetablierung stünde die Verweigerung der Wissenschaft gegenüber den neuen Wortschöpfungen allerdings der Wissenschafts-Praxis-Integration im Wege.

- In Anbetracht der Tatsache, dass die Begriffe Daten, Information und Wissen regelmäßig in einer Reihe genannt werden, stellt alleine das Aufkommen des Begriffs **Knowledge Management** eine logische Folge dar. Nach einer Konzentration auf Data Management und einer Flut von Diskussionen zum Thema Information Management steht Knowledge Management im Zentrum zahlreicher aktueller Tagungen und Veröffentlichungen. Dies ist vergleichbar mit der Entwicklung beim Data Warehouse bzw. Information Warehouse und *Martins* Forderung nach einem Knowledge Warehouse [Mart97, S. 27, Mart98c, S. 415 ff.].[17]

  Damit ergibt sich zwangsläufig die Frage nach der hinter Knowledge Management stehenden Bedeutung:

  ◊ *Probst*, *Raub* und *Romhardt* verstehen Knowledge Management als „… integriertes Invterventionskonzept, das sich mit den Möglichkeiten zur Gestaltung der organisationalen Wissensbasis befaßt." [PRR99, S. 47] Zweck ist die „… zielorientierte Nutzung und Entwicklung von Wissen und Fähigkeiten, welche für den Organisationszweck als notwendig angesehen werden." [PRR99, S. 47] Unter der organisationalen Wissensbasis verstehen sie die „… individuellen und kollektiven Wissensbestände […], auf die eine Organisation zur Lösung ihrer Aufgaben zurückgreifen kann. Sie umfaßt darüber hinaus die Daten- und Informationsbestände, auf welchen individuelles und organisationales Wissen aufbaut." [PRR99, S. 46]

  ◊ *Huang*, *Lee* und *Wang* definieren Knowledge Management als „… organizing and structuring institutional processes, mechanisms, and infrastructures to create, store, and reuse organizational knowledge." [HLW99, S. 113]

  ◊ *Frank* bezeichnet Knowledge Management als „… concepts and theories that help to generate, organize, and leverage knowledge in social settings." [Fran99, S. 5]

  ◊ *O'Dell* und *Grayson* sehen Knowledge Management als „… conscious strategy of getting the right knowledge to the right people at the right time and helping people share and put information into action in ways that strive to improve organizational performance." [OdGr98, S. 6]

  ◊ *Soeffky* bezeichnet als Aufgabe des Knowledge Managements „… das Sammeln, Organisieren, Übertragen und Verteilen von Wissen unter Einbeziehung von Lernprozessen…" [Soef99, S. 22]

  Damit zielen alle in die Richtung, Wissen effizient und effektiv zu sammeln, zu speichern, neu zu erzeugen und verfügbar zu machen. Unklar bleibt aller-

[17] Auch wenn in diesem Fall die zeitliche Reihenfolge des Aufkommens der Begriffe Information Warehouse und Data Warehouse unklar ist. [DeMu88, S. 60 ff., Inmo96 (erste Auflage 1990)]

dings, was unter Wissen verstanden wird. Es wird zwar regelmäßig als eine höherwertige Form von Daten und in vielen Fällen als höherwertige Form von Information gesehen, doch gehen die einzelnen Definitionen je nach Blickwinkel (Decision Support, Deduktion, Management, Künstliche Intelligenz, Philosophie, Psychologie, Systemwissenschaften et cetera), weit auseinander. [Bell97, Colo98, S. 87, Dahr97, S. 5, Fran99, S. 6 f., HLW99, S. 13, HoWh96, S. 24, NoTa95, S. 58 ff., PRR99, S. 36 ff., Seng90, S. 5 ff., Turb95, S. 126 ff.][18]

Ebenso wird beim Knowledge Management je nach Blickwinkel der Fokus auf Individuen, die Organisation oder Informationstechnologie bzw. Daten und Informationen [InHa94, S. 91 f.], Entscheidungsprozesse [HoWh96, S. 32 ff.], Lernprozesse [Dörn97, S. 107 ff., NoTa95, S. 58 ff., Seng90, S. 139 ff.] oder Wissensgenerierung [Colo98, S. 85 ff., Dahr97, S. 2 ff.] gelegt. Von besonderer Bedeutung ist das Organisationale Lernen. [PrBü94, S. 15 ff.]

Auch wenn mithin bisher keine Einigkeit über die Kerninhalte des Knowledge-Management-Konzepts bestehen, so ist doch festzuhalten, dass die Zielsetzung eine interdisziplinäre, aber im Schwerpunkt betriebswirtschaftliche Aufgabenstellung beinhaltet, die über das MSS-Konzept hinausgeht. Informations- und Kommunikationstechnologie stellt jedenfalls ein Instrumentarium zur Verfügung, welches zur Lösung der Knowledge-Management-Probleme einen zentralen Beitrag leistet. *Manthey* identifiziert als durch Informationstechnologie unterstützungsfähige Knowledge-Management-Prozesse [Mant99]:

◊ Erfassen und Kodifizieren

◊ Lokalisieren

◊ Suchen, Weitergeben und Nutzen

◊ Aufbauen

◊ Messen und Dokumentieren

Zwar werden teilweise rechnergestützte Knowledge-Management-Lösungen in Form einfacher Hypertext-Anwendungen realisiert [Fran99, S. 5 ff.] (z. B. Intranet, Lotus Notes), doch zeigen *Whinston* und *Holsapple*, dass zu diesem Zweck ein breites Spektrum wie beispielsweise „Text management, Database management, Forms management, Report generation, Business graphics, Spreadsheet analysis, Solvers, Programming [and] Rule management“ zur Verfügung stehen [HoWh96, S. 281].

Die Integration dieser Techniken und weiterer Systeme und die Absicht, Wissen zu sammeln, zu speichern, neu zu erzeugen und verfügbar zu machen, ist auch das erklärte Ziel von MSS. Insofern kann Knowledge Management als

[18] Der Vergleich der einzelnen Definitionen würde den Rahmen dieser Arbeit übersteigen.

eine Erweiterung des MSS-Konzepts um z. B. Organisationales Lernen und Organisationskonzepte (Managementtheorie) gesehen werden.

Deshalb ist die Entstehung des Begriffs Knowledge Management vorwiegend positiv zu sehen, sofern es gelingt, diesen weitgehend einheitlich zu verwenden und in einen festen wissenschaftlichen Bezugsrahmen einzubetten. Statt der eingeschränkten Sicht auf Managerarbeitsplätze findet eine Betrachtung der Organisation statt. Anstelle von Daten- bzw. Informationsaspekten steht der gesamte Wissensprozess im Vordergrund.

- Der von *Inmon* verwendete Begriff[19] **Corporate Information Factory (CIF)** beinhaltet die den MSS-Front-Ends vorgelagerten Systeme. Darunter fallen maximal der Operational Data Store (ODS), das Data Warehouse inklusive ETL-Funktionalität und Metadatenverwaltung sowie die Data Marts. Die CIF besteht jedoch zumindest aus einem Data Warehouse. Die Zusammensetzung der CIF-Bestandteile ist von den Unternehmensbedürfnissen bzw. dessen IT-Struktur abhängig. [Inmo99, S. 21 ff.]

  Zu erwarten ist eine Erweiterung des Begriffs zur Corporate oder Enterprise Knowledge Factory. Jedoch stellt sich die Frage, weshalb MSS-Front-Ends aus dieser Definition ausgeklammert werden, können MSS-Front-Ends doch sicherlich zu den informations- bzw. wissens-produzierenden Applikationen gerechnet werden. Weil die *inmonsche* Definition der CIF die den MSS-Front-Ends vorgelagerten Systeme bezeichnet, ist sie dienlich und findet folglich in dieser Arbeit Verwendung.

- *Chamoni* und *Gluchowski* verwenden für ihren Sammelband den Begriff **Analytische Informationssysteme (AIS)** [ChGl99a], da die Bezeichnung „… einerseits noch nicht durch ein bestimmtes Begriffsverständnis belegt ist und andererseits zum Ausdruck kommt, dass die Informationsversorgung betrieblicher Fach- und Führungskräfte zu Analysezwecken im Vordergrund steht.“ [ChGl99b, S. 11]

  Schinzer und *Bange* definieren AIS als Werkzeuge zur Selektion und Speicherung entscheidungsrelevanter Informationen, zur entscheidungsorientierten Modellierung und Auswertung sowie zur ungerichteten Informationsselektion und -analyse. Sie nennen beispielhaft als Bestandteile: MIS, DSS, EIS, Data Warehouse, BIT, OLAP und Data Mining. [ScBa99, S. 47 ff.]

  Notwendig ist die Unterscheidung der Begriffe AIS und Informationssystem an sich, um eine Abgrenzung zwischen beiden ziehen zu können. Die Bezeichnung Informationssystem ist auch für die operativen Systeme gültig und wird daher als Oberbegriff für alle „… Systeme zur Speicherung, Wiedergewinnung,

[19] Allerdings gibt *Mucksch* zu bedenken, dass die „... vom debis Systemhaus, Geschäftsstelle Führungsinformationssysteme geprägte Information Factory.“ [Muck99, S. 175] eine andere Bedeutung habe.

Verknüpfung und Auswertung von Informationen…“ [Dude93, S. 317] wie einerseits Administrations- und Dispositionssysteme und andererseits MSS verwendet. *Stahlknecht* und *Hasenkamp* setzen Informationssysteme mit Führungsinformationssystemen gleich [StHa97, S. 426], dies ist allerdings ungewöhnlich [BeSc96, S. 9 ff., Dude93, S. 317 ff., LaLa95, S. 8 ff., Sche98b, S. 5 f., OHE96, S. 199 ff.].

AIS entsprechen inhaltlich dem von *Mertens* und *Griese* verwendeten Ausdruck der Planungs- und Kontrollsysteme [MeGr93, S. 1 ff.], gestatten mit dem Begriff Analyse jedoch ein erweitertes Spektrum gegenüber Planung und Kontrolle.

Insgesamt fällt der geringe Bedeutungsunterschied zwischen AIS und MSS auf. *Chamoni* und *Gluchowski* haben fast zeitgleich zur Veröffentlichung eines MSS-Buches [GGC97] einen AIS-Sammelband [ChGl99a][20] herausgegeben. Beide Bücher unterscheiden sich zwar anhand der Klassifikation bzw. Gliederung, inhaltlich jedoch kaum. Der historische Begriff MSS und der inzwischen sicherlich bezeichnendere des AIS können mithin synonym benutzt werden.

Alle diese neuen Ansätze fokussieren nicht mehr auf die Betrachtung von Managerarbeitsplätzen:

- Business Intelligence Tools und Knowledge Management heben besonders den Wissensaspekt hervor und verweisen noch stärker auf das Potenzial externer und semistrukturierter Information bzw. von Wissen.
- Knowledge Management und Corporate Information Factory betonen besonders den ganzheitlichen Unternehmens- bzw. Organisationsbezug und die aktive Erzeugung/Generierung von Informationen und Wissen.
- BIT bringen jedoch keine weitergehenden Neuerungen. Sie werden in dieser Arbeit mit MSS-Front-Ends gleichgesetzt (siehe Abbildung 5).
- Die Corporate Information Factory stellt eine geeignete Begrifflichkeit für die den MSS-Front-Ends vorgelagerten Systeme dar.
- Analytische Informationssysteme werden synonym zu MSS verwendet, wobei letztere Bezeichnung aufgrund ihrer Historie einen größeren Bekanntheitsgrad aufweist und deshalb den Kernbegriff bildet.
- Knowledge Management wird als umfassenderes Konzept verstanden. Es umfaßt die Aufgaben von MSS vollständig. Das Unternehmens-Knowledge-Management-Konzept steht in Wechselwirkung mit dem MSS-Konzept und darf deshalb nicht losgelöst betrachtet werden. Trotzdem werden die über MSS

[20] Die erste Auflage erschien 1998.

hinausgehenden Aspekte des Knowledge Managements an dieser Stelle nicht weiter betrachtet, da dies über den Rahmen dieser Arbeit hinaus ginge.

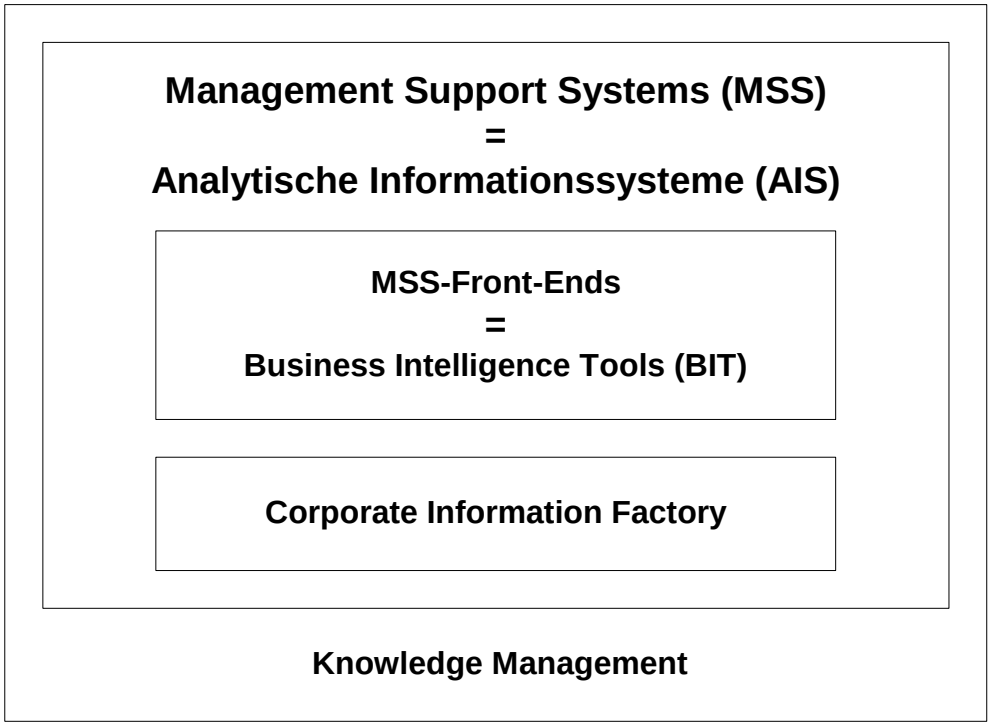


Abbildung 5: Einordnung neuer Begriffe im MSS-Umfeld

2.1.6 MSS-Definition

Insgesamt lassen sich Management Support Systems damit definieren, als Systeme zur Unterstützung von Personen[21] bei der Versorgung mit und Analyse von Daten bzw. Informationen, um im Sinne des Unternehmenszieles bessere Entscheidungen zu treffen und Probleme effizienter lösen zu können. Dazu wird unter Zuhilfenahme von Informations- und Kommunikationstechnologie vorhandene Information bzw. vorhandenes Wissen gesammelt und gespeichert, neue Information bzw. neues Wissen erzeugt und den Mitarbeitern sowie anderen interessierten Personen Bedürfnisstruktur-adäquat zur Verfügung gestellt.

---

[21] *Chamoni*, *Gluchowski* und *Gabriel* unterscheiden Führungskräfte, Fachkräfte, Sachbearbeiter und Hilfskräfte. Die beiden ersten Gruppen bezeichnen sie zusammen als Manager und beschränken die Anwendung von MSS auf diese Manager-Gruppe. [GGC97, S. 1 ff.] Diese Einschränkung entspricht jedoch nicht den Gegebenheiten (siehe oben). Es hat sich gezeigt, dass MSS selbst für Hilfskräfte und in der Fertigung tätige Arbeiter Nutzenpotenziale aufweisen können. Sogar die Beschränkung auf die Gruppe der Mitarbeiter ist inzwischen zu eng gefasst. Auch für Kunden, Lieferanten und Dritte mit berechtigtem Interesse kann der Zugang zu einem MSS fruchtbar sein. So wird beispielsweise der Öffentlichkeit die Benutzung eines OLAP-Systems der Universität Osnabrück über einen Web-Client ermöglicht, um damit einer gewissen Informationspflicht nachzukommen. [Stif99]

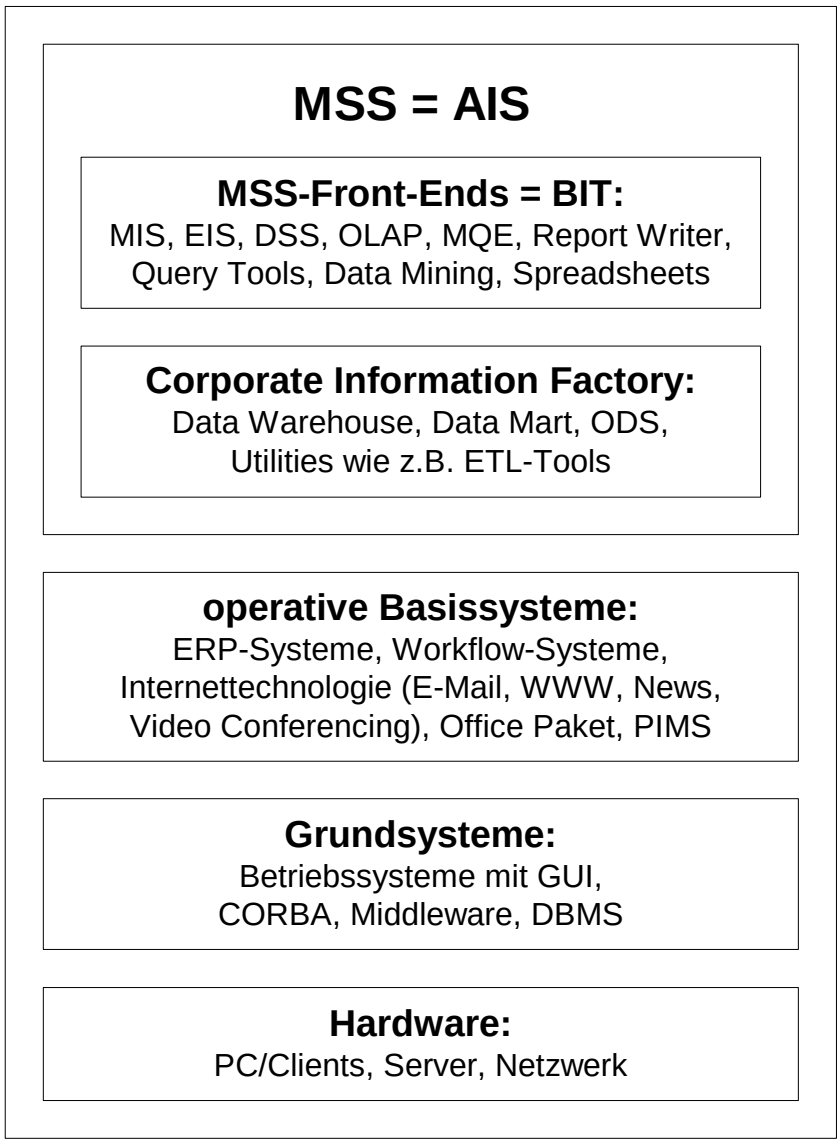


Abbildung 6: Definition des Begriffs Management Support Systems

Zum Betrieb eines Unternehmens-MSS müssen heute verschiedene Komponenten vorausgesetzt werden bzw. sind notwendig (siehe Abbildung 6). Auf der Hardware setzen Grundsysteme auf, die wiederum die Funktionsfähigkeit der operativen Basissysteme gestatten.

- Als Hardware werden vernetzte Client-Server-Umgebungen vorausgesetzt.
- Zu den Grundsystemen gehören leistungsfähige benutzerfreundliche Betriebsysteme mit grafischer Oberfläche (GUI) (z. B. Solaris, Linux, NT), DBMS (z. B. Oracle, Informix, DB/2) und Middleware (z. B. CORBA).
- Operative Basissysteme sind
  - ◊ ERP- und Workflow-Systeme (z. B. Baan, SAP/R3),

- ◊ Office-Pakete (Textverarbeitung, Tabellenkalkulation, Präsentationsprogramm, Grafikanwendung, Datenbank-Front-End und Personal Information Manager (PIMS, z. B. Kalender), z. B. Star-Office von Sun) und
- ◊ Internettechnologie (E-Mail, WWW, News Groups und Video Conferencing).

MSS werden mit Analytischen Informationssystemen gleichgesetzt und können weiter untergliedert werden in MSS-Front-Ends und Corporate Information Factory. MSS-Front-Ends, die auch als Business Intelligent Systems bezeichnet werden, sind beispielsweise MIS, EIS, DSS, OLAP, Report Writer, MQE, Query Tools, Data Mining bzw. KDD Tools und Spreadsheet Systems. Die Corporate Information Factory besteht aus dem Data Warehouse, Data Marts, dem Operational Data Store sowie Hilfsprogrammen wie den ETL-Tools.

Diese MSS-Definition weicht in einigen Punkten fundamental von bisherigen Definitionen ab. Dies ist notwendig, um den entwicklungsbedingten realen Gegebenheiten der MSS-Landschaft Rechnung zu tragen und so eine im *popperschen* Sinne zweckdienlichere Definition zu bilden [Popp94, S. 20 ff.]. Besonders die Abkehr von der arbeitsplatzbezogenen hin zu einer funktionsbedürfnisorientierten Definition ist von Bedeutung für den Gegenstand der MSS-Forschung und -Praxis. Hardware, Grundsysteme und operative Basissysteme sind dadurch kein unmittelbarer Gegenstand der Betrachtung mehr.

MSS unterstützen damit ein breites Aufgabenspektrum und bilden einen grundlegenden Baustein zur Realisation einer Knowledge-Management-Konzeption, eines umfassenden Controlling-Ansatzes und zur Unternehmensteuerung insgesamt. Sie bilden das logische Komplement zu den operativen Informationssystemen [ChGl99b, S. 10 ff.] und ermöglichen dadurch auch neue Organisationsformen.

Nach der ursprünglichen Fokussierung der MSS-Forschung auf das höhere Management, dienen MSS heute vielmehr der Aufgabe, die Verlagerung von Entscheidungskompetenzen auch auf die unteren Mitarbeiterebenen zu unterstützen und somit auf allen Hierarchiestufen möglich zu machen.[22] Die Notwendigkeit zur Ablösung des tayloristischen Prinzips zu Gunsten angereicherter Arbeitsplätze mit dezentralisierten Entscheidungskompetenzen ergibt sich aus dem

---

[22] Informations- und Kommunikationstechnologie fand bereits vielfach Verwendung, um im operativen Geschäft andere Organisationsformen implementieren zu können. Ein herausragend positives Beispiel ist das Osnabrücker Bürgeramt. Anstelle der besonders für die „Kunden“ ineffizienten Aufteilung der Aufgaben wie beispielsweise Paßstelle, Meldewesen, Beglaubigungen von Kopien oder Kasse wurden kombinierte Arbeitsplätze geschaffen. Der „Kunde“ bekommt alle „Leistungen“ an einer Stelle. Die höhere Effizienz der Prozesse und die gestiegene Zufriedenheit beider Seiten sind offensichtlich. Die Verlagerung auch von Entscheidungskompetenzen ist eine vielversprechende Konsequenz. Hierfür sind zwei Schritte notwendig. Erstens müssen die neuen „Entscheider“ mit der notwendigen Information bzw. dem notwendigen Wissen versorgt werden. Zweitens müssen die bisherigen Entscheider oder eine entsprechende Instanz in die Lage versetzt werden, die getroffenen Entscheidungen zu kontrollieren und in ihren Folgen zu bewerten.

Zwang, immer schneller flexibel auf Änderungen des Marktes reagieren zu müssen. Die wachsende Schwerfälligkeit der durch die Fusionierungswelle und den Verdrängungswettbewerb stetig an Größe zunehmenden Unternehmen verlangt deshalb eine Überkompensierung dieses Effektes. Veränderte Organisationskonzepte in Verbindung mit neuen Informations- und Kommunikationstechnologiepotenzialen stellen dabei einen, wenn nicht gar den entscheidenden Einflußfaktor dar. MSS bieten zur Lösung dieser Problematik nicht nur die Chance den Produktionsfaktor Information überhaupt verfügbar bzw. sinnvoll einsetzbar machen zu können, sondern im Gegensatz zum ursprünglichen Ansatz das höhere Management geeignet unterstützen zu wollen, das Potenzial einer Informationsversorgung und Entscheidungsunterstützung aller Mitarbeiter.

MSS lassen sich auch nach den verschiedenen Schichten in der Gesamtarchitektur klassifizieren. Diesbezüglich wird auf Abschnitt 2.4 verwiesen, in welchem die einzelnen Schichten im Detail beschrieben werden. Es handelt sich um MSS-Front-Ends, das Data Warehouse als eine vorgelagerte rekonziliierende Ordnungs- und Speicherkomponente und Vorsysteme als Datenquellen wie die operativen Administrations- und Dispositionssysteme oder externe Datenquellen wie statistisches Material von Marktforschungsinstituten und Internetdiensten-Informationen.

Der MSS-Basisbaustein ist das Data Warehouse, da es zentral die Daten für die Analyse-Front-Ends bereitstellt. Von der Qualität des Data Warehouse ist die Analyse-Nützlichkeit aller aufsetzenden Werkzeuge unmittelbar/direkt abhängig. Diesem Umstand Rechnung tragend ist das Data Warehouse Gegenstand dieser Arbeit und wird in den nächsten Kapiteln detailliert beschrieben.

2.2 Entwicklungspfad des Data-Warehouse-Konzepts

Die Wissenschaft der Informations- und Kommunikationssysteme ist noch jung. Die Anfänge liegen um 1960. Es verwundert mithin nicht, dass gelegentlich von einer noch unausgereiften Wissenschaft gesprochen wird. [Inmo96, S. 1 ff.] Dies wird auch anhand der historischen Entwicklung des Data-Warehouse-Konzepts deutlich. Versuch und Irrtum dominieren einzelne Phasen. Immer wieder werden Irrwege eingeschlagen und es entstehen geradezu chaotische IT-Organisationen. Die revolutionäre Entwicklung der Hardware-Technologie [OHE96, S. 3 ff.], Fortschritte im Softwarebereich [Your93, S. 21 f.] und sich wandelnde Management-Strategien [Stae91, S. 21 ff.], Organisations-Konzepte [GeRo92, S. 11 ff.] und Unternehmenskulturen [Sche97a, S. 1 ff.] sind die wesentlichen Einflussfaktoren dieses Prozesses. Sie stehen in vielfacher Wechselwirkung und bedingen sich gegenseitig. Welche der einzelnen Faktoren kausal Entwicklungs-auslösend wirken oder Ergebnis sind, ist dagegen nur schwerlich zu identifizieren.

Das Data-Warehouse-Konzept stellt in diesem Prozess eine konsequente Entwicklung dar. Es ist keine revolutionäre Erfindung und entspringt auch nicht den Überlegungen einer einzelnen Person. Das Data Warehouse mußte entstehen wie höhere Programmiersprachen, Netzwerke oder grafische Benutzeroberflächen und kann deshalb nicht mit anderen Modewellen verglichen werden. *Zachman* meint:

> „Although it was from my close friend Bill Inmon that I (along with many) learned about the data warehouse, I am sure there were others who were 'inventing' the concept at the same time, because the data warehouse is clearly an idea whose time has come. […] Data warehouse is indeed a function of the times."[23] [Zach97, S. VII]

Anhand der historischen Entwicklung des Data-Warehous-Konzepts wird leicht verständlich, welche einzelnen Funktionen ein Data Warehouse zu erfüllen hat. Damit wird der Grundstein für die sich anschließende Zielsetzung und ideale Architektur gelegt. Die Diskussion erfolgt in sechs Phasen: Informationstechnologie zur Prozessautomatisierung, Software-Engineering und Datenorientierung, Benutzer-Fokussierung und Dezentralisierung, Client/Server-Strukturierung, Wettbewerbsfaktor Information sowie Data Warehouse.

Diese Phasen sind jedoch nicht streng nach einer zeitlichen Reihenfolge angeordnet. Vielmehr wird versucht, die einzelnen Entwicklungstrends bzw. die sich ursächlich bedingenden Entwicklungsstufen zu verdeutlichen. Die Phasen überschneiden sich deshalb teilweise sowohl zeitlich als auch bezüglich anderer alternativer Klassifikationskriterien (z. B. organisatorische versus technologische Wirkungen).

2.2.1 Informationstechnologie zur Prozessautomatisierung

In den Anfängen des IT-Zeitalters steht die Prozess-Automatisierung im Vordergrund. Repetitive strukturierte Tätigkeiten, wie Berechnungen durchführen oder Rechnungen schreiben und verbuchen, können besser Rechner-gestützt durchgeführt werden. Zeit- und Kostenersparnisse stehen im Vordergrund. Auch die Zuverlässigkeit vieler Prozesse kann gesteigert werden. Es entsteht ein erhebliches Rationalisierungspotenzial, bei welchem zunehmend menschliche Arbeit ersetzt wird.

---

[23] Bereits lange vor der ersten Data-Warehouse-Veröffentlichung von *Inmon* gibt es eine umfangreiche und zum damaligen Zeitpunkt bereits erfolgreich erprobte Data-Warehouse-Konzeption von *Devlin* und *Murphy* im Auftrag von IBM. Das Gesamt-Konzept wurde als EBIS-Architektur (EBIS = EMEA Business Information System; EMEA = IBM Europe, Middle East and Africa) bezeichnet, der Kern als Business Data Warehouse. [DeMu88, S. 60] Einzelne Ideen werden bereits sehr früh von *Stahlknecht* aufgegriffen. [Stah73, S. 93] *Inmon* muss jedoch im *schumpeterschen* Sinne als Data-Warehouse-Pionier bezeichnet wrden.

In dieser frühen Phase stehen Funktionen und Algorithmen im Mittelpunkt der Software- bzw. Systementwicklung. Daten spielen eine untergeordnete Rolle. Die in den Programmen verarbeiteten Daten werden mit den Programmen in Dateien gespeichert. Die Nutzung der Daten für andere Anwendungen ist mit einem hohen Aufwand verbunden. Die Nichtexistenz direkt adressierbarer Speichermedien (Direct Access Storage Device, DASD) verstärkt diese Problematik.

Als Folge ist es entsprechend schwierig, Auswertungen anhand der erzeugten operativen Daten vornehmen zu können. Die im Vergleich zur Unterstützung des operativen Bereichs relativ geringe Priorität für MSS-Funktionalitäten verhindert Bestrebungen, dies zu ändern und Daten tiefergehend auszuwerten weitgehend.

Im Laufe der Zeit entsteht immer neuer Unterstützungsbedarf durch Informationstechnologie im betrieblichen Umfeld. Die Zahl der Funktionen und Applikationen steigt exponentiell. Damit wächst auch das Datenaufkommen und folglich auch die Redundanz zwischen den einzelnen Datenbeständen. Die Synchronisation der Dateien untereinander wird notwendig. Die Entstehung stark inkonsistenter Datenbestände ist unvermeidlich.

2.2.2 Software-Engineering und Datenorientierung

Mit zunehmender Verschärfung der sich entwickelnden Probleme verstärkt sich die Forderung, bei der Softwareproduktion systematischer und nach ingenieurswissenschaftlichen Prinzipien vorzugehen. Insbesondere die Strukturierte Analyse (SA) kann hierbei Fortschritte für den Entwicklungsprozess erzielen und fördert die Funktions- und Datenintegration. [Dema79, S. 3 ff.] Es entstehen zunehmend integrierte Applikationen und ersetzen die vorherigen Insellösungen.

Doch auch die Strukturierte Analyse ist primär funktionsorientiert. Erst die Erfindung von direkt adressierbaren Speichermedien und Datenbankverwaltungssystemen (hierarchische DBMS, Netzwerk-DBMS, relationale DBMS) schafft die Entwicklung zu einer datenorientierten Software-Entwicklung. [Your93, S. 27] Bachmann- und Entity-Relationship-Modelle bzw. -Diagramme entstehen als konzeptionelles Entwurfsmittel. Algorithmen und Daten werden getrennt. Die Algorithmen werden in Programmen als Funktionen und Prozeduren implementiert. Die Daten werden gemeinsam von einem DBMS verwaltet.

Das RDBMS endlich ermöglicht Redundanzfreiheit und physische und logische Datenunabhängigkeit. Die Daten liegen in einem Applikations-unabhängigen Schema vor. Die Anfrage beliebiger Daten wird möglich. Die deklarative Anfragesprache SQL erleichtert die Informationsgewinnung enorm. RDBMS erweitern ihre Konzeption um weitere konsistenzsichernde Eigenschaften wie Integritäts-

regeln (Constraints). Es entstehen Sprachen der vierten Generation (4GL), die prozedurale und datenorientierte Sprachelemente kombinieren.

Es entstehen immer komplexere integrierte betriebliche Informationssysteme, die als Administrations- und Dispositionssysteme [Mert93, S. 10 f.], im Produktionsbetrieb als Produktionsplanungs- und -steuerungssysteme (PPS) [Sche90b, S. 14 ff.], im Handelsbetrieb als Warenwirtschaftssysteme (WWS) [BeSc96S, 12 f.] und neuerdings als Enterprise Resource Planing Systems (ERP) bezeichnet werden. Auch darüber hinaus gehende Konzepte wie das Computer Integrated Manufacturing (CIM) entstehen, die weitere Komponenten eines Produktionsbetriebes integrieren. [Sche90b, S. 14 ff.]

Algorithmen verlieren zunehmend an Bedeutung. Die Daten rücken in das Zentrum des Interesses. [Your93, S. 27] Das Datenvolumen steigt erheblich. Auswertungen werden in vielen Fällen erst ermöglicht — auf jeden Fall deutlich vereinfacht.

2.2.3 Benutzer-Fokussierung und Dezentralisierung

Trotz der entstehenden Möglichkeiten, Daten abfragen und auswerten zu können, geschieht dies in der Regel nur sehr eingeschränkt. Bereits die frühen MSS-Anläufe erbringen aus den selben Gründen nur unzufriedenstellende Ergebnisse. Die verfügbare Hardware ist nicht leistungsfähig genug. Die Bedienung der Systeme ist zu kompliziert und wenig anwenderfreundlich. Das IT-Know-how der Endanwender ist zu gering.

Doch selbst nach der Entwicklung des relationalen Datenbankverwaltungssystems bleibt den Endanwendern der Zugriff auf die Daten oft noch versperrt. Zwar müssen keine eigentlichen Programme mehr geschrieben werden, doch auch der Umgang mit SQL ist für den Normalanwender in vielen Fällen zu kompliziert. Kleine Fehler im SQL-Code können nicht nur falsche Ergebnisse liefern, sondern auch das produktive Geschäft eines Unternehmens belasten bzw. gefährden, z. B. durch Anfragen die das DBMS stark belasten, wodurch beispielsweise, als eine Folge, manche Unternehmen Endanwendern den direkten Zugang zum DBMS verbieten. [Kell96, S. 53]

Der Zugriff auf die Daten bzw. auf Analysen liegt mithin immer noch zentral bei einer DV-Abteilung bzw. bei einem Rechenzentrum, auf das der Endanwender angewiesen ist. Folglich können die Daten bzw. Analysen nicht direkt vom Fachspezialisten abgefragt bzw. erzeugt werden, wodurch vielfältige Schwierigkeiten entstehen.

Die IT-Spezialisten haben in der Regel zu wenig Fachwissen, um die Anfragen adäquat erfüllen zu können, was zu Fehlern, Verzögerungen und Unzufriedenheit auf beiden Seiten führt. Zentrale Einrichtungen sind gewöhnlich zu starr in

ihren Strukturen und zu weit weg von den Erfordernissen der Endanwender. Sie sind deshalb oftmals unflexibel und nicht bedürfnisgerecht. Dies wird besonders an starren Regeln und an einem viel zu langsamen Reaktionsvermögen deutlich. So dauert es oft Wochen, Monate oder Jahre, bis einzelne Berichte oder anderweitige Dienstleistungen geliefert werden. [Inmo96, S. 1 ff.] Der Nutzeffekt von Informationen ist allerdings stark an deren Aktualität geknüpft.[24]

Eines der Hauptprobleme von zentralistischen Organisationsformen im Allgemeinen und zentralen DV-Abteilungen im Speziellen ist jedoch das „Closed-Shop-Syndrom". [Jani82, S. 174 ff.] Anstatt sich als Service-Abteilung der Fachspezialisten zu verstehen, wie *Devlin*[25] es fordert, entwickeln die DV-Mitarbeiter eigene Ziele. Es bilden sich Machtstrukturen heraus, die durch die Technologie gestützt werden und nur schwerlich wieder zu beseitigen sind. [Jani82, S. 174 ff.] Der Fachspezialist steht dem hilflos gegenüber. Dem Management fehlt in der Regel das Know-how und/oder das Duchsetzungsvermögen, daran etwas zu ändern.[26]

Einen Einschnitt in diese zentralistischen Strukturen bringt der Personal Computer (PC). Es wird erstmals ermöglicht, eigenständig und unabhängig vom Rechenzentrum zu handeln. Die Flexibilität steigt durch die Eigenverantwortung. Ergebnisse lassen sich schnell erzielen. Kommunikationsbarrieren fallen weg. Fachspezialisten bekommen schrittweise IT-Know-how.

Die Bedürfnisse des Endanwenders rücken in den Mittelpunkt. Es entstehen verstärkt PC-Anwendungen. Benutzerfreundlichkeit wird zum zentralen Thema der IT-Branche. Grafische Benutzeroberflächen für Betriebsysteme (Xerox, MacOS, Windows, X) und Anwendungen entstehen. Benutzerführungen und Hilfesysteme gehören zum Standard. *Devlin* meint dazu:

> „The development of the microcomputer — or, as it later became known, the PC — along with the first data-oriented, end-user application, the spreadsheet, drove the emergence of the end user. With these first crude tools, end users began to stand on their own feet, taking control of their own data for the first time, and working beyond the shadow of the IS shop. While the data available was restricted and higly localized, the experience of independence was a major advance for end users, giving them the confidence

---

[24] Dieser Zusammenhang wird jüngst immer deutlicher. Gerade der harte Wettbewerbsdruck in globalen Märkten lenkt das Interesse auf Indikatoren wie „Time-to-Market". Oder: „Wer zu spät kommt, den bestraft das Leben!" Zeit ist damit zum überlebensbestimmenden Faktor eines Unternehmens geworden.

[25] „Companies that survived and prospered beyond [...] learned, in general, to promote and encourage the new and relatively untamed species of end users, and IS shops began to look for ways to feed them the data they needed." [Devl97, S. 10]

[26] Wie in vielen Fällen, bei denen es um organisatorische oder psycho-soziale Probleme geht. [DeLi91, S. 1 ff., Stae91, S. 131 ff.]

and basic skills to take advantage of the later technical developments of the era." [Devl97, S. 9]

Dadurch gelingt ein weiterer Durchbruch. Der Endanwender kann selber die Kontrolle über die Daten und die mit diesen Daten zu analysierenden Applikationen übernehmen. Wie *Inmon* meint: „The end user ends up 'owning' the data once he or she takes control of it." [Inmo96, S. 6] Der „spielerische" Umgang mit den Daten wird gefördert. Gerade dieser Spiel-Charakter ist es, der das Potenzial zu hoch produktiven Analysen bewirkt.

Dieser Prozess, in welchem der Endanwender zunehmend auch IT-Know-how gewinnt und Einfluß auf die Entwicklung bekommt, erzeugt Rückkopplungen und damit einen sich selbst verstärkenden Prozess. Die Informationstechnologie orientiert sich stärker an den Bedürfnissen der Benutzer und führt dadurch zu besseren Ergebnissen, wodurch wiederum neue Anforderungen entstehen. Die Informations-Qualität steigt.

2.2.4 Client/Server-Strukturierung

Die neuen IT-Freiheiten der Endbenutzer wirken sich zwar produktiv aus, zeigen jedoch auch zahlreiche Nebenwirkungen. Es entstehen Insellösungen, Informationsinseln, ein Applikations-Chaos, Wartungsprobleme und ein Know-how-Lag der Anwender.

Anwender- und abteilungs-spezifische Insellösungen erzeugen Datenredundanz und Medienbrüche. Dies wiederum führt zwangsläufig zu erheblichem Arbeitsmehraufwand und — wesentlich schlimmer — zu falschen und widersprüchlichen Daten und Informationen. [BBB+97, S. 5 ff., Cw98, S. 16, Inmo96, S. 2 ff.] Die inkonsistenten und mangelhaften Informationen führen eventuell zu falschen Entscheidungen und zu einem Vertrauensverlust in die Informationstechnologie insgesamt. [Inmo96, S. 2 ff.] Statt weitere Daten und eine bessere Qualität der Daten zu fordern, liegt der Schwerpunkt darauf, immer neue Informationen aus dem vorhandenen zu schöpfen. [Devl97, S. 10]

Die Informationsinseln führen zu unerwünschten organisatorischen Effekten wie der Schaffung unkontrollierter Machtstrukturen und Abhängigkeiten.[27] An vielen Stellen fehlen die benötigten Informationen, die jedoch an anderer Stelle des Unternehmen vorhanden sind. [BBB+97, S. 5 ff.]

[27] In diesem Zusammenhang zeigt sich immer wieder, wie sich einzelne Mitarbeiter selber wertvoll und unentberlich machen. Nicht durch herausragende Leistung, sondern durch das gezielte Zurückbehalten von Informationen.

Die Anwender-Freiheit zur PC-Selbstgestaltung hat Folgen. Eine Vielzahl verschiedener Applikationen und Utilities wird installiert. Individuelle bedarfsgerechte Einstellungen werden vorgenommen. Es entsteht ein Applikations-Chaos.

Zwar steigt das IT-Know-how der Anwender, doch wachsen die Anforderungen an diese erheblich schneller. Die revolutionäre Entwicklung der Hardware und der exponentielle Anstieg der Software-Komplexität stellen selbst für Spezialisten eine Überforderung dar. Das Know-how-Lag der Anwender steigt folglich weiter, ohne dass dieses ausgeglichen wird. Die Anwender wollen mit Computer-Unterstützung mehr leisten, als sie können — es besteht eine Diskrepanz zwischen ihrem Technologie-Beherrschungs-Vermögen und gewünschtem Einsatz-Potenzial.

Das Applikations-Chaos erzeugt einen hohen Wartungsbedarf. Ebenso die PC-Arbeitsplätze, die jeweils unterschiedlich sind und einzeln dezentral gewartet werden müssen. Anwender haben einen kontinuierlichen Schulungsbedarf und benötigen schnell und flexibel Support bei all ihren IT-Problemen.

Insgesamt erweist sich die fehlende Austauschbarkeit von Daten als eine der maßgeblichen Herausforderungen. Es entstehen Netzwerke, in dem die einzelnen PC miteinander vernetzt werden. Daten können ausgetauscht, Software sowie andere Ressourcen wie Drucker oder Fax gemeinsam genutzt und auf Datenbanken von jedermann zugegriffen werden.

Es entsteht das Client/Server-Prinzip. Dadurch wird ein Teil der dezentralisierten IT wieder zentralisiert. Das in vielen Unternehmen vorherrschende IT-Chaos wird koordiniert. Die Vorteile sowohl zentraler DV wie gebündeltes Know-how, reduzierter Wartungsaufwand und die Koordination gemeinsamer IT (z. B. Administrations- und Dispositionssysteme), als auch dezentraler DV wie Flexibilität, Schnelligkeit, Selbstverantwortung und Motivation sollen realisiert werden.

Beispielsweise werden die Unternehmensdaten in einem zentralen DBMS verwaltet und mit einer Zugriffsrechtestruktur versehen. Die Benutzer können je nach Aufgaben-Erfordernissen Lese-, Schreib- oder gar keine Rechte auf die Daten bekommen. Client-basierte MSS-Front-Ends erlauben über standardisierte Schnittstellen den Zugriff auf den zentralen Datenbestand. Die Daten müssen zwar zentral verwaltet werden, um deren Korrektheit zu sichern, die Analyse kann allerdings effektiver dezentral erfolgen.

2.2.5 Wettbewerbsfaktor Information

Unternehmen wachsen im Laufe der Zeit immer schneller und agieren in immer größeren Märkten. Effizienz ist dadurch zunehmend schwieriger realisierbar. Der Wettbewerbsdruck steigt. Marketing und Controlling bekommen einen herausragenden Stellenwert. Information wird zum entscheidenden Faktor. Der IT-

Fokus wechselt von der Produktivitätssteigerung im operativen Geschäft auf die Gewinnung entscheidungsrelevanter Informationen.

Vor diesem Hintergrund gelingt es erstmalig, IT-Projekte bzw. IT-Investitionen nicht nur anhand von Kosteneinsparungen und direkten Produktivitätssteigerungen zu rechtfertigen.[28] Selbst wenn sich Kosteneinsparungen bei Data-Warehouse-Projekten ergeben, spielen diese insgesamt eine eher untergeordnete Rolle. Das Nutzenpotenzial ist jedoch nur äußerst schwer zu bewerten — kaum zu quantifizieren. Diese Erkenntnis beginnt sich zu verbreiten.

Es ergeben sich unterschiedliche Lösungsalternativen. MSS-Front-Ends setzen direkt auf den operativen Datenbasen auf. [Inmo96, S. 5 ff.] Integrierte Standardsoftware-Pakete werden um MSS-Module ergänzt. [Rieg97, S. 8 ff.] MSS-Front-Ends bilden eigene Datenbasen, wie beispielsweise EIS mit einer EIS-Datenbasis. [Rieg93]

- **Verwenden MSS-Front-Ends die operativen Daten direkt**, entstehen Probleme aus drei Richtungen: Die operativen Daten sind prinzipiell nicht direkt für MSS geeignet, sie sind nicht integriert und bereinigt und deren physische Verwendung durch MSS ist problematisch.

  1. Die direkte Verwendung operativer Daten ist ungeeignet. Operative Datenbasen beinhalten gewöhnlich wenig Semantik und ein geringes Ausmaß an Metadaten.[29] [Heue97, S. 127 ff.] Ein wesentlicher Anteil der Detaildaten ist für MSS-Zwecke ungeeignet.[30] Dies erschwert das Auffinden und die Analyse der benötigten Daten. Auf der anderen Seite fehlen MSS-relevante Datenelemente, weil deren Erfassung oder Speicherung für operative Prozesse keinen Wert aufweist.

     Historisierung und Versionierung werden in der Regel nicht durchgeführt. Daten werden nur kurz- bis mittelfristig gespeichert und verlieren dann ihre Bedeutung. Das Versehen mit Zeitstempeln ist folglich unnötig. Schemaänderungen werden durchgeführt, aber nicht abgebildet. Updates der Daten bewirken eine Abhängigkeit der Analysen/Reports von dem Zeitpunkt ihrer Erstellung.

---

[28] Obwohl viele der ersten Data-Warehouse-Implementierungen trotzdem mit der Begründung entstehen, dass es effizienter sei, eine zentrale Lösung für MSS-relevante Daten zu erstellen, als unzählige Schnittstellen- und Kopierprogramme zu pflegen. Die Überzeugungskraft vieler IT-Manager, Data-Warehouse-Projekte mit diesem geringen direkten Nutzen durchzuführen, hat überhaupt erst die Chance gegeben, das eigentliche Potenzial entwickeln zu können, welches nur langfristig realisierbar ist, wie *Devlin* bemerkt. [Devl97, S. 15 ff.]

[29] Insbesondere die für Endbenutzer bedeutende Geschäftssemantik - der Business Kontext - ist gering ausgeprägt. Die Metadaten sind in der Regel auf technische Metadaten beschränkt, soweit überhaupt vorhanden. Bei großen Standardsoftware-Paketen wie SAP/R3 oder Baan werden selbst die für DBMS elementaren Constraints nicht verwendet und stattdessen in den Programmcode implementiert. Die Portabilität der Software zu verschiedenen DBMS ist dadurch vereinfacht.

[30] SAP/R3 hat beispielsweise einige tausend Tabellen.

2. Setzen MSS-Front-Ends direkt auf der operativen Datenbasis auf, verlangt dies zumindest eine integrierte operative Datenbasis auf der Grundlage eines Unternehmensweiten Datenmodells (UDM). Dies ist auch das Anliegen integrierter Standardsoftware-Pakete wie Baan. Die Realität sieht allerdings anders aus. Auch eine Software wie Baan deckt in den seltensten Fällen das gesamte Funktionsspektrum eines Unternehmens bzw. einer Verwaltung ab, so dass weitere nicht integrierte Datenbasen bestehen.

   Die Datenmodelle von Standardsoftware entsprechen auch nach zahlreichen Anpassungen nie dem UDM. Datenmodelle auch bedeutender Hersteller wie Baan und SAP weisen zahlreiche Redundanzen auf, Erzeugen Inkonsistenzen und sind weniger datenorientiert[31] als vielmehr nach Prozessen, Funktionen oder sogar Masken gegliedert.

   Als Folge ergeben sich zahlreiche Inkompatibilitäts-Probleme bei der gemeinsamen bzw. übergreifenden Auswertung von Daten. Homonyme, Synonyme, unpassende Einheiten oder ein unterschiedlicher Zeitbezug sind Beispiele dafür. Zusätzlich ist die Datenqualität oft zu schlecht, um Informationen gewinnen zu können.

   Setzt sich die operative Datenbasis nicht zumindest im Kern aus einem integrierten Administrations- bzw. Dispositionssystem zusammen, so verschärft sich die Problematik mit zunehmender Anzahl der einzelnen Applikationen (Legacy Systems).

3. Physische Gründe für die Vermeidung des direkten Aufsatzes von MSS-Front-Ends auf die operativen Datenbasen sind vor allem zu hohe Belastungen der operativen Systeme. Dies kann zur Beeinflussung des operativen Geschäfts führen, was es unbedingt zu vermeiden gilt.

   Analytische Fragestellungen bedingen in der Regel Anfragen über eine sehr große Anzahl von Daten und verbrauchen daher erheblich mehr Ressourcen als operative Systeme, die meist nur auf einzelne Datensätze oder -attribute zugreifen.

   Werden in den operativen Datenbasen auch historische Daten persistent gehalten, um den Anforderungen der MSS zu genügen, wächst das Volumen des Datenbestandes unverhältnismäßig stark an und stellt dann ebenfalls eine Belastung dar.

- Bei der **Ergänzung integrierter Standardsoftware-Pakete um MSS-Front-End-Module** gilt obiges analog. Die MSS setzen einzig auf den operativen Daten der Applikation auf. Damit beschränken sich diese MSS auf die

[31] Verwendet werden fast ausschließlich RDBMS. Es liegen mithin relationale Datenmodelle zugrunde, welche auf einer klaren Theorie basieren. Wird hier der Begiff datenorientiert verwendet, ist eine theoriekonforme Modellierung bezeichnet.

Auswertung der dort vorhandenen Daten. Außerdem erfordern sie abhängig von deren Leistungsumfang den Umgang des Anwenders mit vorhandenen Datenredundanz- und -hygiene-Problemen. Die physischen Schwierigkeiten kommen zum Tragen. Der Funktionsumfang der MSS-Module ist insgesamt stark eingeschränkt. [Rieg97, S. 8 ff.] Ein Beispiel ist das Modul Baan-EIS von Baan IV, das hauptsächlich Ishikawa-Diagramme verwendet.

- **Verwenden MSS-Front-Ends eigene Datenbasen**, die durch das Kopieren der Daten aus den einzelnen Systemen entstehen, können bereits einige der gravierendsten Mängel behoben werden. Die physische Belastung der operativen Systeme wird reduziert. Einzig das Kopieren verlangt Ressourcen, kann jedoch während Niedrig-Auslastungs-Zeiten erfolgen. Es kann auf Daten aus verschiedenen Quellen zugegriffen werden. Die Daten können integriert, vereinheitlicht und bereinigt werden. Allerdings sind die Datenbasen oft proprietär. Es sind unzählige Schnittstellen und Kopierprogramme für die einzelnen Datenbasen (Quell- und Zielsysteme) notwendig.

  Das Konzept findet eine starke Verbreitung. Jeder Anwender kopiert sich die für ihn interessanten Daten zusammen. Es entsteht im Laufe der Zeit eine „natürlich gewachsene Struktur“ [Inmo96, S. 8] von Anwendungen und Kopierprogrammen. Diese „Spider Web Systems“ [Inmo96, S. 8] oder „Daisy Chains of Data“ bezeichnet *Devlin* als „Pest“ [Devl97, S. 33, S. 36 ff.]:

  ◊ Die Anwender haben das Problem, dass sie die Daten nicht finden, die sie benötigen. Es kostet Zeit und Mühe, zwischen den unterschiedlichen Werkzeugen und Schnittstellen zu wechseln, die nötig sind, um an die Daten zu kommen, die auf den verschiedenen Systemen verteilt sind. Außerdem gibt es in der Regel mehrere potentielle Quellen. Diese Quellen differieren punktuell. Welche die richtige ist — sofern dies überhaupt auf eine Quelle zutrifft — ist unklar.

  ◊ Sie können nicht an die Daten kommen, die sie brauchen. Immer wieder ist die Hilfe von IT-Experten notwendig. Diese Hilfe zu bekommen, ist aus den bereits erläuterten Umständen schwierig (siehe oben). Dazu kommt, dass einige Daten nicht ohne teilweise erhebliche Veränderungen an den operativen Systemen zu bekommen sind.

  ◊ Sie verstehen die Daten nicht, die sie finden. Die Daten sind oft schlecht dokumentiert. Die vorhandenen Dokumentationen sind meist in sehr technischer, für Benutzer unverständlicher Sprache abgefaßt.

  ◊ Sie können die Daten, die sie finden, nicht benutzen. Die Ergebnisse entsprechen oft nicht den Erwartungen, weil Bedeutung (Semantik) oder der Zeitbezug der Daten ungeeignet sind. Die Daten müssen zum Teil manuell in die Umgebung des Benutzers transformiert werden.

- ◊ Die IT-Experten haben das Problem, dass die Entwicklung der Kopierprogramme kompliziert und aufwendig ist. Es werden immer wieder ähnliche Funktionen benötigt, die zu finden aufgrund von organisatorischen Barrieren und technischen Begrenzungen schwierig ist. Die Logik, Daten bei der Transformation von den operativen Systemen in eine für Endbenutzer geeignete Form zu bringen, ist komplex und ständigen Änderungen unterworfen.
- ◊ Die Wartung und Pflege von Kopierprogrammen bringt ernsthafte Probleme mit sich. Veränderungen in einer einzigen Datenquelle verursachen Anpassungen in vielen abhängigen Kopierprogrammen, was jeweils eine genaue Analyse bedingt. In vielen Fällen sind Abhängigkeiten zwischen Kopierprogrammen jedoch undokumentiert und auch nicht bekannt. Aufgrund der komplexen Transformationsprozesse bzw. Transformationslogik und des erheblichen Anpassungsbedarfs wandeln sich die Transformationsprogramme schnell in „Spaghetti-Code".
- ◊ Der Sekundärspeicherbedarf wächst rasend. Die unkontrollierten Kopien von Datenmassen führen zu einem enormen Datenaufkommen.
- ◊ Die Daten-Administration ist komplex. Das Fehlen von standardisierten Datendefinitionen (Metadaten) führt zum Verlust der Kontrolle über die Informationsumgebung. Für jedes Datenelement existieren viele Quellen.

2.2.6 Data Warehouse

Zur Lösung des gesamten Problemkomplexes entsteht die Idee, analog zur einheitlichen Datenbasis der operativen Systeme [BeSc96, S. 61 ff.] eine einheitliche Datenbasis für MSS bereitzustellen, das Data Warehouse[32] . Die MSS-relevanten Daten[33] der operativen Datenbasen werden extrahiert, bereinigt, vereinheitlicht und geladen. Die Erkenntnis über die Bedeutung externer Daten und interner nicht operativer Daten führt dazu, dass auch diese integriert werden, sofern MSS-Relevanz besteht. Geschäftssemantik und Metadaten bekommen besonderes Gewicht. Es entsteht eine rekonziliierte Datenbasis, die als Datenquelle für alle Arten von MSS-Front-Ends dient.

Diese rekonziliierte MSS-Datenbasis schafft für die Anwender die Möglichkeit auf die Daten selbständig zuzugreifen, mit beliebigen MSS-Front-Ends zu arbei-

---

[32] Genaugenommen handelt es sich um einen Teil der Data-Warehouse-Datenbasis, die rekonziliierte Datenschicht (siehe 2.4.2). Die Data-Warehouse-Datenbasis besteht noch aus weiteren Schichten. Das Data Warehouse umfasst außer dessen Datenbasis noch vielfältige Funktionen um dessen Zweck erfüllen zu können.

[33] Welche Daten Inhalt des Data Warehouse werden, bestimmt sich einerseits durch den Informationsbedarf der Anwender, andererseits durch die Inhalte der verfügbaren Datenquellen (siehe 2.3.1).

ten, dies ohne die zentrale DV-Abteilung zu bewältigen, ohne die Datenquellen zu kennen oder verstehen zu müssen, ohne sich hauptsächlich um Inkonsistenzen, Datenhygiene, Synonyme, Homonyme, Einheits-Umrechnungen (z. B. Zeitbezug, Gewicht), verschiedene Systeme (z. B. Oracle oder Sybase, ISAM oder RDBMS, PC oder Mainframe, 32-Bit Intel oder 64-Bit Sun, Standort in London oder Berlin) Gedanken machen zu müssen und ohne verschiedene Zugriffs-Werkzeuge und Schnittstellen zu beherrschen und zu wissen, an welcher Stelle diese einzusetzen sind. Der Weg zu Unternehmens-kritischen Informationen wird für Endbenutzer so vielfach überhaupt erst geschaffen, in jedem Fall vereinfacht. Das typische Beispiel [Inmo96, S. 8] der durch verschiedene Sachbearbeiter erstellten Berichte, die sich in ihren Aussagen fundamental widersprechen, wird unwahrscheinlicher. Statt der Lösung technischer Probleme kann der Fokus auf die Nutzung der Information selber gelegt werden.

Die Entwicklung und Wartung der „natürlich gewachsenen“ Informationssysteme wird erheblich vereinfacht. Anstelle von undokumentierten Abhängigkeiten von n Quell-Systemen * m Ziel-Systemen gibt es ausführlich dokumentierte Schnittstellen von n Quell-Systemen + m Ziel-Systemen zu genau einer rekonziliierten Datenbasis. Die Existenz einer rekonziliierten Datenbasis erlaubt nicht nur die für Unternehmens-Analysen wichtige Historisierung der Daten, sondern auch die Versionierung der Datenstrukturen. Damit kann die Nutzung als Informationsbasis langfristig gesichert werden. Die Datenbereitstellung wandelt sich von einem in großer Anzahl manuelle Eingriffe benötigenden Prozess zu einer weitgehend automatisierten Aufgabe.

Eine detaillierte Diskussion des aktuellen Data-Warehouse-Konzepts erfolgt in Abschnitt 2.3. Die konkrete Umsetzung wird in Abschnitt 2.4 anhand der Data-Warehouse-Architektur besprochen.

2.3 Data-Warehouse-Konzept

Der Entwicklungspfad des Data-Warehouse-Konzepts verdeutlicht die einzelnen Anforderungen an das Data Warehouse. Das Data Warehouse dient als Grundlage für die verschiedensten Arten von MSS-Front-Ends. Es hält alle MSS-relevanten Daten persistent — analog zu einem Lagerhaus — und bietet Strukturen und Funktionen, um die Daten mit den MSS-Front-Ends nutzen zu können. Das Data-Warehouse-Konzept ist mithin Benutzer- bzw. Front-End-getrieben. Demnach bedarf es der Daten, diese Daten ordnende Strukturen und geeignete Funktionen in Benutzer- und MSS-Front-End-adäquater Form. Benutzer-adäquat, damit die unterschiedlichen Benutzer die benötigten Daten ohne fremde Hilfe und effizient finden, verstehen und zur Analyse selektieren bzw. extrahieren können. MSS-

Front-End-adäquat, um die Daten in einer für das jeweilige Werkzeug verwendbaren Form bereitzustellen.[34]

Demzufolge ist der entscheidende Punkt des Data Warehouse, **welche** Daten „gelagert" werden und **wie** diese Daten „geordnet" sind. Dabei können Waren in einem herkömmlichen Lager lediglich einfach kriteriell sortiert sein, solange es sich um materielle Waren handelt. Datenbankverwaltungssysteme bieten jedoch durchaus die Möglichkeit, sowohl logisch durch Sichten als auch physisch mehrere Ordnungen bzw. Sortierungen parallel anzubieten. Der Interessenkonflikt, der durch die nicht-komplemetären Bedürfnisse der unterschiedlichen Anwender bzw. Erfordernisse der MSS-Front-Ends bedingt wird, kann infolgedessen gemindert werden.

Was diesbezüglich nicht durch Ordnung erreicht werden kann, muss durch bessere Suchmechanismen bzw. Suchmethoden erfolgen. Auch dafür gibt es fortgeschrittene rechnergestützte Ansätze. Werden Daten oder allgemein Wissen zur Problemlösung bzw. zur Entscheidungsunterstützung benötigt, gibt es prinzipiell nur zwei Möglichkeiten: Eine bessere Strukturierung, also Ordnung der Daten oder ein besseres Suchen im Datenbestand. Das Data-Warehouse-Konzept verfolgt primär das erste Ziel. MSS wie Data-Mining-Tools setzen ihren Schwerpunkt auf das Suchen, erzeugen dann allerdings durch die Wissensgenerierung neue Strukturen.

Eine verbesserte Ordnung der Daten erhöht die Effizienz und Effektivität der Unternehmens-Entscheidungen. Die Ordnung der Daten wirkt auf die MSS, welche den Benutzern helfen, was sich wiederum auf die Unternehmens-Entscheidungen auswirkt. Mithin wirkt sich eine Verbesserung[35] der Struktur des Data Warehouse positiv aus. Die Menge der (relevanten) Daten wirkt ebenfalls in diese Richtung, steht jedoch dem positiven Effekt der Strukturierung entgegen. Der Gewinn durch die Menge der im Data Warehouse zu verwaltenden Daten muss also mit dem negativen Effekt auf die Struktur abgewogen werden. Die Entscheidungsqualität steigt nur unterproportional mit der Informationsmenge.

Vor allem müssen die Menge und Ordnung der durch das Data Warehouse verwalteten Daten unter ökonomischen Gesichtspunkten betrachtet werden. Der Aufwand darf den Nutzeffekt nicht überschreiten. Sind beispielsweise die Kosten

---

[34] Wie später gezeigt werden wird, muss es keine unbedingte Trennung zwischen Data Warehouse und MSS-Front-End geben. Anstelle der Verwendung vieler einzelner Spezialwerkzeuge gibt es durchaus auch einen integrativen Ansatz bei welchem die Grenzen zerfließen. Gerade das objektorientierte Paradigma vollzieht die Trennung nicht in dieser Weise. Vor- und Nachteile gelte es zu diskutieren. Trotzdem bedarf es auch bei einem integrativen bzw. objektorientierten Ansatz geeigneter Strukturen und Funktionen für die Daten, um den MSS-Funktionsumfang realisieren zu können.

[35] Eine Verbesserung muss nicht zwangsläuftig ein Mehr an Struktur sein. Überstrukturierungen können im Gegenteil sogar einen negativen Effekt haben. Dieses paradoxe Phänomen entsteht allerdings in der Folge einer ungeeigneten Stukturierung und wird nicht weiter betrachtet.

des Betriebs eines Data Warehouse größer als der kausal zurechenbare Erlös, ist zu prüfen ob eine Reduzierung[36] der Datenmenge oder -strukturierung dieses Verhältnis zu ändern in der Lage ist.[37]

Praktisch bedeutet das, dass sinnvoll nur eine Teilmenge der für die unternehmerischen Entscheidungen relevanten Daten durch das Data Warehouse verwaltet werden kann. Die anderen Informationen müssen auf anderen — dann effizienteren — Wegen bezogen werden oder deren Beschaffung ist überhaupt unökonomisch, der Prohibitivpreis zu hoch. Auch muß der Strukturierungs-Prozess der Daten auf ein sinnvolles Maß beschränkt bleiben.

Beispielsweise ist es mengenmäßig weder realistisch noch zweckmäßig, den Inhalt des gesamten World Wide Web (WWW) mit einem Data Warehouse verwalten zu wollen. Eine effiziente Suche ist dort angebrachter. Besonders interessante Teile eines WWW-Angebotes zu speichern — als Link oder überdies als physische Kopie — kann durchaus sinnvoll sein. [Meie98, S. 105 ff.] Strukturell wird es für gewöhnlich unangemessen sein, in einem Unternehmens-Data-Warehouse Videos in einzelne Bilder, diese wiederum in Strukturen (z. B. Haus, Katze) und Unterstrukturen (z. B. Fenster, Katzenkopf) und deren Beziehungen untereinander oder etwa in Pixel zu zerlegen.

2.3.1 Data-Warehouse-Daten

Den Kern des Data Warehouse bilden die Daten aus den operativen Systemen. Historisch sind sie die primäre Quelle und aufgrund ihres relativ hohen Strukturiertheitsgrades[38] und quantitativen Charakters verhältnismäßig leicht zu ordnen und mit aktueller Softwaretechnologie (RDBMS, 4GL) zu handhaben bzw. zu organisieren. Typische Beispiele für operative Daten sind Daten über Kunden, Lieferanten, Mitarbeiter, Artikel, Aufträge, Verkäufe, Rechnungen, Buchungskonten, Kostenstellen oder Produktions-Maschinen, die sich in weitere Entitäten zergliedern lassen, durch Attribute beschrieben werden und Beziehungen untereinander aufweisen.

---

[36] Es wird hier von dem Fall ausgegangen, dass bereits ein zu hoher Aufwand mit dem Data Warehouse betrieben wird. Der umgekehrte Fall ist mindestens ebenso realistisch und sollte dann entsprechend den gegenteiligen Effekt haben.

[37] Die Bewertung ist allerdings in hohem Maße problematisch. Die Kosten sind oft schwer zurechenbar, aber aufgrund ihres Charakters in der Regel näherungsweise zu bestimmen. Der Nutzeffekt ist dagegen kaum zu bewerten und zu quantifizieren. Der weitverbreitete Effekt, sich deshalb auf die bewert- und quantifizierbaren Faktoren („hard facts“) zu beschränken, führt systematisch zu grob falschen Ergebnissen, da die „soft facts“ den weitaus größeren Nutzenanteil ausmachen.

[38] Zwar gibt es auch semistrukturierte operative Daten, diese Differenzierung liefert im Kontext dieser Arbeit allerdings keinen zusätzlichen Erkenntnisgewinn und wird infolgedessen vernachlässigt. Unter operativen Daten werden mithin die strukturierten Daten aus den Administrations- und Dispositionssystemen verstanden.

Eine zweite Stufe bilden die externen strukturierten Daten. Branchenvergleichsdaten, statistisches Material der Statistischen Landesämter oder Inhalte aus Online-Datenbanken[39] (z. B. Reuters) — wie Börsenkurse, Rohstoffpreise, der Dollarkurs oder aktuelle Länder-Temperaturen — sind hierfür Beispiele. So wurden bei einem Projekt mit der Drogeriemarktkette Ihr Platz Daten des Marktführers Schlecker beim Marktforschungsinstitut Nielsen zugekauft und bei der Bonanalyse als Vergleichswerte verwendet. Ein typisches Ergebnis ist, dass bei Schlecker mehr Waren pro Bon als bei Ihr Platz gekauft werden. [BHJ+96]

Semistrukturierte[40] bzw. qualitative[41] Daten aus internen und externen Quellen bilden eine Ergänzung [Rieg96] des strukturierten Data-Warehouse-Datenbestandes. Die Bedeutung semistrukturierter bzw. qualitativer Daten und als Folge die Notwendigkeit diese in das Data Warehouse zu integrieren ist herrschende Meinung. So meinen beispielsweise *Uhr* und *Breuer*:

> „Während einerseits unbestritten ist, daß Unternehmen vom und für den Markt leben und die wichtigsten Informationen für die Unternehmensführung folglich von außen kommen, sind in MSS häufig nur Informationen aus internen Quellen verfügbar. Erst die Verknüpfung von internen und externen Informationen liefert jedoch eine brauchbare Beurteilungs- und Entscheidungsgrundlage für Führungskräfte.“ [UhBr98, S. 7]

Auch *Mertens* betont den erheblichen Anteil der Einflussnahme dieser Daten am Entscheidungsprozess. [Mert98c, S. 9 ff.] Praktische Umsetzungen sind bisher allerdings noch selten. [UhBr98, S. 7 ff.]

---

[39] Praktisch stehen der Integration von Daten aus Online-Datenbanken zur Zeit noch rechtliche und unternehmens-politische Barrieren im Weg. Mittel- bis langfristig ist die Realisation des Marktpotenzials durch neue oder wandlungsfähige Unternehmen zu erwarten.

[40] Der Begriff semistrukturiert findet Verwendung, da zum Ausdruck gebracht werden soll, dass einerseits semistrukturierte Daten weniger leicht zu strukturieren sind denn die als strukturiert bezeichneten Daten, andererseits semistrukturierte Daten durchaus nicht strukturlos sind. Wird ein Text mit quantitativen Attributen für beispielsweise die Menge der Wörter, Sätze und Absätze oder auch qualitativen Merkmalen wie etwa Lesbarkeit, Spannungsbogen oder Humor (z. B. gut, mittel, schlecht) beschrieben, handelt es sich um strukturierte Daten. Texte weisen zwar eine syntaktische Struktur auf, wie Sätze und Absätze, diese Strukturierung (Modellierung der Struktur) ist allerdings mit größerem Aufwand verbunden. Die inhaltliche bzw. semantische Strukturierung ist ganz erheblich viel schwieriger zu realisieren. Eine klarere Definition zur Unterscheidung von strukturierten und semistrukturierten Daten ist an dieser Stelle nicht zu leisten. Letztlich ist der Strukturiertheitsgrad eine Frage der Sichtweise des Betrachters oder besser: des Modellbildners. Dieser kann durch die Modellierung Strukturen in den Daten „erkennen“ oder „erzeugen“.

[41] Die Bezeichnung qualitative Daten wird im MSS-Umfeld immer wieder synonym zu semistrukturierten Daten verwendet, quantitative Daten entsprechend mit strukturierten Daten gleichgesetzt. [Mert98c, S. 9 ff.] Teilweise wird qualitativ auch im Sinne einer Bewertung (Qualität) verstanden, womit also bewertende Daten gemeint sind. In diesem Fall ist quantitativ und qualitativ kein Gegensatzpaar mehr. Dieser Gegensatz-Charakter wird gleichwohl in der Regel angenommen. Das *Gabler Wirtschaftslexikon* definiert quantitative Daten als kardinal skaliert, qualitative als nominal skaliert und Daten welche nach einer Ordinalskala skaliert werden als beiden Kategorien nicht zurechenbar. [Gabl97] Um selbige Definitionslücke zu schließen, werden in dieser Arbeit die nach der Nominal- und Ordinalskala eingeteilten Daten als qualitative Daten bezeichnet. Quantitative Daten sind kardinal skaliert.

Das sich jüngst stark verbreitende Konzept des Knowledge Management fokussiert genau diesen Teil (siehe Abschnitt 2.1.5). Leider werden als Problemlösungs-Werkzeuge üblicherweise Information-Retrieval-Systems, Dokumenten-Management-Systeme oder Intranet/Internet-Lösungen vorgeschlagen. [RKM00, S. 372 f.] Der primäre Zweck des Data Warehouse die Unternehmensdaten sämtlich zu integrieren/rekonziliieren wird damit außer Acht gelassen.

Beispiele sind Dokumente aus Text- (Kapitel, Absatz), Ton- (Sprache, Musik, Geräusche), Bild- (Fotos, Grafiken, Tabellen) oder Video-Elementen in medialer Form von Büchern, Artikeln, Web-Sites oder CD-ROMs. Die Inhalte können jeweils statisch oder dynamisch sein. Ein Zeitungsartikel wird nicht mehr verändert werden — im Zweifelsfall in einer überarbeiteten Neuauflage erscheinen. Tabellen, Grafiken oder Textteile können dynamisch aus einer Datenbank generiert werden, was vielfach für WWW-Seiten geeignet erscheint.

Sofern diese internen und externen bzw. strukturierten und semistrukturierten Daten für die operativen Systeme oder für MSS von Bedeutung sind, spricht *Devlin* von Geschäftsdaten. [Devl97, S. 44 ff.] Die MSS-relevanten Geschäftsdaten werden im Folgenden als MSS-Daten bezeichnet, die für das operative Geschäft benötigten Geschäftsdaten als operative Daten.

Davon abgegrenzt werden die Metadaten. Metadaten beschreiben die Bedeutung und die Struktur der Geschäftsdaten, sowie deren Entstehung, Zugriff und Verwendung. *Devlin* unterscheidet drei Arten von Metadaten [Devl97, S. 54 ff.]:

1. Die während des Designs und der Konstruktion der Data-Warehouse-Applikationen und Datenbankschemata erzeugten Metadaten, als **Built-Time Metadata**[42] bezeichnet, sind in Form von Datenmodellen als auch Konzept- und Ablaufbeschreibungen in CASE-Tools, Benutzerdokumentationen oder im schlechtesten Fall nur implizit im Code bzw. den Datenbankstrukturen vorhanden.
2. Um die Prozesse des Data Warehouse zu steuern, zu dokumentieren und zu überwachen, werden die als **Control Metadata** bezeichneten Metadaten benötigt. [Mert98b, S. 70] Sie sind Bestandteil des DBMS, der Data-Warehouse-Prozeduren oder spezieller Bewirtschaftungswerkzeuge (ETL-Tools). Beispiele hierfür sind die Algorithmen zur Vereinheitlichung und Bereinigung der Daten, die Mappings zwischen Quell- und Zieldaten, Constraints bzw. Geschäftsregeln und von herausragender Bedeutung die Zugriffsrechtestruktur.
3. Die zur Interpretation der Daten durch den Endbenutzer notwendigen Daten werden als **Usage Metadata** bezeichnet. Beispielsweise müssen Begriffe definiert („Welcher Umsatz ist gemeint?“ und „Woraus setzt sich dieser Umsatz

[42] *Mertens* übersetzt Built-Time Metadata mit Aufbau-Zeit-Metadaten, Control Metadata mit Kontroll-Metadaten und Usage Metadata mit Gebrauchs-Metadaten. [Mert98b]

zusammen?“) oder die Datenquelle und der diesbezügliche Kontext verdeutlicht werden. Dies wird besonders anhand der *riegerschen* Informations-Facette deutlich (siehe Abbildung 7). Wesentlich sind aber auch die Beschreibung der Datenstrukturen und der Datenzuständigkeiten („Ownership“ und „Stewardship“).

Abbildung 7: Informationsobjekt im Kontext (Informations-Facette) [Rieg93]

Da das Data Warehouse die Grundlage von Analytischen Informationssystemen darstellt, übernehmen die Metadaten eine herausragende Rolle. Für die Analyse ist das Verständnis der Daten notwendige Voraussetzung. Insbesondere zur Steigerung der Endanwender-Selbständigkeit bedarf es einer hohen Verständlichkeit der Daten.

Metadaten haben damit vorwiegend Ordnungscharakter. Es stellt sich mithin die Frage inwieweit Metadaten selber Strukturen abbilden bzw. struktur-erzeugend wirken. Je semantisch höherwertiger die Modellierung ist, desto weniger explizite Metadaten sind notwendig, um den selben Informationsgehalt abzubilden bzw. zu erreichen. Daten an sich sind jedoch auch bereits struktur-beschreibend. Das macht eine klare Abgrenzung schwierig.

In Anbetracht ihrer Bedeutung für MSS ist die über Systeme und Anwendungen verteilte Speicherung der Metadaten inadäquat. Vielmehr müssen alle Metadaten (Built-Time, Control und Usage Metadata) integriert verwaltet und analog zu den MSS-Daten bereitgestellt werden. Die Idee einer solchen Metadatenbank ist

identisch mit der Intention eines Data Warehouse, der Schaffung einer integrierten MSS-Datenbasis, bezogen auf die Ebene der Metadaten. Die Entwicklung eines solchen Konzepts ist mithin folgerichtig. Es gibt bereits erste Standardisierungsbemühungen für Metadaten-Schnittstellen. [Dbms96, Kell96, S. 184 ff., Mdc00]

Die Metadaten können integraler Bestandteil der Data-Warehouse-Datenbasis sein, also gemeinsam mit den MSS-Daten abgelegt werden oder deren Speicherung und Verwaltung kann gesondert in einer eigenen Metadatenbank erfolgen, eventuell durch ein eigenes Metadatenbankverwaltungssystem (Meta Database Management System, MDBMS).

Letztere Variante begünstigt die Verwaltung auch Data-Warehouse-externer MSS-relevanter Metadaten, also die gemeinsame Verwaltung aller MSS-Metadaten. Beispielsweise müssen in einem solchen Metadatenbankverwaltungssystem auch die Daten darüber verwaltet werden, welche Analysen es gibt, wo es diese gibt und mit welchem MSS-Werkzeug diese zu lesen bzw. zu bearbeiten sind. [RiKl00, S. 89 ff.]

Aufgrund der Ähnlichkeit beider Konzepte und der fehlenden scharfen Trennung zwischen den MSS-Daten- und MSS-Metadaten scheint jedoch auch die erste Variante der Integration der Metadaten in die Data-Warehouse-Datenbasis vielversprechende Potenziale aufzuweisen und im Falle einer effizienten Machbarkeit dem zentralen Gedanken der integrierten bzw. rekonziliierten Datenbasis in gesteigertem Maße nahe zu kommen.

2.3.2 Data-Warehouse-Datenmodell

Während die Daten den Inhalt des Data Warehouse abbilden, ist das Modell das strukturierende Element. Ein Modell ist eine nach der Modellierungsintention des Modellbildners verkürzte Abbildung bzw. Repräsentation. *Stachowiak* unterscheidet das Abbildungsmerkmal, das Verkürzungsmerkmal und das Pragmatische Merkmal. [Stac73, S. 131 ff.] Dabei wirkt das Pragmatische Merkmal auch auf das Verkürzungsmerkmal.

Modelle lassen sich prinzipiell bezüglich ihrer Erklärungsintention (Pragmatisches Merkmal) in Beschreibungsmodelle, Erklärungsmodelle und Entscheidungsmodelle unterteilen. Während sich MSS im Schwerpunkt mit letzteren beiden beschäftigen, bildet die grundlegende Schicht des Data Warehouse (der Reconciled Data Layer, siehe Abschnitt 2.4.2) analog zu den operativen Systemen ein Beschreibungsmodell. Allerdings muss das Data Warehouse in später aufsetzenden Schichten auch in der Lage sein, Erklärungs- und Entscheidungsmodelle bereitzustellen.

Da der Betrachtungsgegenstand Unternehmen sind, werden Modelle von Unternehmenssystemen gebildet. Je nach Sicht auf das Unternehmenssystem werden jeweils verschiedene Modelle zur Beschreibung verwendet (Verkürzungsmerkmal). Es gibt Methoden zur Modellierung von Daten (z. B. ERM), Funktionen (z. B. Funktionsbaum), Algorithmen (z. B. Nassi-Shneiderman-Diagramm, NSD), Prozessen (z. B. Datenfluss-Diagramme, DFD), Ereignissen (z. B. Zustandsautomaten), Materialflüssen (z. B. Lindner-Diagramme) oder Aufbauorganisationen (z. B. Organigramm). [Balz93b, S. 40 ff.]

Da es Zweck des Data Warehouse ist, die MSS-Daten abzubilden, werden Datenmodelle gebildet. Es gibt verschiedene Methoden zur Datenmodellierung, die sich in konzeptionelle und logische Datenmodellierungs-Methoden unterscheiden lassen. Konzeptionelle Modelle wie das Entity Relationship Modell (ERM), semantische Datenmodelle, funktionale Datenmodelle oder objektorientierte Modelle (z. B. die Coad/Yourdon-Methode oder UML) bilden die Vorstufe zu den logischen Datenmodellen und werden bei der Software-Entwicklung in diese übertragen. Logische Datenmodelle wie das Netzwerk- oder das Relationenmodell bilden die Nutzersicht bestimmter Datenbankverwaltungssysteme und sind, sofern kein Produktstandard existiert, proprietär. Beispielsweise gibt es kein explizites logisches objektorientiertes Datenmodell, jedoch ein Smalltalk-spezifisches[43] .

Konzeptionelle Datenmodellierungs-Methoden werden in der Regel als unabhängig von den logischen Datenmodellen beschrieben. [Voss94, S. 48 ff.] Da eine konzeptionelle Datenmodellierungs-Methode allerdings nicht sinnvoll für ein logisches Datenmodell zu verwenden ist, das weniger Semantik beinhaltet, ist diese Unabhängigkeit nicht wirklich gegeben. In der Praxis werden deshalb auch z. B. für hierarchische Datenmodelle Bachmann-Diagramme, für Relationenmodelle ERM[44] verwendet.

Zur Implementierung eines Data Warehouse werden zur Zeit so gut wie ausschließlich relationale DBMS verwendet. Teilweise wird sogar davon ausgegangen, dass das Data-Warehouse-Konzept untrennbar mit der relationalen Datenbanktechnologie verknüpft ist [DeMu88, S. 60 ff.], die Eignung anderer Technologien wird jedenfalls ausgeschlossen [Burl97, S. 3 ff., Devl97, S. 404, Inmo96, S. 33 ff.]. Auf der logischen Ebene findet mithin das relationale Datenmodell Verwendung, auf der konzeptionellen Ebene das Entity Relationship Modell.

---

[43] In Folge der unabhängigen Entwicklung der verschiedenen Smalltalk-Dialekte sind auch diese Datenmodelle inzwischen unterschiedlich. Zum Beispiel weisen die Datenmodelle von GemStone und VisualWorks differierende Strukturen auf.

[44] Es sei jedoch an dieser Stelle bemerkt, dass im Laufe der Zeit vielfältige Erweiterungen und Varianten des *chenschen* ERM entstanden sind. [Balz96, S. 138 ff., Sinz90, S. 17 ff.] Mithin kann nicht mehr nur von „dem“ ERM gesprochen werden. Der Methoden-Kern weist allerdings mehr Ähnlichkeit auf, als dies für den Vergleich objektorientierter Entwurfsmethoden der Fall ist.

Daraus ergibt sich insgesamt die Notwendigkeit der Modellierung eines Unternehmens-Datenmodells (UDM) [Mert98a, S. 54 ff., Kell96, S. 61 ff.], analog zum operativen Unternehmens-Datenmodell (OUDM) [ScHa92, S. 166 ff., Sche88, S. 567 ff., Sche90a, S. 46 ff., Sche97b, S. 417 ff.]. Da das Ziel die Abbildung der MSS-relevanten Daten ist, wird dieses im Folgenden als MSS-Unternehmens-Datenmodell (MSS-UDM) bezeichnet.

Das MSS-UDM und das operative UDM folgen verschiedenen Modellierungs-Intentionen (Pragmatisches Merkmal). Das OUDM ist das konzeptionelle Datenmodell zur Abbildung des operativen Geschäfts eines Unternehmens. Es ist das Ergebnis einer Organisations- bzw. Systemanalyse. Es dient heute in der Regel als Basis bei der Auswahl und Anpassung eines Standard Administrations- und Dispositionssystems, oder sofern dies nicht sinnvoll möglich aber dennoch ökonomisch ist, der Entwicklung einer solchen umfassenden integrierten Software für das operative Geschäft. Das MSS-UDM ist das konzeptionelle Datenmodell des Data-Warehouse-Kerns, der grundlegenden Schicht.

Beide UDM bilden folglich verschiedene Daten und Strukturen ab. Beispielsweise spielt die Zeit eine herausragende Rolle im MSS-UDM und führt zu deren strukturellen Abbildung in Form von Historisierung und Versionierung, während für das OUDM hauptsächlich aktuelle Zustände modelliert werden, alte Strukturen deshalb außerhalb des Interesses liegen. Operatives UDM und MSS-UDM sind jedoch nicht gänzlich verschieden. Es gibt einen elementaren Kern, der bei beiden identisch ist. Die Unternehmens-Datenmodelle können mithin nicht losgelöst voneinander betrachtet werden. Folglich muss es Verknüpfungen (Mappings) zwischen identischen und abgeleiteten Daten geben. Daraus ergibt sich ein übergeordnetes unternehmensweites Datenmodell, das generische Unternehmens-Datenmodell (GUDM) mit den sich teilweise überlappenden Teilmodellen MSS-UDM und OUDM.

Nur die integrative Modellierung des GUDM, MSS-UDM und OUDM sowie deren Umsetzung in eine logische und physische Schicht kann die Zielsetzung des Data-Warehouse-Konzepts überhaupt erfüllen. Zwar sind ohne UDM dennoch kurzfristig erfolgreiche Teillösungen möglich, doch vermögen diese nicht, das vorhandene Potenzial verwirklichen zu können. Mit dem Data-Warehouse-Konzept hat dies dann wenig gemein. Die erläuterte Modellierung ist notwendige Bedingung und beinhaltet den Schlüssel des Data-Warehouse-Ansatzes.[45] *Zachman* bemerkt treffend:

> „I am describing this historical context for data warehouse to observe that anyone who is building a data warehouse and ignores the data architecture issue clearly does not un-

[45] Allerdings ist die Modellierung der UDM auch der schwierigste/anspruchsvollste Teil der Data-Warehouse-Entwicklung.

derstand the problem. A data warehouse that is not derived from an enterprise data model is simply one more legacy file and ultimately will further contribute to the enterprise's disappointment and frustration with its information systems." [Zach97, S. VIII]

Das Ideal einer betrieblichen Informations-Architektur besteht in der Umsetzung des generischen Unternehmens-Datenmodells in eine zumindest logisch gemeinsame Datenbasis. Der auf der operativen Ebene durch die Administrations- und Dispositionssysteme und auf der MSS-Seite durch das Data Warehouse vollzogene jeweilige Integrationsprozess, müßte folglich in eine gemeinsam integrierte Lösung münden. Beispielsweise könnte ein GUDM-ERM in ein logisches Schema eines RDBMS überführt und dann implementiert werden. Den *coddschen* Entwurfsregeln folgend läge dann jedes Datum maximal einmal (redundanzfrei normalisiert) vor. [Date00, S. 384 ff.] Ergänzend implementierte Integritätsregeln sicherten die Konsistenz.

Auf der physischen Ebene gibt es vielfältige Argumente gegen eine Integration von MSS- und operativen Daten. Unterschiedliche Anforderungen an die Hardware, Zuverlässigkeit und Zeitfenster sind Beispiele. [Inmo96, S. 18 f.] Diese Probleme werden jedoch im Zuge der fortschreitenden Entwicklung an Bedeutung verlieren. Auch ist eine Trennung von logischer und physischer Ebene verwirklichungsfähig. Verteilte Datenbanken sind bereits Normalität und verwirklichen entsprechende Konzepte. [Date00, S. 651 ff., ElNa94, S. 703 ff.]

Was allerdings problematisch ist, ist der Versuch, diese Idealform der logischen Datenintegration zu verwirklichen. Der Modellierungsprozess eines Unternehmens-Datenmodells ist aufgrund des Umfangs äußerst aufwendig und bedingt durch dessen Komplexität von einem ausgesprochen hohen Schwierigkeitsgrad. Schon im operativen Bereich sind die Datenmodelle integrierter Standarsoftware-Pakete wie SAP R/3 oder Baan IV weit davon entfernt dem Idealbild eines UDM zu entsprechen.[46] Redundanz innerhalb und besonders zwischen einzelnen Modulen ist der Normalfall. Inkonsistenzen sind vielfach vorhanden. In Anbetracht dessen ist die Realisierung eines gesamt-integrierten Informationssystems als Utopie zu werten. Es wird nicht gelingen, ein solches Informationssystem zu entwerfen, zu implementieren und den jeweiligen Veränderungen anzupassen.

Auf der anderen Seite darf diese Erkenntnis nicht dazu führen, dass auch die konsistente Modellierung von OUDM, MSS-UDM und GUDM als nicht zu realisieren bewertet wird und deshalb entfällt. Ohne UDM können Informationssysteme nicht sinnvoll funktionieren. Das Data Warehouse dient gerade zur Lö-

[46] Der Grund hierfür ist nicht primär darauf zurückzuführen, dass es sich um Standardsoftware handelt. Individualsoftware hat ceteris paribus eine weit geringere Komplexität aufzuweisen, doch selbst diese Komplexität ist mit bisherigen Methoden nicht ausreichend zu handhaben. Der Modellierungsprozess ist eine primär menschliche Aufgabe. Menschen sind im Umgang mit komplexen Systemen allerdings sehr beschränkt. [Dörn97, S. 58 ff.]

sung der aufgrund der mangelnden konzeptionellen Arbeit an den operativen Systemen entstandenen Problemen. Es deckt kurz- bis langfristig den Bedarf an MSS-Daten. Ob es gelingen wird, gesamt-integrierte Unternehmenslösungen auch auf den über die konzeptionelle Ebene hinausgehenden Schichten zu realisieren, wird in späteren Jahren gezeigt werden.

2.3.3 Data-Warehouse-Engineering

Ein Data Warehouse ist Software und durchläuft demzufolge einen Software-Entwicklungsprozess. Um der Forderung einer ingenieurs-wissenschaftlichen Entwicklung von Software zu entsprechen, wird von Software-Engineering oder als umfassenderen Begriff von Systems-Engineering gesprochen, der dann auch andere Aspekte des die Diskurswelt betreffenden Systems wie beispielsweise die Unternehmens-Organisation beachtet. Dementsprechend findet hier der Ausdruck Data-Warehouse-Engineering Verwendung.

Das Data-Warehouse-Engineering hat zwei Ziele:

1. die Qualität des resultierenden Data Warehouse (Software-Qualität), die Data-Warehouse-Qualität und
2. die Qualität des Entwicklungs-Prozesses des Data Warehouse, das Data Warehousing [Mart98b, S. 29 ff.].

Ein Bewertungskatalog für Software-Qualität ist beispielsweise aus der Norm DIN ISO 9126 zu entnehmen (siehe insbesondere Abschnitt 3.2.3) [Balz98, S. 258]:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Das Data Warehousing setzt sich zusammen aus Entwicklung, Wartung und Betrieb des Data Warehouse. Es bestehen Abhängigkeiten zwischen dem Software-Prozess und dem Software-Produkt. Während sich die Effektivität des Prozesses direkt in der Software-Qualität niederschlägt, ist die Data-Warehousing-Effizienz durch Zeit und Kosten bestimmt. Die Data-Warehouse-Qualität bestimmt wiederum den Aufwand für Betrieb und Wartung des Systems. Die hohe Bedeutung dieses Zusammenhangs liegt auch darin, dass die Entwicklung bzw. Weiterentwicklung des Data Warehouse ein ständiger Prozess ist. [Mart98b, S. 29 ff.]

Die *taylorsche* Aussage über die Beständigkeit von — insbesondere zukünftiger — Software „change is the only constant“ [Tayl92, S. 5] gilt für das Data Warehouse mithin um so deutlicher.

Die Qualität von Data Warehouse **und** Data Warehousing sind folglich zentrale Themen. Software-Engineering und Software-Qualität werden ausführlich in Abschnitt 3.2 diskutiert und mit den entsprechenden Konzepten des objektorientierten Paradigmas vorgestellt (siehe die Abschnitte 3.3 bis 3.5).

2.4 Data-Warehouse-Architektur

In diesem Kapitel wird die Data-Warehouse-Architektur erläutert. Dies geschieht anhand der einzelnen Schichten des Data Warehouse. Dabei lassen sich die einzelnen Schichten grob in drei Kategorien einteilen: Die Quell-Schichten, den Data-Warehouse-Kern und die anwendungs-orientierten Schichten. Diese Einteilung entspricht in etwa auch dem klassischen EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe) konventioneller Anwendungen. Die Daten werden aus den Quell-Schichten eingelesen/geladen (Input), im Kern gespeichert (Speicherung anstelle einer Verarbeitung) und in aufsetzenden Schichten Anwender- bzw. Anwendungs-gerecht aufbereitet und als Output-Schnittstelle zur Verfügung gestellt (Output).

Die diskutierte Data-Warehouse-Architektur ist eine idealtypische Realisierung des Data-Warehouse-Konzepts. Die einzelnen vorgestellten Schichten sind verschieden intensiv ausgeprägt, was von den Umständen des Anwendungsfalles abhängt, d. h. des jeweilig bezogenen Unternehmens. Konkret hat dies zur Folge, dass eventuell auch einzelne Schichten zusammengefasst werden können, also nicht in jedem Fall alle Schichten explizit realisiert werden müssen. Geeignete Anhaltspunkte werden gegeben.

Der Vergleich zwischen den zur Zeit realisierten Data-Warehouse-Implementierungen wie auch dem aktuellen Stand der Literatur auf der einen Seite und der entwickelten idealtypischen Architektur auf der anderen Seite erfolgt unter 2.5. Eine darüber hinausgehende Konzeption wird in Kapitel 4 vorgestellt.

2.4.1 Quell-Schichten

Entsprechend den zu integrierenden Daten (siehe Abschnitt 2.3.1) ergeben sich auch die Quell-Schichten. Den Kern der Quell-Daten bilden die Daten der operativen Systeme, die operativen Daten. Erst in einem Folgeschritt wird der Anteil der MSS-Daten, welcher aus den operativen Daten gewonnen wird durch weitere Daten ergänzt. [Devl97, S. 63 ff.] Weitere Daten sind beliebige MSS-relevante

Daten aus sowohl nicht operativen internen als auch externen Datenquellen (siehe Abbildung 8).

Die Daten sind strukturiert oder semistrukturiert. Sie werden in verschiedenen Datenbank-Management-Systemen, Dateiverwaltungssystemen oder einfach nur in einzelnen Dateien gespeichert. Die DBMS unterscheiden sich nach der zugrundeliegenden Technologie und dort wiederum nach einzelnen Produkten. Relationale DBMS von Oracle, Sybase, Informix oder IBM (DB/2), objektorientierte DBMS von Objectivity, GemStone, Versant oder O_2, hierarchische DBMS, Netzwerk-DBMS oder Flat-Files wie ISAM und VSAM sind Beispiele für die ersten beiden Kategorien. Dokumente in unterschiedlichster Form liegen in der Regel nur als einzelne ungeordnete Dateien vor. Beispiele hierfür sind Briefe als Word-Dokument (z. B. MS-Word-Format (doc) Version 6.0) oder als HTML-Datei (z. B. Version 2.0), Grafiken im JPEG-Format oder Videos als MPEG-Datei.

Der erste Schritt der Integration der operativen Daten zu einer gemeinsamen Datenbasis führt zum Operational Data Store (ODS) (siehe Abbildung 8). [Inmo-99, S. 12 f.] Die in der Regel vielfach verschiedenen nicht-integrierten operativen Systeme (Legacy Systems) werden zu einer gemeinsamen operativen Datenbasis zusammengefasst. Dadurch ist der ODS nichts anderes als die Umsetzung des operativen UDM (siehe Abschnitt 2.3.2). Der ODS muss alle für die operativen Systeme notwendigen Daten integrieren. Der Operational Data Store ist damit eine den Prinzipien der Datenmodellierung folgende [Date00, S. 330 ff.], streng normalisierte Datenbasis.

Ziel bei der Erstellung des ODS ist gleichzeitig die Schaffung einer operativen Datenbasis, welche die Grundlage für ein zukünftiges integriertes operatives System (Administrations- und Dispositionssystem) darstellt. So kann auf dem ODS Stück für Stück bzw. Modul für Modul ein neues integriertes operatives System entstehen. Alternativ kann bei der Einführung einer operativen Standardsoftware OUDM und ODS als Grundlage der Unternehmensanforderungen dienen, die Überführung der weiterhin notwendigen Alt-Daten in das neue System erheblich vereinfachen und den Aufwand für die Neuimplementierung der Schnittstellen zwischen Data Warehouse und dem neuen operativen System verringern.

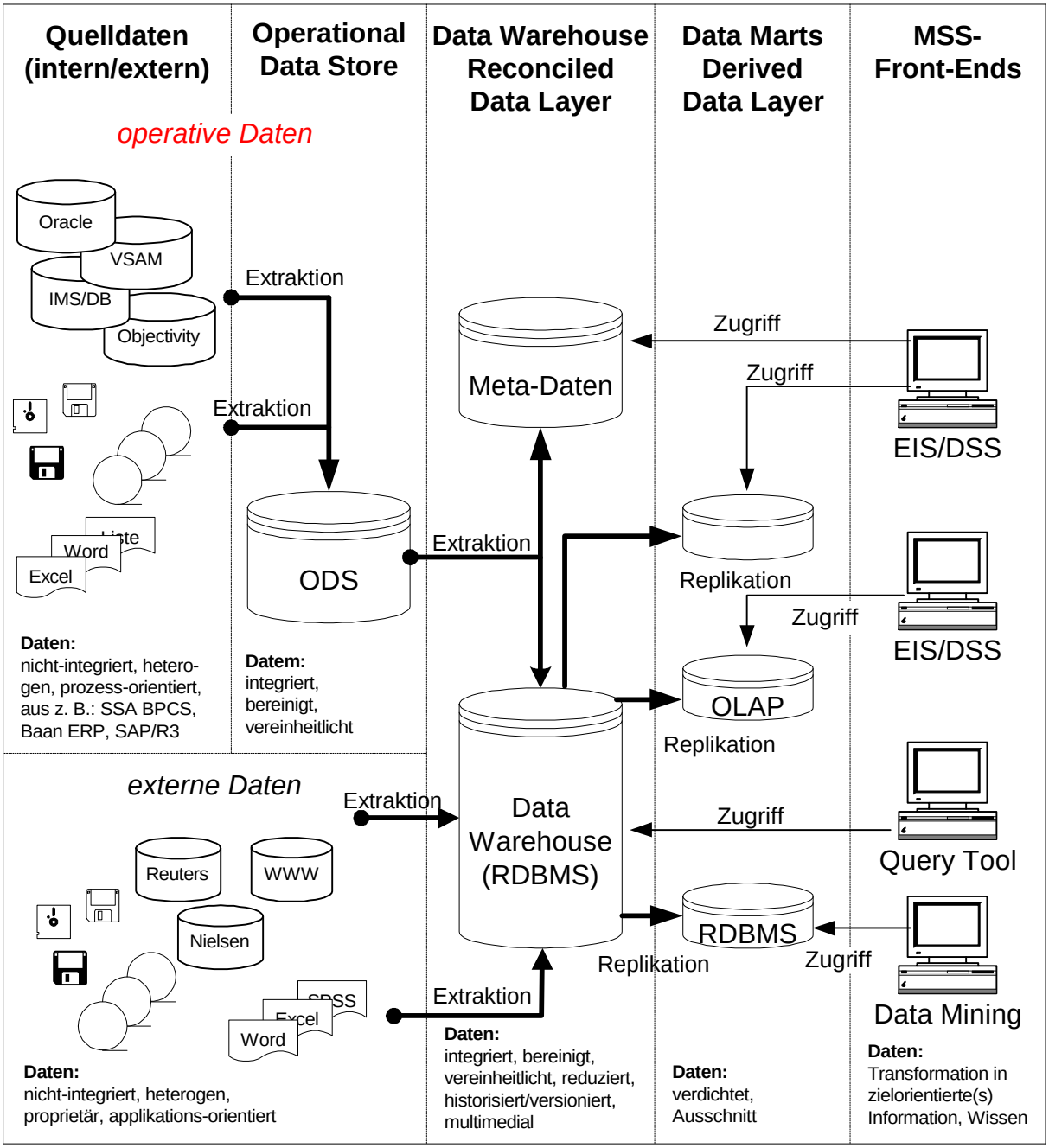


Abbildung 8: Data-Warehouse-Architektur

Ein weiteres mit dem ODS verfolgtes Ziel ist die Befriedigung des kurzfristigen operativen Informationsbedarfs. Beispielsweise möchte ein Kundenberater einer Bank vor einer bevorstehenden Kreditvergabe den Gesamt-Kontostand eines Kunden kennen. Gibt es verschiedene Applikationen und Datenhaltungskomponenten für Girokonto, Wertpapierdepot, Immobilienanlage et cetera, ist der Berater gezwungen, die Bestände aus den einzelnen Applikationen herauszusuchen

und mit einem Taschenrechner oder Spreadsheet-Blatt zu saldieren.[47] Für solche Informationen, die als Grundlage aktuelle Daten benötigen, ist das Data Warehouse nicht geeignet. Der ODS stellt hierfür eine alternative Lösung bereit und soll auch entsprechende Anfragen mit Ad-hoc-Charakter möglich machen.

Durch die Erstellung eines Operational Data Store wird bereits ein wesentlicher Teil des Data-Warehouse-Realisations-Prozesses erbracht. Die Daten aus den Legacy Systems müssen zu einer rekonziliierten operativen Datenbasis bereinigt, vereinheitlicht und integriert werden. Der dafür benötigte Aufwand ist erheblich. Synonyme, Homonyme und unterschiedliche Einheiten der Daten (z. B. pro Geschäfts- oder Kalenderjahr, in Meilen oder Kilometer) müssen vereinheitlicht bzw. inhaltlich verbunden werden. Inkonsistenzen in und zwischen den Datenquellen müssen bereinigt, Redundanzen beseitigt werden.

Die mangelnde Datenhygiene erschwert diesen Prozess in hohem Maße. Die Daten der Legacy Systems liegen oft in einer solch unbrauchbaren Form vor, dass deren Auswertung nicht sinnvoll möglich ist. Die Brauchbarkeit der Daten und damit auch der Erfolg eines Data-Warehouse-Projektes insgesamt hängt unmittelbar von der Qualität der Quelldaten ab. [Hack97, S. 43 f.]

Damit verbunden ist die Erzeugung der Mappings zwischen den Legacy-Daten und den ODS-Daten. Sie stellen einen Teil der Data-Warehouse-Metadaten dar und werden zusammen mit den Rekonziliierungs-Algorithmen in eine entsprechende Metadatenhaltungskomponente integriert.

Die Schritte zur Erstellung des Operational Data Store sind für ein Data Warehouse zum größten Teil unverzichtbar. Einzig auf die Rekonziliierung der nicht MSS-relevanten operativen Daten kann für das Data Warehouse verzichtet werden. Ob ein ODS explizit neben den operativen Systemen erstellt werden muss, hängt von verschiedenen Faktoren ab:

- Gibt es anstelle eines integrierten operativen Systems lediglich eine Landschaft heterogener Legacy Systems, sollte eventuell zuerst dieser Mangel beseitigt werden und nur eine kurzfristige die notwendigsten Informationsbedürfnisse befriedigende Data-Warehouse-Lösung realisisert werden.
- Bei der Einführung eines integrierten Administrations- und Dispositionssystems ist zu beachten, dass inzwischen Standardlösungen der Normalfall im operativen Bereich sind. Da Standardsoftwarepakete ein festes Datenmodell aufweisen, wird aber ein unabhängig erstellter ODS kaum Einfluss auf die Datenbasis dieser Standardsoftware haben. Einen ODS parallel zu Systemen

[47] Es handelt sich dabei um ein realistisches Beispiel, das in vielen ähnlichen Fällen geradezu den Normalfall darstellt..

wie SAP R/3 oder Baan IV zu betreiben ist allerdings enorm aufwendig[48] und wird gewöhnlich weit mehr Kosten verursachen als an explizitem Nutzen zu erwarten ist.

- Besteht eine integrierte operative Lösung, es können aber trotzdem keine Informationen aus obigem Beispiel durch diese Systeme geliefert werden, ist dies ein Mangel des Systems an sich, der beseitigt werden sollte. Ad-hoc-Anfragen müssen natürlich zu realisieren sein.
- Der Data-Warehouse-Fokus und dessen Aktualität wird jedoch den gewöhnlichen Informationsbedarf der Anwender decken können.

2.4.2 Data-Warehouse-Kern

Der Data-Warehouse-Kern besteht auf der Ebene der Daten aus den MSS-Daten und auf der Ebene der Metadaten aus den Data-Warehouse-Metadaten. Die MSS-Daten werden konzeptionell durch das MSS-UDM abgebildet (siehe Abschnitt 2.3.2) und logisch durch eine rekonziliierte Datenschicht implementiert, den Reconciled Data Layer (RDL). [Devl97, S. 69 ff.]

Der Reconciled Data Layer bildet tatsächlich den Kern des Data Warehouse. Er stellt das Data Warehouse i. e. S. dar. Der RDL ist die grundlegende Datenbasis des Data Warehouse, die das Data-Warehouse-Konzept realisiert (siehe die Abschnitte 2.2 und 2.3). Auch der Reconciled Data Layer ist damit eine den Prinzipien der Datenmodellierung folgende normalisierte Datenbasis.[49] Nach *Devlin* ist ein RDL „detailed, historical, consistent, modeled [and] normalized“ [Devl97, S. 128].

Die unterste Schicht des RDL bilden die rekonziliierten MSS-relevanten operativen Daten (siehe die Abschnitte 2.3.1 und 2.4.1). Sie werden entweder aus dem Operational Data Store oder direkt aus den operativen Systemen gewonnen. Rein operative Daten wie beispielsweise applikationsspezifische Daten zur Benutzersteuerung oder Statusattribute werden aufgrund ihrer fehlenden MSS-Relevanz nicht in den RDL integriert. Redundante oder fehlerhafte Daten dürfen ebenfalls nicht integriert werden.

Ergibt die Informationsbedarfsanalyse den Bedarf für weitere Daten, die für Analysen zu integrieren sind, bis dahin aber nicht aus den operativen Datenquellen gewonnen werden können, müssen die operativen Systeme eventuell verän-

---

[48] Tausende von Attributen sind nicht ungewöhnlich. Nach jedem Update müssen die Mappings und Schnittstellen überarbeitet werden.

[49] Es sei bereits an dieser Stelle bemerkt, dass dies eine von der herrschenden Meinung [GGC97, S. 266 ff. Kimb96, S. 30, Mcgu98, MuBe98, S. 3 ff.] in bedeutendem Maße abweichende Auffassung darstellt. Es wird auf Abschnitt 2.5 verwiesen, in welchem eine kritische Betrachtung dieser Auffasung erfolgt.

dert werden, um zukünftig entsprechende Fehler in den Daten zu vermeiden oder diese Daten überhaupt erst durch das System erfassen zu lassen.

Die auf diese Weise gewonnene redundanzfreie rekonziliierte Schicht der MSS-relevanten operativen Daten muss in einen Folgeschritt historisiert und versioniert werden. Jedes Datenelement wird mit einem Zeitstempel versehen und ist damit Zeitpunkt- bzw. Zeitraum-bezogen. Kein Datenelement darf im RDL ohne Zeitstempel existieren, was ansonsten eine immerwährende Gültigkeit bedeutete. Verschiedene Autoren schlagen sogar vor, das Datum als Element des Schlüssels eines Datensatzes aufzunehmen.[50] [Devl97, S. 97 ff., Inmo96, S. 33 ff., Kimb96, S. 15] Es spielt mithin keine Rolle, ob es sich um Stamm- oder Bewegungsdaten oder anders ausgedrückt um Fakt- oder Dimensionsdaten handelt — alle müssen mit einem Zeitstempel versehen, d. h. historisiert werden.

Die Historisierung der Daten ist jedoch nicht hinreichend. Bedingt durch den ständigen Wandel des Data Warehouse — das Data Warehousing — [Mart98b, S. 29 ff.] ändern sich auch die Strukturen des Reconciled Data Layer. Strukturelle Änderungen des RDL erfordern eine entsprechende Anpassung — eine Schemaevolution und damit auch eine Versionierung der betroffenen Strukturen. Beispiele sind das Hinzufügen und Löschen von Attributen, das Ändern des Namens, des Typs oder der Domäne eines Attributes, das Hinzufügen oder Löschen von Entitäten (Relationen) sowie die Änderung von Constraints. [Heue97, S. 393 f.] Auch die Modifikationen von Methoden sind Veränderungen deren Versionierung eventuell notwendig ist.

Während in operativen Systemen der Zeitbezug von Daten und Strukturen häufig eine untergeordnete Rolle spielt, ist dieser im Data Warehouse elementar. Beispielsweise werden operative Daten nach einer anforderungsadäquat festgelegten Periode archiviert oder einfach gelöscht. So werden Auftragsdaten in der Regel nicht über einen Horizont von mehr als ein bis zwei Jahren für das operative Geschäft benötigt. Stammdaten wie z. B. die Adresse eines Kunden werden üblicherweise mit aktuellen Daten überschrieben; die alte Adresse ist von geringem Interesse.

Auch der alte Zustand nach einer Schemaevolution ist für das operative Geschäft gewöhnlich bedeutungslos. Beispielsweise können Attribute zu einem Datensatz hinzugefügt werden, wie die E-Mail-Adresse eines Kunden. Oder es kann die Aufteilung einer Relation in zwei getrennte Tabellen erforderlich werden, wie bei einer möglichen Veränderung, die bewirkt, dass Artikel ab einem festgelegten Zeitpunkt nicht mehr nur von einem sondern von mehreren Lieferanten bezogen werden können. Weitere Fälle sind etwa die Veränderungen der bezogenen Re-

---

[50] Dies setzt wiederum ein RDBMS voraus und führt zu einem sprechenden Schlüssel. Die Verwendung von sprechenden Schlüsseln ist allerdings in hohem Maße problematisch. [Fink93, S. 25 f.]

gionen, die sich durch die fünf neuen Bundesländer ergeben oder Unternehmens-Umstrukturierungen wie von einer funktionalen hin zu einer Produkt-bezogenen (Profit-Center) Organisation.

Im Data Warehouse ist dagegen durchaus von Wichtigkeit, in welchem Zeitraum ein Kunde eine Region bewohnt, d. h. welche Adresse wann Gültigkeit besitzt. Ansonsten könnten unter anderem dessen Umsätze nicht mehr korrekt zugeordnet werden, was nachträglich zu falschen Analysen führen würde. Und es kann von bedeutendem Interesse sein, wann sich welche Datenstrukturen verändert haben. Vor dem Zeitpunkt, an welchem die Datenstrukturen überhaupt nicht die Möglichkeit boten, dass Artikel von mehreren Lieferanten gleichzeitig bestellt werden konnten, ist die Erstellung entsprechender Analysen auch nicht sinnvoll.

Gibt es mehr Regionen bzw. abweichende Regionsaufteilungen, müssen die Veränderungen herausgerechnet werden, um den Vergleich zu den alten Strukturen weiterhin korrekt zu ermöglichen. Das selbe gilt bei obiger Unternehmens-Umstrukturierung, bei welcher versucht werden muss, beispielsweise die neuen Deckungsbeiträge pro Produktsparte und allen Regionen mit den Deckungsbeiträgen pro Region und allen Produktsparten vergleichen zu können. Auch die Veränderung der Datenherkunft oder eines Rekonziliierungs-Algorithmus kann eventuell von Bedeutung sein, so dass in diesem Fall die Versionierung der Methoden notwendig ist.

Im zweiten Schritt werden die sonstigen strukturierten MSS-Daten aus den internen und externen Quellen in den Reconciled Data Layer integriert. Auch diese Daten müssen historisiert und versioniert werden. Das selbe gilt für den dritten Schritt, in welchem die semistrukturierten MSS-Daten integriert werden.

Dieser Kern des Data Warehouse, der Reconciled Data Layer, muss alle MSS-relevanten Daten enthalten und deren Integrität bzw. Datenqualität langfristig sichern. Was im RDL nicht vorhanden ist, kann nicht für MSS-Zwecke verwendet werden. Mängel in der Qualität der Daten und in den Strukturen wirken sich auf alle MSS und damit auch auf die Unternehmensentscheidungen aus. Vernachlässigungen bei der Erzeugung des RDL, wirken mithin auf das gesamte MSS-Konzept. Die Qualität des MSS-UDM und des RDL sind folglich mitbestimmende Faktoren des Unternehmenserfolges. Ein minderwertiger oder fehlerhafter RDL kann deshalb die Tauglichkeit des gesamten MSS-Konzepts infrage stellen.

Die Data-Warehouse-Metadaten gehören ebenfalls zum Data-Warehouse-Kern. Sie sind elementar für den Betrieb des Data Warehouse und die Informationsgewinnung aus dem RDL mithilfe der MSS-Front-Ends bzw. durch die Endanwender (siehe Abschnitt 2.3.1).

Auch die Metadaten müssen in der Folge Built-Time-Metadaten, Control-Metadaten und Usage-Metadaten integriert werden. Dies muss parallel mit den Schritten beim Aufbau des Reconciled Data Layer geschehen, d. h. erst werden

die Metadaten der operativen MSS-Daten integriert, dann die Metadaten der strukturierten externen Daten und zuletzt die Metadaten der semistrukturierten Daten.

Die Metadaten müssen genauso wie die MSS-Daten bereinigt, vereinheitlicht und integriert werden. Sie dürfen weder fehlerhaft sein, noch darf die Metadatenbasis Redundanzen aufweisen. Außerdem werden analog auch die Metadaten historisiert und versioniert.

Werden auch die über den Fokus des Data Warehouse i. e. S. hinausgehenden MSS-Metadaten, d. h. die Differenz zwischen MSS-Metadaten und Data-Warehouse-Metadaten in den Kern des Data Warehouse integriert (siehe Abschnitt 2.3.1), ist dafür ein weiterer Schritt notwendig. Ein Beispiel für MSS-Metadaten, die nicht Data-Warehouse-Metadaten sind, sind Metadaten über bereits erzeugte Analysen: welches Ziel bzw. welchen Inhalt hat die Analyse, wo ist diese Analyse zu finden und mit welchem MSS-Front-End wurde diese Analyse erzeugt bzw. kann sie gelesen werden. Diese Metadaten sind keine Data-Warehouse-Metadaten, da sie sich nicht unmittelbar auf Daten des Data Warehouse beziehen.

Aufgrund des fließenden Übergangs zwischen MSS-Daten und MSS-Metadaten ist die Trennung zwischen den Datenbasen des Reconciled Data Layer und der Data-Warehouse-Metadatenbasis nur schwer zu vollziehen. Werden die Metadaten des Data Warehouse in eine zusätzliche Metadatenbasis integriert, müssen die Data-Warehouse-Metadaten kopiert werden. Diese redundante Haltung erschwert die konsistente Haltung der Metadaten weiter.

2.4.3 Anwendungs-orientierte Schichten

Die auf dem Reconciled Data Layer aufsetzenden, d. h. von diesem abgeleiteten Schichten, bilden den Derived Data Layer (DDL). Während der RDL die Kernschicht des Data Warehouse mit dem Ziel bildet eine langfristig konsistente Datenbasis mit hoher Datenqualität und einem hohen Maß an abgebildeter Semantik zu sein, gilt dies für die aufsetzenden Schichten in weit aus geringerem Maße. Vielmehr ist das Ziel der einzelnen Schichten des DDL jeweils kurz- bis mittelfristig als Output-Schicht des Data Warehouse zu fungieren.

Der DDL bietet Schichten bzw. Sichten auf dem RDL um MSS-Front-Ends bzw. Endanwender oder Endanwendergruppen (z. B. Abteilungen) bedarfsgerechter mit MSS-Daten zu versorgen, als dies mit dem RDL möglich ist. Spezielle Kopien von Ausschnitten aus dem RDL werden als Data Marts bezeichnet. Data Marts können Daten unter anderem auf einem höheren Aggregationsniveau, d. h. mit einer höheren Daten-Granularität, in speziellen Datenschemata oder auch beliebig transformiert speichern. Bei *Devlin* werden Data Marts als Business In-

formation Warehouses (BIW), der RDL als Business Data Warehouse (BDW) bezeichnet. [Devl97, S. 125 ff.]

Der RDL ist in seiner Gesamtheit eventuell zu komplex. Viele Anwender beziehen den Großteil ihrer Information aus einem kleinen Teil des Datenbestandes. MSS-Front-Ends benötigen in der Regel spezielle Datenstrukturen oder verlangen sogar eine eigene, oft proprietäre Datenbasis. Häufig genannte Argumente für den DDL sind auch das Erfordernis für spezielle Semantiken und die Notwendigkeit zur Erzeugung eines für die Analysen angemessen kleinen Zeitfensters.

OLAP und Data Mining (siehe Abschnitt 2.1.3) sind die gegenwärtig populärsten MSS-Applikationen. [BaSe97b, S. 1, ChGl99a, S. 1 ff., MuBe98, S. IX ff.] Demzufolge werden derzeit in besonderem Maße DDL-Schichten für OLAP und Data Mining diskutiert und realisiert.

Ein OLAP-Data-Mart speichert die Daten entweder als Arraystrukturen in einer speziellen multidimensionalen proprietären Datenbank oder mithilfe eines RDBMS. Wird für ein OLAP-Data-Mart relationale Technologie verwendet, liegt ein spezielles Datenschema zugrunde. Verwendung finden Varianten des Star-Schemas, wie das Snowflake-Schema, das Galaxie-Schema, das Fact-Constellation-Schema oder Kombinationen aus diesen Grundformen. [Hahn99, S. 152 ff.]

Die zugrunde liegende Idee ist es, dass, konzeptionelle OLAP-Modell eines Würfels, d. h. von Faktwerten (Measures) und Dimensionen (siehe Abschnitt 2.1.3) in den Tabellen des Relationenmodells abzubilden. So gibt es in der Grundform des Starschemas eine Fakttabelle, in welcher die Faktwerte enthalten sind und Dimensionstabellen mit den die Faktwerte beschreibenden Dimensionswerten. Außer den Faktwerten enthält die Fakttabelle auch die Referenzen (Fremdschlüssel) auf die Dimensionstabellen bzw. die zugehörigen Datensätze der Dimensionstabellen.

Dadurch gelingt es, die Semantik der mehrdimensionalen OLAP-Analyse in den Datenstrukturen des OLAP-Data-Mart abzubilden und die Anwendung z. B. bei Anfragen mit SQL oder einem ROLAP-Werkzeug (siehe Abschnitt 2.1.3) für den Endanwender erheblich zu vereinfachen. [Mcgu98] Das Beschreibungsmodell des RDL wird zu einem Erklärungsmodell erweitert.

Auch das Performanz-Ziel kann auf diese Weise realisiert werden. Beispielsweise werden die Daten denormalisiert, indem auf die erste Normalform verzichtet wird, so dass Wiederholungsgruppen entstehen und folglich jeweils zwei Tabellen zu einer Tabelle zusammengelegt werden können und berechnete Werte (z. B. Aggregationen) physisch gespeichert werden, mithin auf die dritte Normalform verzichtet wird. Durch diese Denormalisierung kann bei der Anfrage auf viele Berechnungsfunktionen und das Joinen von Tabellen verzichtet werden, was in den meisten Fällen zu einer beträchtlichen Performanz-Steigerung führt.

Allerdings gibt es auch bei dieser Art von Datenmodellen Zielkonflikte. Das Star-Schema etwa verzichtet auf die Abbildung der Dimensionshierarchie-Semantik (z. B. Artikel, Artikelgruppe, Warengruppe, Sortimentsgruppe oder Tag, Monat, Jahr) und damit auf Anwenderfreundlichkeit zugunsten einer weitestgehenden Denormalisierung und damit höheren Performanz. Dadurch wird die Speicherung von aggregierten Werten, was eine höhere Semantik und Anfragegeschwindigkeit bewirkt, enorm kompliziert. [Mcgu98]

Weiterhin ist zu beachten, dass die erzeugten Schemata nicht nur nach einer für den Datenbestand bzw. den Geschäftskontext optimalen Kombination erzeugt werden müssen. MSS-Werkzeuge weisen in der Regel zusätzliche Besonderheiten auf, die Strukturen des zu entwickelnden Schemas vorgeben. Das bedeutet, dass gewisse Strukturen eines Datenmodells vom MSS-Front-End, in diesem Fall ROLAP-Front-End, nicht verwendet werden können.[51] Datenschema oder MSS-Front-End sind dann — je nach Sichtweise — unbrauchbar.

Analog verhält es sich bei Data-Mining-Techniken. Data-Mining-Tools arbeiten gemeinhin nicht auf beliebigen Datenstrukturen. Multikriterielle Analysen sind beispielsweise noch Gegenstand der Forschung. Demzufolge müssen nach den Anforderungen der Analyse und des Data-Mining-Produktes Data-Mining-Data-Marts erzeugt werden. Da diese Data Marts einen beträchtlichen Umfang annehmen können, meist aber nur kurzfristig gebraucht werden, ist die temporäre Erzeugung von Data-Mining-Data-Marts in vielen Fällen die empfehlenswerteste Lösung.

Für bestimmte Anwender- bzw. Front-End-Anforderungen kann es zweckmäßiger sein, anstelle eines expliziten Data Marts Sichten (Views) auf dem Reconciled Data Layer zu definieren. Dadurch kann ein Anwender etwa davor bewahrt werden, sich mit versionierten Strukturen befassen zu müssen, indem ihm eine gemeinsame Sicht, d. h. in einer Version, aller von ihm betrachteten Datenstrukturen angeboten wird.

Über die Schichten bzw. Sichten des Derived Data Layer können ebenfalls die Rechtestrukturen geregelt werden. Aufgrund des hohen Geschäftswertes der Informationen eines Data Warehouse spielen Rechtestrukturen in diesem Zusammenhang eine herausragende Rolle. *Devlin* und *Murphy* integrieren bereits im ersten Entwurf des Data Warehouse einen entsprechenden umfangreichen Security Layer. [DeMu88, S. 76]

Aufgabe des Derived-Data-Layer-Konzepts ist es mithin, jeweils benötigte Schichten und Sichten für Anwender und MSS-Front-Ends bereitzustellen. Weil

---

[51] Die Cognos-Suite kann beispielsweise keine kombinierten Schlüssel für einen OLAP-Würfel verwenden. Mit Business Objects können keine nach der *kimballschen* Methode aggregierten Werte im Starschema verarbeitet werden.

diese Schichten aus dem Reconciled Data Layer gewonnen werden, ist die Korrektheit gewährleistet. Die DDL-Schichten weisen jedoch vielfache Redundanzen innerhalb einer Schicht und zwischen den Schichten auf. Werden die DDL-Schichten im Laufe der Zeit nicht mehr handhabbar, inkonsistent oder bekommen andere Mängel, bleibt dies für gewöhnlich ohne größere Folgewirkungen. Die Daten können jederzeit erneut aus dem RDL gewonnen und bedarfsgerecht bereitgestellt werden. Die Lebensdauer einzelner DDL-Schichten bzw. -Sichten dürfte deshalb in der Regel auf kurz- bis mittelfristige Perioden begrenzt bleiben.

2.5 Kritik an bisherigen Data-Warehouse-Ansätzen

Die Kritik an bisherigen Data-Warehouse-Umsetzungen besteht aus unterschiedlichen Richtungen. Die empirische Kritik richtet sich zum Einen an die momentan in der Praxis vorzufindenden Implementierungen und zum Anderen an die in der Literatur üblichen oder gar herrschenden Meinungen. Diese Kritik besteht hauptsächlich darin, die unter 2.2 bis 2.4 erarbeitete Data-Warehouse-Konzeption mit jener in Praxis und Literatur[52] abzugleichen.

Die konzeptionelle Kritik geht über diese Punkte hinaus, indem sie die bisherige Konzeption an sich infrage stellt. Dies ist im Sinne dieser Arbeit verständlich und erklärtes Ziel. Insbesondere die Hypothese der ausschließlichen Eignung von relationaler Datenbanktechnologie verbunden mit konventioneller[53] Programmierung zur Data-Warehouse-Realisation wird negiert und es wird versucht, dafür einen entsprechenden Nachweis zu erbringen.

Die Entwicklung der konventionellen Softwaretechnologie erschwert diese Unterscheidung allerdings. Erweiterungen von C um beispielsweise ein Klassenkonzept zu C++ und von relationalen DBMS um Stored Procedures, Trigger oder Tupel-Identitäten zu objekt-relationalen DBMS [Burl97, S. 20 ff.] sind Beispiele hierfür.

Weiterhin ist die Differenzierung zwischen den unmittelbar und den nur mittelbar dem Ansinnen des Data-Warehouse-Konzepts entgegenstehenden Effekten schwierig, die aufgrund der Verwendung konventioneller Technologie entstehen. Einige der nur mittelbar zurechenbaren Effekte werden fälschlicherweise als

---

[52] Literatur wird hier auch als Synonym für Forschung verwendet. Allerdings ist Literatur der umfassendere Begriff und hilft bei der Vermeidung einer Klassifikation in Literatur, die als Forschungsergebnis zu betrachten ist, und Literatur, welche dieser Kategrorie nicht zugerechnet werden kann.

[53] Konventionelle Programmierung, Softwareentwicklung, Systementwicklung et cetera wird in Abgrenzung zu objektorientierter Programmierung, Softwareentwicklung, Systementwicklung et cetera verstanden und meint gewöhnlich die Verwendung relationaler Datenbanktechnologie auf der Datenseite und prozeduraler Programmiersprachen auf der Funktions- bzw. Algorithmenseite. Üblicherweise werden Sprachen wie C, Pascal oder Cobol mit eingebundenen SQL-Statements benutzt, sogenannte 4GL (Fourth Generation Languages).

Fehler von Praxis und Literatur angesehen, obgleich sie mit relationaler Technologie durchaus zu realisieren sind, dies aber zu ineffizient ist.

Eine streng getrennte Diskussion nach empirischer und konzeptioneller Kritik ist aus diesen Gründen unangebracht. Statt dessen wird die Kritik analog zu Abschnitt 2.4 anhand der Data-Warehouse-Schichten diskutiert und entsprechend zu Abschnitt 2.3.3 um die Data-Warehouse-Engineering-Aspekte ergänzt.

2.5.1 Quell-Schichten

Das Data-Warehouse-Problem beginnt bereits bei den Quellsystemen. Mangelnde Qualität von Daten und Datenmodell der zugrunde liegenden Systeme können zum Teil nur mit erheblichem Aufwand, zum Teil überhaupt nicht ausgeglichen werden. Sind Daten nicht vorhanden, können sie auch nicht für das Data Warehouse genutzt werden und stehen damit auch nicht für aufsetzende MSS zur Verfügung. Genauso verhält es sich mit fehlerhaften Daten. Nicht alle Datenfehler können mithilfe von Algorithmen oder sogar per Hand korrigiert werden.

Beispielsweise zeigte sich in einem durchgeführten Projekt, dass Scannerdaten, die codiert gespeichert wurden strukturelle Fehler aufwiesen. [BHJ+96] Die Decodierung war damit nicht mehr vollständig möglich. Die fehlerhaften „Kassenbons“ konnten nicht in das Data Warehouse integriert werden. Als Folge ergaben sich damit von den realen Daten abweichende Ergebnisse für beispielsweise Umsatz oder Absatzmenge.

Inwieweit ein Data-Warehouse-Projekt in solchen Fällen sinnvoll ist, ist zumindest fraglich. Die Datenqualität der Quelldaten ist jedoch regelmäßig von erheblich schlechter Qualität, was alle bisher durchgeführten Projekte bestätigen. Die Hauptursache liegt in der schlechten Software-Qualität der operativen Systeme. Für *Yourdon* sind hierfür unsystematische Vorgehensweisen, kurzfristig orientierte Handlungsperspektiven, ökonomischer Druck, schlechtes Management und die Ignoranz gegenüber psycho-sozialen Faktoren verantwortlich. [Your93, S. 1 ff.]

Ein wesentlicher Aspekt ist, dass immer wieder auf die Erstellung eines OUDM verzichtet wird. Dies hat direkte Auswirkungen auf das Data Warehouse (siehe Abschnitt 2.3.2). Anstelle einer ingenieurs-wissenschaftlichen Vorgehensweise wird der Software-Erstellungsprozess immer noch als eine Art künstlerischer Beruf missverstanden [Your93, S. 61 ff.]. Doch selbst wenn in aufwendigen Verfahren ein OUDM entsteht und sogar als Grundlage für das realisierte System Verwendung findet, ist die Übereinstimmung von OUDM und Software von nur kurzfristiger Dauer. Während die Software einem ständigen Wandel unterzogen

ist, bleibt das OUDM meist statisch.[54] Ein für das Data Warehouse zu verwendendes OUDM ist mithin in vielen Fällen nicht vorhanden.

Selbst wenn ausgereifte operative Standardsoftwarelösungen wie etwa Baan IV analysiert werden (siehe Abschnitt 5.1), zeigt sich auch deren mangelhaftes Datenmodell. Dies verdeutlicht, um wieviel schlimmer die Auswirkungen von Insellösungen von Legacy Systems sind. Die Systeme sind zum großen Teil überhaupt nicht nach Datenmodellierungs-Gesichtspunkten [Date00, S. 330 ff.] modelliert. Die Anbindung dieser Systeme an ein gemeinsames OUDM ist nur mit erheblichem Aufwand möglich. Schlechte Datenmodelle verzichten auf die Abbildung von Semantik und begünstigen die Entstehung mangelnder Datenqualität.

Ein erfolgversprechendes Data-Warehouse-Projekt beginnt folglich bei den Quelldaten. Hierfür muss ein integriertes operatives System, ein jeweils (dynamisch) aktuelles OUDM und hohe Datenqualität angestrebt werden. Die für MSS erforderlichen Daten müssen durch die operativen Systeme erfasst werden.

Zwar ist es oft sinnvoll, ein erstes rudimentäres Data Warehouse auf den Legacy Systems zu implementieren, da es ansonsten, zugunsten der Verbesserungen an den operativen Systemen, eventuell nie zu einer Data-Warehouse-Implementierung kommt, doch hängt der Erfolg letztlich ganz wesentlich von den Quellsystemen ab.

Sind die operativen Systeme zu heterogen, sollte zumindest temporär ein Operational Data Store realisiert werden. Doch auch dies geschieht nur in seltenen Fällen. Dementsprechend wird ein ODS-Konzept auch nur selten in der Data-Warehouse-Literatur thematisiert. Selbst wenn ein ODS propagiert wird, hat es mit dem ODS-Konzept [Inmo99, S. 12 ff., InHa94, S. 27 ff.] wenig Gemeinsamkeiten.

Beispielsweise fordert *Kimball* einen ODS einzig dazu, den aktuellen Informationsbedarf zu befriedigen. Er fordert, die Implementierung eines ODS mithilfe des Star-Schemas zu realisieren. [Kimb97] Das Star-Schema ist jedoch in keiner Weise dazu geeignet, das OUDM abzubilden. Es bildet eine einzige Sicht — die multidimensionale — ab, kann nur stark eingeschränkt Semantik abbilden und erzeugt hohe Redundanz. Das Star-Schema ist für einen Operational Data Store mithin völlig ungeeignet.

Ein erhebliches MSS-Potenzial bilden die über die operativen Daten hinausgehenden Daten (siehe Abschnitt 2.3.1). So wird regelmäßig die Integration von externen, semistrukturierten oder qualitativen Daten gefordert. [Devl97, S. 59 ff., InHa94, S. 113 ff., MuBe98, S. IX ff., UhBr98, S. 7 ff.]. Wie dies allerdings kon-

[54] Trotz erheblicher Fortschritte im CASE-Bereich gibt es für dieses Problem bisher keine adäquaten Lösungskonzepte.

kret zu bewerkstelligen ist, bleiben die Autoren in der Regel schuldig. Praktische Beispiele bilden deshalb bisher die Ausnahme.

Gründe hierfür sind, dass bisherige Data-Warehouse-Implementierungen noch in der Anfangsphase stecken und mit der Integration der operativen Daten üblicherweise begonnen wird, viele Entscheidungsträger primären Wert auf quantifizierbare Daten legen, die Integration semistrukturierter Daten erheblich schwieriger zu bewerkstelligen ist und letztlich das Relationenmodell zur Integration semistrukturierter Daten ungeeignet ist.

2.5.2 Data-Warehouse-Kern

Eine Zielsetzung, zumindest jedoch ein wesentlicher Entstehungsgrund für das Data-Warehouse-Konzept ist die mangelhafte Qualität der Quellsysteme. Es scheint jedoch, als würde genau jenes Qualitätsniveau auch für das Data Warehouse angestrebt werden. Die Modellierung eines UDM wird vernachlässigt, Data-Warehouse-Implementierungen werden wieder als Insellösungen realisiert und auf einen Reconciled Data Layer wird verzichtet.

Die Modellierung eines MSS-UDM oder gar dessen Integration mit dem OUDM zum GUDM werden kaum realisiert. Zwar gibt es inzwischen einzelne Autoren, die den Stellenwert der Modellierung eines UDM als Basis des Data Warehouse verdeutlichen [Bric98, Devl97, S. 92 ff., Kell96, S. 61 ff., Zach97, S. VIII], doch bilden sie nach wie vor die Ausnahme. Dementsprechend gering ist die Bedeutung in der Praxis. Die Modellierung wird als zu kompliziert, zeitaufwendig und kostenintensiv angesehen.

Statt dessen entstehen kurzfristige Insellösungen. Es werden einzelne Data Marts realisiert, deren Zielsetzung in einer lokalen kurzfristigen Befriedigung des Informationsbedarfs besteht. Es entstehen hoch redundante inkonsistente Data Marts, meist auf spezielle MSS-Applikationen zugeschnitten, deren Weiterentwicklung und Wartung kaum möglich ist. Dadurch ist der Zielsetzung des Data-Warehouse-Konzepts nur wenig gedient. Das Ergebnis unterscheidet sich nur geringfügig von dem Zustand der Spider Web Systems.

Aufgrund dessen wird auch regelmäßig auf die Erstellung eines übergreifenden Informationskonzepts verzichtet. Regelungen über Zugriffsrechte und die bedürfnisadäquate Informationsverteilung unterbleiben daher. Informationsinseln, bisherige Machtstrukturen und Informationsintransparenz haben weitgehend Bestand. Trotzdem werden gerade diese nicht-integrierten Data Marts implementiert und auch in der Literatur erfolgt nur wenig diesbezügliche Kritik [Cw-98, S. 16]. Teilweise wird zwar die Notwendigkeit eines UDM erkannt, aber von einem zentralen Data Warehouse zugunsten „flexibler Data Marts“ [Dobe98] abgeraten, ohne zu erkennen, dass dies nicht zu realisieren ist.

Auch in den Fällen zentraler Data Warehouses kann nur in seltenen Fällen von der Realisation eines Reconciled Data Layer gesprochen werden. Datenmodellierung ist im Data-Warehouse-Bereich bisher keine übliche Technik. Doch selbst wenn von Datenmodellierung für ein Data Warehouse die Rede ist [BaSe97b, S. 10, Burl97, S. 49 ff. Edel97, S. 31 ff., Hahn99, S. 152 ff., Holt98, S. 143 ff., Kimb96, S. 8 ff., Mcgu98, Raut97, S. 5 ff.], wird nur in seltenen Ausnahmefällen von einem RDL-Konzept [Devl97, S. 70 ff., Mert98a, S. 58 ff.] ausgegangen. Fast alle diesbezüglichen Artikel beschreiben das Data Warehouse als reine OLAP-Datenbasis und verzichten auf die Abbildung reichhaltiger Semantik, die Integration semistrukturierter Daten, Versionierung und die Historisierung von Dimensionsdaten.

Die aktuelle Bedeutung von OLAP-Analysen wird derart hoch bewertet, dass Data Warehouses meist nur multidimensionale Strukturen abbilden. Faktisch erfolgt eine Gleichsetzung von OLAP-Datenbasis und Data Warehouse. Data Warehouses werden fast ausschließlich als Star-Schema-Variante realisiert. Die Verwechslung reicht sogar soweit, dass einzelne Autoren [BaSe97b, S. 6, Edel97, S. 46 f.] die Überlegung anstellen, ob es besser sei, ein Data Warehouse mit relationaler Technologie als ROLAP oder mit multidimensionalen Datenbanken, also einer MOLAP-Datenhaltungskomponente wie Oracle Express Server, zu erstellen.

Mit Star-Schema-Varianten oder einer MOLAP-Datenbank kann ein Reconciled Data Layer nicht erzeugt werden!

Date bemerkt:

> „**Caveat**: We remark immediately that one thing these areas[55] all have in common is that good logical design principles are rarely followed in any of them! […]
>
> Unfortunately […] vendors and users of decision support systems often fail to distinguish adequately between logical and physical issue;[56] in fact, they often forgo logical design entirely. As a consequence, attempts to deal with the various characteristics discussed above tend to be **ad hoc** and often lead to insurmountable difficulties in trying to balance correctness, maintainability, performance, scalability, and usability requirements.“ [Date00, S. 695, S. 697]

OLAP deckt einen bedeutenden Teil des MSS-Spektrums ab. Nach *Thomsen* können unter Zuhilfenahme der OLAP-Technologie 90 % der aktuellen Fragestellun-

---

[55] *Date* bezieht sich auf die folgenden Begriffe: „… **data warehouse**, **data mart**, **operational data store**, **online analytical processing** (OLAP), **multi-dimensional databases**, and **data mining** (among others).“ [Date00, S. 697]

[56] „Data warehouse and OLAP specialists tend to be especially guilty on this count; they often argue that relational design is simply ‘wrong’ for decision support, claiming that the relational model is incapable of representing the data and must be circumvented. Such arguments almost always hinge on failure to distinguish the relational model from its physical implementation.“

gen beantwortet werden. [Thom97, S. XV ff.] Wird aber bedacht, dass nach einer Studie des Weißen Hauses in Washington Geschäftsführungsmitglieder amerikanischer Unternehmen ihre Entscheidungen zu über 95 % auf der Grundlage von externen Informationen treffen [Saxe96, S. 50], wird das hinter dem OLAP-Konzept liegende Data-Warehouse-Potenzial deutlich.

Zwar sind die meisten Anwender-Fragestellungen tatsächlich einfach und deshalb oftmals mit OLAP effizient zu beantworten, doch liegt der MSS-Fokus noch weit hinter dieser Analyseform (siehe Abschnitt 2.1). *Burleson* bemerkt, dass die Anforderungen der Anwender mit den eröffneten Möglichkeiten wachsen. Von „MIS (standard reports and forcasts of sales)", über „hypothesis testing" und „model building" hin zu „discovery of unknown trends" wachsen die Informationsbedürfnisse immer weiter an. [Burl97, S. 34]

Die Reduzierung auf nur eine Analyseform ist mithin nicht nachvollziehbar. Wieder einmal werden kurzfristige Zielsetzungen einer durchdachten Konzeption vorgezogen. Millionenbeträge werden fehlinvestiert. Anstelle von Investitionen in das Datenmodell, die Datenqualität und die Software-Qualität wird die Antwortgeschwindigkeit optimiert. Die Entwicklung von „Garbage Warehouses" [Mart97, S. 25] ist mithin „vorprogrammiert".

Dies trifft nicht nur auf die mit hoher Wahrscheinlichkeit auftretenden Inkonsistenzen zu. Zwar wird im Zusammenhang mit dem Data Warehouse von einem ständigen Entwicklungsprozess (Data Warehousing) gesprochen, doch wird dabei offensichtlich übersehen, dass ohne RDL strukturelle Änderungen kaum abbildbar sind. Konsequenterweise wird von Versionierung abgeraten. Selbst auf die Historisierung von Dimensionsdaten solle besser verzichtet werden, da das Datenmodell dadurch zu komplex in der Handhabung werde. [Kimb96, S. 100 ff.]

Ein inkorrektes Datenmodell mit der Folge falscher Informationen kann jedoch keine Alternative darstellen.[57] Zieht beispielsweise ein Kunde in eine andere Region, wird auch sein bisheriger Umsatz der neuen Region, dem neuen Vertreter oder der neuen Filiale zugerechnet oder der Kunde wird — als eine pragmatische Lösung — mit entsprechenden Folgen als Neukunde angelegt. Eine inflexible statische Data-Warehouse-Lösung wird den heutigen Anforderungen nicht gerecht, es wird jeweils nur ein Bruchteil seines Potenzials realisieren können oder sogar unbrauchbar sein.

Aus dem Data Warehouse können außerdem nur diejenigen Informationen gewonnen werden, die expliziter oder impliziter Bestandteil bzw. mit angemessenem Aufwand zu gewinnen sind. Der Grad an Semantik spielt dabei eine zentra-

[57] In diesem Zusammenhang sei an den bekannten Software-Engineering-Ausspruch erinnert: „For every problem there is a simple solution, and it's always wrong!" [zitiert nach Quot99] Oder um mit *Einstein* zu sprechen: „Everything should be made as simple as possible, but not simpler!" [zitiert nach Mins88, S. 17]

le Rolle. Deshalb eignet sich etwa ein Snowflake-Schema für die Beantwortung vieler multidimensionaler Fragestellungen besser als ein normalisiertes relationales Datenmodell. Umgekehrt beinhaltet ein normalisiertes relationales Datenmodell in der Regel mehr semantische Ausdruckskraft als ein Star-Schema.

Doch das Relationenmodell ist insgesamt in seiner semantischen Ausdruckskraft erheblich beschränkt. Zur Behebung dieses Mangels wurden in den letzten Jahrzehnten weitere Datenmodelle wie das semantische Datenmodell, das NF^2-Datenmodell, das eNF^2-Datenmodell, funktionale Datenmodelle und objektorientierte Modelle entwickelt. [Heue97, S. 131 f.] Beispielsweise können semistrukturierte Daten mit relationaler Technologie[58] oft nur als BLOB abgelegt werden. Die Beschränktheit der Datentypen zwingt zur atomaren Zerlegung und Rekonstruktion von Informationsobjekten (siehe Abbildung 7 auf Seite 52). Ausführliche Betrachtungen zu diesem Punkt erfolgen in Abschnitt 3.4.3.

Auch Metadaten sind Bestandteil der meisten Data-Warehouse-Konzeptionen. [InHa94, S. 70 ff., GrWa98, S. 15, Muck99, S. 172 ff., Poe96, S. 31 f.]. Die konkrete Spezifikation und Umsetzung bleiben die Autoren in der Regel schuldig. In den positiven Fällen der Metadatenintegration handelt es sich jedoch zumeist um technische Metadaten. Die im Data-Warehouse-Kontext elementaren Gebrauchs-Metadaten (Usage-Metadata), insbesondere auch die MSS-Metadaten werden bisher kaum erzeugt, obwohl *Devlin* und *Murphy* besonderen Wert auf genau diese Metadaten legen und deren Implementierung bereits in ihrer ersten Data-Warehouse-Konzeption fordern. [DeMu88, S. 71 ff.]

Ein weiterer Punkt ist die Integration der Metadaten. Zwar gibt es inzwischen Projekte zur Entwicklung von Metadaten Repositories (Meta Database Management Systems) [Huff97, S. 225 ff.] und erste Schnittstellen-Standards [Kell96, S. 184 ff.], doch fehlt nach wie vor die Diskussion darüber, wo die Trennlinie zwischen Daten und Metadaten zu ziehen ist. Es bleibt folglich unklar, welche Daten Bestandteil der Data-Warehouse-Datenbasis i. e. S. oder der Metadatenbank sind. Insbesondere das Problem zur Vermeidung von Redundanzen oder — sofern diese nicht vollständig zu vermeiden sind — die Entwicklung von Mechanismen zur Integritätssicherung der Datenbestände sind hoch prioritäre Fragestellungen.

2.5.3 Anwendungs-orientierte Schichten

Aufgrund des regelmäßig fehlenden RDL ist auch die Ableitung anwendungsorientierter Schichten (Derived Data Layer) problematisch. Im Falle einer Data-

[58] Es sei angemerkt, dass moderne RDBMS wie Oracle inzwischen vielfältige Erweiterungen aufweisen, die über das relationale Datenmodell hinausgehen. [Burl97, S. 20] Die Speicherung von BLOB ist bereits eine solche Erweiterung.

Warehouse-Datenbasis in Form einer Star-Schema-Variante werden gewöhnlich Ausschnitte extrahiert, die etwa abteilungsspezifisch sind. Diese Data Marts weisen eine andere Datengranularität, weitere Aggregationen und einen eingeschränkten Datenumfang auf. Damit ist das Spektrum des DDL meist schon erschöpft. Die vielfältigen MSS-Potenziale bleiben ungenutzt.

Die kritiklose Verwendung des Star-Schemas ist auch für multidimensionale Strukturen unangebracht, selbst bei der Ableitung aus dem RDL. Die hohe Redundanz wirft durchaus Probleme auf. Die immer wieder behauptete Volatilität, aufgrund derer Inkonsistenzen nicht auftreten könnten, ist real nicht gegeben. Die Performanz kann nicht in jedem Fall von denormalisierten Tabellen gesteigert werden. Die Semantik der Dimensionshierarchien kann nur äußerst kompliziert abgebildet werden. [Mcgu98]. MSS-Front-Ends können diese Abbildung in der Regel nicht verwenden. Bei multiplen Hierarchien verschärft sich das Problem. Dasselbe gilt für aggregierte Werte.

Date hält die Modellierung der Star-Schema-Varianten überhaupt für schlechte Modellierungsformen. Seine Kritk umfasst sechs Hauptpunkte [Date00, S. 713 f.]:

1. „First of all, it is **ad hoc** (it is based on intuition rather than principle). This lack of discipline makes it difficult to change the schema in a proper fashion when (for example) new types of data are added to the database or when dependencies change. In fact, star schemas are often constructed by simply editing a previous design, and that previous design in turn is often constructed by trial and error.
2. Star schemas are really physical, not logical, though they are talked as if they were logical. The problem is, that there is really no concept of logical design, as distinct from physical design, in the star schema approach.
3. The star schema approach does not always result in a legitimate physical design (i. e., one that preserves all of the information in a relationally correct logical design, as distinct from physical design). This shortcoming becomes more apparent the more complex the schema becomes.
4. Because there is little discipline, designers often include several different types of facts in the fact table. As a consequence, the rows and columns of the fact table typically do not have a uniform interpretation. What is more, certain columns then typically apply only to certain type of facts, implying that the columns in question must permit nulls. As more and more types or facts are included, so the table becomes more and more difficult to maintain and understand, and access becomes less and less efficient. For example, we might decide to modify the shipments table to track part purchases as well as part shipments. We will then need some kind of ‘flag’ column to show which rows corre-

spond to purchases and which to shipments. A proper design, by contrast, would create a distinct fact table for each distinct type of fact.

5. Again because of the lag of discipline, the dimension tables too can become nonuniform. This error typically occurs when the fact table is used to maintain data pertaining to differing levels of aggregation. For example, we might (mistakenly) add rows to the shipments table that show the total part quantities for each day, each month, each quarter, each year, and even the grand total to date. Notice first that this change causes the time period identifier (#TP) and quantity (QTY) columns in the table SP to have nonuniform meanings. Suppose now that the FROM and TO columns in the dimension table TP are each replaced by a combination of YEAR, MONTH, DAY, etc., columns must now all permit nulls. Also, a flag column will probably be needed too, in order to indicate the type of the applicable time period.
6. The dimensional tables are often less than fully normalized.[59] The desire to avoid joins often leads designers to bundle distinct information together in those tables that would better be kept separate. In the extreme case, columns that simply happen to be accessed together are kept together in the same dimension table. It should be clear that following such an extreme, and nonrelational, 'discipline' will almost certainly lead to uncontrolled — and probably uncontrollable — redudancy."

Vielfach fließen in die Data Marts weitere Daten ein, die nicht durch das Data Warehouse verwaltet werden. [Devl97, S. 57 ff.] Dadurch können wiederum Redundanzen und Inkonsistenzen entstehen. Solche Ergänzungen sollten deshalb entweder nur über den RDL in die Data Marts eingehen oder zumindest durch andere Maßnahmen mit den anderen Data Marts konsistent gehalten werden.

Die Ergänzung der Data-Warehouse-Daten durch die Benutzer muss jedoch gewährleistet sein, sind es doch die Anwender, die in vielen Fällen über relevante Daten verfügen und diese deshalb über das Data Warehouse zur Verfügung stellen müssen. Dies betrifft vor allem semistrukturierte Daten und Metadaten. Zeitschriftenartikel, Web-Links, Definitionen von Kennzahlen, selbst erstellte MSS-Analysen nebst Beschreibung oder Erläuterungen zu Berichten sind Beispiele hierfür. Diese Möglichkeit wird den Benutzern bisher in der Regel verweigert. Die Einspeisung der Daten erfolgt ausschließlich über die zentrale DV-Abteilung. Ein solches Konzept kann allerdings nur mit genauen Reglementierungen und Integritätsmechanismen erfolgen.

---

[59] „Here is some advice from a book on data warehouses: '[Resist] normalization … Efforts to normalize any of the tables in a dimensional database solely in order to save disk space [**sic**] are a waste of time … The dimension table must not be normalized … Normalized dimension tables destroy the ability to browse.' [Kimb96]" [Date00, S. 714]

Auch der Derived Data Layer wird bisher fast ausschließlich relational realisiert. Dadurch ist die Nutzung etwa erweiterter objektorientierter MSS-Applikationen nur sehr eingeschränkt möglich (siehe Kapitel 4). Tatsächlich finden vorwiegend relationale MSS-Applikationen Verwendung, deren Software-Qualität in den meisten Fällen als mangelhaft gewertet werden muss.

2.5.4 Data-Warehouse-Engineering

Die Kritik an den bisherigen Data-Warehouse-Ansätzen trifft aber nicht nur die einzelnen Schichten des Data Warehouse bzw. dessen Konzeption. Auch die über die Funktionalität hinausgehende Software-Qualität des Data Warehouse und die Qualität des Entwicklungs-Prozesses weisen regelmäßig Mängel auf.

Die bei der Entwicklung sonstiger Software üblichen ingenieurs-wissenschaftlichen Prinzipien werden vernachlässigt. Vorgehensmodelle sind bislang unzureichend entwickelt, spezielle CASE-Tools bis jetzt rar und ETL-Tools noch unausgereift. Die Integration von CASE- und ETL-Tools in das Data Warehouse und speziell die Metadatenbasis sind praktisch nicht existent.

Als Folge sind Data-Warehouse-Projekte ineffizient, werden vielfach erfolglos abgebrochen [Cw98, S. 16] und das erzeugte System leidet unter mangelnder Zuverlässigkeit, Robustheit und Änderbarkeit.

Ein Grund ist auch die Verwendung konventioneller Technologien. Die Fortschritte im Bereich des Systems-Engineering werden nicht genutzt (siehe Abschnitt 3.2.1) und damit die bisherigen Mängel wie z. B. der Impedance Mismatch oder der Strukturbruch in der Entwicklung (Analyse, Design, Implementierung) [CoYo96, S. 47 ff., CNM97, S. 1 ff.] in Kauf genommen. Teilweise gibt es sogar Rückschritte zu verzeichnen. Es steigt nicht nur die Gefahr von Integritätsverletzungen, indem die Erzeugung erheblicher Datenredundanz propagiert wird, zusätzlich wird sogar der Verzicht auf Constraints empfohlen. Ausgerechnet bei einem unternehmenskritischen System wie dem Data Warehouse die Daten- und Software-Qualität zu vernachlässigen, kann zu erheblichen negativen Auswirkungen führen.

Die Beschränkung der Data-Warehouse-Konzeption einzig auf relationale Technologien [DeMu88, S. 60 ff.] ist zu eng und kann nicht gerechtfertigt werden. Auch der Hinweis objektorientierte Technologien seien ungeeignet, da sie die Integration von Methoden und Daten anstrebten, Methoden aber keine Rolle spielten, weil der Schwerpunkt des Data Warehouse auf den Daten liege [Devl97, S. 404], ist nicht haltbar. Methoden spielen durchaus eine zentrale Rolle im Data-Warehouse-Konzept. Außerdem hat *Devlin* damit den Strukturaspekt des objektorientierten Modells übersehen, der inzwischen auch auf RDBMS und sogar spe-

ziell für die Data-Warehouse-Implementierung übertragen wird. [Burl97, S. 20 ff.]

Gerade in Anbetracht sich zukünftig verstärkender Tendenzen, die Data-Warehouse-Quellsysteme objektorientiert zu entwickeln [Tayl92, S. 19 ff.], muss eine Beschränkung des Data-Warehouse-Konzepts auf RDBMS infrage gestellt werden. Hierbei ist an Web-Technologie, an Multimediadatenbanken und insbesondere auch an operative Systeme zu denken. Reichhaltige Objekte in atomare Daten bzw. semantikarme Datenstrukturen aufzuspalten, um daraus wiederum Informationsobjekte zu generieren, ist unvernünftig.

Die Verwendung relationaler Technologie zur Verwaltung von Metadaten ist in besonderem Maße ungeeignet. Ein Großteil der Metadaten kann nur schlecht in die starren Strukturen des Relationenmodells eingefügt werden. Die mangelnde semantische Ausdruckskraft des Relationenschemas erschwert die notwendige Abbildung der Metadaten und vielfältigen Verknüpfungen untereinander erheblich.

KAPITEL 3

OBJEKT-PARADIGMA

Ziel dieses Kapitels ist, Klarheit über das Objekt-Paradigma (objektorientiertes Paradigma, **Objektorientierung**) zu erlangen, damit die Potenziale dieses Paradigmas für objektorientierte MSS (OMSS) und speziell das Object Warehouse — dessen Anforderungen in Kapitel 2 erarbeitet worden sind — beurteilt werden können.

In den letzten Jahren sind viele Artikel und Bücher zum Thema Objektorientierung erschienen, ein Großteil der erhältlichen Applikationen mit dem „Qualitätssiegel“ objektorientiert versehen worden und auch eine Ausweitung des Objekt-Paradigmas auf andere Bereiche wurde versucht.

Auf die typische Frage eines Laien, was denn unter der „Objektorientierung“ zu verstehen sei, können viele Antworten gefunden werden, doch nur selten eine für den Fragesteller zufriedenstellende. Das Objekt-Paradigma ist komplexer Natur. Es entstammt unterschiedlichen Gebieten und verfolgt dementsprechend heterogene Zielsetzungen. Außerdem ergeben sich auch innerhalb der einzelnen Gebiete konkurrierende Intentionen, so dass sich viele nicht-komplementäre Ziel-Interdependenzen innerhalb und zwischen den einzelnen Gebieten ergeben.

Es kann also bereits im Vorhinein erkannt werden, dass es nicht „das Objekt-Paradigma“ gibt. Dennoch lässt sich zum einen ein gemeinsamer Kern erkennen, zum anderen muss versucht werden die Zielsetzungen der jeweiligen Gebiete abzuwägen und je nach Problemstellung bestmöglich auszugestalten.

Koshafian und *Abnous* fassen die Ziele des Objekt-Paradigmas wie folgt zusammen [KhAb95, S. 3]:

- „Modeling the real world as close to a user's perspective as possible.
- Interacting easily with a computational environment, using familiar metaphors.

- Constructing reusable software components and easily extensible libraries of software modules.
- Easily modifying and extending implementations of components without having to recode everything from scratch.“

Um die notwendige Klarheit über das Objekt-Paradigma zu gewinnen, wird zunächst die Frage beantwortet, was ein Paradigma überhaupt ausmacht. Dazu werden die vom Objekt-Paradigma betroffenen Gebiete sequenziell betrachtet und es wird versucht, deren diesbezügliche Hauptintention zu identifizieren, wie auch deren Wechselwirkung mit den anderen Gebieten aufzuzeigen.

Nach der Klärung der inhaltlichen Bedeutung und Intention des Begriffs Paradigma erfolgt dessen Eindordnung in das Systems Engineering mit weiteren notwendigen begrifflichen Klärungen. Dem schließen sich die Analysen der unterschiedlichen vom Objekt-Paradigma betroffenen Gebiete an, die in die Kategorien Entwicklungs-Phasen, Applikations-Schichten und weitere Objekt-Paradigmata eingeteilt werden.

3.1 Begriff des Software-Paradigma

Paradigmata sind Denkmuster die das wissenschaftliche Weltbild, die Weltsicht einer Zeit prägen. [Balz96, S. 40] Paradigma ist ein

> „von *Th. Kuhn* in die Wissenschaftstheorie eingeführter Begriff für die eine Wissenschaft in einem bestimmten Zeitraum prägenden allgemein akzeptierten Auffassungen. Hierzu zählen sowohl methodologische Konzepte als auch intuitive Grundeinstellungen zu Phänomenen.“ [Gabl97]

Kuhn versteht unter einem Paradigma

> „eine wissenschaftliche Gemeinschaft, d. h. Wissenschaftler, die miteinander kommunizieren und Gemeinsamkeiten haben, die für die wissenschaftliche Zusammenarbeit wichtig sind. Sie beschäftigen sich mit ähnlichen Problemen und haben gleichartige Auffassungen über die Vorgehensweise bei Problemlösungen; sie akzeptieren übereinstimmend bestimmte Theorien, Methoden und Forschungsergebnisse, nehmen gleiche Standpunkte zu wissenschaftstheoretischen Standpunkten ein, verwenden die zentralen Fachbegriffe in gleicher Weise und arbeiten mit Lehrbüchern, deren Inhalt zumindest sehr ähnlich ist. Paradigmata erlangen einen Status, weil sie bei der Lösung einiger Probleme, welche ein Kreis von Fachleuten als brennend erkannt hat, erfolgreicher sind als die mit ihnen konkurrierenden. Einzelne Misserfolge stellen aber nur einzelne Theorien oder Problemlösungen in Frage, aber nicht das Paradigma.“ [zitiert nach Behr91, S. 4766 f.]

Ein Software-Paradigma besteht aus speziellen Methoden, Werkzeugen und Prinzipien. Das grundlegende Prinzip dabei ist eine spezielle Sichtweise auf die betrachteten Dinge — die „Brille" des Betrachters. Beispielsweise erzeugt die Systemtheorie die Sicht auf die Diskurswelt als System. [Popp95, S. 227 ff.] Demzufolge modelliert die Systemanalyse ein Unternehmen als System mit Elementen und Beziehungen. [Kral94, S. 5 ff.] Innerhalb dieser Systemsicht gibt es wiederum verschiedene Sichtweisen, wie Prozess-Sichten, Sichten auf die Aufbauorganisation oder Algorithmensichten.

Entscheidend sind also auch bei einem Software-Paradigma die Sichtweisen und die verwendeten Metaphern, die dadurch den gesamten Problemlösungsprozess prägen. *Yourdon* gibt ein Beispiel:

> „Es gibt einen einfachen aber entscheidenden Unterschied zwischen der OO-Gemeinde[60] und der SA/SD-Gemeinde auf diesem Gebiet: die OO-Gemeinde nimmt an, dass ein Muster vorhanden ist, und die Aufgabe besteht lediglich darin, dass man es findet. Wenn die SA/SD-Gemeinde dagegen einem Problem gegenübersteht, das nicht in ein altbekanntes Grundmuster passt, dann fängt sie von vorne an und versucht vollständig neue Lösungen zu erfinden, die in ein Muster passen." [Your93, S. 103]

Yourdon zeigt auch, dass es sogar zu kulturellen Unterschieden kommt. [Your93, S. 101 ff.]

Booch differenziert fünf Programmier-Paradigmen [Booc96, S. 59]:

- prozedurorientiert (Algorithmen)
- objektorientiert (Klassen und Objekte)
- logikorientiert (Ziele, oft ausgedrückt in eimem Prädikatenkalkül)
- regelorientiert (If-Then-Regeln)
- bedingungsorientiert (Invarianten-Beziehungen)

Balzert grenzt drei Paradigmen der Software-Technik gegen einander ab [Balz96, S. 40 f.]:

- das strukturierte Paradigma
- das objektorientierte Paradigma
- das wissensbasierte Paradigma

Bei der Informationssystem-Entwicklung wird in der Regel zwischen drei wesentlichen Paradigmen unterschieden [Your93, S. 96 ff.]:

- strukturierte Techniken (SA/SD, DFD, Funktionen/Algorithmen)

---

[60] Teilweise wird sogar von „Gemeinden" (im englischen Sprachraum von communities) gesprochen. Die Assoziation zur „Glaubensgemeinde" liegt nahe und dürfte in vielen Fällen treffend sein.

- Information Engineering (ERM, RDBMS, Daten)
- Objektorientierung

Welche Methoden und Prinzipien zum Objekt-Paradigma gehören ist jedoch unklar — teilweise sogar hart umstritten. Mithin ist es nicht möglich, „das objektorientierte Paradigma“ zu definieren. Dies hat verschiedene Gründe.

Das Objekt-Paradigma entstand aus unterschiedlichen wissenschaftlichen Disziplinen wie dem Software Engineering, der Programmierung, der Datenbanktheorie, der Forschung zur Human Computer Interaction (HCI) und der Künstlichen-Intelligenz-Forschung. *Booch* sieht auch einen Einfluss durch Philosophie und Kognitionswissenschaft. [Booc96, S. 55 ff.] Doch auch Bereiche wie die Organisationstheorie werden durch das Objekt-Paradigma betroffen und dementsprechend gestaltet. [KGZ94, S. 137 ff.] In der Folge sind die jeweiligen Zielsetzungen des Paradigmas sehr unterschiedlich und oft nicht komplementär. Dementsprechend verhält es sich mit der Ausgestaltung der jeweiligen objektorientierten Prinzipien.

Einerseits führt diese Erkenntnis dazu, dass die einzelnen objektorientierten Prinzipien je nach Gebiet unabhängig betrachtet werden sollten, andererseits bestehen aber durchaus gemeinsame Prinzipien, wie eine gemeinsame Sprache und Sicht der Dinge. Außerdem sind all diese Gebiete untereinander verknüpft. Die Künstliche Intelligenz ist ein Software-Bestandteil. Die Benutzerschnittstelle, die Applikation und die Datenbank sind Schichten einer Anwendung. Ein Informationssystem ist als sozio-technisches System eng mit der Organisation verzahnt. [KGZ94, S. 137 ff.]

> „*Yonezawa* und *Tokoro* stellen fest: Der Begriff Objekt entstand in den frühen 70er Jahren fast gleichzeitig und nahezu unabhängig von einander in verschiedenen Bereichen der Informatik, um Ideen zu beschreiben, die sich zwar verschieden darstellen, irgendwie jedoch in Zusammenhang stehen.“ [zitiert nach Booc96, S. 54]

Um darüber Klarheit zu erlangen, müssen verschiedene Ebenen differenziert werden (siehe die Abschnitte 3.3 bis 3.5). Bevor dies jedoch geschieht, muss die Frage beantwortet werden, welcher Zweck mit einem Software-Paradigma überhaupt verfolgt wird. Hierzu erfolgt die Einordung bezüglich des Systems Engineering (siehe Abschnitt 3.2).

3.2 Software-Paradigma-Zielsetzung

Nach einer Klärung wichtiger Begriffe im Bereich des Systems Engineering und einer entsprechender Einordnung dieser Arbeit, erfolgt eine Diskussion über die Qualität von Software-Paradigmata. Anschließend werden vertiefend mögliche Qualitätskriterien zur Beurteilung von Software angesprochen.

3.2.1 Systems Engineering

Systems Engineering wird hier als Begrifflichkeit gewählt, um die Intentionen von Software Engineering, Software-Technik, Systemanalyse, Systementwicklung und Software-Entwicklung zusammenzufassen. Der Zusammenhang zwischen diesen Termini wird sowohl begrifflich wie auch inhaltlich unterschiedlich beurteilt.

Während Systemanalyse gelegentlich zur Bezeichnung der frühen Prozessphasen (von der Istanalyse bis zum Sollkonzept) und Systementwicklung für die späteren Phasen (von der Spezifikation bis zu Wartung und Pflege) des Gesamtprozesses verwendet werden, so werden beide doch für gewöhnlich synonym gebraucht. [Gabl97] Problembereich und Zielsetzungen beider Begriffe sind identisch.

Systemanalyse und -entwicklung beinhalten die frühen Phasen der Istanalyse des Unternehmens und der Sollkonzeption eines neuen Systems, die mittleren Phasen der Realisierung mit eventueller Auswahl von Standardsoftware, dem Design, der Implementierung und dem Test der Software beziehungsweise der Anpassungen der Standardsoftware und der Implementierung der veränderten Organisation sowie den späten Phasen mit Einführung, Wartung und Pflege.

Der Begriff Systementwicklung kam später auf und wird insofern vom *Gabler-Wirtschaftslexikon* als der „korrektere“ Begriff angesehen, da sich die Systemanalyse nicht nur auf analytische Tätigkeiten beschränkt. [Gabl97]

Betrachtungsgegenstand von Systemanalyse und -entwicklung sind in der Regel Unternehmen oder Verwaltungen, die als sozio-technische Systeme aufgefasst werden. Als relevante Systembestandteile werden deshalb insbesondere Software, Hardware, Personal und die Organisation angesehen. [Gabl97]

Software-Technik wird als deutsche Übersetzung von Software-Engineering verwendet. [Balz96, S. 35 f.] Mit Software Engineering wird versucht, eine ingenieurs-wissenschaftliche Vorgehensweise für den gesamten Software-Erstellungs-Prozess zu erreichen. Damit wird eine systematische, Prinzipien folgende Vorgehensweise, im Gegensatz zu einer intuitiv-chaotischen Vorgehensweise wie sie etwa von manchen Künstlern bevorzugt wird, angestrebt. [Dude93, S. 655 ff.] Ziel ist die Verbesserung des Entwicklungsprozesses und des resultierenden Produkts.

Die Teilgebiete des Software Engineering sind [Balz96, S. 35 ff.]:

- Software-Entwicklung
- Software-Management
- Software-Qualitätssicherung
- Software-Wartung und -Pflege

Während die Systemanalyse aus der Wirtschaftsinformatik heraus entstanden ist, hat sich das Software Engineering unabhängig von den Erkenntnissen der Systemanalyse aus der Angewandten Informatik als Konsequenz der „Softwarekrise“ entwickelt. [Kurb97] Mithin gibt es weitgehende Überschneidungen beider Disziplinen. *Kurbel* meint:

> „Im Gegensatz zu der umfassenden Sicht der Systemanalyse, beschränkt sich das Software-Engineering jedoch nur auf einen Teilaspekt, nämlich auf die mit der Software verbundenen Fragen.“[61] [Kurb97]

Deutlich wird der unterschiedliche Fokus anhand der Bezeichnungen „Software“ oder „System“. So bezeichnen *Gladis* und *Pretzsch* die Frage als wesentlich, ob CASE für Computer Aided Software Engineering oder für Computer Aided Systems Engineering steht. Ersteres ist für den komplexen System-Entwicklungs-Prozess eine zu enge Sichtweise. Die Beschreibung auch nichtautomatisierter Aufgaben eines personellen Aufgabenträgers und organisatorische Aspekte dürfen nicht außer acht gelassen werden. Im Vordergrund steht das Mensch-Maschine-System des Unternehmens. Die Software ist kein Selbtzweck und muss in die Strukturen des Unternehmens eingebunden und von den Anwendern akzeptiert werden. [GlPr93, S. 204]

Demzufolge ist der Begriff des Systems Engineering der allesumfassende und findet hier in dieser Form Verwendung.

3.2.2 Software-Paradigma-Qualität

Nach der Klärung was ein Software-Paradigma ist und welche wesentlichen Ausprägungen es bisher gibt, muss die Frage beantwortet werden, wann ein Software-Paradigma als ein erfolgreiches Paradigma gewertet werden kann. Zwar ist der *kuhnschen* Wissenschaftstheorie inhärent, dass der Vergleich der einzelnen Paradigmata untereinander nur schwer zu bewerkstelligen ist, doch muss es zumindest möglich sein, sich der Zielsetzung eines Software-Paradigmas soweit bewußt zu werden, dass Erkenntnisfortschritte beurteilt und somit überhaupt erbracht werden können.

Ein Software-Paradigma ist dann erfolgreich, wenn es kausal zur Qualitätssteigerung des Systems Engineering beiträgt.[62] Das bedeutet, dass durch den Einsatz des Paradigmas erstens das betrachtete System (die Diskurswelt, z. B. ein

---

[61] Fachvertreter des Software Engineering werden dies oftmals anders beurteilen. *Balzert* beispielsweise beschreibt Systemanalyse als Teilgebiet des Software Engineering und meint über die Systemanalyse: „Die Ergebnisse müssen sich in einer Produkt-Definition wiederspiegeln.“ [Balz96, S. 96]

[62] Konsequenterweise müsste insofern auch anstelle von Software-Paradigma der Terminus Systems-Engineering-Paradigma verwendet werden. Die weitere Verwendung von Software-Paradigma wird der Lesbarkeit und Verständlichkeit des Textes allerdings einen besseren Dienst erweisen und findet deshalb Verwendung.

Unternehmen) verbessert wird und zweitens der System-Entwicklungs-Prozess verbessert wird. Das grundsätzliches Ziel von Software Engineering sieht *Meyer* darin, zur Produktion von Qualitätssoftware beizutragen. [Meye90, S. 2]

Die Qualität des betrachteten Systems hängt in der Regel hauptsächlich von der Qualität der Software und der Organisation ab sowie deren Zusammenspiel im Gesamtsystem, d. h. wie gut Software und Organisation zusammenpassen. Die Qualität des System-Entwicklungs-Prozesses hängt hauptsächlich davon ab, ob unter sonst gleichen Bedingungen weniger Kosten entstehen, der Prozess schneller geht, d. h. weniger Zeit erfordert und zu einem besseren Produkt führt.

Hierbei wird bereits deutlich, dass der System-Entwicklungs-Prozess erheblichen Einfluss auch auf die Qualität des Produkts hat. Da wiederum das Software-Paradigma ganz erheblichen Einfluss auf den System-Entwicklungs-Prozess hat, wird auch die Bedeutung dieses mittelbaren Einflusses des Paradigmas auf das Produkt ersichtlich. Umgekehrt wirkt sich aber auch die Qualität des Produkts direkt auf den Prozess aus. Je besser das Produkt ist, desto weniger Wartung wird erforderlich und desto einfacher sind Änderungen zu bewerkstelligen.

Der Fokus dieser Arbeit liegt auf dem Software-Aspekt. Die Interdependenzen mit den anderen Systemfaktoren fallen nicht in den primären Betrachtungsbereich. Organisatorische Aspekte werden folglich nur am Rande betrachtet bzw. finden dort Erwähnung, wo ein unmittelbarer Bezug zur Software besteht (siehe Abschnitt 3.5.2). Gleichwohl wird versucht, das Software-Paradigma immer im Kontext des Gesamtsystems zu betrachten.

Folglich wird im nächsten Abschnitt näher auf mögliche Kriterien für Software-Qualität eingegangen (siehe Abschnitt 3.2.3). Kriterien zur Beurteilung der Qualität von System-Entwicklungs-Prozessen werden nicht weiter vertiefend behandelt. Allerdings erfolgt die Betrachtung der Auswirkungen des Software-Paradigmas auf Vorgehens-Methodiken (siehe Abschnitt 3.3.5).

3.2.3 Software-Qualität

> „Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse erfüllen." [zitiert nach Balz98, S. 257][63]

Jacobson, *Christerson*, *Jonsson* und *Övergaard* unterteilen in interne und externe Sichtweisen auf die Software, um beurteilen zu können, ob und inwieweit es sich dabei um gute Software handelt. [JCJÖ95, S. 69]

---

[63] nach DIN ISO 9126

Die externe Sichtweise erfolgt durch alle, die das System in irgendeiner Weise nutzen (Endbenutzer, Administratoren [Heue97, S. 160 f.] und IT-Manager [JCJÖ95, S. 69]) und betrifft Anforderungen wie das schnelle Liefern von korrekten Ergebnissen, Zuverlässigkeit, leichte Erlern- und Nutzbarkeit, Effektivität et cetera. [JCJÖ95, S. 69]

Die interne Sichtweise erfolgt durch die Software-Entwickler und diejenigen, die die Software warten müssen (EDV-Profis [Meye90, S. 2]). Sie haben Anforderungen wie eine leichte Änder- und Erweiterbarkeit, leichte Verständlichkeit, das Vorhandensein von wiederverwendbaren Bausteinen, leichte Testbarkeit, Kompatibilität mit anderen Systemen, Portabilität sowie mächtige und einfache Entwicklungsmöglichkeiten [JCJÖ95, S. 69], z. B. durch entsprechende Entwicklungsumgebungen, wie etwa VisualWorks [Howa95, S. 45 ff., PaDi95a, S. 101 ff.] sie bietet.

Auch *Meyer* und *Heuer* unterteilen die Software-Qualitäts-Kriterien in analoger Weise, d. h. in äußere und innere Faktoren [Meye90, S. 2] bzw. Benutzer-Kriterien und Programmierer-Kriterien [Heue97, S. 161]. *Heuer* hebt bei letzteren die Modularisierbarkeit von Programmen und Strukturiertheit von Methoden hervor [Heue97, S. 161]; *Meyer* zusätzlich die Lesbarkeit des Codes [Meye90, S. 2]. *Taylor* sieht einen Schwerpunkt auf den Kriterien Wiederverwendbarkeit und flexibler Änder- bzw. Erweiterbarkeit. [Tayl92, S. 246 ff.]

Meyer betont außerdem, dass die inneren Faktoren einzig Mittel zum Zweck der Realisierung der äußeren Faktoren sind, dafür allerdings wesentlich bestimmend. [Meye90, S. 2]

Die wichtigsten äußeren Qualitätsfaktoren sind für *Meyer* Korrektheit, Robustheit, Erweiterbarkeit, Wiederverwendbarkeit, Kompatibilität (insbesondere durch Standards), Effizienz bei der Hardware-Nutzung, Portabilität, Verifizierbarkeit, Integrität und Benutzerfreundlichkeit. [Meye90, S. 3 ff.]

Balzert wählt einen anderen Ansatz, da bisher keine Einigkeit über den Qualitätsbegriff an sich besteht. Er unterteilt in fünf Ansätze, die unterschiedliche betriebliche Sichtweisen auf das Produkt wiederspiegeln [Balz98, S. 256]:

1. transzendenter Ansatz
2. produktbezogener Ansatz (Entwicklung)
3. benutzerbezogener Ansatz (Marketing/Vertrieb)
4. prozessbezogener Ansatz (Fertigung)
5. kosten-/nutzen-bezogener Ansatz (Finanzen)

Der Schwerpunkt dieser Arbeit liegt auf dem produktbezogenen Ansatz und kommt damit den oben beschriebenen Ansätzen nahe. Zumindest die Intentionen der anderen Ansätze sollten jedoch nicht außer Acht gelassen werden, denn was

bei den oben beschriebenen Ansätzen fehlt, sind beispielsweise die ökonomischen Restriktionen, die gewöhnlich einen ganz entscheidenden Einfluss auf die produktbezogene Qualität haben [Your93, S. 167 f.].[64]

Dunn betont, dass Qualität insbesondere daran gemessen werden muss, inwieweit das Produkt „verkaufbar" ist. Funktionen, die nicht dem Anforderungsprofil entsprechen, werden auch nicht bezahlt. [Dunn93, S. 4 ff.]

Ein gut geeigneter Bewertungskatalog für produktbezogene Software-Qualität ist beispielsweise aus der Norm DIN ISO 9126 zu entnehmen. Die sechs Bewertungskriterien werden im Anhang der Norm durch Teilmerkmale ergänzt (die in Klammern aufgeführt sind) [Balz98, S. 258 ff.]:

- **Funktionalität** (Richtigkeit, Angemessenheit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit)
- **Zuverlässigkeit** (Reife, Fehlertoleranz, Wiederherstellbarkeit)
- **Benutzbarkeit** (Verständlichkeit, Erlernbarkeit, Bedienbarkeit)
- **Effizienz** (Zeitverhalten, Verbrauchsverhalten)
- **Änderbarkeit** (Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit)
- **Übertragbarkeit** (Anpaßbarkeit, Installierbarkeit, Konformität, Austauschbarkeit)

Diese Kriterien werden als Leitfaden für die Bewertung der Object-Warehouse-Konzeption dienen.

3.3 Entwicklungs-Phasen

Im Folgenden werden die einzelnen Entwicklungsphasen OOA, OOD und OOP betrachtet. Zur Diskussion wird auch diese Reihenfolge des Entwicklungs-Prozesses verwendet, da die OOA das grundlegende Objektmodell des Software-Entwicklungs-Prozesses liefert und die Modellierung des Systems die Modellierungs-Prinzipien des Objekt-Paradigmas in seiner reinsten Form verwirklicht, d. h. die grundlegenden Metaphern des Objekt-Paradigmas am konsequentesten umsetzt.

Innerhalb des OOD-Kapitels werden besonders wichtige Design-Prinzipien des Objekt-Paradigmas erläutert und in Bezug zur OOA gesetzt. Im Abschnitt der OOP wird dann die Entwicklung von den prozeduralen zu den objektorientierten Programmiersprachen aufgezeigt und die implementierungs-technischen Abweichungen bzw. Zielkonflikte zu OOA und OOD angeschnitten.

---

[64] Nach Aussagen von *Krallmann* sind Software-Projekte durch Kosten- und Zeitbudgets beschränkt.

Zunächst erfolgt allerdings die Einordnung von OOA, OOD und OOP in einen Bezugsrahmen, so dass auch Klarheit über die wichtigsten Begriffe besteht. Abschließend wird auf die objektorientierten Vorgehens-Methodiken eingegangen, da auch diese ganz erheblich von den Vorgehens-Methodiken der strukturierten Techniken abweichen.

3.3.1 Bezugsrahmen

Bei der objektorientierten Systementwicklung erfolgt die Einteilung üblicherweise nicht nach den oben beschriebenen (siehe Abschnitt 3.2.1) frühen, mittleren und späten Phasen der Systementwicklung, sondern wird in objektorientierte Analyse (OOA), objektorientiertes Design (OOD) und objektorientierte Programmierung (OOP) unterschieden. [CoYo96, S. 17 ff., CoYo94, S. 15 ff., CoNi94, S. 25 ff.]

Hierbei führt eine ungenaue Begriffsabgrenzung immer wieder zu Verwechslungen und in dieser Konsequenz zu mangelhaften Systemen. Die objektorientierte Analyse wird oftmals mit einer objektorientierten Systemanalyse verwechselt. Tatsächlich ist die OOA jedoch nur ein Teilbereich des Systemanalyse-Prozesses, selbst wenn diese Systemanalyse dem objektorientierten Paradigma folgt. Auch die OOA und die frühen Phasen der Systemanalyse sind nicht identisch. Die OOA konzentriert sich ausschließlich auf die Ermittlung der Anforderungen für die zu erstellende Software. [CoYo96, S. 73 ff., CNM97, S. 1 ff., Booc96, S. 56 ff., Balz95, S. 13 ff.]

Damit beschränkt sich die OOA auf die Sollkonzeption eines objektorientierten Informationssystems, d. h. einzig auf den Software-Aspekt des Entwurfs. OOA, OOD und OOP sind deshalb als die frühen, mittleren und späten Phasen der reinen Software-Entwicklung zu verstehen (siehe allerdings Abschnitt 3.3.5).

Die objektorientierte Systemanalyse ist bisher nur selten Gegenstand der Diskussion. Die diesbezügliche Literatur (z. B. *Heide Balzerts* Buch „Methoden der objektorientierten Systemanalyse“ [Balz95, S. 13 ff.]) beschränkt sich in der Regel auf die Betrachtung von OOA. *Helmut Balzert* differenziert nicht zwischen Systemanalye und OOA, sondern zwischen OOA und objektorientierter Unternehmensmodellierung, womit ein wesentlicher Teil von Istanalyse und Sollkonzeption gemeint ist. Er meint: „Bei einem OOA-Modell wird ein Software-Produkt modelliert, bei einem Unternehmensmodell ein Unternehmen.“ [Balz98, S. 722] Es ist außerordentlich bemerkenswert, dass die Erkenntnisse der Systemanalyse beim Objekt-Paradigma bisher derart unterrepräsentiert sind.

Da sich der Kern dieser Arbeit mit der Software-Entwicklung beschäftigt, wird auch hier die Einteilung in OOA, OOD und OOP verwendet. Es ist aber essen-

ziell immer präsent zu behalten, dass Informationssysteme sinnvoll nur im Unternehmenskontext entwickelt werden können.[65]

Bemerkenswert ist auch die Reihenfolge des Entstehens von OOA, OOD und OOP, was die Kausalität der Entstehung des Objekt-Paradigmas verdeutlicht, d. h. die Richtung der Beeinflussung. Während die Anfänge der OOP bereits 1967 durch Simula verfügbar und mit Beginn der achtziger Jahre durch Smalltalk 80 vollständig kommerziell erhältlich waren, entstanden erst in dieser Folge das OOD und erst wesentlich später die OOA.[66] Dies zeigt, dass sich Analyse und Design an den gegebenen Möglichkeiten von Hard- und Software orientieren, wie dies auch in den Anfängen mit den strukturierten Methoden der Fall war. [Your-93, S. 71 f.]

Obwohl also die OOA eine Folge von OOD und OOP ist und OOD wiederum eine Folge der OOP, bildet diese Reihenfolge nicht die Grundlage für die folgende Diskussion. Stattdessen werden die einzelnen Intentionen der jeweiligen Objekt-Paradigmata in der Sequenz der Software-Entwicklung OOA, OOD und OOP abgearbeitet. Die Konzepte des Objekt-Paradigmas können auf diese Weise in strukturierterer Form dargestellt werden.

3.3.2 Objektorientierte Analyse

Nach Klärung des Gegenstands, der Zielsetzung und der Grundidee der objektorientierten Analyse (OOA), erfolgen vertiefte Diskussionen über die Grundkonzepte und Vorgehensweisen, die im Fazit zusammengefasst werden. Die Grundkonzepte bilden weitestgehend auch für die anderen Objekt-Paradigmen den Ausgangspunkt der Diskussion. Die Vorgehensweise der OOA ist bereits maßgeblich für den gesamten System-Entwicklungs-Prozess und deshalb von besonderer Bedeutung.

3.3.2.1 Gegenstand

Gegenstand der OOA ist die Ermittlung der Benutzeranforderungen für eine objektorientierte Software und das Erzeugen einer entsprechenden Spezifikation.

---

[65] Bei der Verwendung von Standardsoftware sind deshalb umfangreiche Anpassungen notwendig, die entweder an der Software, der Organisation oder sonstigen betroffenen Systemkomponenten des Unternehmens vorgenommen werden müssen.

[66] Nach *Balzert* entstand die OOA erst zehn Jahre nach OOP und OOD im Jahre 1991 durch *Coad* und *Yourdon*. [Balz96, S. 328] *Yourdon* verweist allerdings auf *Shlaer* und *Mellor*, die bereits 1988 OOA als Thema behandelten. [Your93, S. 92] *Yourdon* bemerkt auch, dass es von der Strukturierten Programmierung (SP) bis zur Strukturierten Analyse (SA) zehn Jahre gedauert hat [Your93, S. 92], was dann in etwa dem Zeitraum der Entwicklung beim Objekt-Paradigma entspricht.

Wie bei Istanalyse und Sollkonzeption der Systementwicklung ist damit auch bei der OOA die Modellierung des betrachteten Systems (Diskurswelt, Domäne) bzw. der zu erstellenden Software ein wesentlicher Teil.

Folglich meinen *Argila* und *Yourdon* korrekt: „Truly the core of any OOA is the process of what we call information modeling." [YoAr96, S. 7] Und auch *Balzert* meint entsprechend: „Ein wesentliches Ziel des Requirements Engineering[67] ist es, ein vollständiges, konsistentes und kohärentes Systemmodell zu erstellen." [Balz93b, S. 66]

3.3.2.2 Zielsetzung

Mithin verlangt die OOA die Erstellung von Beschreibungsmodellen, die anhand verschiedener Diagramme dokumentiert werden und auf diese Weise die Systembeschreibung bilden. Je nach Paradigma gibt es eine Reihe verschiedener Diagramme, d. h. Basistechniken, die das System aus einem ganz bestimmten Blickwinkel betrachten bzw. modellieren. Jedes einzelne Paradigma wiederum verwendet jeweils verschiedene Basistechniken bzw. Diagramme.

- Bei der strukturierten Methode stehen die funktionale Dekomposition, das Datenflußdiagramm (DFD) und Prozessbeschreibungen in algorithmischer Form (z. B. Pseudo-Code und Struktogramm/Nassi-Shneiderman-Diagramm) im Vordergrund.
- Bei der datenorientierten Methode (Information Engineering) bildet gewöhnlich eine Form des Entity-Relationship-Modells (ERM) das Zentrum des Interesses.
- Beim Objekt-Paradigma steht das Objekt im Mittelpunkt, dessen Struktur im Klassendiagramm und dessen Verhalten im Interaktionsdiagramm abgebildet wird.

Die Modellbildung zeigt, wie bedeutend der Einfluss der jeweiligen Sichtweise auf das betrachtete System für das Ergebnis des Modells ist (siehe Abschnitt 2.3.2). Das bedeutet, dass je nach Diagramm (DFD, ERM, Klassendiagramm) sehr unterschiedliche Modelle entstehen. Das Modell hängt also unmittelbar von den Konzepten des Paradigmas und der Unterstützung dieser Konzepte durch die Diagramm-Techniken ab — d. h. letztlich der Unterstützung dieser Diagramm-Techniken bei der Modellbildung.

---

[67] *Balzert* übersetzt Requirements Engineering mit Systemanalyse, meint dies aber im Sinne einer Software-Spezifikation (siehe Abschnitt 3.2.1). [Balz96, S. 92]

3.3.2.3 Grundidee

Der Grundgedanke des Objekt-Paradigmas ist, eine der Natur weitgehend entsprechende Abbildung zu finden. *Kay* wollte eine Modellierung der Diskurswelt, die der Sicht von Kindern auf das System möglichst nahe kommt [Kay93, S. 1 ff.] und folglich von Kindern leicht — und in der Folge von allen Beteiligten „kinderleicht“ — verstanden werden kann.[68]

Die Welt als Zusammenspiel von Objekten zu sehen entspricht diesem Gedanken. Es gibt viele Analogien in diesem Zusammenhang, wie Lego-Bausteine, Fraktale, Organismen, den PC mit seinen modularen Bauteilen (z. B. Motherboard, Karten und Schnittstellen) oder eben auch Unternehmen.

Diese Gedankenwelt in eine Programmiersprache zu übertragen, war *Kays* **grundlegende** Idee. Bei der Programmiersprache Simula von *Dahl* und *Nygaard* fand er bereits einige dieser Basiskonzepte realisiert. [Balz96, S. 164 f.] Wie der Name bereits verdeutlicht (Simula = **Simu**lation **La**nguage), ist das Ziel von Simula die Simulation von Systemen. Es ist also auch die grundlegende Idee von Simula, die Diskurswelt möglichst nahe abzubilden, um mithilfe dieses Modells die Realität zu simulieren und dann als Folge entsprechend aussagekräftige Rückschlüsse auf das Originalsystem zu erhalten.

Software kann mithin ebenso als Simulation eines realen Systems interpretiert werden — im Fall eines Informationssystems etwa als Simulation eines Unternehmenssystems. [John99] Diese Sichtweise ist für den Software-Entwicklungs-Prozess durchaus fruchtbar. Dieser Idee folgt in gleicher Weise das Modellierungs-Konzept der OOA. Auch steht einer identischen Übertragung bzw. Erweiterung dieses Konzepts auf die objektorientierte Systemanalyse nichts entgegen.

3.3.2.4 Grundkonzepte

Zentraler Betrachtungsgegenstand der OOA ist das Objekt. Objekte werden durch Attribute (Daten) beschrieben und enthalten Methoden (Funktionen). Die Objekte bilden „aktive autonome“ Einheiten, die in Beziehungen stehen und über „Nachrichten kommunizieren“.

Es gibt mithin zwei wesentliche Sichtweisen auf das OOA-Modell, eine statische, die die Strukturen durch Attribute und Beziehungen beschreibt und eine dynamische, die das Verhalten abbildet. Zur Abbildung der statischen, d. h. strukturellen Eigenschaften der OOA wird ein Klassen- bzw. Objekt-Diagramm erzeugt. Für die dynamischen Eigenschaften, d. h. die Nachrichtenflüsse zeichnen bei-

[68] Im Gegensatz dazu meint *Yourdon*, dass sich die bisherigen Strukturierten Techniken nicht an der Denkweise der Menschen, sondern an der Hardware orientieren. [Your93, S. 21 f.]

spielsweise *Coad* und *Yourdon* Nachrichtenverbindungen in das Klassen-Diagramm ein. [CoYo96, S. 194]

Es hat sich jedoch gezeigt, dass diese Form der Darstellung unzureichend ist.[69] Deshalb hat *Coad* seine Methode um einen Szenario-View ergänzt. [CNM97, S. 466 ff.] Diese Form des Szenario-Views entspricht prinzipiell dem Interaktions-Diagramm von *Jacobson*, *Christerson*, *Jonsson* und *Övergaard* [JCJÖ95, S. 215 ff.], welches als erstes solcher Diagramme bei der objektorientierten Systementwicklung angesehen wird [Balz96, S. 198 ff.].

Zunächst werden die statischen Elemente des Objekt-Paradigmas der OOA (Objekte, Attribute, Klassen- und Vererbungskonzept, Assoziations- und Aggregations-Beziehungen, Kardinalitäts- und Rollen-Semantik, Constraints bzw. Geschäftsregeln sowie Subsysteme) erläutert und anschließend durch die dynamischen Elemente (Methoden und Nachrichten) bzw. die eng damit verbundenen Konzepte (Polymorphismus sowie Kapselung) komplettiert.

Ergänzt wird die Sichtweise des Systems, als ein Zusammenspiel von Objekten, um das **Klassen- und Vererbungskonzept**. Dabei werden gleichartige Objekte in Klassen eingeteilt. Diese Analogie entstammt der *platonischen* Ideenlehre [Kay93, S. 3, Liu96, S. 30, Wegn87, S. 177], in der es eine ideale Form, die Klasse, und verschiedene Ausprägungen, also Exemplare (Objekte, Instanzen) gibt.

Garder wählt den Vergleich mit einer Backform für Pfefferkuchenmänner und den darin gebackenen Pfefferkuchenmännern, die sich, obwohl sie der selben Form entstammen, doch in einzelnen Ausprägungen/Details unterscheiden. [Gaar00, S. 106 f.]

Zwischen diesen Klassen gibt es, z. B. analog zur Tier- und Pflanzenwelt, Vererbungsbeziehungen. Vererbungsbeziehungen bilden eine Spezialisierungs-Hierarchie bzw. im umgekehrter Sichtweise eine Generalisierungs-Hierarchie ab. [Heue97, S. 187 ff.] Das grundlegende Kriterium dabei ist die „IS-KIND-OF“-Beziehung (IS-A, A-KIND-OF (AKO)). Beispielsweise ist der Mensch eine Art von Säugetier.

Es wird davon ausgegangen, dass es zwischen den in Vererbungsbeziehung stehenden Klassen der Objekte Gemeinsamkeiten von Strukturen, d. h. Daten/Attributen (z. B. Anzahl der Beine, ob Säuger oder maximales Gewicht) und gemeinsame Fähigkeiten, d. h. Methoden/Funktionen (z. B. kann schwimmen, ist Fleischfresser oder Ausprägung von Sprachfähigkeit) gibt.

Die Strukturen der Attribute und Funktionen sind mithin für alle Objekte einer Klasse identisch. Die Ausprägungen der Attributwerte der Objekte variieren.

---

[69] Andere Autoren wie beispielsweise *Booch* unterscheiden noch weitere Sichten bzw. Diagramme. [Booc96, S. 41 f.] Von dieser Betrachtung soll im Folgenden allerdings abgesehen werden, da ein weiterer Erkenntnisgewinn in diesem Zusammenhang nicht zu erwarten ist.

Dies ist eine Festlegung bzw. Definition der Grundprinzipien der Vererbung durch das der OOP und in der Folge auch mittelbar der OOD und OOA zugrundeliegende Smalltalk-80-System. [GoRo89, S. 55 ff.] Alternative Ausgestaltungen der Vererbungs-Analogie sind denkbar (siehe Abschnitt 3.3.4.5). [Wegn87, S. 168 ff.]

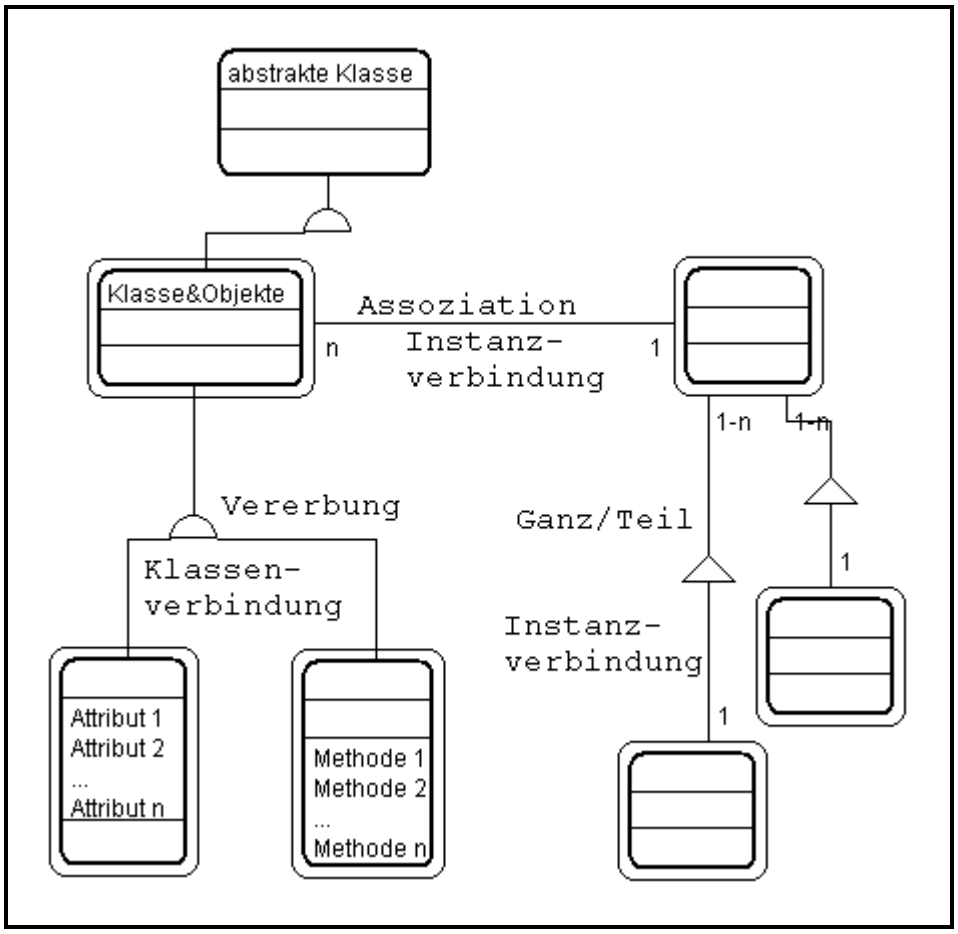


Abbildung 9: Coad/Yourdon-Notation im Klassendiagramm [CNM97, Einband]

Außer der Vererbungs-Beziehung zwischen Klassen gibt es **Assoziations- und Aggregations-Beziehungen** zwischen Objekten. Beispielsweise die Coad/Yourdon-Methode macht den Unterschied zwischen Klassen- und Instanz-Beziehungen deutlich (siehe Abbildung 9). Instanzverbindungen (Aggregation als Verbindung mit Dreieck-Symbol, Assoziation als direkte Verbindung ohne Symbol) gehen von Instanz (äußerer Rand des Klassen/Objekt-Symbols) zu Instanz, während die Klassenverbindungen (Vererbung als Verbindung mit Halbkreis-Symbol) von Klasse zu Klasse gehen (innerer Rand des Klassen/Objekt-Symbols).

Ohne die Verbindung zwischen Objekten kann deren Bezug nicht hergestellt werden. Beispielsweise muss ersichtlich sein, welche Rechnung zu einem Kunden bzw. zu welchem Kunden eine Rechnung gehört.

Obwohl die Assoziation teilweise als Spezialfall der Aggregation angesehen wird [Reen96, S. 159 ff.], bildet die Aggregation eine weitergehende Semantik ab. In vielen Fällen ist die Grenze zwischen Aggregation und Assoziation allerdings fließend. Demzufolge wird im Zweifelsfall nur das Konstrukt der Assoziation gewählt. [CoYo96, S. 121] Oftmals wird die Aggregation auch als Klassenbeziehung gesehen, da sie bei der Klassendefinition von Objekten angegeben wird, also kein eigenständiges (Teil-)Objekt wie bei der Assoziation entsteht, zu dem dann eine Beziehung erzeugt wird. [Heue97, S. 276 ff.]

Die Assoziation verbindet zwei Objekte in der Form einer „kennt"-Verbindung. [Vett95, S. 97 ff.] Beispielsweise „kennt" ein Mann seine Ehefrau. Bei der Aggregation handelt es sich um eine „Ganz/Teil"-Beziehung (Whole/Part-Structure). Beispielsweise ist ein Autoreifen Teil eines Autos. Der Reifen kann nicht gleichzeitig Teil eines anderen Autos sein.

Diese Beziehungsform ist von erweiterter Bedeutung. Die Existenz eines Ganzen (z. B. ein Team) kann von seinen Teilen (z. B. Team-Mitglieder) abhängen, wie auch die Existenz der Teile (z. B. Körperteile) vom Ganzen (z. B. einem Menschen) existenziell abhängig sein kann.

Die Objekt-Beziehungen in Form von Assoziation und Aggregation werden um die **Kardinalitäts-Semantik** bzw. Optionalität [Vett95, S. 97 ff.] und die **Rollen-Semantik** [Balz96, S. 333 ff.] ergänzt bzw. spezifiziert.

Die Kardinalitäts-Semantik gibt die Wertigkeit einer Instanzverbindung an, beispielsweise wie viele Fernsehsender eine Person besitzen darf oder sogar besitzen muss. Dies kann von null Fernsehsendern (Optionalitäts-Bedingung) bis zu unendlich vielen sein oder von null bis z. B. maximal zwei. Bei einem Menschen wird man genau zwei Augen voraussetzen und kann insofern für eine Applikation eines Augenarztes für jeden Patienten immer genau zwei Augenobjekte modellieren.

Die Rollen-Semantik gibt an, welche Rolle eine Beziehung zwischen zwei Objekten spielt. Etwa kann eine Person in der Beziehung eines Unternehmens ein Mitarbeiter sein, in der Beziehung zu einer anderen Person der Vater.

Auch **Constraints** bzw. **Geschäftsregeln** stellen einen essenziellen Teil des Informationssystems dar [RMK99] und müssen durch die OOA ermittelt und modelliert werden. Beispielsweise könnte die Bedingung für den Betrieb eines Fernsehsenders darin bestehen, mindestens achtzehn Jahre alt zu sein, keine Vorstrafen zu haben und die deutsche Staatsbürgerschaft zu besitzen.

Als weiteres Konstrukt teilen *Coad* und *Yourdon* das Objektmodell in **Subsysteme** ein („Subjects"), was im wesentlichen der Übersichtlichkeit und damit der Komplexitätsreduktion dient. Die Subsysteme können später eventuell auch als Module (im Sinne eines Segments) realisiert werden, d. h. als größere Einheiten

von Objekten, die möglichst unabhängig sind, z. B. durch eine möglichst schmale Schnittstelle. [CoYo96, S. 134 ff.]

Außer dieser statischen Betrachtung, d. h. der Struktur des Systems ist die Abbildung der System-Dynamik (Verhalten) von essenzieller Bedeutung.[70] Die Dynamik des Systems entsteht durch Interaktion der Objekte. Die Objekte kommunizieren, indem sie sich gegenseitig **Nachrichten** (Botschaften) senden. Versteht ein Objekt eine Nachricht, so führt es daraufhin eine seiner **Methoden** aus und sendet eine Antwort zurück an das Objekt, welches die Nachricht geschickt hatte.

Eng verbunden mit diesem Konzept ist die **Kapselung**. Weder auf die Attribute (oder etwa deren Werte), noch auf die Methoden eines Objekts kann von einem anderen Objekt aus zugegriffen werden. Das Innere eines Objekts ist mithin vor den anderen Objekten verborgen. Wenn ein Objekt einem anderen Objekt eine Nachricht sendet, wird es zwar ein Ergebnis bekommen, doch ist es für das sendende Objekt unerheblich wie das Empfängerobjekt zu diesem Ergebnis gekommen ist. [Knig96, S. 13 f.]

Booch nennt als Beispiel das Autofahren. [Booc96, S. 69 ff.] Man betätigt zwar Knöpfe oder Hebel (Nachricht senden) und weiss, wie das Auto reagieren wird (Antwort des Objekts), doch wie das Auto oder dessen Teile (Objekte) dies bewerkstelligen (Methoden und Attribute), ist für den Fahrer unerheblich.

Damit in Zusammenhang steht auch der **Polymorphismus**. Verschiedene Objekte können die selbe Nachricht empfangen bzw. verarbeiten, reagieren jedoch individuell verschieden darauf, d. h. führen jeweils verschiedene Methoden aus. Beispielsweise werden ein Hund(eobjekt) und ein Mensch(enobjekt) unterschiedliche Jahresangaben machen, wenn sie nach ihrem Alter befragt werden, auch wenn sie den selben Zeitraum damit bezeichnen.[71]

Dies ist eine weitere Unterstützung der Autonomie der Objekte. Ein Objekt kann auf diese Weise unterschiedlichen anderen Objekten jeweils die selbe Nachricht schicken, ohne dass sich das sendende Objekt darüber „Gedanken machen“ muss, wie das Empfänger-Objekt diese Nachricht verarbeitet.

---

[70] Dies trifft insbesondere zu, wenn bedacht wird, dass bei den strukturierten Methoden eben diese Dynamik im Mittelpunkt der Betrachtung steht. Gerade die Sichtweise auf Prozesse wird vielfach als genauso natürlich empfunden wie die Sichtweise des Objekt-Paradigmas. Deshalb wird diese Prozess-Sicht oftmals weiterhin als einzig nutzbringendes Paradigma angesehen [Sche98a, S. 1 ff., Sche98b, S. 10 ff.], was auch zu entsprechenden Konsequenzen in der Unternehmensgestaltung führt [HaCa96, S. 71 ff.]. Sogar *Balzert* glaubt, dass das grundlegende Konstrukt der objektorientierten Unternehmensmodellierung eine Prozess-Sicht und damit auch eine Untergliederung nach Prozessen sei. [Balz98, S. 721 ff.] Damit schlägt *Balzert* allerdings eine dem Objekt-Paradigma entgegenstehende Modellierung und Gestaltung des Systems vor.

[71] Hunde altern in den ersten Jahren ihres Lebens ca. sieben mal schneller als Menschen, später in etwa fünf mal schneller.

Beispielsweise braucht ein am Alter von anderen Objekten interessierter nicht nur nicht zu wissen, wie sich das jeweilige Alter des Hundes und des Menschen aus obigem Beispiel berechnen, er muss dabei nicht einmal wissen, ob es sich um einen Hund oder einen Menschen handelt. Er fragt die beiden lediglich: „Wie alt bist Du?“

Üblicherweise wird das Konzept des Polymorphismus auf Objekte beschränkt, die in einer Vererbungsbeziehung stehen. [Balz96, S. 167 f.] Diese Einschränkung des Konzepts ist aber unnötig (siehe auch Abschnitt 3.3.4).

3.3.2.5 Vorgehensweise

Damit verdeutlichen die OOA-Konzepte, dass die objektorientierte Analyse in zwei wesentliche Sichtweisen auf das System unterteilt wird, die statische Sichtweise der Strukturen und die dynamische Sichtweise des Verhaltens. Das substanzielle des Objekt-Paradigmas ist allerdings, dass Struktur und Verhalten simultan modelliert werden, d. h. dass die gemeinsam zu einem Objekt gehörigen Daten und Methoden auch zusammen in diesem Objekt gekapselt werden müssen.

Durch eine einseitige Sichtweise auf das System wird dies jedoch in der Regel verhindert werden. Dies ist deshalb so elementar, da die grundlegenden Modellierungs-Entscheidungen bereits bei der OOA fallen. Bei der OOA wird das Kern-Modell gebildet, welches durch das OOD erweitert und durch die OOP implementiert wird. *Argila* und *Yourdon* meinen:

> „In OOA, the difficult part of the problem is establishing what are the ‘things in the real world’ […]. These will establish the basic building blocks from which the system will be constructed.“ [YoAr96, S. 7]

Es ist jedoch nicht möglich, Struktur und Verhalten wirklich gleichzeitig, d. h. mit auch identischer Priorität zu modellieren. Eine der beiden Sichtweisen wird zumindest tendenziell zu einem stärker strukturierenden Effekt führen. Dementsprechend gibt es unterschiedliche Schwerpunkte bei der Vorgehensweise der OOA. *Balzert* unterscheidet drei prinzipielle Methoden [Balz96, S. 358]:

1. Orientierung an den Informationen (Daten) des Systems, z. B. *Shlaer* und *Mellor* [ShMe96, S. 1 ff., ShMe98, S. 11 ff.] oder *Rumbaugh et al.* [RBP+93, S. 27 ff.].
2. Orientierung an der Funktionalität (Verhalten) des Systems, z. B. *Wirfs-Brock*, *Wilkerson* und *Wiener* [WWW90, S. 61 ff.], *Rubin* und *Goldberg* [RuGo92, S. 48 ff.] oder *Jacobson*, *Christerson*, *Jonsson* und *Övergaard* [JCJÖ95, S. 69 ff.].
3. Synthese von 1 und 2, z. B. *Coad* und *Yourdon* (Schwerpunkt auf 1) [CoYo96, S. 74 ff.] oder *Booch* (Schwerpunkt auf 2) [Booc96, S. 215 ff.].

Die beiden extremen Vorgehensweisen der Datenorientierung und Funktionsorientierung führen jeweils zu zwei unterschiedlichen Effekten.

Erstens begünstigen beide Formen die Bildung von getrennten Daten- und Funktionsobjekten, indem anstelle einer objektorientierten Modellierung die konventionellen Techniken beibehalten werden. Dies führt im ersten Fall zu „Information Engineering“ und im zweiten Fall zu „funktionaler Dekomposition“.

Besteht beispielsweise beim Objekt-Paradigma auch Einigkeit darüber, dass die Strukturierung des Systems in Objekte erfolgt, die Daten und Funktionen gemeinsam kapseln und mithin keine Trennung nach Daten und Funktionen stattfindet, werden doch immer wieder Systeme als dem Objekt-Paradigma zugehörig zu klassifizieren versucht, die das System in Daten-Objekte und Funktions-Objekte untergliedern (siehe Abschnitt 3.3.3). [YoAr96, S. 29 ff.]

Zweitens wird eine datenorientierte Vorgehensweise tendenziell zu einer Strukturierung des Systems anhand der Datenstrukturen führen, eine funktions-orientierte Vorgehensweise zu einer Strukturierung anhand der Funktionen.

Dieser Effekt hängt ganz wesentlich von der Sichtweise auf das System ab. Bei einer datenorientierten Vorgehensweise mit einer „Daten-Brille“ müssen die Funktionen auf die nach den Daten strukturierten Objekte verteilt werden, bei einer funktions-orientierten Vorgehensweise mit einer „Funktions-Brille“ müssen die Daten auf die nach Funktionen strukturierten Objekte verteilt werden.

Der **erste** Effekt, bei dem weiterhin konventionelle Methoden verwendet werden, zeigt sich beispielsweise bei der Methode zur objektorientierten Unternehmensmodellierung von *Balzert*, bei welcher *Balzert* eine prozessorientierte Systemzerlegung vorschlägt. [Balz98, S. 721 ff.] Dies entspricht genau dem Vorgehen bei der strukturierten Methode. Infolgedessen ist es auch nicht verwunderlich, dass *Balzerts* Diagramm-Vorschlag weitgehend mit dem DFD der strukturierten Analyse identisch ist.

Shlaer und *Mellor* orientieren sich mit ihrer datenorientierten Vorgehensweise am klassischen Information Engineering und führen diese unverständlicherweise sogar stark angelehnt an die Modellierung von Relationenschemata durch. [ShMe96, S. 10 ff.] Darauf aufbauend ergänzen sie Prozessmodelle. [ShMe98, S. 6 ff.]

Auch *Yourdon* und *Argila* schlagen in ihrer 3-View-Modeling-Methode die Kombination von Entity-Relationship-Models, Data-Flow-Models und State-Transition-Models vor und insofern eine Kombination aus klassischer strukturierter Analyse und Information Engineering, wie bei der *hatley-pirbhaischen* Methode SA/RT [Balz93b, S. 71 ff.]. [YoAr96, S. 31 ff.]

Diese Vorgehensweise verhindert jedoch ein objektorientiertes Vorgehen. Geschäftsprozess-Sichten sind Untergliederungen nach Prozessen und damit auch nach Funktionen. Wird nach Prozessen gegliedert, können die Daten diesen Pro-

zess-Objekten lediglich „angehängt“ werden. Echte Datenmodellierung ist dann nicht mehr möglich und erzeugt genau die Inkompatibilitäts-Probleme, die entstehen, wenn das DFD-Team und das ERM-Team ihre Ergebnissse der Analyse zu verbinden versuchen [Your93, S. 25 f.].

Dies führt entweder zu hoher Daten-Redundanz, der Trennung in Daten-Objekte und Prozess-Objekte oder zu einem von der „natürlichen“ Modellierung weit entfernten System. Explizite Geschäftsprozess-Modellierung und das Objekt-Paradigma sind mithin zwei konkurrierende Paradigmen.

Der **zweite** Effekt, bei welchem die Objekte anhand von Funktionen oder anhand von Daten strukturiert werden, zeigt sich in einer ausgeprägten funktionsorientierten Form bei der CRC-Karten-Methode von *Beck* und *Cunningham*. Die Methode besteht aus drei wesentlichen Elementen [BeCu89, S. 52]:

1. **C**lasses
2. **R**esponsibilities
3. **C**ollaborators

Die Objekte, d. h. Klassen der Objekte werden als Abbildung natürlicher Elemente gebildet. Dann werden die Methoden der einzelnen Klassen und die zur Erledigung dieser Methoden notwendigen anderen Klassen ermittelt bzw. gebildet. Objekte können also Aufgaben erledigen und andere Objekte zur Erledigung von Aufgaben auffordern. [Beck91, S. 63 ff., Beck93c, S. 205 ff.]

Dieses Konzept geht damit von streng gekapselten Objekten aus. Es sind nur die nach außen sichtbaren Methoden eines Objekts bekannt, dessen Schnittstelle (Protokoll). Von den Implementierungen und Attributen der Objekte wird abstrahiert. Dies entspricht dem Konzept der abstrakten Datentypen (ADT) und zeigt wie implementierungsnah dieser Ansatz ist, statt wirklich den Versuch einer natürlichen Modellierung unternehmen zu wollen. [Beck94a, S. 219 ff., Beck94b, S. 239 ff.]

Wilkinson erweitert diesen Ansatz zu einer Methode für OOA und OOD. [Wilk95, S. 81 ff.] *Wirfs-Brock*, *Wilkerson* und *Wiener* verwenden die CRC-Karten-Methode als Grundlage ihrer OOA-Methode. [WWW90, S. 31 ff.] *Jacobson*, *Christerson*, *Jonsson* und *Övergaard* kommen mit ihrem Use-Case-Ansatz zu einer sehr ähnlichen Vorgehensweise. [JCJÖ95, S. 69 ff.] Und auch *Rubin* und *Goldberg* verwenden das Verhalten als primäres Gestaltungskriterium für das System. Attribute werden erst im zweiten Schritt ihrer Methode zu den Objekten hinzugefügt und auch dies erfolgt einzig nach dem Kriterium der Notwendigkeit für das Objektverhalten. [RuGo92, S. 48 ff.]

Durch diese Gliederung des Objektmodells anhand der Funktionalitäten der Objekte und in der Folge der Abstraktion von den Attributen, ist eine spätere Verteilung der Attribute auf die Objekte notwendig, die sich danach richtet, welche

Methode eines Objekts welche Attribute benötigt. Irgendwann müssen die Attribute schließlich doch modelliert werden und dies erst während der Implementierungsphase zu tun hat Folgen.

Insgesamt bedeutet dies dann konkret, dass beide Extremformen der OOA, sowohl die daten- als auch die funktions-orientierte, ungeeignet sind, um Objektmodelle nach den Zielsetzungen des Objekt-Paradigmas der OOA erzeugen zu können. Die Entscheidung muss allerdings zugunsten einer primären Vorgehensweise fallen, entweder der funktions-orientierten und damit auch zu einer Redundanz der Daten oder der datenorientierten und damit auch zu einer Redundanz der Funktionen. Daten **und** Funktionen in realen Systemen redundanzfrei zu bekommen, ist nicht möglich.

Dementsprechend meint *Meyer*:

> „Beim Grobentwurf einer Systemarchitektur ist der Softwareentwerfer vor eine grundsätzliche Wahl gestellt: Soll die Struktur auf den Aktionen[72] oder den Daten beruhen? In der Antwort auf diese Frage liegt der Unterschied zwischen herkömmlichen Entwurfsmethoden und dem objektorientierten Ansatz.
>
> Es gibt Gründe für hybride Ansätze, denn letztendlich spielen sowohl Daten als auch Aktionen ihre Rolle in der Struktur eines Programms. Die wichtigste Frage ist jedoch, was als erstes Kriterium zur Beschreibung der Zerlegungsstrukturen auf höchster Ebene dienen soll. Dieser Frage kann man sich nicht entziehen: Der Entwerfer muss sich entscheiden." [Meye90, S. 44]

Auch *Heuer* ist dieser Ansicht, indem er ausdrückt:

> „Zunächst einmal muss beim Software-Entwurf grundsätzlich entschieden werden, ob die Modularisierung anhand der Funktionen oder anhand der Daten oder Objekte vorgenommen werden soll. Die herkömmliche Sicht im Software-Entwurf ist die Modularisierung anhand von Funktionen. Die objektorientierte Technik ist dagegen die Modularisierung anhand der Daten oder Objekte." [Heue97, S. 169]

Da bei Informationssystemen, insbesondere bei Management Support Systems die Auswirkungen inkonsistenter Daten ceteris paribus erheblich kritischer sind als unterschiedlich implementierte Algorithmen,[73] muss die Entscheidung zu-

---

[72] '[…] die Begriffe 'Funktion', 'Aktion' und 'Prozess' [werden] austauschbar verwendet und ebenso 'Daten' und 'Objekte' […]" [Meye90, S. 44]

[73] Diese Formulierung ist mit Bedacht gewählt. Zwar könnte die Formulierung auch lauten: „Da bei Informationssystemen, insbesondere bei Management Support Systems die Auswirkungen inkonsistenter Daten ceteris paribus erheblich kritischer sind als inkonsistente Algorithmen, muss die Entscheidung zugunsten der primär datenorientierten Vorgehensweise erfolgen." Auch dies wäre korrekt, da die Daten im Vordergrund des Interesses stehen. Im Einzelfall kann das natürlich anders aussehen. Werden beispielsweise logisch unterschiedliche Rekonziliierungs-Algorithmen verwendet, entstehen auch inkonsistente Daten, was erhebliche Folgen hat. Doch ist das beim Objekt-Paradigma nicht die Alternative. Bei gleichen Methoden in verschiedenen Objekten ist oftmals die Implementierung differierend, das Ergebnis jedoch identisch. Sind hingegen gleiche Daten in unterschiedlichen Objekten unterschiedlich, ist keine Konsistenz mehr gegeben.

gunsten der primär datenorientierten Vorgehensweise erfolgen (siehe auch Abschnitt 3.4.3). Außerdem ist die Bildung der Objekte selbst auch datenorientiert — Objekte sind Daten.[74]

Folglich meint auch *Riel*:

> „While action-oriented software development is involved with functional decomposition through a very centralized control mechanism, the object-oriented paradigm focuses more on the decomposition of data with its corresponding functionality in a very decentralized setting. […] Many developers correctly claim that the action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. They then claim, that the object-oriented paradigm focuses exclusively on the data, ignoring the functionality of the system until it is required. This is not possible, because the behavior of a system often drives the decomposition of data. It is preferable to think of the object-oriented paradigm as keeping data in the front of a developer's mind while keeping functionality in the back of his or her mind." [Riel96, S. 29, S. 31]

Auch *Khoshafian* und *Abnous* drücken dies aus durch:

> „The decomposition of the object space in an object-oriented application is data-based. This means the data objects are the central entities in an object-oriented program." [KhAb95, S. 41]

3.3.2.6 Fazit

Damit stellt die OOA als grundlegender Schritt der Modellierung des Objektmodells die wesentlichen Metaphern des Objekt-Paradigmas zur Verfügung und verfolgt damit das primäre Ziel einer natürlichen semantisch-reichhaltigen Abbildung der Diskurswelt.

Die wichtigsten Konzepte der OOA sind das Objekt, die Klasse, Attribute, Methoden und Beziehungen zur Beschreibung der Struktur und Nachrichtenflüsse zur Beschreibung des Verhaltens. Im Vordergrund steht die datenorientierte Gliederung der Objekte (Attribute und Beziehungen), die um die Methoden ergänzt wird.

Genau dieses Modell, d. h. auch diese Paradigmen-Sicht auf OOD und OOP zu übertragen ist das Ziel. [CoYo96, S. 47 ff.] Zwar liefert die OOA tatsächlich das Grundmodell, folgt damit auch den grundlegenden Prinzipien des Objekt-Para-

---

[74] Objekte sind nicht immer Abstraktionen realer Dinge. Es gibt beispielsweise auch Gründe zur Bildung von Prozess-Objekten. [WWW90] Dies sollte jedoch der OOA-Zielsetzung entsprechend die Ausnahme sein. Allerdings ist hierbei zwischen Prozess-Objekten zu unterscheiden, die reine Funktionsobjekte (Manager, Controller) darstellen und solchen Prozess-Objekten, die Prozess-Abstraktionen abbilden, wie beispielsweise eine Buchung oder ein Projekt. Letztere sind als echte Objekte zu behandeln, erstere möglichst zu vermeiden.

digmas, doch die konkurrierenden Objekt-Paradigmen-Sichten werden dies im Folgenden einschränken.

3.3.3 Objektorientiertes Design

Nach einer Einordnung des objektorientierten Designs (OOD) in den Software-Entwicklungs-Prozess wird ausführlich auf Design-Heuristiken und Design-Patterns (Entwurfsmuster) eingegangen — es werden deren Entwicklungsrichtungen, Zielsetzungen und Einsatzgebiete diskutiert und durch wichtige Beispiele ergänzt. Anschließend erfolgt eine ausführliche Abgrenzung zur OOA.

3.3.3.1 Gegenstand

> „Objektorientiertes Design ist eine Designmethode, die den Prozess der objektorientierten Zerlegung beinhaltet sowie eine Notation für die Beschreibung der logischen und physikalischen wie auch statischen und dynamischen Modelle des betrachteten Systems.“ [Booc96, S. 58]

Während sich die OOA auf die Entwicklung der Domäne (Problem Domain, Anwendungsgebiet) konzentriert, d. h. auf den Entwurf des Software-Systems, das die Diskurswelt betrifft, wird beim objektorientierten Design diese Objekt-Schicht der Domänen-Objekte (Domain Objects) ergänzt.

Das Objektmodell der OOA [Balz96, S. 864]:

- beschreibt keine technischen Lösungen
- enthält keine Optimierungen
- enthällt keine Objektverwaltung

Die weiteren OOD-Schichten bzw. OOD-Komponenten sind die Mensch-Maschine-Kommunikation (Human Computer Interaction, HCI), das Datenmanagement (oder besser: die Anbindung eines Datenbank Management Systems bzw. Einbindung eines Objectbase Management Systems) und das Taskmanagement (Verarbeitungssteuerung). [CoYo94, S. 43]

Während bei der OOA immer noch versucht wird, einen weitgehend implementierungs-unabhängigen Entwurf zu erzeugen, werden beim objektorientierten Design Anpassungen an die Zielprogrammiersprache vorgenommen [Balz96, S. 864 ff.].

Die Literatur ergibt kein klares Bild. Es bleibt unklar, was genau objektorientiertes Design ist und vor allem wie OOD von der OOA auf der einen Seite und von der OOP auf der anderen Seite abgegrenzt werden kann. Es ist einzig augenfällig, dass etwas zwischen OOA und OOP ist und dass der Detaillierungsgrad

von der OOA über das OOD bis zur OOP zunimmt bzw. das Abstraktionsniveau abnimmt. [YoAr96, S. 13 ff.]

OOA, OOD und OOP sind mithin nicht unabhängig von einander. Je besser das Ergebnis der OOA ausfällt, desto weniger werden nachträgliche Korrekturen durch den Design-Prozess stattfinden müssen. Je besser das Ergebnis des OOD ist, desto weniger zusätzliche Arbeit erfordert die OOP, insbesondere auch dadurch, dass dann im Idealfall entsprechend mehr Code aus dem Design generiert werden kann.[75] Aus diesem Grund beobachtet *Yourdon* eine Verlagerung des Schwerpunkts von der Codierung auf Design und Analyse.

Außerdem ist bereits die Implementierungs-Unabhängigkeit der OOA nicht gegeben. Das Ergebnis der OOA hängt nicht nur von der verwendeten Methodik ab (z. B. *Shlaer* und *Mellor* [ShMe96, ShMe98], *Wirfs-Brock*, *Wilkerson* und *Wiener* [WWW90] oder *Coad* und *Yourdon* [CoYo96], siehe Abschnitt 3.3.2), sondern auch vom Implementierungsmodell der verwendeten Komponenten (GUI, OOP und DBMS, siehe Abschnitt 3.4).[76]

Auch *Gamma*, *Helm*, *Johnson* und *Vlissides* meinen:

> „Die Wahl einer Programmiersprache ist wichtig, weil sie den eigenen Blickwinkel beeinflusst." [GHJV96, S. 5]

Alpert, *Brown* und *Woolf* betonen sogar, dass es fundamentale Unterschiede beim Entwurf gibt, die sich sogar auf die Sichtweise auf Probleme und mögliche Lösungsansätze beziehen, je nachdem welche Programmiersprache der Entwickler später verwenden wird, d. h. in der Regel verwendet. Einen programmiersprachen-unabhängigen Entwurf halten sie für ungeeignet, da dieser Potenzial unverwirklicht lässt und zu sehr schlechter Software („very poor programs") führt. [ABW98, S. 4 f.]

Das Ergebnis der OOA wirkt also bestimmend auf das OOD und dessen Ergebnis wirkt auf die OOP. Die Konzepte der OOP bedingen aber wiederum das Ergebnis des OOD und dieses das Ergebnis der OOA.

---

[75] Heutzutage immer noch keinen Code aus den Spezifikationen mithilfe von CASE-Umgebungen zu generieren, ist nicht nachvollziehbar. Selbst wenn es sich um ein nur einfaches Werkzeug wie den VisualDesigner von ArsNova handelt, können doch zumindest aus den Klassendiagrammen die Klassen, Instanzvariablen (inklusive Initialisierung, einfacher Typprüfung und Zugriffsmethoden), Methodenrümpfe, Kommentare, Beziehungssemantiken (inklusive der Integritätsicherung von Inversen) generiert werden. Konzeptionelle Erweiterungen des VisualDesigner sind leicht zu bewerkstelligen. [Arsn95] Code-Generierung ist auch die einzige durchführbare Methode, um die Spezifikation und damit auch die elementar wichtige Dokumentation des objektorientierten Programm-Codes [Rusk00, S. 84] konsistent mit der Software zu halten. (Siehe aber auch Abschnitt 3.3.5.)

[76] Es ist durchaus möglich, die OOA unabhängig von Design und Implementierung und damit auch unabhängig von den verwendeten Werkzeugen (GUI, OOP, DBMS) durchzuführen. Dies wird in der Praxis in der Regel nicht geschehen. Das OOA-Ergebnis müsste auf einem sehr hohen Abstraktionsniveau erfolgen, so dass die Design-Phase eine erneute Analyse erforderte, um die restliche Semantik für das Objektmodell zu ermitteln.

Als Folge wird hier unter OOD der Teil der Software-Entwicklung verstanden, in welchem die wesentlichen Design-Entscheidungen (Design-Prinzipien) des Informationssystems getroffen werden. Design-Entscheidungen lassen sich in zwei Kategorien einteilen. Auf einer höheren Abstraktionsebene gibt es Design-Heuristiken [Riel96, S. XI ff.], die sich mit prinzipiellen Design-Entscheidungen des Objekt-Paradigmas beschäftigen. Auf einer niedrigeren Abstraktionsebene gibt es Design-Patterns (Entwurfsmuster) [GHJV96, S. 101 ff.].

Dementsprechend werden im Folgenden zuerst das Konzept der Design-Heuristiken erläutert und wesentliche Heuristiken diskutiert, da diese einen elementaren Bestandteil des Objekt-Paradigmas darstellen. Als zweites wird aufgrund ihrer Bedeutung in der Praxis — und auch als Teilaspekt des Objekt-Paradigmas[77] — die Entstehung und der Zweck von Design-Patterns erörtert.

3.3.3.2 Design-Heuristiken

Der bedeutungsvollste Aspekt des objektorientierten Designs bezüglich des Objektmodells ist, dass die Objekte als möglichst autonome aktive Einheiten angesehen werden, d. h. die Verteilung von Attributen und Methoden auf die unterschiedlichen Klassen die grundlegende Aufgabe des OOD darstellt.

Oftmals kann bei Objektmodellen die Einteilung in „aktive“ und „passive“ Objekte beobachtet werden. [Clau96, S. 151 ff.] Die aktiven Objekten beinhalten die Funktionen und können insofern als Prozedur-Objekte (oder sogar als Hauptprogramm [Riel96, S. 29 ff.]) bezeichnet werden, die passiven Objekte sind Datenspeicher und entsprechen deshalb den Entitäten des ERM. Eine ähnliche Form sind die Einführung von so genannten „Manager-Klassen“. [Appe95, S. 139]

Coad und *Nicola* betonen, dass ein Objekt „lebe“. Sie teilen den Grad des Objektdenkens in drei Ausprägungen ein [CoNi94, S. 27]:

1. **niedrig:** Aufteilung der Anwendung nach Funktionen anstelle der Aufteilung der Anwendung nach Objekten
2. **mittel:** Manager und Datenkapselung — Schwerpunkt auf den Managern, wodurch die anderen Objekte nur noch wenige Aufgaben haben; im Extremfall nur noch Zugriffsmethoden
3. **hoch:** „Objekte an der Arbeit — Verlegung jeder Aktion in das Objekt, das genug kann, um sie direkt auszuführen“

Taylor drückt dies aus, indem er verlangt, dass die Objekte die „Intelligenz“ zu bekommen haben, die zu ihnen gehört und diese eben nicht in Manager-Objekte

---

[77] *Yourdon* sieht das objektorientierte Design als die Zusammensetzung von einzelnen Mustern (siehe das Zitat unter Abschnitt 3.1 auf Seite 81). [Your93, S. 102 ff.]

oder dergleichen gelegt wird, die diese Objekte dann manipulieren. [Tayl92, S. 112] Hierbei versteht *Taylor* unter der „Intelligenz“ die Anwendungs-Logik.

Beispielsweise muss ein Kundenobjekt Auskunft über seinen Gesamtumsatz geben können, ein Rechnungsobjekt über die Gesamtsumme, ein Buchobjekt, ob und wenn ja, an wen es zur Zeit ausgeliehen ist und — so abstrus dies im ersten Moment klingen mag — selbiges Buchobjekt muss eventuell die Fähigkeit (Methode) besitzen sich auszuleihen, sich nach der Rückgabe durch den Ausleiher wieder für ausleihbar zu erklären und sich wieder in das Regal einzusortieren.

Ein elementares Konzept des Objekt-Paradigmas ist mithin die Forderung, weitgehend auf passive Objekte und stark aktive Objekte zu verzichten, d. h. die Systemintelligenz so zu verteilen, dass aktive autonome Objekte entstehen. *Wirfs-Brock*, *Wilkerson* und *Wiener* drücken dies aus durch [WWW90, S. 63 f.]:

- „Evenly distribute system intelligence.
- State responsibilities as generally as possible.
- Keep behavior with related information, if any.
- Keep information about one thing in one place.
- Share responsibilities among related objects.“

Dabei schränken *Wirfs-Brock*, *Wilkerson* und *Wiener* ein, dass es nicht möglich sei, die Intelligenz tatsächlich völlig gleich zu verteilen, da dies so auch nicht in natürlichen Systemen vorkomme. Sie meinen: „Within any system, some classes can be viewed as being relatively ‘smart’, while others seem less so.“ [WWW90, S. 64]

Hinkle und *Johnson* fordern in dieser Konsequenz auf der Implementierungsebene sogar, dass alle Sprachelemente aktive Objekte sein sollten. Obwohl bei Smalltalk bis auf wenige Ausnahmen alle Sprachelemente Objekte sind („classes, methods, processes, contexts, global variables, class variables“), schlagen sie ein Konzept zur Implementierung von Instanzvariablen als Objekt vor und in der Folge von aktiven Instanzvariablen und aktiven Klassen. [HiJo96, S. 14 ff.]

Riel sieht den Schwerpunkt des objektorientierten Designs in der gleichen Richtung. Er betont die Kapselung als notwendige Voraussetzung: „All data should be hidden within its class!“ [Riel96, S. 13] Kein Objekt dürfe Details eines anderen Objekts kennen. Daten (Attribute) und Funktionen (Methoden) sind geheim. Die Objekte sind autonom und sorgen selber für die Erledigung der von ihnen angebotenen Services, die durch das öffentliche Protokoll angeboten werden.

Auf dieser Grundlage betont *Riel* die elementaren OOD-Prinzipien:

- „Distribute system intelligence horizontally as uniformly as possible.“ [Riel96, S. 33]

- „Do not create god classes/objects in your system.“ [Riel96, S. 33]
- „Beware of classes that have many accessor methods defined in their public interface. Having many implies that related data and behavior are not being kept in one place.“ [Riel96, S. 33]
- Beware of classes that have too much noncommunicating behavior.“ [Riel96, S. 33]
- „Keep related data and behavior in one place.“ [Riel96, S. 20]
- „Spin off nonrelated information into another class.“ [Riel96, S. 20]
- „Model the real world when ever possible.“ [Riel96, S. 39]
- „Eliminate irrelevant classes from your design.“ [Riel96, S. 42]
- „Eliminate classes that are outside the system.“ [Riel96, S. 42]
- „Do not turn an operation into a class.“ [Riel96, S. 43]
- „Agent classes are often placed in the analysis model of an application. During time, many agents are found to be irrelevant and should be removed.“ [Riel96, S. 47]

Auch *Riel* macht damit deutlich, dass der Kern des objektorientierten Designs darin besteht, aktive autonome Objekte zu schaffen, mithin die Trennung von Daten und Funktionen aufzuheben und die Intelligenz auf die Objekte zu verteilen, in welche sie gehört. Gott-Klassen — also „aktive“ Objekte („Behavioral Form“, Manager, Controller, Agenten) und „passive“ Objekte („Data Form“, Entity-Classes) — sind auf jeden Fall zu vermeiden und zeigen, dass der System-Designer strukturiertes Design („Action-Oriented Paradigm“) betreibt. [Riel96, S. 32 ff.]

> „This separation [von Daten und Funktionen in Controller- und Entity-Klassen] should be considered a violation of a fundamental principle of the object-oriented paradigm, namely, data and behavior are bidirectionally related as a conceptual block.“ [Riel96, S. 48]

Knight verdeutlicht diese Design-Heurisitk sehr plastisch anhand seiner „Dilbert-Prinzipien“, die zu den selben Design-Entscheidungen führen [Knig96, S. 13 f.]:

- „Never do any work that you can get someone else to do for you.
- Avoid responsibility.
- Postpone Decisions.“

Design-Heuristiken sind mithin prinzipielle Hilfestellungen für das objektorientierte Design. Sie sollen dem Designer grundsätzliche Lösungen anbieten und Argumente liefern, unter welchen Umständen ein bestimmtes Design geeigneter

ist als eine Alternativlösung. Konkrete Lösungen für ein Design oder sogar Code-Fragmente sind dagegen kein Bestandteil von Design-Heuristiken.

3.3.3.3 Design-Patterns

Design-Patterns bieten für wiederkehrende Probleme konkrete Lösungen. Diese Idee ist bereits sehr alt. Beispielsweise gibt es beim ältesten bekannten Brettspiel „GO“, bei welchem mit schwarzen und weißen Steinen Muster[78] gelegt werden, solche wiederkehrenden Muster, sogenannte „Joseki“, die jeweils benannt sind und als bestmögliche Lösungen in der jeweiligen Situation gelten. [Kage84] Eine wichtige Aufgabe eines neuen GO-Schülers ist das Lernen dieser „Joseki“. „Joseki“ sind gute Lösungen, verdeutlichen das Prinzip des Spiels bzw. guter Lösungen und erleichtern die Kommunikation mit anderen GO-Spielern.

Populär wurde das Pattern-Konzept durch *Alexander*. Er erklärt, wie Architekturen durch Patterns beschrieben werden können und entwirft eine Pattern-Language (Sprache aus Mustern). [AIM77, S. IX ff., Alex79, S. 167 ff.] *Alexander*, *Ishikawa* und *Silverstein* meinen, dass mithilfe von 264 Mustern alle möglichen Architekturen beschrieben werden können. [AIM77, S. XIX ff.]

Yourdon erwähnt, dass er bereits 1973, durch eine frühere Arbeit von *Alexander* im Jahre 1971 angeregt, das Konzept der Patterns auf die Software-Entwicklung übertragen hat. Er erwähnt auch bereits 1992 die elementare Bedeutung der Entwurfsmuster für das objektorientierte Paradigma. [Your93[79], S. 102 ff.] *Coad* erläutert 1992 das Konzept der objektorientierten Patterns und stellt konkrete OOA- und OOD-Patterns vor [Coad92, S. 152 ff.], die er 1995 durch eine umfangreiche Sammlung ergänzt [CNM97[80], S. 381 ff.]. Inzwischen gibt es umfangreiche Patterns-Sammlungen [CoSc98, MRB98, VCK98], die die Vielschichtigkeit und den praktischen Nutzen bezeugen.

Die Verwendung von objektorientierten Patterns wurde durch *Gamma*, *Helm*, *Johnson* und *Vlissides* populär — seitdem als „Gang of Four (GoF)“ bekannt [ABW98, S. 1]. Sie konnten durch ihre konkreten Design-Patterns (Entwurfsmuster) für C++ eine starke Verbreitung des Konzepts und der Implementierung ihrer vorgestellten Patterns erreichen.[81] [GHJV96, S. 101 ff.] Auch die Namen ihrer Entwurfsmuster sind bereits zum Standard avanciert.

---

[78] Muster wird hier nicht im Sinne eines Bildes verstanden, sondern ebenso wie Design-Patterns bildet sich ein solches Muster aus Struktur und Dynamik (Verhalten).

[79] Die amerikanische Originalausgabe „The Decline and Fall of the American Programmer“ erschien 1992.

[80] Die erste Auflage erschien 1995.

[81] Der Grundkatalog der von *Gamma, Helm, Johnson* und *Vlissides* entworfenen Muster stammt bereits aus *Gammas* Dissertation von 1991.

Gamma, *Helm*, *Johnson* und *Vlissides* definieren Entwurfsmuster als:

> „… Beschreibungen zusammenarbeitender Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen." [GHJV96, S. 4]

Als Hauptziel des objektorientierten Entwurfs sehen sie das Erzeugen von wiederverwendbaren und flexiblen Entwürfen, damit die zu erstellende Software sowohl den vorliegenden spezifischen Anforderungen genügt, als auch allgemein genug ist, um zukünftigen Problemen und Anforderungen begegnen zu können. Die Revision von Entwürfen soll vermieden oder zumindest minimiert werden. [GHJV96, S. 33]

> „Der Schlüssel zur Maximierung von Wiederverwendung liegt im Voraussehen von neuen Anforderungen und von Änderungen existierender Anforderungen sowie im Entwurf Ihrer Systeme derart, dass sie sich entsprechend entwickeln können." [GHJV96, S. 33]

Dementsprechend sehen *Gamma*, *Helm*, *Johnson* und *Vlissides* acht Punkte als wesentliche Probleme des objektorientierten Designs, die es durch geeignete Entwurfsmuster zu beheben gilt:

1. „Erzeugen eines Objekts unter expliziter Nennung seiner Klasse." [GHJV96, S. 33]
2. „Abhängigkeit von spezifischen Operationen." [GHJV96, S. 33]
3. „Abhängigkeit von Hardware- und Softwareplattformen." [GHJV96, S. 34]
4. „Abhängigkeit von Objektrepräsentationen oder Objektimplementierungen." [GHJV96, S. 34]
5. „Algorithmische Abhängigkeiten." [GHJV96, S. 34]
6. „Enge Kopplung." [GHJV96, S. 34]
7. „Erweiterung von Funktionlität durch Unterklassenbildung." [GHJV96, S. 35]
8. „Unmöglichkeit, Klassen bequem zu ändern." [GHJV96, S. 35]

Ihre Entwurfsmuster dienen dazu, dass Entwickler statt jedes Problem von Grund auf neu anzugehen, erfolgreiche Lösungen wiederverwenden. Die „Gang of Four" stellt dafür drei Kategorien von Patterns vor: „Creation Patterns", „Structural Patterns" und „Behavioral Patterns". [GHJV96, S. 101 ff.]

Sie erklären ausführlich die Funktionsweise und den Zweck der einzelnen Entwurfsmuster. Die elementaren Bestandteile sind der Mustername, das Problem, für welches das Muster sinnvoll erscheint, die Lösung des Musters und die sich daraus ergebenden Konsequenzen. [GHJV96, S. 3 f.]

Da die einzelnen Entwurfsmuster der „Gang of Four“ stark von der hauptsächlich verwendeten Programmiersprache C++ abhängig sind, ist ihre Verwendung für andere Programmiersprachen nur eingeschränkt möglich. Damit auch Smalltalk-Programmierer eben diese Entwurfsmuster anwenden können, haben *Alpert*, *Brown* und *Woolf* eine entsprechende Smalltalk-Version, d. h. Übertragung auf das Smalltalk-Paradigma herausgegeben. [ABW98, S. 1 ff.]

Viele Probleme des C++-Paradigmas ergeben sich bei Smalltalk nicht, dafür sind für bestimmte Probleme deutlich andere Lösungen notwendig. Es handelt sich jedoch prinzipiell um die selben Entwurfsmuster, soweit diese auch für Smalltalk sinnvoll verwendbar sind.

Auf einer noch konkreteren Ebene befinden sich die von *Beck* für Smalltalk entworfenen Muster (praktische Gebrauchsmuster) [Beck97, S. 13 ff.], die bereits als Implementierungs-Muster bezeichnet werden können, da dort sehr speziell auf Smalltalk-Probleme zugeschnittene Lösungen diskutiert werden — Smalltalk-Programmier-Probleme.

Beispielsweise wird diskutiert, in welchen Fällen es sinnvoll ist, innerhalb einer Methode eines Objekts direkt auf die Attribute (Instanzvariablen) des Objekts zuzugreifen oder nur indirekt über dessen Accessing-Methoden.

Alle diese Design-Patterns stellen einen wichtigen Teil der Software-Entwicklung und speziell des objektorientierten Designs dar.[82] Wiederverwendung wird hier auf eine sehr erfolgreiche Weise praktiziert. Insbesondere werden erfolgreiche Umsetzungen des Objekt-Paradigmas — analog zu einem Joseki — zum Einsatz gebracht. Die Diskussion der einzelnen Entwurfsmuster würde an dieser Stelle zu weit führen.[83]

3.3.3.4 Abgrenzung

Es wird deutlich, dass es prinzipielle Unterschiede zwischen dem Paradigma der OOA und dem des OOD gibt. Während bei der OOA versucht wird, die Diskurswelt so „natürlich“ wie möglich abzubilden, bestehen diesbezüglich Einschränkungen durch das objektorientierte Design. Beispielsweise muss auch bei der OOA die System-Intelligenz möglichst sinnvoll und gleichmäßig auf die Objekte verteilt werden, doch gibt es beim OOD auch differierende Kriterien wie diese Intelligenz verteilt werden sollte.

---

[82] Design anhand von Entwurfsmustern wird inzwischen auch bei der konventionellen Entwicklung verwendet. Beispielsweise hat *Peterson* ein Pattern für ein anfrageoptimiertes relationales Datenbank-Design in Form eines Star-Schemas entwickelt. [Pete98, S. 163 ff.]

[83] Es sei jedoch an dieser Stelle erwähnt, dass Design-Patterns trotzdem eine besondere Bedeutung für diese Arbeit haben, da es als eine Zielsetzung angesehen werden muss, auch für den Entwicklungsprozess eines Object Warehouse entsprechende Entwurfsmuster zu finden.

Ein Beispiel sind die Agenten-Klassen, als eine Ausprägung der Gott-Klassen, die in vielen Fällen agierende Personen darstellen, die zwischen Objekten vermitteln. Typische Beispiele sind ein Bibliothekar oder ein Verkäufer. Bei der OOA kann die Modellierung eines solchen Bibliothekars sinnvoll sein[84], da es in der Regel der Bibliothekar sein wird, der ein Buch verleiht, entgegennimmt, einsortiert et cetera.

Dadurch entsteht aber ein typisches „aktives" Gott-Objekt, das andere „passive" Gott-Objekte manipuliert und im schlimmsten Fall nicht nur die Aktionen der beteiligten Objekte anstößt, sondern auch noch vollständig durchführt und so nur noch die neuen Werte der Attribute setzt, z. B. des Buches oder des Bücherregals.

Beim objektorientierten Design wird man in vielen Fällen feststellen, dass der Bibliothekar zumindest für den Systemzweck der Software kein weiterer Bestandteil ist und folglich eliminiert wird.[85] Die System-Aufgaben werden mithin von den Objekten Buch, Bücherregal et cetera selbst erledigt und brauchen keinen Vermittler.

Dass dies in einigen Fällen nicht mehr sehr „natürlich" sein kann, zeigt das extreme Beispiel von *Page-Jones*:

> „On an object-oriented farm there is an object-oriented cow with some object-oriented milk. Should the object-oriented cow send the object-oriented milk the uncow yourself message, or should the object-oriented milk send the object-oriented cow the unmilk yourself message?" [zitiert nach Riel96, S. 45 f.]

Ein weiterer essenzieller Punkt des objektorientierten Designs ist z. B. die Minimierung der Abhängigkeiten durch die Minimierung der Klassen, mit denen eine Klasse kommuniziert, der Anzahl der Nachrichtenverbindungen zwischen zwei Klassen, der Anzahl der Nachrichten zwischen zwei Klassen und der Methoden im Protokoll einer Klasse. [Riel96, S. 57 ff:] Ebenso wie die oben erläuterten acht Punkte von *Gamma*, *Helm*, *Johnson* und *Vlissides*, die alle als Hauptziel die Schaffung eines flexiblen, leicht wiederverwendbaren, anpassbaren und wartbaren Designs haben.

> „Das schwierige am objektorientierten Entwurf ist die Zerlegung eines Systems in Objekte. Hierbei spielen viele Faktoren eine Rolle: Kapselung, Granularität, Abhängigkeit, Flexibilität, Laufzeitverhalten, Evolution, Wiederverwendbarkeit und so weiter. Sie alle beeinflussen in einander oft widersprechender Weise die Zerlegung." [GHJV96, S. 17]

---

[84] Bei der objektorientierten Systemanalyse wird er als Systembestandteil modelliert werden müssen. An dieser Stelle ist lediglich die Modellierung der Software Gegenstand der Betrachtung (siehe Abschnitt 3.2.1).

[85] Das ist nicht damit gleichzusetzen, dass dieser Bilbliothekar auch für das Unternehmenssystem überflüssig wird und dadurch wegrationalisiert wird, kann aber in einigen Fällen dazu führen. Es ist hier nur gemeint, dass der Bibliothekar nicht mehr Modellierungsgegenstand der Software ist. Als Benutzer wird er in der Regel eine wichtige Rolle für das System spielen.

Bei der OOA ist dies nicht der Fall. Im Gegenteil könnte bei der OOA der Ablauf eines Prozesses durch viele Kommunikationspartner (Objekte) und einen ausgeprägten Nachrichtenaustausch modelliert werden. Beispielsweise könnte eine Auftragsannahme im Extremfall derart erzeugt werden, dass ein Kunde der Auftragsklasse die Nachricht „neu“ schickt, dieser Auftrag dann vom Kunden Attribut für Attribut die Daten für den Auftrag erfragt et cetera — analog der Kommunikation zwischen einem Kunden und einem Verkäufer.

Mithin entstammen zwar viele Objekte eines Entwurfs der OOA, doch während des objektorientierten Designs werden auch Klassen modelliert, für die es keine Entsprechung in der realen Welt gibt. Einige Klassen weisen einen höheren, andere einen niedrigeren Abstraktionsgrad auf und haben deshalb keine physische Entsprechung in der realen Welt. *Gamma*, *Helm*, *Johnson* und *Vlissides* nennen solche Objekte als Beispiel, die einen Prozess oder Algorithmus repräsentieren. Diese Objekte erscheinen aber nicht in der Natur, sind jedoch ein essenzieller Bestandteil flexibler Entwürfe. [GHJV96, S. 17]

Beck zeigt konkret am Beispiel von Collections, Variablen und Methoden, wodurch und in welchen Fällen zusätzliche Objekte entstehen können und inwiefern diese zusätzlichen Klassen den Entwurf verbessern. Als essenzielle Bewertungskriterien sieht *Beck* [Beck94a, S. 221, Beck94b, S. 239 ff.]:

- Einfachheit
- Lesbarkeit
- Wartbarkeit/Erweiterbarkeit

Dies entspricht auch wesentlich den Kriterien von *Howard*, der wiederum die Zerlegung dergestalt fordert, dass der Programm-Code einfach und lesbar sein muss, um leicht verständlich für andere Entwickler zu sein, da Software hauptsächlich gewartet und erweitert wird und das eben in den meisten Fällen von anderen. [Howa95, S. 295 ff.]

Gamma, *Helm*, *Johnson* und *Vlissides* bringen den Grund für die unterschiedlichen Zielsetzungen von OOA und OOD auf den Punkt, indem sie meinen:

> „Die strikt an der realen Welt orientierte Modellierung führt zu einem System, das vielleicht die heutige Realität wiedergibt, nicht aber unbedingt die von morgen. Die sich während eines Entwurfs ergebenden Abstraktionen sind der Schlüssel dazu, einen Entwurf flexibel zu machen.“ [GHJV96, S. 17]

Es zeigen sich jedoch nicht nur Unterschiede zur OOA, selbst zwischen verschiedenen Abstraktionsstufen des objektorientierten Designs wie den Design-Heuristiken [Riel96, S. 11 ff.], den Entwurfsmustern [GHJV96, S. 101 ff., ABW98, S. 29 ff.] und den Implementierungs-Mustern (Practice Patterns) [Beck97, S. 13 ff.] gibt es konkurrierende Zielsetzungen.

3.3.4 Objektorientierte Programmierung

Zunächst wird der Gegenstand der objektorientierten Programmierung (OOP) geklärt, dem sich ein kurzer Überblick über die Entwicklung der OOP anschließt, da das Objekt-Paradigma hauptsächlich aus der OOP entstanden ist (siehe Abschnitt 3.3.1) und auf diese Weise auch die sich daraus entwickelten Konzepte besser eingeordnet werden können.

Nach den Grundkonzepten werden in Anbetracht der besonderen Bedeutung von Smalltalk speziell die Konzepte dieser OOPL diskutiert. Dem schließt sich eine detaillierte Diskussion von Typen, Klassen und Vererbung an. Am Ende erfolgt eine Abgrenzung der OOP-Konzepte von denen der OOA und des OOD.

3.3.4.1 Gegenstand

Zunächst stellt sich die Frage, was objektorientierte Programmierung (OOP) ist und wann eine Programmiersprache als eine objektorientierte Programmiersprache (OOPL) klassifiziert werden kann.

Eine Programmiersprache gilt dann als objektorientiert, wenn sie die Konzepte der Objektorientierung ausreichend unterstützt. *Stroustrup* drückt dies treffend aus:

> „If the term ‘object-oriented programming language’ means anything, it must mean a language that has mechanisms that support the object-oriented style of programming well. There is an important distinction here: A language **supports** a programming style if it provides facilities that make it convenient (reasonably easy, safe, and efficient) to use that style. A language does not support a technique if it takes exceptional effort or skill to write such programs; in that case, the language merely **enables** programmers to use the technique. For example, you can write structured programs in Fortran and type-secure programs in C, and you can use data abstraction in Modula-2, but it is unnecessarily hard to do so because those languages do not **support** those techniques.“ [Stro88, S. 10 f.]

3.3.4.2 Entwicklungspfad

Damit stellt sich die Frage, welche Konzepte die objektorientierte Programmierung beinhaltet. Es gibt mindestens zwei Entwicklungswege zur objektorientierten Programmierung.

Erstens wird die *kaysche* Idee des Objekt-Paradigmas zu verwirklichen versucht (siehe Abschnitt 3.3.2). Zu diesem Zweck wird von *Ingalls*, *Goldberg* und *Kay* auf der Grundlage von Simula die Sprache Smalltalk entwickelt [GoRo89, S. VI ff.]. [Kay93, S. 4]

Zweitens entwickeln sich aufgrund der steigenden Software-Anforderungen auch die Konzepte anderer Programmiersprachen weiter bzw. führen zu neuen Programmiersprachen, was sehr ähnliche Entwicklungsrichtungen anstößt.

Die Entwicklung ging über Assembler und FORTRAN durch das die Konzepte Variablen, Arrays, Kontrollstrukturen (Iteration und Entscheidung) eingeführt werden, zu höheren Sprachen wie PL/1, COBOL und Algol 60. Mit Algol 60 wird das Konzept der lokalen Variablen und damit auch von Blöcken eingeführt, was eine erste Form der Kapselung bedeutet. [KhAb95, S. 13]

Während bei Simula der Block bereits als Objekt interpretiert wird, werden die Blöcke ansonsten funktions-orientiert zerlegt. Damit war die Grundlage der *dijakstraschen* strukturierten Programmierung gelegt und die strukturierte Analyse und das strukturierte Design ermöglicht. [Balz96, S. 215 ff.] Pascal verwirklicht die Konzepte der strukturierten Programmierung durch Unterprogramme in Form von Prozeduren und Funktionen, Parameterübergabe, das Typkonzept sowie lokale Variablen.

In Modula-2 wird das Konzept der Unterprogramme zum Modulkonzept erweitert und um Datenkapselung ergänzt. [BrBu92, S. 11 f.] Zwar können die Daten dadurch bereits zusammen mit den Funktionen gespeichert werden, doch die Gliederung der Module erfolgt weiterhin funktions-orientiert.

Erst das Konzept der Datenabstraktion verbunden mit der Entwicklung der abstrakten Datentypen (ADT) und die Umsetzung in beispielsweise Ada führt zu einer Änderung dieses Paradigmas. Durch die abstrakten Datentypen erfolgt die Strukturierung der Module anhand der Datenmodellierung und ergänzt die Daten-Module um die zugehörigen Operationen (Funktionen). Außerdem verstecken ADT die Implementierung und Repräsentation der Module vor den anderen Modulen (Information Hiding). [KhAb95, S. 14] Dies ist der entscheidende Schritt zur Realisierung eines Objekt-Konzepts.

Der nächste Schritt ist das Klassenkonzept und die Vererbung. Ab diesem Punkt laufen die beiden Wege der Entwicklungen der Programmiersprachen zusammen. Es entstehen objektorientierte Programmiersprachen wie Smalltalk, C++, Objectiv C, Eiffel oder Java.

3.3.4.3 Grundkonzepte

Dementsprechend klassifiziert *Wegner* drei Arten von Objekt-Sprachen (siehe Abbildung 10). Sprachen, die abstrakte Datentypen, d. h. Objekte als grundlegende Elemente verwenden sind objektbasiert. Gehört jedes Objekt zusätzlich zu einer Klasse, handelt es sich um eine klassen-basierte Sprache. Sind die Klassen zusätzlich in einer Klassenhierarchie angeordnet, so dass Vererbung realisiert wird, handelt es sich um eine objektorientierte Sprache. [Wegn87, S. 169 ff.]

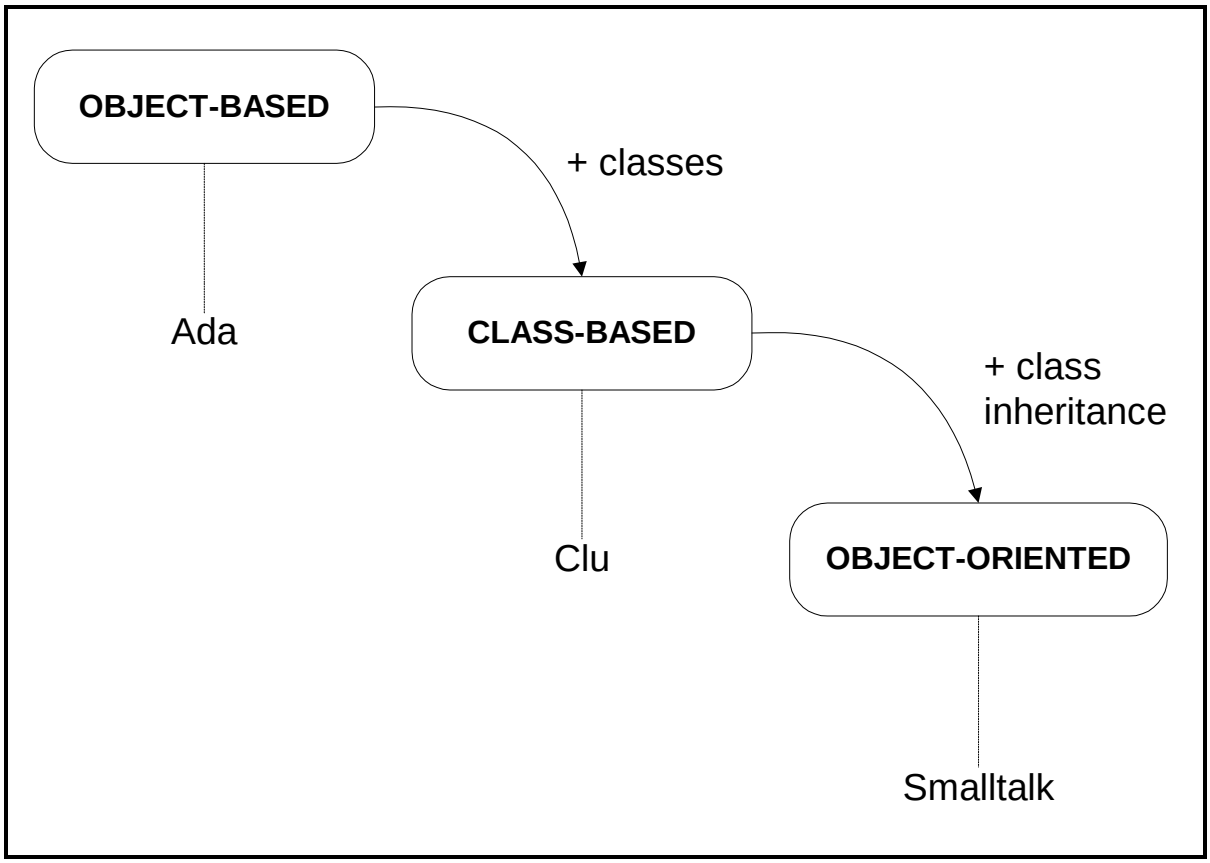


Abbildung 10: Klassifizierung von Objekt-Sprachen nach *Wegner* [Wegn87, S. 169]

Diese Definition gilt gemeinhin als akzeptiert [Booc96, S. 57 f., Liu96, S. 28, Vett95, S. 87]. *Booch* definiert OOP deswegen folgendermaßen:

> „Objektorientierte Programmierung ist eine Implementierungsmethode, bei der Progamme als kooperierende Ansammlungen von Objekten angeordnet sind. Jedes dieser Objekte stellt eine Instanz einer Klasse dar, und alle Klassen sind Elemente einer Klassenhierarchie, die durch Vererbungsbeziehungen gekennzeichnet ist.“ [Booc96, S. 57]

Wegner ergänzt, dass es zusätzlich nützliche aber nicht notwendige Eigenschaften von objektorientierten Programmiersprachen wie strenge Typisierung, Nebenläufigkeit und Persistenz gibt. [Wegn87, S. 170 ff.]

Von einzelnen Autoren wird das Konzept allerdings abgewandelt bzw. ergänzt. So erweitern *Khoshafian* und *Abnous* das Konzept um die Objektidentität (OID) und die Idee konkurrierender Prozesse. Als wesentliche Merkmale sehen sie [KhAb95, S. 16]:

- „Abstract data types
- Inheritance
- Object identity
- Concurrency“

Die **Objektidentität** besagt, dass jedes Objekt einmalig ist und dessen OID während des gesamten Lebenszyklus des Systems erhalten bleibt. Dies ist sehr realitätsnah und bildet auch deswegen ein elementares Konzept des Objekt-Paradigmas. Wesentlich dabei ist, dass die OOPL die gesamte Verwaltung der OID übernimmt, so dass die Identität eines Objekts nicht durch den Programmierer beeinflusst werden kann.

Dies ersetzt auch das Konzept der Fremd- und Primärschlüssel des Relationen-Modells, wodurch ein Zwang zu sauberen Konzepten besteht und beispielsweise die Verwendung von sprechenden Schlüsseln verhindert wird. [Heue97, S. 286 ff.]

Concurrency meint „… multiple independent activities within a system, including […] communication." [ToSc89, S. 79] Die Kommunikation zwischen Objekten erfolgt unabhängig und konkurrierend. [KhAb95, S. 16] Durch diese Nebenläufigkeit von Prozessen können die Prozesse konkurrieren.

Es wäre nicht sehr natürlich, wenn nacheinander immer nur zwei Objekte kommunizieren könnten bzw. gleichzeitig immer nur ein Objekt eine Nachricht schicken kann.[86] Dieses Konzept gehört zwingend zur Sicht des Nachrichtenaustauschs, ist für bestimmte Programmanforderungen unverzichtbar und sollte deshalb zumindest konzeptionell durch die OOPL unterstützt werden. [Nier89, S. 13 f.]

Meyer dagegen bildet aufeinander aufbauende, d. h. sequenzielle (Reife-)Stufen für eine objektorientierte Programmiersprache. Für „wahre Objektorientiertheit" verlangt er die Erfüllung aller „sieben Stufen zur objektbasierten Glückseligkeit" [Meye90, S. 65]:

1. „**Objektbasierte modulare Struktur**: Systeme werden auf der Grundlage ihrer Datenstruktur modularisiert." [Meye90, S. 65]
2. **„Datenabstraktion**: Objekte müssen als Implementierungen abstrakter Datentypen beschrieben werden." [Meye90, S. 65]
3. **„Automatische Speicherplatzverwaltung**: Unbenutzte Objekte sollten ohne Programmiereingriff vom unterliegenden Sprachsystem freigegeben werden." [Meye90, S. 66]
4. **„Klassen**: Jeder nicht-einfache Typ ist ein Modul, und jeder Modul höherer Ebene ist ein Typ." [Meye90, S. 66]

[86] Man stelle sich eine Universität vor, in der gleichzeitig immer nur ein Dozent etwas erzählen darf, so dass die anderen Dozenten in den anderen Hörsälen auf eine Pause des anderen warten müssten. Auch kann das Phänomen empirisch beobachtet werden, dass manche Kommunikationspartner gleichzeitig sprechen, selbst wenn es sich dabei um völlig verschiedene Dinge handelt.

5. **„Vererbung**: Eine Klasse kann als Einschränkung oder Erweiterung einer anderen definiert werden.“ [Meye90, S. 67]
6. **„Polymorphismus und dynamisches Binden**: Programmelemente dürfen sich auf Objekte aus mehr als einer Klasse beziehen, und Operationen dürfen unterschiedliche Realisierungen in unterschiedlichen Klassen haben.“ [Meye-90, S. 66]
7. **„Mehrfaches und wiederholtes Erben**: Man kann Klassen deklarieren, die Erben von mehr als einer Klasse sind und mehr als einmal von einer Klasse erben.“ [Meye90, S. 68]

3.3.4.4 Smalltalk-Konzepte

Es gibt inzwischen eine Vielzahl an verschiedenen objektorientierten Programmiersprachen. Die einzelnen Konzepte sind zum Teil deutlich unterschiedlich. Eine herausragende Stellung kommt dabei der OOPL Smalltalk zu. Zum einen wird Smalltalk gemeinhin als „reine“ objektorientierte Sprache bezeichnet [ABW98, S. 5, ElNa94, S. 665, Lewi95, S. 18, Shar97, S. XXI], da alle Elemente Objekte sind [HiJo96, S. 14 ff.] (z. B. Klassen, Instanzen, Basistypen, Literale, Methoden, Nachrichten, Blöcke, Fenster, Widgets) und zum anderen, weil Smalltalk der Prototyp bei der Entwicklung des Objekt-Paradigmas der objektorientierten Programmierung ist und insofern einen gewissen normativen Charakter hat. [87]

Smalltalk realisiert

- vollständig abstrakte Datentypen (Methoden und Attribute sind streng gekapselt)[88] , wodurch Datenabstraktion unterstützt wird,
- es gibt Klassen und Vererbung in einem hierarchischen Klassenbaum mit einer Wurzel,
- vom System verwaltete Objekt-Identität,
- dynamisches Binden und Typfreiheit, wodurch Teilmengen- und impliziter[89] Polymorphismus ermöglicht wird,
- konzeptionelle Nebenläufigkeit,

---

[87] „Da Smalltalk so kompromisslos objektorientiert ist, wird es oft auch als Quasi-Norm für OOPLs angesehen.“ [Heue97, S. 246]

[88] Allerdings ist es nicht möglich, Methoden von der Schnittstelle zu entfernen, d. h. Methoden als privat zu deklarieren. Die Methoden selber, d. h. deren Implementierung ist gekapselt.

[89] Durch die Typfreiheit ist der Polymorphismus nicht nur innerhalb eines Klassenbaums realisierbar. Impliziter Polymorhpismus wird auch als Ad-hoc-Polymorphismus bezeichnet. [Liu96, S. 169 f.]

- automatische Speicherverwaltung[90] und
- hardware-unabhängige Bytecode-Compilierung.

Üblicherweise ist ein Smalltalk-Dialekt nur mit einer kompletten Entwicklungsumgebung (z. B. VisualWorks) als Image mit umfangreicher Standard-Klassenbibliothek, Entwicklungs-Werkzeugen (Browser, Debugger, Inspector etc.), GUI-Builder-Framework, Datenbank-Schnittstelle und Just-In-Time-Compiler (JIT)[91] erhältlich. [Lewi95, S. 18, PaDi95a, S. 101 ff., Shar97, S. 181 ff.]

Mehrfachvererbung ist dagegen im Zuge der Entwicklung von Smalltalk aus einer früheren Version wieder entfernt worden[92], da der zusätzliche Aufwand als zum Nutzen unverhältnismäßig hoch angesehen wurde [Liu96, S. 103 f.] und von Mehrfachvererbung ohnehin regelmäßig abgeraten wird [Riel96, S. 131 ff.], insbesondere in Anbetracht der Tatsache, dass überhaupt die Verwendung der Vererbung, also auch der einfachen Vererbung durchaus nicht unproblematisch ist [Beck93d, S. 181 ff., Beck93e, S. 189 ff.] und in vielen Fällen besser durch Aggregation realisiert wird [Liu96, S. 97 ff.].

Smalltalk ist in seinem Konzept konsequent. Da auch Klassen Objekte sind, sind Klassen auch Instanzen. Jede Klasse ist die Instanz einer Metaklasse. Metaklassen sind auch Objekte und deshalb ebenfalls Instanzen einer Klasse.[93] [WaOe97, S. 133 ff.]

Dies geht über das Klassenkonzept des OOA-Paradigmas hinaus. Die Klasse ist nicht nur die Form für Objekte. Klassen haben eigene Instanzvariablen und Methoden. Dies ermöglicht es, Klassen mit in die Applikationslogik einzubeziehen. Beispielsweise können Klassen ihre Instanzen verwalten (Extents) [SST97, S. 165], Constraints realisieren [RMK99], Fenster- und Datenbankschnittstellen-Spezifikationen verwalten [Howa95, S. 121 ff.] oder sich wie andere Instanzen verhalten.

Dieses Konzept ist sehr mächtig und besonders zur Verwaltung von Metadaten der Instanzen sowie gemeinsamen Daten der Instanzen geeignet und deshalb auch für die Metadaten-Repräsentation des Object Warehouse von hoher Bedeu-

---

[90] Auch das Löschen von Objekten erfolgt automatisch durch die Garbage Collection, sobald ein Objekt von keinem Objekt mehr referenziert wird, das zumindest mittelbar durch das SystemDictionary, also einer globalen Variablen referenziert wird, d. h. zu dessen transitiver Hülle gehört.

[91] Diese Methode ist vergleichsweise effizient und bietet insbesondere die Chance, sehr mächtige, hoch flexible Konstrukte zu realisieren, indem zur Laufzeit compiliert werden kann. Dadurch kann nicht nur bei der Entwicklung viel Zeit durch schnelles Compilieren (weil nur der jeweilig veränderte Teil compiliert werden muss) und effizientes Debugging gespart werden, sondern es können einzelne Methoden oder ganze Klassen abhängig von der Applikationslogik erzeugt, verändert oder gelöscht werden. Manche Smalltalk-Umgebungen wie etwa Squeak interpretieren den Code.

[92] Nach einer Aussage von *Georg Heeg* in der GSUG-Mailingliste (http://swiki.gsug.org/list, 27. 4. 2000)

[93] Alle Metaklassen sind Instanzen der MetaMetaClass, die natürlich auch wieder Instanz einer Klasse ist.

tung. Die strenge konzeptionelle Trennung von Klasse und Objekt wird damit aufgegeben. Wie sich gezeigt hat, neigen Anfänger in der Folge allerdings oftmals dazu, statt in Objekten zu denken, Klassenmethoden zu verwenden.[94]

3.3.4.5 Typen, Klassen und Vererbung

Eines der elementaren Konzepte des Objekt-Paradigmas der objektorientierten Programmierung — wenn nicht gar das grundlegendste Konzept für die OOP überhaupt — ist die Einführung von abstrakten Datentypen. *Breutmann* und *Burkhardt* definieren abstrakte Datentypen wie folgt:

> „Abstrakte Datentypen realisieren **Datenabstraktion**, d. h., sie ermöglichen es, anwendungsbezogene und nicht in der Programmiersprache vordefinierte Datentypen zu definieren und gemeinsam mit ihren Operationen zu handhaben." [BrBu92, S. 33]

Ein ADT besteht mithin aus seiner Struktur (Daten) und seinem Verhalten (Funktionen). Diese Definition ist jedoch nicht hinreichend. Der Unterschied zwischen einem gewöhnlichen Typ[95] und einem abstrakten Datentyp besteht in der Kapselung der Daten und Funktionen.[96] Das zentrale Konzept ist folglich die Kapselung, durch welche ADT unabhängig von ihrer Implementierung verwendet werden können. [JCJÖ95, S. 49]

Zwei ADT gelten dann als gleich, wenn ihre jeweiligen Protokolle (Schnittstellen) die selben Methoden aufweisen, d. h. Methodennamen, und in jedem Fall auf eine Nachricht zum jeweils selben Verhalten führen sowie mit dem identischem Ergebnis antworten, d. h. den identischen Rückgabewert haben. Beide ADT müssen also über semantisch gleiche Methoden und die für diese Methoden — in welcher Form auch immer — gleichwertigen Attributwerte verfügen. Es wird jedoch vollständig von der Implementierung der Daten (Attribute und deren Typ) und Funktionen (Programm-Code) abstrahiert. Dementsprechend meinen *Gamma*, *Helm*, *Johnson* und *Vlissides*: „Ein Typ ist ein Name, der eine bestimmte Schnittstelle bezeichnet." [GHJV96, S. 18]

Dadurch ergibt sich die Frage, wann ein Typ ein Subtyp bzw. Supertyp eines anderen Typen ist. Hierzu besteht die Forderung, dass ein Typ immer durch seinen Subtyp substituiert werden können muss, wodurch ein Typ deshalb auch immer sein eigener Subtyp und Supertyp ist. [Liu96, S. 198 ff.] Wenn beispielsweise En-

---

[94] Dies hat sich bei Kursen und Projekten an der Universität Osnabrück, Fachgebiet BWL/Wirtschaftsinformatik II gezeigt.

[95] Konventionelle Datentypen, etwa des Relationenmodells, bilden lediglich die Struktur ab und bestehen insofern einzig aus Daten. [Heue97, S. 51 ff.] Hier wird Typ bereits im Sinne einer Beschreibung durch Daten und Funktionen verstanden.

[96] Trotzdem wird im Folgenden der üblichen Praxis entsprechend [Liu96, S. 13, GHJV96, S. 18 f.] Typ im Zusammenhang mit der OOP als abstrakter Datentyp verstanden.

te Subtyp von Vogel ist, dann muss es möglich sein, an jeder Stelle eines objektorientierten Programms an der ein Vogel korrekt ist auch eine Ente einzusetzen. *Liu* meint dementsprechend:

> „This property is called **substitutability**: an instance of a subtype may be substituted wherever something of the supertype is expected and the program will still work." [Liu96, S. 200]

Die Substituierbarkeit eines Supertyps ist mithin für alle seine Subtypen gegeben. Daraus ergibt sich eine Typhierarchie. Wird eine Typhierarchie durch Vererbung verwirklicht, heißt dies Typvererbung [Liu96, S. 198 ff.], Subtyping [JCJÖ95, S. 64] oder Schnittstellenvererbung [GHJV96, S. 18 ff.]. Gerade der letzte Ausdruck betont die Kapselungseigenschaft der ADT.

Unter welchen Bedingungen ein Typ Supertyp eines Subtypen ist und in der Folge durch diesen Subtyp substituiert werden kann, um Konsistenz zu wahren, ist umstritten. *Liu* unterscheidet vier Formen [Liu96, S. 202 ff.]:

1. **Anarchy** nennt *Liu* den Fall, bei dem die einzige Bedingung ist, dass ein Subtyp mindestens die selben Methoden im Protokoll hat wie sein Supertyp. Ansonsten gibt es keine weiteren Restriktionen. Auch was diese Methoden bewirken ist beliebig. Smalltalk realisiert dieses Konzept. Die Typprüfung erfolgt zur Laufzeit und beschränkt sich vollständig darauf, zu prüfen, ob ein Objekt über die Methode verfügt, die durch die Nachricht aufgerufen werden soll.[97]

2. Als **Conformance** bezeichnet er es, wenn zusätzlich zum Fall der Anarchy auch die Parametertypen der Methoden übereinstimmen („conform"). Dies ist eben dann der Fall, wenn dadurch Substituierbarkeit garantiert werden kann.

   Der Rückgabeparameter der betrachteten Methode des Subtyps muss deshalb ein Subtyp des Rückgabeparameters der gleichen Methode des Supertyps sein („covariant"). So muss zum Beispiel obige Ente auf die Nachricht „Lege ein Ei" mit einem Vogelei oder einem Entenei antworten, sofern das Entenei ein Subtyp von Vogelei ist und der Vogel auf die Nachricht „Lege ein Ei" mit einem Vogelei antwortet.

   Für die Übergabeparameter (Eingangsparameter) gilt dasselbe mit umgekehrter Wirkung. Der Übergabeparametertyp einer Subtyp-Methode muss ein Su-

---

[97] Smalltalk ist insofern als eine typisierte Sprache aufzufassen, indem der Typ einzig durch die Schnittstelle (Signaturen des Protokolls) bestimmt wird. Dies entspricht dem Konzept der ADT (wobei semantische Prüfungen wie üblich außerhalb der Betrachtung liegen). Die Prüfung der korrekten Typzurordnung, also der Nachrichten zu den Methoden, erfolgt zur Laufzeit. Diese Interpretation ist allerdings kritisch zu betrachten, da in diesem Fall die begriffliche Unterscheidung zwischen typisierten Sprachen und untypisierten Sprachen keinen definitorischen Nutzen erbringt. *Maier*, *Stein*, *Otis*, und *Purdy* beschreiben dies wie folgt: „Unlike most programming languages that support abstract data types, Smalltalk associates types with values, not the slots holding the values." [MSOP90, S. 205]

pertyp des Übergabeparameters der Supertyp-Methode sein („contravariant“). Beispielsweise muss ein Professor als Übergabeparametertyp der Methode „unterrichte“ entweder „schweren Lehrstoff“ oder „Lehrstoff“ festlegen, nicht hingegen „sehr schweren Lehrstoff“, wenn davon ausgegangen wird, dass „sehr schwerer Lehrstoff“ ein Subtyp von „schwerer Lehrstoff“ ist und dieser wiederum von „Lehrstoff“, Professor ein Subtyp von Dozent ist und Dozent als Übergabeparametertyp „schweren Lehrstoff“ festlegt.

Mit dem Parametertyp „Lehrstoff“ kann der Professor selbstverständlich auch „sehr schweren Lehrstoff“ unterrichten, da ersterer ein Subtyp des letzteren ist. Eine Einschränkung des Parametertyps auf „sehr schweren Lehrstoff“ verhinderte die Substituierbarkeit des Supertyps Dozent durch den Subtyp Professor, auch wenn dies ökonomisch sinnvoller sein sollte.

3. Die Forderung nach **Behavioral Consistency** geht noch weiter und verlangt ergänzend die Verwirklichung von Conformance auch für die Semantik der Methoden. Eine Subtyp-Methode muss in allen Fällen anwendbar sein, wenn die gleiche Methode des Supertyps angewendet werden kann. Beispielsweise muss ein Professor in mindestens den Fällen unterrichten können, in welchen der Dozent dies kann.

   Oft passiert es jedoch, dass die Methode des Supertyps bei der Anwendung auf den Subtyp keinen Sinn ergibt. Deshalb müssen analog zu den Einschränkungen der Parametertypen Vor- und Nachbedingungen spezifiziert werden. Die Vorbedingungen (Pre-Conditions) der Subtyp-Methode werden deshalb erleichtert, die der Nachbedingungnen (Post-Conditions) erschwert. Rudimentär werden hierzu von Eiffel Mechanismen realisiert.

4. **Rigidity** verbietet die Neudefinition von Methoden überhaupt. Dadurch werden die beschriebenen Konsistenzprobleme vermieden. Jeder Subtyp beinhaltet alle Methoden, deren Implementierung und Parametertypen des Supertyps. Substituierbarkeit ist damit vollständig gewährleistet, auf Polymorphismus wird damit jedoch verzichtet, wodurch eine solche Sprache dann auch nicht mehr in die Klassifikation einer OOPL fiele.

Abstrakte Datentypen werden durch Klassen implementiert. Klassen stehen in einer Vererbungsbeziehung. In diesem Fall repräsentieren Oberklassen Supertypen und Unterklassen Subtypen. Um Anarchy, Conformance, Behavioral Conformance oder Rigidity zu realisieren, können Klassen Attribute und Methoden in entsprechenden Varianten vererben. *Khoshafian* und *Abnous* unterscheiden vier Formen der Attributvererbung und drei Formen der Methodenvererbung [KhAb95, S. 96 f.]:

Attributvererbung:

1. Keinerlei Redefinition der Instanzvariablen. (Rigidity)

2. Unabhängige Redefinition des Typs der Instanzvariablen. (Anarchy)
3. Beschränkte Redefinition des Typs der Instanzvariablen dahingehend, dass die Instanzvariablen Subtyp der jeweiligen Instanzvariable der Oberklasse sein dürfen. (Conformance)
4. Versteckte Definition der Instanzvariablen, so dass die Instanzvariablen um Konflikte zu vermeiden in jeder Unterklasse neu definiert werden.

Methodenvererbung:

1. Keinerlei Redefinition der Methoden (Overriding). (Rigidity)
2. Unabhängige Redefinition der Methoden ohne Beschränkung der Parametertypen oder des Verhaltens. Es sind also auch Methoden mit völlig anderem Verhalten oder sogar leere Methoden möglich. Wird eine Methode nicht überschrieben, wird die Methode der Oberklasse auf das Objekt angewendet. (Anarchy)
3. Beschränkte (constraint) Redefinition der Methoden. Werden Methoden überschrieben, muss die Signatur einer Subtypmethode ein Subtyp der Signatur eines Supertyps sein. Dies betrifft Bedingungen bezüglich der Attributtypen und/oder der Semantik von Methoden. (Conformance, Behavioral Conformance)

Obwohl diese Typvererbung eine Schnittstellenvererbung ist, d. h. von den Instanzvariablen und Methodenimplementierungen abstrahiert wird, besteht durch die Vererbung der Methoden und Attribute doch eine Abhängigkeit zwischen Unter- und Oberklasse.[98]

Üblicherweise werden Instanzvariablen und Methoden an die Unterklasse vererbt. Methoden können zwar in der Regel redefiniert werden, müssen aber in jedem Fall auch für den Subtyp sinnvoll im Sinne des Nachrichtensenders sein, selbst wenn die strengen Regeln der Behavioral Conformance missachtet werden.[99] Bei Instanzvariablen ergibt sich ein weiteres Problem. Werden Instanzvariablen im Subtyp nicht benötigt, was bei der Schnittstellenvererbung durchaus sinnvoll und von Bedeutung sein kann, können diese bei den meisten Sprachen trotzdem nicht aus der Unterklasse entfernt werden.

Liu nennt Quadrat und Rechteck als Beispiel. Ein Quadrat ist ein spezielles Rechteck bei welchem alle vier Seiten gleich lang sind. Quadrat erbt folglich von Rechteck und kann an jeder Stelle eines Programms ein Rechteck substituieren. Da ein Rechteck zwei verschieden lange Seiten aufweist, erfordert ein Rechteck

---

[98] *Liskov* hat darauf hingewiesen, dass das Geheimnisprinzip durch die Vererbung durchbrochen wird, weil Änderungen einer Oberklasse Auswirkungnen auf die Unterklasse haben. [zitiert nach Heue97, S. 188]

[99] Behavioral Conformance findet außer der rudimentären Form in Eiffel bisher nur in einigen Forschungsprototypen Verwendung. Der erforderliche Aufwand wird hierbei für unverhältnismäßig hoch angesehen.

eine Instanzvariable mehr. Erbt Quadrat von Rechteck, so hat jedes Quadrat eine Instanzvariable zuviel. [Liu96, S. 201] Dieser Effekt führt nicht nur zu möglicherweise erheblicher Speicherplatzverschwendung, sondern birgt auch die Gefahr von Dateninkonsistenz.

Die Umsetzung des Konzepts der Typhierarchie mithilfe der Klassenvererbung ist also durchaus problematisch. Außerdem sind bei der OOP auch andere Zielsetzungen von hoher Priorität. Insbesondere die Wiederverwendung der Klassenhierarchie spielt eine bedeutende Rolle. Oftmals werden Klassenhierarchien derart gebildet, dass, analog zu Bibliotheken konventioneller Programmiersprachen, Klassenbibliotheken zugekauft und dann durch Unterklassenbildung den speziellen Bedürfnissen der Anwendung angepasst werden können. Dies gilt auch für die Bildung der eigenen Klassen, werden diese nach selbigem Kriterium gebildet.

Als Ergebnis werden Klassenhierarchien nicht nach dem Prinzip der Typvererbung und Substituierbarkeit erzeugt, sondern zur Wiederverwendung. Analog zur konventionellen strukturierten Programmierung, werden Module etwa nach dem Kriterium der „Allgemeinen Benutzbarkeit“ [Balz96, S. 915 ff.] gebildet. Dies bedeutet, dass Klassen spezielle Funktionalitäten bieten, die von anderen Klassen genutzt werden können. *Jacobson*, *Christerson*, *Jonsson* und *Övergaard* bezeichnen diese Form der Vererbung als „Reuse“ und beschreiben deren Zweck wie folgt:

> „The most common reason for using inheritance is that it simplifies the reuse of code. Reuse can, in principle, occur in two different ways in combination with inheritance. The first is that two classes are found to have similar parts; these parts are extracted and placed in an abstract class, which both the original classes inherit. This abstract class represents the common parts of both classes and need not always be meaningful in itself. The other way to reuse is to start form a class library. Find a class which have the operations that you need, inherit this class and make the required modifications.“ [JCJÖ95, S. 64]

Als ein Beispiel wird auf diese Weise die Mehrfachvererbung genutzt, um Objekte bestimmter Klassen persistent zu machen. Hierzu erbt die persistent zu machende Klasse ergänzend die Eigenschaften einer die Persistenz realisierenden Klasse. [Schm91, S. 53 ff.]

Mithin gibt es zwei sehr unterschiedliche Zielsetzungen bei der Konstruktion von Klassenhierarchien, entweder eine Typhierarchie (Schnittstellenvererbung, Subtyping) oder eine Klassenhierarchie (Implementierungsvererbung, Subclassing, Reuse). Für *Gamma*, *Helm*, *Johnson* und *Vlissides* bildet die erste Form die Grundlage für die Implementierung ihrer Entwurfsmuster. [GHJV96, S. 18 ff.] Bei Smalltalk wird ausschließlich die zweite Form realisiert. [GoRo89, S. 55 ff.]

Bei Java besteht die Möglichkeit, beide Varianten parallel zu implementieren. [Liu96, S. 207]

Dies zeigt, wie unterschiedlich das Konzept der Vererbung bereits innerhalb der OOP verstanden wird, obwohl dahingehend Einigkeit herrscht, dass gerade die Vererbung den Unterschied zwischen einer objektbasierten und einer objektorientierten Programmiersprache ausmacht [Wegn87, S. 169].

Deshalb gilt die Klassenvererbung oftmals nicht als probates Mittel und wird dann etwa durch „Objektkomposition" ersetzt. [GHJV96, S. 26 f.] Außerdem kann Vererbung auch ohne Klassen über Delegation an andere Objekte erfolgen, wie dies etwa bei der Prototypensprache Self der Fall ist. [KhAb95, S. 84] Während *Liu* keine Zweifel daran hat, dass trotzdem auch Self eine voll objektorientierte Programmiersprache ist [Liu96, S. 28], sieht *Heuer* die Notwendigkeit von Klassen als existenziell an und klassifiziert Self folglich als nicht-objektorientiert [Heue97, S. 190 f.].

Beck durchbricht das Vererbungskonzept durch die Realisierung von instanzspezifischem Verhalten. Er implementiert unterschiedliche Methoden für Objekte der selben Klasse. [Beck93a, S. 157 ff., Beck93b, S. 167 ff.] Tatsächlich gibt es auf der OOP-Ebene Argumente dafür, diese Beschränkung der Klassenvererbung zu durchbrechen. Auch konzeptionell kann es Gründe geben, dass Objekte derselben Klasse nicht nur unterschiedliche Attributwerte sondern auch unterschiedliches Verhalten aufweisen.

3.3.4.6 Abgrenzung

Die Verschiedenheit des Objekt-Paradigmas der OOP begründet sich in den abweichenden Zielsetzungen zu OOD und noch deutlicher zur OOA. Außer „Kapselung, Granularität, Abhängigkeit, Flexibilität, Laufzeitverhalten, Evolution, Wiederverwendbarkeit und so weiter" [GHJV96, S. 17] spielen konkret auch:

- „Elegance and readability of Code
- Brevitiy of Code
- Encapsulation and functionality
- Code reuse and development efficiency
- Flexibility of interface management
- Safe component access" [Howa95, S. 356 f.]

eine entscheidende Rolle. Der Code selbst (Lesbarkeit und Einfachheit), Effizienzgründe bezüglich der Entwicklung, Wartung, Erweiterbarkeit und der Laufzeit sowie die Wiederverwendung von Code, Daten und Klassen stehen im Vordergrund.

Während bei der OOP konfliktäre Zielsetzungen bezüglich der Verwirklichung von Typhierarchie und Klassenhierachie bestehen und wie eine solche jeweils auszugestalten bzw. zu realisieren ist, fokussiert sich das objektorientierte Design auf die Typvererbung und die OOA auf IS-KIND-OF-Hierarchien (IS-A, A-KIND-OF (AKO)) bzw. Spezialisierung und Generalisierung.

Heuer setzt IS-KIND-OF-Vererbung im OOP-Bereich mit Generalisierung und Spezialisierung gleich bzw. definiert letztere als zwei mögliche duale Ausprägungen der ersteren Form. Er sieht IS-KIND-OF-Beziehungen im wesentlichen als Möglichkeit zur Abbildung von Integritätsbedingungen. [Heue97, S. 187 ff.] *Liu* setzt die IS-KIND-OF-Verebung mit der Typvererbung gleich [Liu96, S. 198 ff.] und *Jacobson*, *Christerson*, *Jonsson* und *Övergaard* sehen diese als Vererbungsform auf konzeptioneller Ebene [JCJÖ95, S. 64 f.].

Auch darüber, welche Aufgabe einer Klasse beim Objekt-Paradigma zukommt, herrscht keine Einigkeit.

Meyer sieht Klassen als Implementierungsmöglichkeit von ADT. [Meye90, S. 65 ff.] „Die Implementierung eines Objekttyps wird durch eine Klasse vorgenommen, die eine Menge von Objekten mit gleichen Attributen und Methoden bezeichnet." [Heue97, S. 180]

Atkinson et al. sehen eine Klasse als Objektfabrik an. [ABD+89] „Eine Klasse ist eine Objektfabrik, in der während der Laufzeit beliebig viele Objekte hergestellt werden können, die dann jeweils Attributwerte und Methoden besitzen. Jeder Klasse wird ein Objekttyp zugeordnet, der die Attribute und Methoden der Klasse darstellt." [Heue97, S. 180]

Riehle setzt dem entgegen:

> „Es ist ein vergleichsweise neuer Trend, Klassen nur als Implementierungskonzepte und nicht auch als Modellierungskonzepte zu verstehen. Historisch betrachtet wurde in Simula-67 der Begriff des Objekts als 'konkretes Phänomen' und der Klasse als 'Abstraktion von konkreten Phänomenen' definiert. Ziel ist es, einen Realitätsausschnitt mithilfe von Objekten und Klassen zu modellieren." [GHJV96, S. 23]

Die unterschiedlichen Konzepte der OOPL wirken sich auch entsprechend auf die Umsetzung der Programme aus. Beispielsweise ist die Substituierbarkeit ohne Typhierarchien problematisch. Typisierte Sprachen bieten mehr Sicherheit, während untypisierte Sprachen mehr Flexibilität bieten. Dadurch werden auch die Formen des Polymorphismus geprägt, die dann jeweils sehr unterschiedlich sein können.

Für die Umsetzung von Constraints bzw. Geschäftsregeln gibt es in der Regel keine Konzepte bei OOPL. Und auch für die Aggregationssemantik werden in OOPL in den meisten Fällen keine Konzepte angeboten [King89, S. 26], so dass diese durch Assoziation ersetzt werden müssen, obwohl gerade die Aggregation

als wesentliches Konzept der Objektorientierung gesehen wird und, anders als die Assoziation, die eine Objekt-Beziehung darstellt, zum Teil als Klassenbeziehung interpetiert wird [WWW90, S. 70, Heue97, S. 275 ff.].

3.3.5 Vorgehens-Methodik

Vorgehens-Methodiken haben ebenfalls Einfluss auf das Objekt-Paradigma. Der grundlegendste Unterschied ist der, dass das klassische phasenweise Vorgehen wie im Wasserfallmodell bei der Entwicklung objektorientierter Software oftmals in radikaler Form abgelehnt wird. [Beck00, S. 3 ff., CoSh95, S. 1 f.] Die Einheitlichkeit bzw. Durchgängigkeit des Objektmodells von OOA, OOD und OOP [CoYo96, S. 47 ff., CoYo94, S. 37 ff.] im Gegensatz zu den Strukturbrüchen bei SA, SD und strukturierter Programmierung begünstigen diesen Prozess, was auch Rücksprünge in frühere Phasen und die Zerlegung nach Komponenten begünstigt. [Your93, S. 92 ff.]

Hierbei lassen sich verschiedene Formen unterscheiden. *Conell* und *Shafer* [CoSh95, S. 1 ff.] sowie *Pomberger* und *Blaschek* [PoBl96, S. 1 ff.] heben in besonderem Maße die Bedeutung des Prototyping für die Entwicklung hervor. Kurze Entwicklungszyklen zwischen den einzelnen Versionen der Prototypen, eine enge Benutzerkopplung — insbesondere über Benutzeroberflächen — und die Zerlegung des Entwicklungsprozesses nach Subsystemen anstelle von Phasen sind die grundlegenden Merkmale.

Pomberger und *Blaschek* nennen als weitere wesentliche Unterschiede zwischen dem objektorientierten Software Engineering in Abgrenzung zur konventionellen Entwicklung [PoBl96, S. 30 f.]:

- Design und Implementierung sind keine getrennten Prozesse mehr, da untersucht werden muss, welche Komponenten zur Lösung der Probleme bereits vorhanden sind und das Design je nach Wahl der Programmiersprache zu völlig verschiedenen Ergebnissen führt.
- Die Länge der Implementierungsphase ist reduziert, so dass Teilprodukte früher fertiggestellt sind, um getestet werden zu können, wodurch wiederum früher Feedback über die Korrektheit von Design und Implementierung erfolgt und führt in der Folge früher zu einer Korrektur der Fehler.
- Die Konzentration muss nicht nur auf das zu erstellende Produkt, sondern auch auf die Pflege und Wartung der Klassenbibliothek durch den Klassen-Bibliothekar erfolgen.
- Die Auswirkungen von Änderungen an der Klassenbibliothek auch auf andere Anwendungen muss beachtet werden.

- Die bei der Entwicklung einer Anwendung entstandenen Klassen müssen, sofern sinnvoll, verallgemeinert und für andere Entwickler nutzbar gemacht werden.

Dabei tritt besonders die Bedeutung der Wiederverwendung hervor. Beachtet werden müssen sowohl die Verwendung der vorhandenen als auch Erzeugung neuer Komponenten. Solch mögliche Komponenten können einzelne Klassen, Klassenbibliotheken, Frameworks, Patterns oder Design-Heuristiken sein. *Lorenz* nennt als mögliche Komponenten: „Base Classes", Frameworks", „Business Domain Components" und „Object Models". [Lore95, S. 2]

Conell und *Shafer* grenzen ihren Ansatz sowohl von „Prespecification" wie auch vom „Hacking" ab. [CoSh95, S. 1 f.] Letzterer entspricht beispielsweise dem Vorgehen des *beckschen* Extreme Programming (XP), welches sich ebenfalls deutlich von der Notwendigkeit zu „Prespecification" abgrenzt [ABB+98, S. 25 ff., Beck00, S. 43 ff.].

Beck bezeichnet den Modellierungsprozess als unzweckmäßig und fängt deshalb verzögerungsfrei mit der Implementierung an. Die Testphase wird mit der Implementierungsphase kombiniert, indem zu jedem Teil des Systems (z. B. einer Klasse) Tests implementiert werden (z. B. wird noch vor der eigentlichen Funktion eine Testmethode geschrieben), wobei *Beck* zwischen Unit Testing, Integration Testing und Running Tests unterscheidet. [Beck94c, S. 278 f., Beck99a]

Extreme Programming ist implementierungs- und sogar sprachen-zentriert [Yip99]. Das Vorgehen besteht aus den vier Aktivitäten [Beck00, S. 43 ff.]:

- Coding
- Testing
- Listening
- Designing (Refactoring)

Beck betreibt mithin echtes „Hacking". Er codiert ein Stück, führt Tests durch, holt sich Rückkopplung beim Benutzer und versucht schließlich, das Ergebnis dieses Prozesses in sauberes Design zu wandeln. [Beck00, S. 43 ff.] Wesentlich ist auch, dass *Beck* im Unterschied zur Idee der Entwurfsmuster — die möglichst viel Code-Flexibilität erbringen sollen, da Änderungen von *Gamma*, *Helm*, *Johnson* und *Vlissides* als sehr teuer angesehen werden [GHJV96, S. 33] — empfiehlt, nie mehr als im jeweiligen Moment notwendig zu implementieren: „To go fast, we build only what we really need, thus keeping the system lean and mean." [ABB+98, S. 26]

Extreme Programming verwendet einen inkrementellen Ansatz, wird als „just-in-time" software development bezeichnet, vermeidet im Gegensatz zum Ansatz von *Conell* und *Shafer* die Implementierung der Benutzeroberfläche solange es mög-

lich ist und hält formales OOA und OOD für genauso wertlos wie eine explizite Dokumentation. [Beck00, S. 43 ff.]

Das zeigt, dass Vorgehens-Methodiken einen erheblichen Einfluss auf das Objekt-Paradigma haben. Deshalb sind die vorgestellten Ansätze — die auch aufgrund ihrer Bedeutung in der Praxis [ABB+98, S. 25 ff.] als repräsentativ angesehen werden können — im Sinne des Objekt-Paradigmas äußerst kritisch zu beurteilen. Der Verzicht, je nach Vorgehens-Methodik, auf OOA, OOD, Entwurf, Dokumentation und zum Teil sogar methodisches Vorgehen überhaupt, steht dem objektorientierten Software-Entwicklungs-Prozess im Sinne der hier diskutierten OOA-, OOD- und OOP-Paradigmata entgegen.

Khoshafian und *Abnous* halten die gründliche Modellierung eines Systems und damit auch OOA und OOD für essenzielle Voraussetzungen eines qualitativen Systems. Sie meinen [KhAB95, S. 12]:

> „It is interesting to notice that when it comes to manufacturing 'hardware,' usually there is a very serious effort to make sure the design specifications are detailed, complete, and precise. The cost of manufacturing a product is just too high — especially for large items such as automobiles or airplanes. Unfortunately, when it comes to software, there is still — even after a couple decades of object-oriented programming — a temptation to launch into hacking and implementation before designing all the details.
>
> This very serious software engineering issue cannot be overemphasized. By its very nature, the 'cost' for developing software and the satisfaction of having 'something that runs or executes' often means that software engineers rush to implement the system — and they pay the cost of implementation without design later (bug fixes, maintainability, performance tuning, etc.). In the earlier days of object-orientation, the potential was to have software integrated circuits. It was hoped and dreamed that just as hardware integrated circuits have well-specified interfaces and components that fit together, software too can be constructed with very well defined interfaces and modules that fit together perfectly in large, complex systems. Unfortunately, in many situations where software is developed, the reality is quite the opposite. […] The common features in almost all the rapidly hacked systems are almost always terrible design flaws!"

Insgesamt bieten objektorientierte Vorgehens-Methodiken ein erhebliches zusätzliches Potenzial für den Software-Entwicklungs-Prozess. Die durchgängige Sichtweise von OOA bis OOP ermöglicht inkrementelles Vorgehen, einfache Rücksprünge in frühere Phasen, leichte Erweiterbarkeit und verstärkte Wiederverwendung als impliziten Bestandteil des Software-Entwicklungs-Prozesses. Dieses Potenzial gilt es zu nutzen — „Hacking" wie beim Extreme Programming oder den Verzicht auf „Prespecifications" wie bei *Connel* und *Shafer* darf es nicht geben.

3.4 Applikations-Schichten

Nach einem Überblick über verschiedene Formen der Verteilung und einer damit verbundenen Begriffsklärung, erfolgt die Diskussion anhand der Applikations-Schichten grafische Benutzeroberfläche, Objectbase Management System (OBMS) und Applikations-Schicht. Der Schwerpunkt liegt auf dem OBMS-Abschnitt, da OBMS auch den Schwerpunkt dieser Arbeit bilden.

3.4.1 Verteilung

Als ein weiteres wichtiges Merkmal des Objekt-Paradigmas wird die Verteilung (Distribution) angesehen. [ABD+89, CaBa98, S. 2] Damit wird üblicherweise eine Verteilung von Datenbanken gemeint, die zwischenzeitlich auch bei RDBMS als wichtige Funktionalität vorausgesetzt wird. Verteilung betrifft aber viele unterschiedliche Ebenen, die je nach Ausprägung ganz erheblichen Einfluss auf das Objekt-Paradigma aufweisen.

- Es gibt eine physische Verteilung der Datenbestände in einem LAN (z. B. abteilungs-spezifisch) oder auch über ein WAN (z. B. zwischen zwei Unternehmens-Filialen in Berlin und Tokio). [Tayl92, S. 284 ff.] In diesem Fall gibt es trotzdem ein gemeinsames logisches Datenmodell. [PlBr98, S. 413]
- Es gibt eine Verteilung zwischen unterschiedlicher Hardware (z. B. INTEL, RS/6000), die auch von der jeweiligen Bauart abhängt (z. B. RS/6000 43 CT, RS/6000 230) und Software, wie Betriebssystemsoftware (z. B. Windows 2000, MacOS 10, AIX 4.3), Netzwerkbetriebssystemsoftware (z. B. Novell 4.1, Windows 2000), Protokollen (z. B. TCP/IP, SPX/IPX) etc. [Khos93, S. 313]
- Es gibt eine Verteilung auf unterschiedliche DBMS und/oder Modelle (z. B. GemStone, Objectivity). [Khos93, S. 313]
- Es finden unterschiedliche OOP (Java, Smalltalk, C++) für die Erzeugung und Manipulation der Objekte Verwendung. Es ist etwa möglich, Objekte von Anwendungen in verschiedenen Sprachen gleichzeitig in einem OBMS zu speichern (z. B. Objectivity) [Obje97, S. 8-1 ff.] aber auch die OBMS können speziell auf eine Sprache ausgerichtet sein (z. B. Gemstone/S, GemStone/J). [Tayl92, S. 253]
- Die Objekte einer Anwendung werden in Schichten getrennt. Es gibt mindestens drei Schichten, auf welche die Objekte verteilt werden: Domänen-Objekt-Schicht, Applikations-Schicht (Verbindungsschicht) und GUI-Schicht. [Howa-95, S. 375 ff.] Diese Schichten wiederum befinden sich vollständig im Client (z. B. GUI-Objekte), vollständig im Server (z. B. Domänen-Objekte) oder deren Objekte werden zwischen Client und Server verteilt.

- Es gibt eine Verteilung bei der Abarbeitung von Anfragen zwischen Client und Server. Bei SQL ist es in der Regel üblich, aus Effizienzgründen (leistungsfähiger Server, niedrige Netzbelastung) eine Anfrage vom Server abarbeiten zu lassen und das Ergebnis an den Client zu senden. Es gibt aber auch die Variante, die Objekte in den Client zu laden und die Anfrage dort abarbeiten zu lassen. Damit zusammen hängt auch die Ausführung von den Methoden der Objekte. In vielen Fällen fungieren OBMS bisher nur als reine Datenspeicher, die die Methoden der Objekte weder speichern noch ausführen können (z. B. Objectivity).[100] [Obje97, S. 2-8 ff.]

Die klassische Trennung einer Anwendung in Programm (Funktionen bzw. Algorithmen) im Client und Daten auf dem Server (DBMS) ist damit in weiten Teilen aufgehoben. Auch hierbei ist außer der Festlegung, welche Objekte persistent zu machen sind, entscheidend, wie die Anwendung möglichst effizient und flexibel zu realisieren ist.

Das Ziel einer applikations-unabhängigen Datenbasis sowie logischer und teilweise sogar physischer Datenunabhängigkeit, kann damit nicht mehr im selben Maße aufrechterhalten werden wie dies bei der konventionellen Software-Entwicklung (RDBMS) der Fall ist. Zumindest die persistenten Objekte des OBMS müssen analog zu den Daten der Relationenschemata Interapplikationsfunktionalität realisieren.

Da das Grundprinzip des Objekt-Paradigmas die Modularisierung von zusammengehörigen Daten und Funktionen ist, kann es keine Trennung in Programm-Funktionen und DBMS-Daten geben. Daraus ergibt sich insgesamt eine von der konventionellen Programmierung erheblich abweichende Architektur der Anwendungslandschaft.[101]

Aufgrund der Bedeutung der Schichten einer objektorientierten Applikation, werden im Folgenden die unterschiedlichen Objekt-Paradigmata dieser Schichten betrachtet werden. Die Einteilung erfolgt in die Schichten grafische Benutzeroberfläche, Objectbase Management System und Applikations-Schicht.

3.4.2 Grafische Benutzeroberflächen

Bezüglich grafischer Benutzeroberflächen (Graphical User Interfaces, GUI) herrscht in hohem Maße Unklarheit darüber, wann eine solche GUI als eine ob-

[100] Objekte verschiedener Programmiersprachen parallel in einem OBMS zu speichern, deren Methoden gleichzeitig ausführen zu können und diese eventuell auch noch „kommunizieren“ zu lassen, ist aufgrund der sehr unterschiedlichen Konzepte der OOPL bisher schwer vorstellbar.

[101] Inwieweit dann überhaupt noch von einzelnen Anwendungen gesprochen werden kann, ist fraglich. Auch der Klassenhierarchie-Ansatz von Smalltalk zeigt bereits, wie schwierig die Herauslösung einer bestimmten Anwendung aus dieser Art der Architektur zu bewerkstelligen ist.

jektorientierte grafische Benutzeroberfläche bezeichnet werden kann. Auf der anderen Seite sind GUI ein fundamentaler Bestandteil des Objekt-Paradigmas [Kay93, S. 17 ff.] und verdeutlichen auf besondere Weise dessen Konzepte.

Die Hauptursache für die Missverständnisse dafür, unter welchen Bedingungen eine Oberfläche als objektorientiert gilt und wann dies nicht der Fall ist, liegt in der üblicherweise fehlenden aber fundamentalen Unterscheidung zwischen einer konzeptionell objektorientierten Oberfläche und einer Oberfläche, die objektorientiert realisiert ist. [102] Aus diesem Grund werden diese beiden Alternativen im Folgenden differenziert betrachtet.

3.4.2.1 Konzeptionell objektorientiert

Balzert trennt — analog zur System-Entwicklung — zwischen funktions-orientierter und objektorientierter Bedienung. Bei der funktions-orientierten Bedienung wählt der Benutzer eine Funktion oder eine Anwendung mit ihren Funktionen aus und bestimmt anschließend, auf welchen Objekten diese Funktion oder Anwendung ausgeführt werden soll. Bei der objektorientierten Bedienung wählt der Benutzer erst das Objekt und dann eine Funktion aus, die auf diesem Objekt ausgeführt werden kann. „Das Objekt bestimmt sozusagen mit seinen Eigenschaften, welche Funktionen bzw. Operationen auf ihm ausgeführt werden können.“ [Balz96, S. 468]

Ein Beispiel für eine funktions-orientierte Benutzerschnittstelle ist etwa eine UNIX-Shell (z. B. TC-Shell), bei welcher der Benutzer einen Befehl (Funktion) mit seinen Parametern (Daten) und oftmals unzähligen Optionen, die er auswendig kennen muss, an einem Kommandoprompt eingibt. Baan IV bietet eine grafische Benutzeroberfläche die funktions-orientiert ist. Analog zur funktionsorientierten Modularisierung der Software, sind auch die Menüs des GUI in einem hierarchischen Baum angeordnet. Die dementsprechend aufzurufenden Fenster bieten dann die Möglichkeit, die Daten zu bearbeiten.

Diese Differenzierung von *Balzert* greift allerdings zu kurz. Mit der Bezeichnung eines konzeptionell objektorientierten GUI wird mehr verbunden. Die wirklich grundlegende Idee war hierbei wieder der Versuch, eine möglichst nahe Abbildung der Realität zu verwirklichen. Im Gegensatz zu funktions-orientierten Benutzeroberflächen, die sich nach der von-Neumann-Architektur ausrichten

[102] Dieser Mangel der Unterscheidung trifft ebenso auf Software-Produkte insgesamt zu, was bedeutet, dass viele Produkte ein objektorientiertes „Label“ bekommen, weil sie über eine konzeptionell objektorientierte Oberfläche verfügen. Wird weiterhin bedacht, dass GUI an sich als konzeptionell objektorientiert gelten — was auf einen Großteil der heutigen GUI zutrifft — und fast jede kommerzielle Software über ein GUI verfügt, gibt es inzwischen fast ausschließlich objektorientierte Software. Dies wäre jedoch eine den umfangreichen Konzepten des Objekt-Paradigmas wenig dienliche Definition.

[Khos93, S. 7], sollten Bedienoberflächen geschaffen werden, die Metaphern der realen Welt des Benutzers sind und so eine natürliche Bedienung ermöglichen.

Primäres Ziel von *Kay* war es einen Computer für Kinder zu entwickeln. [Kay93, S. ff.] Vor diesem Hintergrund ist verständlich, wenn bei objektorientierten GUI auch immer wieder der Spielcharakter hervorgehoben wird.

Orfali, *Harkey* und *Edwards* meinen dementsprechend:

> „You're […] invited to play with it. […] The OOUI […] is very familiar, especially to kids. It feels like a video game. Kids feel quite at home with drag-and-drop, icons and direct manipulation. The OOUI is a simulation of reality that they can easily recognize." [OHE96, S. 71]

Heuer unterscheidet fünf grundlegende Eigenschaften des von *Kay* und seinen Kollegen entwickelten Konzepts (Xerox „Alto" und „Star") [Heue97, S. 157 f.]:

- „Schreibtisch-Philosophie mit Objekten wie Dokumenten, Ordnern, Eingangskorb und Papierkorb.
- Generische Kommandos; so ist etwa das Kopieren von Dokumenten und Ordnern mit dem gleichen Kommando zu bewerkstelligen.
- Direkte Manipulation von Objekten durch Anklicken, Verschieben und Vergrößern.
- Darstellung von Objekten durch Icons.
- Objekte haben Eigenschaften, die vom Benutzer verändert werden können und die sich dieses Objekt 'merkt'.[103] "

Khoshafian und *Abnous* ergänzen [KhAb95, S. 381 ff.]:

- Pointing devices (selecting, moving, merging)
- Windows (overlapped, tiled, iconified)
- Dialog boxes

Grundlegend für das Objekt-Paradigma von GUI ist also die Simulation der realen Welt durch Objekte und Verhalten (Funktionen).

Die Objekte werden durch aussagekräftige Bilder, wie etwa Icons oder Grafiken repräsentiert. Alle Objekte haben lokale Eigenschaften wie einen Namen, einen

---

[103] *Heuer* sieht in vielen Fällen eine Verletzung dieses Punktes. Etwa können Voreinstellungen einer Grafik nur auf Anwendungsebene verändert werden. Die Änderungen betreffen dann allerdings auch alle anderen Grafiken oder können nicht persistent gehalten werden. Die Grafikeigenschaften sind mithin keine Objekteigenschaften. Damit übernimmt die Benutzerschnittstelle nur das Aussehen, nicht jedoch den objektorientierten Unterbau der Xerox-Oberfläche. *Heuer* meint: „Eine Benutzerschnittstelle mit einem derartigen Verhalten kann eigentlich nicht als objektorientiert bezeichnet werden, da sie eines […] der wesentlichen Prinzipien verletzt. Leider scheint es bei der Vergabe des Prädikats 'objektorientiert' oft nur darauf anzukommen, ob irgendwelche Objekte in diesem System angeklickt werden können." [Heue97, S. 158 f.]

Typ und weitere Attribute, die durch Voreinstellungen (Defaultwerte) belegt werden, jedoch verändert werden können, d. h. durch andere Werte ersetzt werden können. [Heue97, S. 157 ff.] Hierfür ist die Schreibtisch-Metapher ein gutes Beispiel, das sich in dieser Form auch auf andere Anwendungen übertragen lässt.

Bei der Simulation des Verhaltens werden Kommandos nicht überflüssig, sind aber weniger bedeutend. So meint *Balzert* über die objektorientierte Anwendungsentwicklung:

> „Die Interaktion mit der Anwendung erfolgt […] fast ausschließlich über direkte Manipulation. Die Menüauswahl über Menübalken, pull-down-Menüs und Druckknöpfe verliert zugunsten der direkten Manipulation an Bedeutung." [Balz96, S. 495]

Vieles lässt sich durch die direkte Manipulation erheblich einfacher bewerkstelligen, als wenn dafür Befehle notwendig wären — statt sich genau auszudrücken, was der Benutzer erreichen möchte, „vollzieht" er die Handlung einfach. Werden die Prinzipien der direkten Manipulation verstanden, ist das Erlernen der Bedienung auch von neuer Software weniger problematisch, da das bereits Erlernte wiederverwendet werden kann. Werden Kommandos verwendet, wird versucht den Polymorphismus auch hier anzuwenden, indem soweit dies möglich ist, generische Kommandos verwendet werden.

Die Konzeption der objektorientierten GUI entspricht damit im Wesentlichen der Zielsetzung der OOA:

- Auch der Kommunikations-Aspekt spielt eine wesentliche Rolle. *Khoshafian* und *Abnous* betonen deshalb auch die besondere Bedeutung von Dialogboxen, um Rückmeldungen zu geben oder die notwendigen Attribute und Optionen abzufragen [KhAb95, S. 385] — das Objekt selbst verfügt über dieses Wissen. Auch *Orfali*, *Harkey* und *Edwards* meinen: „OOUIs have an insatiable appetite for communications." Die Objekte kommunizieren mit anderen GUI-Objekten, mit der Anwendung und dem Benutzer. [OHE96, S. 69]
- Nebenläufigkeit wird durch die unterschiedlichen Fenster repräsentiert. Die Fenster können sich überlappen und in allen diesen Fenstern können parallel Prozesse ablaufen.
- Auch die Unterscheidung zwischen den einzelnen Applikationen sowie die Unterscheidung zwischen den Applikationen und dem Betriebssystem verwischt zusehends. Die funktionale Aufteilung in Applikationen entfällt damit zusehends. [OHE96, S. 72 f.]
- Auch Vererbung spielt eine Rolle. Objekte werden analog zur OOA in Klassen eingeteilt. Außerdem kommt häufig noch eine andere Art der Vererbung hinzu, die dem Protoypen-Prinzip von Self entspricht. Beispielsweise können Oberflächen von anderen Oberflächen abgeleitet werden (erben) und erben dann auch deren Eigenschaften. Werden die Eigenschaften der Oberfläche ge-

ändert, von der abgeleitet wurde, ändern sich auch die Eigenschaften der zweiten Oberfläche oder besser: die Eigenschaften der Oberflächen-Objekte.

3.4.2.2 Objektorientiert realisiert

Eine objektorientierte Benutzerschnittstelle ist also immer ein GUI. Ein konzeptionell objektorientiertes GUI muss aber nicht unbedingt objektorientiert realisiert sein. *Balzert* hält etwa Motif für flexibel genug, um damit ein objektorientiertes GUI realisieren zu können. [Balz96, S. 469] Auch GUI wie Windows 95 werden teilweise als objektorientiert klassifiziert [KhAb95, S. 390 ff., OHE96, S. 69], obwohl sie nicht objektorientiert realisiert sind. Selbst die Oberfläche des Xerox „Star" ist nicht in einer OOPL implementiert worden. [Heue97, S. 157 ff.]

Eine Oberfäche ist dann objektorientiert realisiert, wenn eine OOPL derart verwendet wird, dass auch die Oberflächen-Elemente aktive autonome Objekte sind. „Everything that you see is an object." [OHE96, S. 72] Beim Model-View-Controller-Konzept (MVC) bzw. ValueModel Framework [Wool94, S. 469 ff.] von VisualWorks etwa sind alle Elemente vollständige Objekte, wie Widgets, Views oder Controller, die während des Programmablaufs über Nachrichten kommunizieren.

Da alle Objekte — bis zu den atomaren Objekten — wieder aus Objekten bestehen, können auch Objekte auf allen Ebenen Nachrichten empfangen. Beispielsweise kann ein Widget (z. B. ein Eingabefeld) von einem beliebigen anderen Objekt die Nachricht „beInvisible", „disable", „moveBy: aPoint" oder etwa „changeColorTo: aColor" erhalten. Ändert sich der Wert des Eingabefelds, bekommt das Widget die Nachricht „update".[104] Muss ein Oberflächenelement neu gezeichnet werden, erhält es die Nachricht „invalidate" oder „invalidate: aRectangle". Welche Aktionen dann jeweils weiter notwendig sind, „wissen" die Objekte selbst.

Abhängig von anderen Objekten, können sogar ganze Unterfenster (Subcanvases) oder Menüs erweitert, verschoben oder vollständig ausgetauscht werden. Das gesamte MVC-Konzept basiert auf der Unterscheidung von Model-, View- und Controller-Objekten, die über Nachrichten kommunizieren, um sich gegenseitig über Veränderungen zu informieren, wodurch auch die saubere Trennung

[104] Das wesentliche an diesem Prinzip ist, dass, vereinfacht, ein Domänen-Objekt nicht einfach das sich geänderte Attribut beim View neu setzt. Das Domänen-Objekt kennt den View auch nicht unmittelbar. Stattdessen registriert sich der View in einer Benachrichtigungsliste des Domänen-Objekts. Ändert sich das Domänen-Objekt dann, bekommen einfach alle Objekte in der Benachrichtungsliste eine Nachricht, was sich geändert hat. Wie die benachrichtigten Objekte diese Nachricht dann behandeln, ist ihnen selbst überlassen. Hat sich beispielsweise etwas für den View Relevantes geändert, muss dieser wiederum das Domänen-Objekt nach dem neuen Wert befragen, den Wert dann aber selbst neu setzen und auch selber dafür sorgen, dass er — der View — neu gezeichnet wird (wie auch immer das dann wieder geht).

von Domänen-Objekten und Oberflächen-Objekten ermöglicht wird. Deshalb meint *Howard*:

> „[…] for it is the lever that transforms VisualWorks from just another GUI builder into the most dynamic, flexible, and extendible application development system available today.“ [Howa95, S. 3]

Es gibt allerdings auch hier wieder ganz andere Ansätze. *Holub* hält diesen Ansatz des MVC für ungeeignet und stellt damit das gesamte Design einer objektorientierten Applikation infrage. Er verlangt fünf Punkte, damit überhaupt von einer objektorientierten Anwendung gesprochen werden kann: „If the system doesn't follow these rules, it isn't object-oriented. It's that simple.“ [Holu99]:

1. „All data is private. Period. (This rule applies to all implementation details, not just the data.)
2. **get** and **set** functions are evil. (They're just elaborate ways to make the data public.)
3. Never ask an object for the information you need to do something: rather, ask the object that has the information to do the work for you.
4. It must be possible to make any change to the way an object is implemented, no matter how significant that change may be, by modifying the single class that defines that object.
5. All objects must provide their own user interface.“

Außerdem hält *Holub* die Betrachtung von Klassen für irrelevant, da sie einzig für den Compiler von Interesse seien. Gegenstand der Betrachtung sind einzelne Objekte oder Objekt-Klassen im Sinne von Mengen. [Holu99]

Die ersten vier Punkte erfordern prinzipiell nichts Neues. Kapselung ist das Grundprinzip. Objekte sind einzig über ihre Schnittstelle definiert. *Holub* verlangt, die Struktur eines Objekts beliebig verändern zu können, ohne dadurch die Notwendigkeit der Änderung des Codes bei den dieses Objekt benutzenden Objekten nach sich zu ziehen. [Holu99]

Um diesem Ziel noch näher kommen zu können, verlangt *Holub* mit Punkt fünf zusätzlich die direkte Kopplung von Domänen-Objekt und GUI-Objekt. Dieser Punkt würde die Domänen-Objekte — ganz im Sinne des Objekt-Paradigmas — noch autonomer und „natürlicher“ machen. Beispielsweise könnte dann ein Auftrags-Objekt seine Initialisierung vollständig selbst durchführen, indem es sein eigenes GUI-Objekt zur Eingabe der Daten erzeugt (Dialogbox) und dadurch in direkte Kommunikation mit dem Benutzer tritt.

Beim MVC-Konzept muss diese Aufgabe durch eine Zwischenschicht, die Applikations-Schicht erfolgen (siehe Abschnitt 3.4.4); die Attribute der Domänen-Ob-

jekte werden über Accessing-Methoden gesetzt (bei VisualWorks über ValueModel als Adapter [Wool94, S. 481]).

Holub erklärt seine Konzeption aus Sicht der GUI-Konstruktion: „This article […] looks at the underlying theory of object-oriented design from the perspective of the user interface." [Holu99] Außerdem beschreibt er dieses Paradigma im Zusammenhang mit Java. Tatsächlich kann es in Verbindung mit Java eine andere Sicht auf die Applikations-Entwicklung geben, da durch die Internet/Browser-Applikationen andere Architekturen notwendig werden.

Für Client/Server-Informationssysteme scheint sein Ansatz dagegen wenig geeignet. Die Trennung zwischen GUI-Objekten und Domänen-Objekten verwischt.

Holub lässt offen, wie mit seinem Konzept verschiedene Views (GUI) für ein Model (Domänen-Objekt) erstellt werden können. Dies wird benötigt, um verschiedene Präsentationen des Models anzubieten oder unterschiedliche Clients auf dem selben Domänen-Modell (Hardware, Betriebssystem, OOPL, GUI-Builder) entwickeln zu können. [GHJV96, S. 5 ff.]

Unklar bleibt auch, auf welcher Aggregationsstufe Objekte ihre eigenen Views realisieren sollen. Für eine erhöhte Wiederverwendung werden Views sinnvollerweise nicht aus einzelnen Attributen, sondern aus Unterviews (Subcanvases) und diese wiederum aus Unterviews zusammengesetzt.

Außerdem können Oberflächen nicht immer nach der Struktur der Domänen-Objekte gestaltet werden. Viele Attribute sind in einem bestimmten View überflüssig oder dürfen etwa aus Datenschutzgründen im Sinne von Stellen-Kompetenzen nicht angezeigt werden. Views eines Informationssystems werden zudem in der Regel nach Aufgaben gestaltet. Ein Telefonverkäufer möchte, als ein Beispiel, genau die Informationen auf einem View, die er für die Auftragsannahme benötigt — nicht mehr und nicht weniger. Aus welchen Domänen-Objekten dieser View besteht, ist für ihn von untergeordnetem Interesse.

Hierbei besteht auch eine direkte Rückwirkung auf die Modellierung der Objekte. *Holubs* Ansatz verleitet dazu, Objekte nach den Anforderungen eines Views zu modellieren. Dies widerspricht allerdings den Modellierungs-Grundsätzen des Objekt-Paradigmas. Dieselben Objekte müssen Attribute und Funktionen für verschiedene (alle betroffenen) Applikationen implementieren.

Wie von *Rieger*, *Krüger* und *von Maur* in einem ähnlichen Ansatz unter Verwendung des Morph-Paradigmas von Squeak gezeigt, ist dieses Konzept zum Verständnis des Objekt-Paradigmas in der Lehre sinnvoll einzusetzen [RKM98, S. 238 ff.], für kommerzielle Anwendungen jedoch weniger geeignet.

Howard fasst die Notwendigkeit der Trennung von Domänen-Objekten und GUI-Objekten wie folgt zusammen:

„This is absolutely essential if we are to build distributed, production worthy, client/-server applications with a persistent store. This is because it is the domain information that persists in a database while the application information resides entirely in the client." [Howa95, S. 375]

3.4.3 Objectbase Management Systems

Objectbase Management Systems (OBMS) sind seit einigen Jahren ein Forschungsschwerpunkt der Informatik [Heue97, S. 9 ff., SST97, S. 1] und dieses ausgeprägte Interesse an OBMS wird auch in den nächsten Jahren ungemindert anhalten. Im Bereich der Database Management Systems gibt es seit nunmehr über dreißig Jahren [Voss94, S. 3] die Bestrebung, Daten langfristig zu speichern, um diese für verschiedene Anwendungsprogramme bereitzustellen und ein effektives Information Retrieval zu ermöglichen. Dafür sind ein „möglichst anwendernahes Abbilden der Diskurswelt" [SST97, S. 56], Datenintegration, Datenkonsistenz und die Effizienz des DBMS maßgebliche Zielkriterien.

Wie in den anderen Bereichen der Softwareentwicklung gibt es auch bei den DBMS sehr unterschiedliche Paradigmata (Netzwerkmodell, semantische Datenmodelle, aktive DBMS, deduktive DBMS, multimediale DBMS, multidimensionale DBMS etc.). Die Übertragung des Objekt-Paradigmas auf diese DBMS-Paradigmata, bietet für viele dieser Ansätze einen adäquaten Lösungsansatz und verspricht sogar möglicherweise einen integrativen Ansatz realisieren zu können. Es führt aber auch dazu, dass die Bandbreite der unterschiedlichen Konzepte innerhalb des OBMS-Paradigmas noch weit größer ist, als dies beispielsweise bei den OOPL der Fall ist. Bedenkt man weiterhin, dass OBMS in der Regel keine alleinstehenden Systeme darstellen, sondern etwa mit OOPL gekoppelt sind, werden die gegenseitigen Interdependenzen der Objekt-Paradigmata deutlich und führen zu einer kaum noch zu überblickenden Vielfalt an Konzepten.

Es lassen sich zwei Hauptrichtungen unterscheiden, die auf das OBMS-Paradigma Einfluss nehmen. Dies sind zum einen die Anforderungen der objektorientierten Programmierung an ein DBMS und zum anderen die Konzepte von Database Management Systems, die mithilfe der Objekt-Technologie verbessert werden sollen.

Dementsprechend meinen *Saake*, *Schmitt* und *Türker*:

„Objektdatenbanksysteme sind zur Zeit ein wichtiges Forschungs- und Entwicklungsgebiet […] Ihre Entwicklung resultiert in erster Linie aus der Idee, das Objektparadigma auch auf Datenbanksysteme anzuwenden, um bestehende Probleme beim Bestehen herkömmlicher, etablierter Datenbanksysteme zu lösen." [SST97, S. 1]

Die Anforderungen an OBMS aus Sicht der OOP unterscheiden sich in vielfacher Weise von denen aus Sicht der DBMS, weshalb die Konzepte dieser prinzipiell

unterschiedlichen Blickwinkel in den folgenden zwei Abschnitten diskutiert werden. Die Konzepte der Evolution und Versionierung von Objekten bzw. des Schemas, der Sichten, der Realisierung von Persistenz, der Typen, Klassen und Vererbung sowie der Anfragesprache werden aufgrund ihrer Bedeutung gesondert behandelt. Abschließend erfolgt die Diskussion über die OBMS-Architektur und die Definition des OBMS-Begriffs.

3.4.3.1 OOP-Blickwinkel

Aus der Sicht eines Anwendungsprogrammierers konventioneller Programmiersprachen erfüllen DBMS primär die Aufgabe der Speicherung von Daten, die die Anwendungsdauer überstehen müssen. Bezogen auf OOPL müssen, analog zu den Daten, Objekte gespeichert werden können. Wie *Manola* meint:

> „… an ODBMS is a DBMS that stores not data, but **objects**: encapsulated combinations of data structures together with associated procedures (methods).“ [Mano94, S. 213]

Der wirkliche Durchbruch in der Datenbanktechnologie gelang mit den relationalen DBMS. Anstelle der engen Kopplung zwischen Anwendungs-Programmiersprache (Application Programming Language, APL [Catt91, S. 120 ff.]) und Datenbank-Programmiersprache (Database Programming Language, DPL)[105] wie noch bei hierarchischen und Netzwerk-DBMS, wird bei RDBMS nicht nur eine Trennung von Daten und Anwendung, sondern vor allem eine strikte Trennung zwischen APL und DPL vorgenommen.

Konventionelle (prozedurale) APL sind zur Verarbeitung von Algorithmen optimiert, DPL für Daten. APL verfügen über umfangreiche Sprach- und Typkonstrukte (z. B. Entscheidungen, Schleifen, Zeiger[106], verschachtelbare Tupel- und

---

[105] DPL wird hier zur Abgrenzung gegenüber APL verwendet. *Cattell* benutzt anstelle von DPL die Bezeichnung Extended Database Query Language. [Catt91] Während eine APL eine Programmiersprache wie Pascal oder C++ ist, besteht eine DPL aus mehreren Bestandteilen. Die ODMG unterscheidet eine Object Definition Language (ODL), Object Manipulation Language (OML) und Object Query Language (OQL). [CaBa98, S. 83 ff.] *Elmasri* und *Navathe* differenzieren zwischen einer Data Definition Language (DDL), die wiederum in eine DDL zur Definition des konzeptuellen Schemas und einer Storage Definition Language (SDL) zur Definition des internen Schemas unterschieden wird, einer View Definition Language (VDL) zur Definition des externen Schemas und einer Data Manipulation Language (DML). SQL bildet jedoch keine Query Language im Sinne einer Anfragesprache — etwa Analog zur OQL der ODMG —, sondern eine Kombination aus DDL, SDL, VDL und DML. Eine Anfragesprache bezeichnen sie als Query Language, wenn diese deklarativ ist und keine Updates auf den Datenbestand erlaubt. Eine APL bezeichnen sie als Host Language (Wirtssprache [LaLo95, S. 27]), eine DPL als Data Sublanguage. [ElNa94, S. 30] *Bertino* und *Martino* unterteilen eine DPL in DDL und DML. [BeMa94, S. 3] *Lang* und *Lockemann* betonen, dass der Teil einer Data Sublanguage, der in eine Host Language eingebettet wird, keine DDL benötigt, sondern einzig aus einer DML besteht. Soll die DDL eingebettet werden, erfordert dies eine vollständige Einbindung des Datenmodells des DBMS in eine Programmiersprache, welche sie dann als Datenbankprogrammiersprache bezeichnen. [LaLo95, S. 27] *Loomis* bezeichnet eine DPL als Database Interface Language. [Loom95, S. 57 ff.]

[106] Zeiger dürfen in diesem Zusammenhang nicht mit dem Hilfskonstrukt des Cursors verwechselt werden, der zur Überwindung des Bruchs zwischen dem „Record-at-a-time-“ und dem „Set-at-a-time“-Paradigma benötigt

Listkonstruktoren), weshalb sie auch als imperative General-Purpose Programming Languages bezeichnet werden [MSOP90, S. 206], aber nur Datensatz-Orientierung (Record-at-a-time-Handhabung [ElNa94, S. 29]), DPL hingegen über Mengen-Orientierung (Set-at-a-time [Mano94, S. 34], Set-oriented [KeMo94, S. 100 f.]) und deklarative Sprachkonstrukte, sind dafür jedoch nicht berechnungsvollständig und verfügen über nur wenige atomare Datentypen (z. B. Character, String, Integer, Real, Boolean)[107] [Date00, S. 111 ff.].

Dies hat zur Folge, dass APL und DPL sehr unterschiedliche Typsysteme und Sprachparadigmata [SST97, S. 176 ff.] (Computational Models [BeMa94, S. 82 f.], Execution Paradigms [KeMo94, S. 100]) aufweisen und die notwendige Kopplung beider Sprachen einen erheblichen Programmieraufwand (Tranfer- bzw. Transformationsroutinen [SST97, S. 176]) erfordert.[108] Der Bruch zwischen den jeweiligen Typsystemen und Sprachparadigmata wird als Impedance Mismatch bezeichnet. [BeMa94, S. 82 f., KeMo94, S. 100 ff.]

Das Hauptziel bei der Gestaltung eines OBMS aus Sicht der OOP ist die Überwindung des Impedance Mismatch. Deshalb wird gefordert, dass Typsysteme und Sprachparadigmata, oder genauer: die Objektmodelle insgesamt, von OOPL und OBMS weitgehend identisch sein sollen. Alle Konzepte einer OOPL sollen auch vom OBMS verfügbar sein.

Es wird eine transparente (im Sinne von unsichtbar, „seamless“ [Loom95, S. 24 f.]) Schnittstelle von der OOPL zum OBMS gefordert. Spezielle Speicherfunktionen (etwa „store“ oder „load“) sind zu vermeiden. [Heue97, S. 522 ff.] Auch die Behandlung von transienten und persistenten Objekten hat weitgehend identisch zu erfolgen. Der Zugriff auf die persistenten Objekte des OBMS soll genauso wie auf die transienten Objekte der Client-Anwendung erfolgen, weshalb auch der Unterschied zwischen dem Hauptspeicher des Anwendungs-Clients und dem Sekundärspeicher des OBMS-Servers transparent wird. [Mano94, S. 199 ff.]

Die Überwindung des Impedance Mismatch ist insbesondere beim Objekt-Paradigma eine wichtige Forderung. Das Ziel des Objekt-Paradigmas ist gerade die Aufhebung der bisherigen strikten Trennung zwischen Daten und Funktionen, d. h. Datenbank und Anwendung, da stattdessen Daten und Funktionen gemeinsam in Objekte gekapselt werden. Etwa meinen *Lausen* und *Vossen*:

> „Ein Datenbanksystem (DBS) basiert, grob gesehen, auf der Idee, Daten und zugehörige Programme zu trennen und unterscheidet sich damit wesentlich von einem Dateisystem.

---

wird, der nur sequenzielle Zugriffe erlaubt und damit für die Handhabung von komplexen Objekten vollständig ungeeignet ist. [KeMo94, S. 101]

[107] Genaugenommen gibt es in RDBMS immer nur die einfache Form der Typkonstruktion [Heue97, S. 276]:

SET OF (TUPLE OF ($A_1 : T_1, \ldots, A_n : T_n$))

[108] Der zusätzliche Aufwand macht in der Regel etwa 30 % des Programm-Codes aus. [BeMa94, S. 83]

> Es sei allerdings bemerkt, dass ein zentrales Anliegen objektorientierter Datenbanken wiederum die Integration von Daten und Programmen ist, die jetzt aber in einem angemessenen methodischen Rahmen (abstrakte Datentypen) vorgenommen wird.“ [LaVo-96, S. 3 f.]

Die Zerlegung komplexer Objekte in flache Speicherstrukturen ist noch aufwendiger, als dies bisher der Fall war. Die reichhaltige Semantik und Struktur des Objektmodells kann nur durch DBMS erhalten bleiben, die ein entsprechendes Objektmodell implementieren, welches die gleiche semantische und strukturelle Ausdruckskraft zu realisieren in der Lage ist. Für primitive Dateisysteme oder RDBMS trifft dies nicht zu. Hilfsweise kann diese semantische und strukturelle Lücke einzig durch zusätzlichen Applikations-Code geschlossen werden, der dann allerdings in den Anwendungen implementiert werden muss, obwohl gerade diese Vorgehensweise dem Objekt-Paradigma widerspricht.

Dies führt dazu, dass das Objektmodell bzw. Typsystem und das Sprachparadigma einer OOPL direkt als DPL verwendet werden. Das Typsystem der OOPL dient direkt als Datenbankmodell des DBMS. Als APL und DPL dient die identische OOPL, z. B. entweder C++ oder Smalltalk. Transfer- bzw. Transformationsprozesse sind für den Anwendungsprogrammierer transparent. *Loomis* meint dementsprechend:

> „The programmer would like to use the object DBMS as if it were part of the language. In fact, many programmers would prefer that the object database were simply invisible. Object DBMS products adress these two integration requirements[109] by using the object programming language as the database interface language.“ [Loom95, S. 23]

Diese Art des einem OBMS zugrunde liegenden Objektmodells bezeichnen *Saake*, *Schmitt* und *Türker* als objektorientiertes Datenmodell (OODM). [SST97, S. 7]

Persistenz von Objekten kann auf verschiedene Weisen realisiert werden, etwa durch Dateisysteme, Objekt-Manager, RDBMS oder durch objekt-relationale DBMS (ORDBMS). Die beschriebene Art der Objekt-Persistenz kann dadurch allerdings nicht erreicht werden. Es gibt immer einen Impedance Mismatch.

Dies ist auch der Grund, warum auch kein noch so durch objektorientierte Funktionalitäten angereichertes ORDBMS (z. B. OID, Stored Procedures, geschachtelte Relationen, erweitertes Typsystem) dazu in der Lage ist, die Forderungen aus Sicht der OOP zu erfüllen. Werden diese Forderungen nicht erfüllt, geht, außer dass ein enormer zusätzlicher Programmieraufwand notwendig wird, ein erhebliches Potenzial des Objekt-Paradigmas insgesamt verloren, wie etwa die semanti-

---

[109] Die beiden Anforderungen von *Loomis* sind:

„1. the object DBMS introduces its own data model or shares the programming language’s type system, and

2. there is a simple mapping between transient and persistent objects“ [Loom95, S. 23]

sche Ausdruckskraft, das Typsystem, die Einfachheit, die Natürlichkeit oder die Speicherung „echter“ Objekte.

Zusätzlich muss das OBMS auch noch über die klassischen DBMS-Funktionalitäten verfügen. Smalltalk etwa bietet durch das Image-Konzept Persistenzeigenschaften, wodurch die obigen Forderungen realisiert werden können. Die transitive Hülle eines Objekts wird persistent, indem eine Referenz des System-Dictionaries auf dieses Objekt erzeugt wird.[110] Damit können allerdings keine Multiuser-, Netzwerk- oder sonstigen essenziellen DBMS-Funktionalitäten erreicht werden.

Deshalb werden außer der Forderung, dass Objekte persistent in einer redundanzfreien und konsistenten Objektbasis vorliegen müssen, auch aus Sicht der OOP Eigenschaften von einem OBMS gefordert wie Mehrbenutzerbetrieb, Concurrency Control, Recovery, Transaktions-Management, Schema-Evolution, Integritätssicherung, Zugriffskontrolle, Sekundärspeicher-Verwaltung mit Clusterung, Indizierung, Replikation und Anfrageoptimierung sowie eine deklarative Anfragesprache (siehe Abschnitt „DBMS-Funktionalität“ auf Seite 145).

3.4.3.2 DBMS-Blickwinkel

Parallel zur Entwicklung der objektorientierten Programmiersprachen gibt es auch eine ständige Weiterentwicklung von Datenbankmodellen und Datenbank-Management-Systemen, die aus Bestrebungen zur Beseitigung bisheriger Mängel [Voss94, S. 337 ff.] und aus neuen Anforderungen an diese Systeme [KeMo94, S. 3 ff.] resultieren.

Es lassen sich zwei Entwicklungsrichtungen erkennen. Zum Ersten steht, wie auch beim Objektmodell der OOP, die Bestrebung im Vordergrund, bessere Modelle zur Abbildung der Diskurswelt (Natürlichkeit, Semantik, Struktur) zur Verfügung zu stellen. Zum Zweiten sind es die DBMS-Funktionalitäten, die auf die neuen Konzepte eines OBMS übertragen werden müssen, die durch die neuen Konzepte eines OBMS zusätzlich notwendig bzw. sinnvoll werden und DBMS-Funktionalitäten, die an sich verbessert werden müssen.

Dementsprechend gliedert sich dieser Abschnitt in die zwei Unterabschnitte Modellierung, der die Konzepte erläutert, die primär mit dem Objektmodell verbunden sind und DBMS-Funktionalität, der die Konzepte diskutiert, die primär mit den Funktionen eines DBMS im engeren Sinne zusammenhängen.

---

[110] Dies entspricht dem Persistenz-Konzept der ODMG des Persistence by Reachability (Persistenz durch Erreichbarkeit [SST97, S. 170]) bzw. by Naming (durch Benennung [SST97, S. 167 f.]). [CaBa98, S. 19 f.] Durch Benennung, weil der Eintrag in das SystemDictionary die Bennenung durch ein Symbol (z. B. „#Kunden“) erfordert. Ein solcher Eintrag wird auch als globale Variable bezeichnet. [BGL95, S. 36 f.,] Durch Erreichbarkeit, weil die transitive Hülle dadurch persistent wird. [HoHo97, S. 78]

Modellierung

Der Durchbruch in der Datenbank-Technologie gelang mit den relationalen DBMS (siehe oben). Das Relationen-Modell ist ein einfaches, aus wenigen Elementen bestehendes Modell, basierend auf der Grundlage einer klaren mathematischen Theorie. Die Vorteile sind die „… Einfachheit, Verständlichkeit und Homogenität des Modells." [Heue97, S. 297] Es ist damit allerdings auch vielen Beschränkungen unterlegen. *Lausen* und *Vossen* meinen etwa:

> „Wir erkennen […], dass bereits bei der Datenmodellierung in traditionellen Anwendungen Modelle wie das relationale an ihre Grenzen stoßen, speziell dann, wenn man angemessen modellieren möchte." [LaVo96, S. 22]

Um die Beschränkungen des Relationen-Modells zu überwinden, entstanden verschiedene andere Datenmodelle [CáLe90b, S. XX ff.], wie direkte Erweiterungen des Relationen-Modells etwa in Form der geschachtelten Relationen-Modelle NF^2 bzw. eNF^2 und Komplex-Objektmodelle [Khos93, S. 126 ff.] oder Modelle mit einem höheren Abstraktionsniveau wie das Entity-Relationship-Modell, wiederum mit Erweiterungen [Sinz90, S. 17 ff.], funktionale Modelle und semantische Datenmodelle (SDM) [KhAb95, S. 21 f.]. Den am weitestgehenden Ansatz realisieren die semantischen Datenmodelle. [CáLe90b, S. XVIII] *Hull* und *King* meinen etwa:

> „… semantic models attempt to provide more powerful abstractions for the specification of database schemas than are supported by the relational, hierarchical, and network models. […] Semantic models were developed to provide a higher level of abstraction for modeling data, allowing database designers to think of data in ways which correlate more directly to how data arise in the world." [HuKi90, S. 3]

Erweiterungen der semantischen Datenmodelle, etwa gegenüber dem Entity-Relationship-Modell, bestehen unter anderem in der Integration von Konzepten wie Spezialisierungsbeziehungen, Integritätsbedingungen und komplexen Datentypen. [SST97, S. 6 ff.]

Heuer erwähnt in diesem Zusammenhang vier Anforderungen, die an Datenbankmodelle zu stellen sind [Heue97, S. 130 f.]:

1. **Abstraktionsvermögen**: Objekte sollen stufenweise abstrahiert werden können (Information Hiding).
2. **Direkte Modellierbarkeit von Anwendungsobjekten**: Ein Objekttyp der Anwendungswelt (z. B. Buch) soll mit genau **einem** Datenbankmodell-Konzept modellierbar sein und nicht zerlegt werden müssen.
3. **Mächtigkeit**: Das Datenbankmodell soll über möglichst viele spezielle semantische Ausdrucksmöglichkeiten verfügen. Beispielsweise soll es eine direkte Abbildung von n:m-Beziehungen geben, die auch über geeignete Integritätsbedingungen verfügen können.

4. **Exakte Definition**: Das Datenbankmodell soll exakt definiert sein, so dass Konsistenz, Vollständigkeit und Redundanzfreiheit gewährleistet werden können.

Heuer betrachtet SDM als Möglichkeit, die Struktur von Anwendungsobjekten direkter und semantisch reichhaltiger darstellen zu können. Zumindest die ersten drei Forderungen betrachtet er für SDM als erfüllt an. *Heuer* sieht außerdem wesentliche Parallelen zwischen dem Strukturteil des Objektmodells und den semantischen Datenmodellen, wobei das Objektmodell um einen Operationenteil und „höhere Konzepte" ergänzt wird. [Heue97, S. 127 ff.]

Dementsprechend meinen auch *Lausen* und *Vossen*:

> „Wesentliches Ziel war zunächst die Herstellung von Möglichkeiten einer adäquateren Beschreibung von Objekten mit komplexer Struktur sowie die Erfassung von semantischer Information. […] Den vorläufigen Abschluss der Evolution von Datenmodellen bilden objektorientierte Modelle, die neben einer Struktur-Modellierung auch eine Modellierung von Objekt-Verhalten erlauben." [LaVo96, S. 11]

Diese Art von Objektmodellen bezeichnen *Saake*, *Schmitt* und *Türker* im Gegensatz zu den mehr programmiersprachen-basierten OODM-Ansätzen als objektorientierte Entwurfsmodelle (OEM), da sie ihre Wurzeln mehr in den ERM und SDM als in den OOPL haben sowie einen höheren Grad an Abstraktion realisieren, wodurch unter anderem auch ein höherer Grad an physischer Datenunabhängigkeit entsteht. [SST97, S. 8 f.]

Dies ist eng verbunden mit den Konzepten des Objekt-Paradigmas der OOA. Doch während die Modellierung der OOA die konzeptionelle Ebene betrifft, sollten die Objektmodelle der OBMS möglichst vollständig und ohne Bruch implementiert werden.

Die Konzepte der SDM oder auch des einfacheren ERM wurden bisher fast ausschließlich zur konzeptionellen Modellierung von Datenbank-Modellen verwendet, doch anschließend, zur Implementierung in ein DBMS, zumeist in die einfacheren Strukturen eines Relationen-Modells übersetzt, da nur vereinzelte „kommerzielle" SDM-DBMS existieren [FGJ+90, S. 241 ff.], sich jedoch nicht durchsetzen konnten [Heue97, S. 127 f.].

Ziel des OBMS-Objekt-Paradigmas ist allerdings, wie gezeigt, eine weitgehend getreue Abbildung des OOA-Objektmodells in das OBMS-Objektmodell zu erreichen. Gefordert wird deshalb, alle Konzepte des OOA-Paradigmas auch im Objektmodell des OBMS bereitzustellen, wie komplexe Objekte und neue Typen (Erweiterbarkeit), Objektidentität (OID), Methoden und Nachrichten, Kapselung, Klassen, Vererbung, Polymorphie, Assoziation- und Aggregation-Beziehungen, Kardinalitäts- und Rollen-Semantik sowie Constraints bzw. Geschäftsregeln (siehe Abschnitt 3.3.2.4).

Die Umsetzung dieser Konzepte in das OBMS-Objektmodell bzw. durch das OBMS hat weitreichende Konsequenzen. Struktur und Semantik der Objekte können direkt im OBMS-Objektmodell hinterlegt werden, gehen also nicht wie bisher bei der Umsetzung in das DBMS verloren bzw. müssen nicht durch die einzelnen Applikationen umständlich nachimplementiert werden.

Saake, *Schmitt* und *Türker* nennen ein sehr plastisches Beispiel für die strukturellen Eigenschaften des Objektmodells eines OBMS. Wird ein Objektmodell verwendet, kann ein Auto-Objekt auf einfache Weise Persistenz erlangen, z. B. durch Benennung (siehe Abschnitt 3.4.3.5). Wird hingegen ein Relationen-Modell verwendet, ist ein aufwendiger Zerlegungsprozess notwendig:

> „Bevor ein Auto in der Garage abgestellt werden kann, muss es in seine tausend Einzelteile zerlegt und in den dafür vorgesehenen Fächern abgelegt werden. Bevor es wieder benutzt werden kann, ist ein komplizierter Zusammenbau notwendig." [SST97, S. 43]

Dadurch wird es auch möglich, semistukturierte Daten in das Objektmodell zu integrieren, diese inklusive Strukturen und Semantik abzubilden und folglich als Ganzes oder auch in Teilen zu verwenden. Beispiele solcher Daten, die auch zunehmend an praktischer Relevanz für den Datenbankbereich gewinnen, sind etwa „lange Texte, graphische Daten, Bilder, Audioinformation (Ton, Sprache, Musik, etc.) und eventuell sogar Videosequenzen." [LaLo95, S. 223]

Insbesondere durch die zunehmende Bedeutung der Internet-Technologie, etwa in Form multimedialer WWW-Dokumente, des Knowledge Managements und von Management Support Systems (siehe Kapitel 2) wird die Integration solcher semistrukturierten Daten immer erfolgskritischer.

Für diese semistrukturierten Objekte müssen geeignete Typkonstruktoren und Klassen sowie entsprechende Methoden vorhanden bzw. erzeugbar sein. Auch die Handhabung dieser im Verhältnis zu den sonstigen, strukturierten Objekten wesentlich umfangreicheren Datenvolumina muss vom OBMS übernommen werden. Dabei ist bei der Modellierung solcher Objekte zwischen rein syntaktischen Kriterien (z. B. Worte, Sätze, Pixel) und einer sinntragenden syntaktischen Struktur zu unterscheiden, die beide notwendig sind. [LaLo95, S. 223 ff.]

Bei den Methoden muss es zum Beispiel die Möglichkeit von fortgeschrittenen Such- und Manipulations-Funktionen geben. Beispiele sind die Suche nach Teilworten, die Berücksichtigung von phonemischen Ähnlichkeiten in Anfragen, die Beziehungen von Textelementen zu Bildausschnitten oder die Verbindung von strukturierten Informationsobjekten mit kartografischen Informationen [WhDa-98, S. 1 ff.].[111] [LaLo95, S. 223 ff.]

---

[111] Es sei an dieser Stelle bemerkt, dass die nachträgliche Einführung des BLOB-Konzeptes bei relationalen DBMS aufgrund der fehlenden Möglichkeit der Behandlung von semistrukturierten Daten all dies nicht gestattet und insofern ungeeignet zur Lösung dieses Problems ist. Eine Integration von semistrukturierten Daten

Larson sieht einen sich verstärkenden Trend bei der Entwicklung von Systemen, die diesen Anforderungen gerecht werden. Er sieht dies in Form von Text-basierten Datenbanksystemen, Multimedia-Datenbanken, speziell für die Verknüpfung von Dokumenten mit Hypertext und Hypermedia und deren Integration mit OBMS-Technologie. [Lars95, S. 191 ff.]

Weiterhin sind in diesem Zusammenhang auch Konzepte zur Abbildung von Sequenzen (Reihenfolgen) notwendig, was durch die reine Mengenbetrachtung des Relationen-Modells bisher auch nicht möglich ist und so zu erheblichen Problemen führen kann [RBK+98, S. 137 ff.]. Auch Assoziation, Aggregation und Spezialisierung können im Relationen-Modell nicht direkt, d. h. mit ausreichender semantischer Ausdruckskraft dargestellt werden. [SST97, S. 39 ff.]

Beispielsweise muss die spezielle Semantik einer Aggregations-Beziehung in Abgrenzung zu einer einfachen Assoziation abgebildet werden, wie etwa der Unterschied zwischen einer essenziellen Abhängigkeit des Ganzen von seinen Teilen bzw. einer essenziellen Abhängigkeit der Teile vom Ganzen. [RMP99, S. 98 ff.]

Dies betrifft auch die Abbildung der Beziehungen. Binäre Beziehungen [KeMo94, S. 244 ff.] dürfen nicht zu inkonsistenten Referenzen führen, auch wenn sie eventuell physisch redundant implementiert werden müssen. Da die ODMG diese binären Beziehungen (Inversen) als Standard definiert hat [CaBa98, S. 36 ff.], ist die korrekte Behandlung essenziell.[112] [RMK99]

Loomis nennt als weiteres wichtiges Beispiel die Konstruktion eines Zeitreihen-Datentyps, der in einer großen Anzahl von Anwendungen benötigt wird. [Loom-95, S. 41] Tatsächlich spielt die Zeitreihe im MSS-Umfeld eine herausragend bedeutende Rolle und wird deshalb in so gut wie jeder MSS-Applikation, wie auch dem Object Warehouse benötigt.

Vom OBMS verwaltete Objektidentitäten gewährleisten die Trennung von Identifikationssemantik und Wertsemantik, die beim Relationen-Modell durch die Simulation in Form eines Primärschlüssels vielfach vermischt wird und in der Folge immer wieder zu Konflikten führt. Stattdessen erspart die OID-Verwaltung des OBMS den Anwendern die explizite Verwaltung und lässt auch eine Unterscheidung von gleichen und identischen Objekten zu. [SST97, S. 80 ff., S. 47] Das OID-Konzept ist also inhärenter Bestandteil des Objektmodells eines OBMS.

---

bzw. von semistrukturierten Objekten in das Objektmodell findet dort deshalb nicht statt. *Heuer* hält dieses Kriterium für zukünftige DBMS als überlebenswichtig. [Heue97, S. 627 ff.]

[112] Objectivity bietet etwa eine Behandlung von Inversen durch das OBMS. Dazu werden bei der Klassendefinition, d. h. der ODL von Objectivity entsprechende binäre Beziehungen definiert und mit den Kardinalitäten 1 oder n versehen, so dass direkte 1:1-. 1:n- und n:m-Beziehungen verwaltet werden können. Andere Kardinalitäten oder Integritätsbedingungen sind allerdings nicht möglich. [Obje97, S. 4-9 ff.]

Insbesondere bei umfangreichen Datenbeständen muss die Verwaltung der Identifikationssemantik vom DBMS übernommen werden, da eine konsistente Handhabung andernfalls nicht zu gewährleisten ist. Außerdem besteht ein ganz wesentlicher Unterschied zwischen einem Primärschlüssel und einer OID darin, dass ersterer nur innerhalb einer Relation gültig ist, während die OID systemweite Gültigkeit besitzt und dadurch ein entsprechend mächtiges Referenzierungskonzept verwirklicht werden kann.

Auch umfangreiche Integritätsbedingungen müssen durch das Objektmodell des OBMS darstellbar sein bzw. durch das OBMS realisiert werden können. In der OOA ermittelte Constraints bzw. Geschäftsregeln müssen umgesetzt werden. Weitere Integritätsbedingungen müssen die langfristige Konsistenz des Objektbestands gewährleisten. *Gray*, *Kulkarni* und *Paton* sehen Integritätsbedingungen als Mittel, Eigenschaften von Daten auszudrücken, die nicht in der Form von Strukturen abgebildet werden können, sondern als logische Restriktionen hinterlegt werden müssen. [GKP92, S. 13]

Integritätsbedingungen lassen sich in statische und dynamische (transitionale und temporale) Varianten unterscheiden. Sie sind jeweils lokal auf Attribute, Objekte oder Klassen bezogen oder Attribut-, Objekt- oder Klassen-übergreifend. Die wichtigsten Integritätsbedingungen betreffen Wertebereichseinschränkungen, Kardinalitätsbedingungen, referenzielle Integritätsbedingungen, Bedingungen bezüglich der Spezialisierungshierarchie und Bedingungen bezüglich der Aggregationshierarchie. Integritätsbedingungen können deklarativ (etwa in den Klassendefinitionen), operational (in den Instanzmethoden) oder regelbasiert (durch ECA-Regeln getriggert) definiert werden. [SST97, S. 343 ff.]

Gerade in Anbetracht der wesentlich höheren Komplexität und Ausdruckskraft von Objektmodellen sind komplexe und dynamische Integritätsbedingungen eine grundlegende Anforderung an OBMS. Sie bilden auch einen bedeutenden Anteil der Metainformationen. [RMK99] *Brodie* definiert ein Datenbankmodell dieser Bedeutung entsprechend als aus statischen Eigenschafften, dynamischen Eigenschaften und Integritätsbedingungen bestehend. [zitiert nach Heue97, S. 130]

Das Objektmodell sollte außer rein mengen-orientierten Zugriffen auch navigierende Zugriffe erlauben, da Navigation in vielen Bereichen eine natürliche und effiziente Suchmethode darstellt. *Chorafas* und *Steinmann* meinen etwa:

> „In real-life applications, the user need to navigate through a network or hierarchy. For example, to find recent invoices of a customer, the user locates the customer account record, then locates the invoice records under that customer, and then moves to the account payable.[113] […] we do have hypertext and hypermedia, and hypermedia facilities should be embedded in object-oriented applications." [ChSt93, S. 201 f.]

[113] Zum Vergleich sieht das selbe Beispiel in SQL-Form so aus [ChSt93, S. 202]:

Es zeigt sich, dass die bei der OOA erläuterten Konzepte (siehe Abschnitt 3.3.2) direkt in das Objektmodell umgesetzt werden können und diese Konzepte, soweit möglich, vom OBMS verwaltet werden, also nicht explizit codiert werden müssen (OID, Aggregation, Inverse etc.). Außerdem zeigt sich, dass die Konzepte der OOA für ein OBMS nicht hinreichend sind und folglich erweitert werden müssen, da die persistenten Objekte langfristig konsistent zu halten und insbesondere auch die Dynamik der Objekte und der Strukturen des Objektmodells, des Objekt-Schemas, abgebildet werden müssen (dynamische Constraints, navigierende Zugriffe, erweiterte Anfrage- bzw. Suchmechanismen etc.).

Der diesbezüglich wichtigste Unterschied ist die Unterscheidung von persistenten und transienten Objekten und die Umsetzung von entsprechenden Konzepten, durch die bzw. anhand derer Persistenz realisiert werden kann. Diese Modellierung bzw. Unterscheidung zwischen persistenten und transienten Objekten erfolgt in den meisten OOA-Ansätzen allerdings nicht [Balz95, S. 273 ff.], da hierfür weitergehende Darstellungsmittel erforderlich sind [KaSc96, S. 23 ff.].

Persistenz ist ein essenzieller Bestandteil des OBMS-Objektmodells, wird jedoch üblicherweise als DBMS-Funktionalität gesehen [Khos93, S. 1 f.], so dass auf den Unterabschnitt „DBMS-Funktionalität“ (siehe Seite 145) beziehungsweise „Persistenzmodelle“ (siehe Abschnitt 3.4.3.5) verwiesen wird.

Als weitere essenzielle Bestandteile, speziell des Objektmodells eines OBMS, also auch des Modellierungs-Prozesses, werden in dieser Arbeit die Konzepte der Versionierung in Form von Instanz-Versionierung, Schema-Versionierung und Instanz-Evolutions-Versionierung, der Instanz-Evolution in Form von Instanzmigration, Instanz-Spezialisierung und Instanzrollen[114] (siehe Abschnitt 3.4.3.3) und das Sichten-Konzept betrachtet (siehe Abschnitt 3.4.3.4).

DBMS-Funktionalität

Die im letzten Unterabschnitt „Modellierung“ aufgestellten Anforderungen wie unter anderen Vererbung, Objektidentität, Aggregationssemantik, Kardinalität und inverse Beziehungen müssen durch das OBMS weitestgehend automatisch, d. h. durch geeignete Konstrukte unterstützt werden. Dies bedeutet etwa eine

```
SELECT
FROM acct_records
WHERE acct_num IN
    (SELECT acct_num
     FROM customer_accts
     WHERE name = “Paul Peters“);
```

[114] Instanz-Evolution [BeMa94, S. 138 ff.], Instanzmigration, Instanz-Spezialisierung und Instanzrollen werden oft auch als Objekt-Evolution [Gepp97, S. 89 ff.], Objektmigration, Objektspezialisierung und Objektrollen [SST97, S. 402 f.] bezeichnet. Da gerade in diesem Zusammenhang die Unterscheidung zwischen Klassen und Instanzen wesentlich ist, die im Folgenden betrachteten Konzepte jedoch prinzipiell für alle Objekte, also auch für Klassen realisierbar sind, wird die spezifischere Ausdrucksform Instanz verwendet.

vollständige Verwaltung der OID durch das OBMS, ohne dass diesbezügliche Eingriffe durch Benutzer möglich oder gar notwendig sind, wie dies im Gegensatz dazu beim Primärschlüssel-Konzept vieler RDBMS[115] der Fall ist.[116]

Neben diesen Anforderungen der Modellierungs-Konzepte, speziell objektorientierter DBMS, werden auch spezifische Funktionalitäten zur Verwaltung persistenter Objekte benötigt, d. h. klassische Datenbank-Funktionen für ein OBMS.

Die klassischen von *Codd* festgelegten Komponenten eines DBMS sind Datenintegration, generische Operationen, Schemakatalog, Benutzersichten, Integritätssicherung, Zugriffskontrolle, Transaktionen, Synchronisation und Datensicherung. [SST97, S. 4 f.] Zumindest diese Funktionen müssen auch durch ein OBMS verwirklicht werden, soweit sie in diesem Kontext weiterhin sinnvoll sind.[117]

Khoshafian und *Abnous* fordern „Persistence, Concurrency, Transactions, Recovery, Querying, Versioning, Integrity, Security, and Performance." [KhAb95, S. 321 f.]

Loomis verlangt „Storage Management, Schema Management, Concurrency Control, Transaction Management, Recoverability, Query Processing, and Access Control." [Loom95, S. 46]

Atkinson et al. führen die Kriterien „Computational Completeness, Extensibility, Persistence, Secondary Storage Management, Concurrency, Recovery, Ad Hoc Query Facility, Distribution, Design Transactions, and Versions" [ABD+89] an.

Saake, *Schmitt* und *Türker* ergänzen den Katalog um Integrität, Sichten, Schema-Evolution und Zugriffskontrolle und kommen insgesamt zu den neun Punkten: Persistenz, Anfragesprache, Transaktionen, Synchronisation im Mehrbenutzerbetrieb, Datensicherung, Integritätssicherung, Sichtdefinition, Zugriffsschutz und Schema-Evolution. [SST97, S. 12 f.]

Datenintegration bzw. Objektintegration ist primäres Ziel der Modellierung und wurde deshalb dort diskutiert (siehe Unterabschnitt „Modellierung" auf Seite 140). Dies gilt ebenfalls für die automatische Verwaltung der Objektidentität (OID), Versionierung, Schemaevolution, Schemakatalog und das Schema Management, die Erweiterbarkeit (Extensibility) sowie eine berechnungsvollständige Programmiersprache (Computational Completeness).

---

[115] Beispielsweise verfügt ORACLE 7 nicht einmal über einen SERIAL. [KoLo95, S. 436 f.]

[116] Es sei allerdings bemerkt, dass kommerzielle OBMS bis jetzt nur einen sehr geringen Teil dieser Anforderungen unterstützen, wie etwa die Verwaltung von Inversen, Kardinalitäts-Constraints, Aggregationssemantik oder Instanz-Migration.

[117] Die Diskussion hierüber ist noch nicht abgeschlossen. *Heuer* fordert beispielsweise auch das Vorhandensein generischer Operationen für OBMS [Heue97, S. 487 ff.], obwohl diese grundlegend den Prinzipien des Objekt-Paradigmas widersprechen.

Generische Operationen sind überhaupt umstritten (siehe Fußnote 117 auf Seite 146) und werden deswegen an dieser Stelle nicht weiter behandelt. Aufgrund ihrer Bedeutung für Anfragesprachen werden sie im Abschnitt „Object Query Language“ (siehe Abschnitt 3.4.3.7) behandelt.

Aus den obigen Punkten ergeben sich insgesamt die folgenden für ein OBMS notwendigen Funktionen:

- Realisierung eines **Mehrbenutzerbetriebs**, bei dem trotz mehrerer konkurrierender Transaktionen von Benutzern bzw. Applikationen, nach jeder Transaktion wieder ein konsistenter Zustand des OBMS entsteht und die Transaktionen zu den beabsichtigten Ergebnissen führen. Dafür notwendig ist **Concurrency Control** (**Synchronisation**), zum Beispiel durch Serialisierung oder Locking.
- **Transaktionen** schließen einzelne Datenbankänderungen zusammen, die gemeinsam ausgeführt werden müssen, damit der Datenbankzustand von einem konsistenten Zustand in den nächsten überführt wird. Eine Transaktion wird deshalb vollständig ausgeführt (commit) oder rückgängig gemacht (Abort, Rollback). Dafür wird auch ein spezielles **Transaction Management** vom OBMS zur Verfügung gestellt. Durch das besondere Modell eines OBMS und Anwendungen mit entsprechend komplexen Strukturen wie etwa CAD oder besonders umfangreichen Transaktionen wie beispielsweise dem Ladevorgang eines Object Warehouse, sind spezielle **lange Transaktionen** (**Design Transactions**) notwendig.
- Durch **Recovery** wird sichergestellt, dass der konsistente Datenbankzustand immer wieder hergestellt werden kann, auch wenn Fehler während Transaktionen durch das System oder die Hardware auftreten.
- Das **Storage Management** (Sekundärspeicher-Verwaltung) muss ebenfalls vollständig durch das OBMS übernommen werden. Die Transformation zwischen Haupt- und Sekundärspeicher erfolgt für den Benutzer transparent (im Sinne von unsichtbar).
- Es muss die Möglichkeit zur **Verwaltung großer Datenmengen** (von Objekten) gegeben sein.[118] Besonders im Zusammenhang mit dem Object-Ware-

[118] Zur Verwaltung großer Datenmengen werden spezielle Speicherkonzepte bzw. -verwaltungen benötigt, die unter anderem Plattenpartitionierung und -verwaltung und parallele Prozessoren unterstützen sowie eine Optimierung von Netzwerkverkehr und Plattenzugriffen erreichen. Außerdem ist es bei großen Datenmengen sinnvoll, einen sehr leistungsfähigen, hochkapazitären Server (Fat Server, hohe Speicher- und Verarbeitungskapazität) für die Verwaltung derartiger Datenvolumina zu verwenden, statt diese Aufgabe von den weniger leistungsstarken Clients durchführen zu lassen.

house-Konzept sind Größenordnungen von 100 Mega Bytes oder sogar Tera Bytes zu erwarten.[119]

- Die **Distribution** (Verteilung) von Daten bzw. Objekten wird immer wichtiger und muss deshalb auch vom OBMS verwaltet werden.
- Physisches Design durch **Clusterung**, **Indizes** und **Replikation** (Performance) ist notwendig, da die geforderte Ausführungseffizienz ansonsten nicht zu realisieren ist.
- Eine komplexe **Zugriffskontrolle** ist eine notwendige Voraussetzung für ein DBMS, da die teilweise hochsensiblen Daten auf diese Weise vor unberechtigtem Zugriff geschützt werden können.[120]
- Ein elementares Konzept von DBMS ist die **Integritätssicherung**. Integritätssicherungen sind zwar ebenfalls primärer Bestandteil des Objektmodells, können jedoch auf unterschiedliche Weise realisiert werden. Die effizienteste Möglichkeit sind deklarative Integritätsbedingungen, die deshalb durch das OBMS bereitgestellt werden müssen. [RMK99] Operationale Integritätsbedingungen müssen und ECA-Regeln sollten ebenfalls umzusetzen sein.
- Auch eine **deklarative Anfragesprache** wird als notwendige Bedingung für ein OBMS angesehen. Dabei stehen vor allem der Ad-hoc-Character, die einfache (deklarative) Bedienbarkeit (im Gegensatz zu einer vollständigen Programmiersprache) und die Optimierbarkeit der Anfragen im Vordergrund des Interesses (siehe Abschnitt 3.4.3.7). Als wesentliches Merkmal von OBMS werden im Zusammenhang mit Anfragesprachen auch navigierende Zugriffe (über Referenzen) genannt.
- **Persistenz** ist das Grundkonzept eines DBMS, welche je nach Modell auf sehr unterschiedliche Weise realisiert werden kann (siehe Abschnitt 3.4.3.5).

3.4.3.3 Evolution und Versionierung

In OOA-Modellen und konventionellen Datenbankmodellen erfolgen vielfach nur statische Betrachtungen der Daten bzw. Objekte, im Sinne einer zeitpunktbezogenen (aktuellen) Gültigkeit der Daten. Doch insbesondere für den MSS-Bereich

---

[119] Es wird allerdings erwartet, dass aus der weitaus geringeren Redundanz des Objektmodells des Derived Object Layers, die durch die Zeiger-Semantik realisiert wird, auch erheblich weniger Speicherkapazität benötigt wird als dies beim konventionellen Analogon, dem Derived Data Layer (z. B. Star-Schema) eines relationalen Data Warehouse der Fall ist.

[120] GemStone realisiert beispielsweise einen Schutz auf Objektebene. Objekte werden Segmenten zugeordnet, die mit Benutzer-, Gruppen- und „Welt"-Rechten der Ausprägung Schreiben, Lesen und „Rechtlosigkeit" versehen werden. [Gems96b, S. 7-4 ff.] Dadurch sind beliebige Zugriffsrechte auf flexible Art und Weise zu erstellen. Es sei angemerkt, dass der Modellierungs-Prozess für die Benutzerrechte mit erheblichem Aufwand verbunden ist. [Gems96b, S. 7-16 ff.]

werden Daten langfristig benötigt, so dass auch die Abbildung deren Dynamik (Historie) im Objektmodell des OBMS, insbesondere auch die Veränderung der Objektstrukturen benötigt wird.

Es wird im Folgenden unterschieden zwischen der Versionierung von Instanzen und Schemata, zwischen der Instanz-Evolution, Instanz-Spezialisierung und Instanzrollen sowie der sich daraus ergebenden Instanz-Evolutions-Versionierung.

Bei der **Versionierung** ist zwischen der Versionierung von Instanzwerten, der Instanz-Versionierung und der Versionierung des Schemas des OBMS-Objektmodells, der Schema-Versionierung zu unterscheiden. Die Instanz-Evolution bildet einen Spezialfall.

Die **Instanz-Versionierung** spielt eine entscheidende Rolle in Anwendungen, die vergangene Zustände von Instanzen abbilden müssen, also Vorgänger-Versionen oder sogar gleichzeitig unterschiedliche Instanz-Versionen benötigen.

Bertino und *Martino* nennen eine Gehaltsdatenbank als Beispiel. [BeMa94, S. 98] Werden die alten Gehälter eines Mitarbeiters durch die jeweils neueren Gehälter überschrieben, ist unter anderem keine Gehaltsentwicklung mehr zu ermitteln. Diese Information kann jedoch in vielen Fällen eine sehr wesentliche Information darstellen.

Khoshafian nennt die Entwicklung seines Buches als Beispiel. Die Endversion dieses Buches besteht aus unterschiedlichen Versionen von Kapiteln bzw. Abschnitten, da er am Ende des Schreibprozesses gelegentlich lieber auf ältere Versionen zurückgreift, sofern ihm diese letztlich doch als besser gelungen erscheinen. [Khos93, S. 271] Üblicherweise werden Änderungen in Dokumenten einfach überschrieben und gehen dadurch verloren.

Beispiele für Anwendungen, in denen eine Versionierung der Instanzen eine grundlegende Anforderung an das System darstellt, sind das Data Warehouse [RiBr99, S. 1 ff.], vor allem das Object Warehouse [RiMa00], Computer Aided Design (CAD) oder Computer Aided Software (bzw. Systems) Engineering (CASE) [KeMo94, S. 3 ff.].

Ändern sich beim Object Warehouse zum Beispiel Stammdaten wie etwa Kunde oder Artikel, müssen diese Änderungen festgehalten werden. Zieht ein Kunde in eine andere Region und diese Änderung wird nicht als neue Version festgehalten, d. h. es wird wie vielfach üblich, der alte durch den neuen Wert überschrieben, werden auch alle bisher getätigten Umsätze der neuen Region zugerechnet. [Maur99, S. 69 ff.] Aber auch für die Simulation von Altzuständen, Simulations-Anfragen [RiBr99, S. 6 f.] und für lange Transaktionen, wie etwa beim Ladevorgang eines Data Warehouse [IWG97, S. 102] ist Versionierung wichtig.

Ändert sich die Artikelspezifikation (z. B. Zusammensetzung, Beschaffenheit oder Ausstattung), muss auch dies festgehalten werden, um weiterhin korrekte

Anfragen zu ermöglichen. Alternativ könnte auch ein neuer Artikel entstehen, doch es hängt von der Domäne ab, ob eine Änderung als so marginal angesehen wird, dass dies lediglich zu einer neuen Version des Artikels führt und nicht zu einem neuen Produkt und diese Änderung gleichzeitig bedeutend genug ist, eine neue Version zu rechtfertigen.[121] Der Bezug zwischen beiden Artikelversionen wird allerdings in den meisten Fällen Informations-Potenzial beinhalten und deshalb abgebildet werden müssen.

Eine neue Version einer Kunden- oder Artikel-Instanz wird dann erzeugt, wenn ein als versions-kritisch festgelegtes Attribut eines Objekts verändert wird.

Zum Teil wird auch eine alternative Modellierung gewählt, indem einzelne versions-kritische Attribute als eigene Objekte realisiert werden, die mit speziellen Versionseigenschaften ausgestattet sind, um die Versionierung genau dieses jeweiligen Attributs abbilden zu können. Etwa wird in vielen Fällen der Verkaufspreis eines Artikels als eigenes Objekt mit entsprechenden Attributen der Gültigkeit, insbesondere Datum und Zeit modelliert werden.

Bei der Instanz-Versionierung wird einfach eine komplett neue Version einer Instanz erzeugt, wenn sich auch nur ein versions-kritisches Attribut ändert. [Lars-95, S. 215]

Bertino und *Martino* zeigen die herausragende Stellung der zeitlichen Dimension für die Modellierung und das Management von Versionen. Sie betonen allerdings, dass das Versionskonzept über die Anforderungen zeit-orientierter Datenmodelle bzw. zeit-orientierter DBMS hinausgeht. Versionsänderungen sind explizite Operationen, die semantisch signifikant und mit Design-Änderungen anderer Objekte verbunden sind. Erforderlich sind die Berücksichtigung von Vererbungsbeziehungen und die Propagierung der Änderungen und (temporären) Arbeitsbereiche. [BeMa94, S. 113 f.]

Bei Anwendungen wie CAD oder CASE ist zusätzlich die Verwaltung von Konfigurationen notwendig. Dabei werden verschiedene Versionen von Instanzen, die

---

[121] Für *Saake*, *Schmitt* und *Türker* ist dieses Problem nicht vordergründig, sie meinen: „… da es sich bei Versionen um dasselbe Realwelt-Objekt handelt, das über die Zeit gesehen, nur seine Attributwerte ändert." [SST97, S. 374] In solchen Fällen ist die Entscheidung einfacher. Doch auch unter solchen Fällen wird es natürlich Fälle geben, bei denen durch Veränderungen ein „neues" Objekt entsteht. In den meisten Fällen wird damit allerdings auch eine Änderung der Klassenzugehörigkeit verbunden sein (siehe unten), was dann nicht mehr in den hier betrachteten Bereich der Instanz-Versionierung fällt. Die Betrachtung des „Artikel-Objekts" ist insofern anders zu betrachten, da Artikel bereits selbst kein konkretes Objekt darstellt. Eine Instanz von Artikel ist — abhängig von der Domäne, d. h. der jeweiligen Modellierungsintention (siehe Abschnitt 2.3.2) — kein konkreter Artikel, sondern beschreibt eine Artikel-Gattung. Andernfalls müsste beispielsweise IKEA mit etwa 40.000 verschiedenen Artikeln ebenso viele Klassen modellieren und für jeden verkauften Artikel eine Instanz erzeugen und verwalten, was gerade in diesem Beispiel unökonomisch sein wird. Eine Artikel-Gattung ist dann allerdings kein Realwelt-Objekt, sondern wiederum eine Abstraktion, so dass die Entscheidung, ob es sich um eine neue Version des selben Artikels oder einen neuen Artikel handelt von der Sichtweise des Modellierers bzw. den Anforderungen der Diskurswelt abhängt.

zusammen gehören bzw. sinnvoll zusammen passen, zu bestimmten Konfigurationen zusammengefasst [KeMo94, S. 437 ff.], da beispielsweise ein verändertes Bauteil auch einen entsprechend veränderten Rahmen benötigt oder Änderungen eines Organigramms auch Veränderungen des Prozessmodells erfordern können. Versionen sind also nicht frei kombinierbar, sondern müssen Konsistenzbedingungen genügen, die durch Konfigurationen kombiniert werden.

Instanz-Versionierung kann auch im Relationen-Modell simuliert werden [Devl-97, S. 97 ff.] und ist dort aufgrund des wesentlich einfacheren Modells auch entsprechend einfacher zu realisieren [BeMa94, S. 121 f.]. Trotzdem wird bereits bei der Verwendung des Relationen-Modells regelmäßig von der Versionierung abgeraten [Kimb96, S. 100 ff.], da diese nur äußerst schwer zu handhaben ist. [RiBr-99, S. 1 ff.] Deshalb ist es essenziell, dass die Versionierung durch das OBMS übernommen wird (Versionierungs-Manager) und nicht wie bei RDBMS manuell codiert werden muss.

Schema-Versionierung ist prinzipiell das gleiche wie Instanz-Versionierung, findet jedoch auf einer Meta-Ebene statt. Während Instanz-Versionierung die Änderung von Attributwerten auf den Ebenen von atomaren Attributen, komplexen Attributen, Instanzen, Klassen-Extensionen (Instanzen einer Klasse, siehe 3.4.3.6) oder Datenbanken betrifft [SST97, S. 374], ist Schema-Versionierung auf eine Änderung der Strukturen des OBMS-Schemas bezogen (Schema-Evolution), d. h. der Metadaten des Schemakatalogs [SST97, S. 370].

Bertino und *Martino* teilen die Schema-Evolution in die drei Kategorien Klassen-Definition (1), Vererbungs-Hierarchie (2) und Klassen-Set (3) ein, wobei Modifikationen bezüglich der Klassen-Definition nach Methoden- (1.2) und Attribut-Änderungen (1.1) differenziert werden (siehe Abbildung 11).[122] *Saake*, *Schmitt* und *Türker* unterteilen dies in nur zwei Evolutionsebenen, Änderungen des Datenbankschemas (2 und 3) und Änderungen der Klassenintension (Klassenimplementierung, 1). [SST97, S. 395 ff.]

Es können sich also nicht nur die Werte der Instanz-Attribute, sondern auch die Struktur und das Verhalten des OBMS-Schemas verändern. Während das erstere den Normalfall für ein DBMS darstellt, werden Änderungen am DBMS-Schema als außergewöhnlich angesehen. Es wird sogar immer wieder davon ausgegangen, dass die Strukturen eines DBMS langfristig konstant bleiben. [Dema79, S. 257 ff., Sche88, S. 567 ff., Voss94, S. 23] Auf der anderen Seite wird immer wieder betont, wie schnell sich der Wandel vollzieht. [Booc96, S. 17 ff., ChGl99b, S. 4 ff., GGC97, S. 1 ff., Tayl92, S. 5 ff.]

---

[122] Obwohl die verschiedenen Taxonomien prinzipiell sehr ähnlich sind, gibt es Variationen je nach Objektmodell des OBMS. Dem Modell von *Bertino* und *Martino* liegt das ORION-Modell zugrunde. [BeMa94, S. 129 ff.]. *Kemper* und *Moerkotte* stellen ein erweitertes Modell speziell für den technische Bereich (CAD etc.) vor. [KeMo94, S. 450 ff.] *Bretl et al.* entwickeln eine Variante für GemStone. [BMO+89, S. 300 ff.]

(1) Modifying the class definition
  (1.1) Modifying attributes
    (1.1.1) Adding attributes
    (1.1.2) Deleting attributes
    (1.1.3) Renaming attributes
    (1.1.4) Modifying the domain's attributes
    (1.1.5) Modifying the inheritance of attributes (attributes inherited from another superclass bear the same name)
    (1.1.6) Modifying the default value attributes
    (1.1.7) Manipulating shared attributes
      (1.1.7.1) Transforming non-shared attributes into shared attributes
      (1.1.7.2) Modifying the value of shared attributes
      (1.1.7.3) Transforming shared attributes into non-shared attributes
    (1.1.8) Modifying composite attributes into non-composite attributes
  (1.2) Modifying methods
    (1.2.1) Adding methods
    (1.2.2) Deleting methods
    (1.2.3) Modifying the name of methods (signature)
    (1.2.4) Modifying the implementation of methods
    (1.2.5) Modifying inheritance of methods (a method inherited from another superclass bears the same name)
(2) Modifying the inheritance hierarchy
  (2.1) Making a class S the superclass of a class C
  (2.2) Removing a class S from the list of superclasses of a class C
  (2.3) Modifying the order of the superclasses of a class C
(3) Modifying the set of classes
  (3.1) Creating new classes
  (3.2) Deleting classes
  (3.3) Modifying class names

Abbildung 11: Taxonomie der Schema-Evolution in OBMS nach *Bertino* und *Martino* [BeMa94, S. 129 ff.]

Dementsprechend meint beispielsweise *Taylor*: „Change is the only constant" [Tayl92, S. 5] (siehe Abschnitt 2.3.3). Diese Änderungen betreffen natürlich auch die Strukturen des DBMS-Schemas. Ändert sich die Diskurswelt, muss sich auch das Objektmodell ändern — sonst passt es nicht mehr. Schema-Änderungen sind mithin keine Ausnahme, sondern die Regel.

Bertino und *Martino* meinen deshalb treffend:

> „… application environments that OODBMS support are characterized by the fact that schema changes are the rule rather than the exception. In fact, it is natural that, for example, during a design application, there are changes in the way in which objects are classified and in whch relations between them are established. This is also due to the length of time that these applications last." [BeMa94, S. 121]

Wie auch *Kemper* und *Moerkotte* ausdrücken:

> „A schema is a set of type definitions — including their definitions of the structure and the behavior. A schema is not fixed once written, but evolves over time in order to capture the ever-changing requirements. New types are introduced, old types deleted, operations modified, errors eliminated, and so on." [KeMo94, S. 441]

Diese Änderungen sind mithin elementar und betreffen direkt das Objektmodell. Folglich sind sie ein inhärenter Bestandteil des Objektmodells.

Während Schema-Änderungen in RDBMS nur eingeschränkt und mit äußerst hohem Aufwand zu bewerkstelligen sind [KoLo95, S. 377 ff.], können diese bei OBMS teilweise dynamisch, d. h. sogar zur Laufzeit, beispielsweise abhängig von Methoden erfolgen [Gems96b, S. 8-1 ff., S. 13-1 ff.]. *Heuer* meint etwa:

> „In objektorientierten Datenbankanwendungen sind nicht nur die Objekte dynamisch, sondern auch die Objekttypen. Ein starres Schema wie in relationalen Systemen ist in OODBSs nicht brauchbar. Dort können (etwa in der aktuellen SQL-Norm) gerade einmal neue Attribute zu einem Relationenschema hinzugefügt werden und komplette Relationenschemata neu erzeugt oder gelöscht werden. Weitere Änderungen sind nicht erlaubt." [Heue97, S. 393]

Wesentlich ist, dass diese Schema-Änderungen (Schema-Evolution) nicht nur durchgeführt werden können. Auch eine Versionierung der Schema-Änderungen, eine Schema-Versionierung ist notwendig. Bestandteil der Metadaten sind dann die Änderungen der Instanz-Versionierung und Schema-Versionierung, also die jeweiligen Zustände von Werten, Struktur und Verhalten wie auch die einzelnen Historien — explizit oder ableitbar.

Ändern sich etwa die Strukturen eines Object Warehouse, ist es noch erheblich schwieriger, Auswertungen auf dem verfügbaren Objekt-Bestand durchzuführen oder überhaupt nicht möglich. Deshalb ist die Schema-Versionierung ein elementarer Bestandteil des Objektmodells (Reconciled Object Layer, siehe Kapitel 4) eines Object Warehouse. Anders ist die Adäquatheit, Konsistenz und Auswertbarkeit nicht langfristig zu gewährleisten. [RMP99, S. 98 ff.]

Der Unterschied zwischen Instanz-Versionierung und Schema-Versionierung verwischt aber. Semantische Konstrukte des Objektmodells sind Metadaten. Beispielsweise können viele der im Relationen-Modell explizit zu formulierenden Metadaten im OBMS-Objektmodell implizit mit den vorhandenen Konzepten abgebildet werden. Metadaten sind mithin integraler Bestandteil des Objektmodells selber. Dasselbe gilt für das Versionierungs-Konzept.

Wird weiterhin bedacht, dass etwa bei GemStone auch Klassen Objekte sind, deren Definition wiederum über Metaklassen dynamisch verfügbar und manipulierbar ist, da diese wiederum Objekte sind, wird das Ausmaß der diesbezüglichen Möglichkeiten ersichtlich. Statische Datenbankmodelle bzw. OBMS-Modelle

sind somit bereits vom Ansatz her wenig nutzbringend — im Kontext dieser Arbeit wertlos.

Auch *Saake*, *Schmitt* und *Türker* sehen den Bedarf, Daten und Metadaten in ähnlicher Weise zu behandeln:

> „Eine grundsätzliche Anforderung ist, dass Metadaten wie 'normale Daten' behandelt werden. Metadaten sollen vom Nutzer bzw. von den Anwendungsprogrammen (Analyse-Tools, CASE-Tools, etc.) wie normale Daten abgefragt und geändert werden können. Eine Schemaänderung ist also nichts anderes, als eine Änderung eines Metadatums." [SST97, S. 392 f.]

Damit ist wiederum eine notwendige Anforderung für ein Object Warehouse bzw. für MSS-Front-Ends erfüllt, da die Bedeutung für den Aufbau und den Betrieb eines Object Warehouse sowie die Informationsgewinnung mit MSS-Front-Ends eine entsprechende Metadaten-Handhabung erfordern (siehe Kapitel 2), inbesondere wenn die MSS-Front-Ends in der Lage sind, objektorientierte Eigenschaften realisieren zu können.

Eine weitere Form der Evolution ist die **Instanz-Evolution**. Die Instanz-Evolution fällt nicht in die beiden bisher betrachteten Kategorien. Die Instanz-Evolution bezeichnet den Wechsel einer Instanz zu einer anderen Klasse. [BeMa94, S. 138 ff., SST97, S. 402 f.] Es ändern sich in der Regel nicht nur die Werte der Instanzen, sondern auch deren Strukturen.[123] Allerdings hat die Änderung der Strukturen einer Instanz keine Schema-Änderung des OBMS-Schemas zur Folge, da die Instanzen innerhalb der bereits verfügbaren Strukturen verändert werden.

Der einfachste Fall der Instanz-Evolution ist die **Instanzmigration**. Hierbei wechselt eine Instanz die Klassenzugehörigkeit (siehe zu diesem Begriff auch Fußnote 114). Beispielsweise kann ein Student in einem universitären Verwaltungssystem nach Beendigung seines Studiums zu einem Wissenschaftlichen Mitarbeiter werden, wodurch sich einige Eigenschaften, d. h. Attribute, Beziehungen und Methoden wie auch die Zugehörigkeit zu Extensionen ändern werden.[124]

Die Objektidentität der Instanz muss unverändert erhalten bleiben. In der neuen Klasse nicht-existente Attribute der Instanz gehen mit ihren Werten verloren.

---

[123] Es kann auch einen Klassenwechsel ohne Änderung der Strukturen und sogar ohne Änderung der Werte der Instanz-Attribute geben, dies ist jedoch ein zu vernachlässigender Spezialfall.

[124] Doch selbst in Fällen, in denen die Eigenschaften der Klassen identisch sind, d. h. im Extremfall keinerlei Veränderungen von Attributen, Beziehungen und Methoden oder sogar der Extensionen (weil zum Beispiel überhaupt keine Verwaltung von Extensionen stattfindet), drückt die Zugehörigkeit zu einer anderen Klasse bereits eine veränderte Semantik aus. Deshalb müssen auch zwei Instanzen, die aufgrund ihrer identischen Attributstrukturen und -werte als gleich gelten, als ungleich betrachtet werden, nachdem eine von beiden Instanzen zu einer anderen Klasse migriert wurde.

Üblicherweise realisieren OBMS dies durch die Zusendung etwa eines move-Kommandos.[125] [SST97, S. 138]

Bei der **Instanz-Spezialisierung** findet keine Migration im eigentlichen Sinne statt. Die Instanz migriert zu einer Klasse, die Unterklasse bzw. Suptyp der bisherigen Klasse der Instanz ist. Da Unterklassen Superklassen nur erweitern, gehen in der Regel keine Eigenschaften der Instanz verloren, sondern werden lediglich erweitert. Dies trifft zumindest auf die Attribute (Struktur und Typ) und die Extensionen zu.

Beim Konzept der **Instanzrollen** gibt es keine Instanzmigration, da die Instanz nicht von einer Klasse zu einer anderen Klasse wechselt. Stattdessen kann eine Instanz bei diesem Konzept eine mehrfache Klassenzugehörigkeit realisieren. Es gibt also einen Rolleneintritt und Rollenaustritt, was als Typerweiterung bzw. Reduzierung der Typkapazität gesehen werden kann. [SST97, S. 139 ff.]

Dadurch kann etwa eine Person gleichzeitig und unabhängig voneinander die Rolle eines Studenten und Mitarbeiters wahrnehmen, d. h. Instanz der Klasse Student und Instanz der Klasse Mitarbeiter sein. Ist das Studium beendet, wird die Klasse Student verlassen, während die Klassenzugehörigkeit zu Mitarbeiter beibehalten werden kann.

Saake, *Schmitt* und *Türker* empfehlen gemeinsame Instanzrollen einer Instanz nur für Klassen mit einer gemeinsamen Superklasse zu realisieren [SST97, S. 139 ff.], wie im Beispiel der Oberklasse Person und den Unterklassen Mitarbeiter und Student. *Bertino* und *Martino* verlangen jedoch eine davon unabhängige Umsetzung des Konzepts. [BeMa94, S. 138 ff.]

Gerade das Rollenkonzept wird von *Saake*, *Schmitt* und *Türker* als essenzielles Modellierungskonzept in einem OBMS gesehen, da Objekte in einem OBMS eine langfristige Lebensdauer aufweisen und in der Folge natürlicherweise Rollenwechseln unterliegen. Als Alternative sehen sie ausschließlich die Verwendung der Mehrfachvererbung, die dann allerdings zu einer „kombinatorischen Klassenexplosion“ (Klassendefinitionen) führe.[126] Außerdem betonen sie die Wichtigkeit einer solchen expliziten „has-role-Beziehung“ bzw. „role-of-Beziehung“:

> „Die Notwendigkeit für das Rollenkonzept ergibt sich aus dem Wunsch nach möglichst natürlicher Unterstützung der menschlichen Klassifikation in Objektmodellen.“ [SST97, S. 139]

---

[125] GemStone etwa bietet die Methode „changeClassTo: neueKlasse“ um eine Klasse zu migrieren [Gems96a, S. 2-290], „migrateInstancesTo: neueKlasse“ [Gems96a, S. 2-92] um alle Instanzen einer Klasse zu einer anderen zu migrieren und „migrateTo: neueKlasse“ [Gems96a, S. 2-92] um eine Migrations-Destination zu setzen, die dann die Migration einer Instanz zu der angegebenen neuen Klasse bewirkt, wenn die Instanz die Nachricht „migrate“ erhält [Gems96b, S. 8-8 f.].

[126] Siehe allerdings die Abschnitte 3.3.3 und 3.3.4, da, aus Design- und Implementierungssicht, für diesen Fall Vererbung überhaupt als ungeeignet erscheint.

Auch für die Instanz-Evolution muss eine geeignete Versionierung durch das OBMS realisiert werden, im Folgenden als **Instanz-Evolutions-Versionierung** bezeichnet. Dadurch werden insgesamt drei Kategorien der Versionierung unterschieden:

1. Instanz-Versionierung
2. Schema-Versionierung
3. Instanz-Evolutions-Versionierung

3.4.3.4 Sicht-Konzept

Das Sicht-Konzept dient dazu, spezielle Sichten auf das OBMS-Objektmodell zu modellieren und nach Außen hin bereitzustellen, beispielsweise in Form von Objekt-Ausschnitten, auf einer höheren Objektgranularität oder auch abgeleiteter Objekte bzw. Objekt-Schemata. Die Realisierung des Sicht-Konzepts wird als für DBMS essenzielle Anforderung gesehen. [Date00, S. 289 ff.]

Mithilfe des Sicht-Konzepts wird das externe Schema implementiert, eine der drei grundlegenden Schichten der Drei-Ebenen-Schema-Architektur des ANSI/-SPARC-Standards (siehe Abbildung 13 auf Seite 181), durch die eines der grundlegenden Ziele von DBMS, die logische Datenunabhängigkeit, erreicht werden soll (siehe Abschnitt 3.4.3.7). [Embl98, S. 22 f.]

Dadurch können unterschiedliche Formen der Schema-Evolution vor den auf dem externen Schema aufsetzenden Applikationen versteckt werden, d. h. Änderungen des OBMS-Objektmodells verlangen zwar eine Anpassung der Sichten, nicht aber der aufsetzenden Applikationen. Beispielsweise kann dadurch ein Operational Data Store ein sauberes integriertes Datenschema realisieren, für die Legacy Systems aber weiterhin die alten Strukturen bereitstellen.

Bertino und *Martino* halten den Sichtansatz auch anstelle einer Schema-Evolution für denkbar. Statt das OBMS-Schema zu ändern, wird nur der View verändert, wodurch beispielsweise beim Löschen eines Attributs dieses nicht verloren geht, sondern lediglich ausgeblendet wird. [BeMa94, S. 138] Beim Hinzufügen eines Attributs oder bei bestimmten strukturellen Änderungen ist dieser Ansatz jedoch nicht adäquat und kann besser durch die Verwendung einer Schema-Versionierung ersetzt werden, die ohnehin ein wichtiges Konzept darstellt, das verfügbar sein sollte.

Sichten dienen aber auch zur Realisierung von Zugriffskontrollen oder Struktur- und Verhaltensabstraktionen für spezielle Anwendungen, beim ODS etwa für proprietäre OLAP-Werkzeuge. Besonders die Bildung von Ausschnitten des OBMS-Objektmodells ist eine wichtige Zielsetzung, da das gesamte Modell zu

umfangreich und zu komplex ist, um durch die Endanwender direkt benutzt werden zu können. [Embl98, S. 22 f.]

Beim Relationen-Modell werden Sichten (Views) normalerweise durch gespeicherte Anfragen realisiert. Doch obwohl das Relationen-Modell durch sein wesentlich einfacheres Paradigma (siehe oben) auch eine ganz erheblich geminderte Realisations-Problematik für das View-Konzept zur Folge hat, sind auch dort noch vielerlei konzeptionelle Fragestellungen zu klären und vor allem entsprechende Lösungen in kommerzielle RDBMS zu implementieren. [Date00, S. 294]

Auch für das OBMS-Objektmodell werden Sichten durch Anfragen (siehe Abschnitt 3.4.3.7) realisiert. Eine weitere Möglichkeit bietet ein struktureller Ansatz, bei dem ein Objekt-Schema etwa durch das Verstecken oder Umbenennen von Klassen oder Attributen erfolgt. Ein solcher Ansatz ist beispielsweise ViCOM, bei dem Datentypen, Attribute, Beziehungen, Methoden, Objekttypen, Klassen und Vererbungsbeziehungen mögliche veränderbare Strukturelemente sind und sich global auf das Schema oder lokal auf eine Klasse beziehen können. [SST97, S. 388 ff.]

Embley unterscheidet dabei zwischen einer vollständigen Sichtbarkeit (public), einer nicht vorhandenen Sichtbarkeit (hidden) und der Möglichkeit von Elementen, auf die nur lesend (read only) zugegriffen werden darf. [Embl98, S. 634 ff.]

Saake, *Schmitt* und *Türker* empfehlen, eine Kombination dieser beiden Ansätze zu verwirklichen, da die Anforderungen beim OBMS weitergehend sind. Etwa meinen sie:

> „Um Modellierungskonstrukte wie Beziehungen und Spezialisierungshierarchien angemessen berücksichtigen zu können, sollte das Ergebnis einer Sichtdeklaration im Falle von ODBMS nicht nur eine einzelne Klasse sein, sondern eine Objektdatenbank. In diesem Kontext kann damit auch nur ein Teil einer Objektdatenbank gemeint sein. Das heißt, eine Sicht kann sich nur auf einen Ausschnitt eines Objektdatenbankschemas (etwa einige ausgewählte Klassen) mit den dazugehörigen Objekten beziehen.“ [SST97, S. 383]

Beispielsweise können Attribute ausgeblendet, umbenannt oder umstrukturiert werden. Beziehungen können von einer unidirektionalen in eine bidirektionale Beziehung erweitert werden oder es kann eine transitive Beziehung einer Klasse A zu einer Klasse C über eine Klasse B unter Ausblendung der Zwischenklasse B in der Sicht realisiert werden.

Außer diesen klassen-intensionalen Veränderungen können Sichten auch die Klassen-Extension betreffen, d. h. die Sicht kann die Menge der Instanzen einer Klasse beschränken, zum Beispiel auf die Mitarbeiter der Mitarbeiter-Extension mit einem Gehalt von über 5.000,- DM pro Monat.

Eine andere Auffassung besteht darin, dass eine eigene Sichtdefinition beim Objekt-Paradigma von geringerer Priorität ist, da ADT über die Kapselung von Methoden und Attributen bereits eine Schnittstelle in Form externer Sichten auf das Objektmodell realisieren. [Catt91, S. 126 ff.] Die Form der Datenunabhängigkeit durch Kapselung ist einer der wichtigsten Vorteile des Objekt-Paradigmas.

Embley sieht Kapselung vor allem als Möglichkeit zur Realisation der physischen Datenunabhängigkeit (Implementation Independence). [Embl98, S. 23] *Cattell* hingegen sieht Kapselung als probates Mittel für physische und logische Datenunabhängigkeit. [Catt91, S. 126 ff.] *Lausen* und *Vossen* meinen etwa, dass logische Datenunabhängigkeit durch ADT erreicht sei, da die Methoden-Implementierung geändert werden könne, ohne die nach außen bereitgestellte Schnittstelle verändern zu müssen. [LaVo96, S. 36]

Bei einfachen Schema-Änderungen, wie etwa dem Hinzufügen oder Umbenennen von Attributen oder dem Modifizieren von Methoden-Implementierungen oder -Signaturen (siehe Abbildung 11 auf Seite 152) kann dieser Mechanismus leicht verwendet werden. Bei komplexeren oder umfangreicheren Schema-Änderungen, wie beispielsweise bei der Änderung einer Klassenstruktur, die etwa die Einführung einer Beziehungsklasse erfordert (siehe obiges Beispiel), ist es kein adäquates Mittel, nur die Schnittstelle der Klassen zu ändern, da sich auch die Logik des Schemas ändert.

Außerdem werden umfangreiche Schema-Änderungen mit der Zeit zu einem Objektmodell führen, das den Anforderungen einer angemessenen Modellierung nicht mehr entsprechen wird und stattdessen zu einem unüberschaubaren „Chaos“ führt. Zumindest mittelfristig ist dann eine Anpassung der abhängigen Applikationen die sinnvollere Lösung.

In Anbetracht, erstens der wesentlichen Zielsetzung des Objekt-Paradigmas, bessere Anpassbarkeit und Erweiterbarkeit zu ermöglichen, zweitens, dem Gegenstand, dass es beim Objekt-Paradigma überhaupt keine getrennten, d. h. „eigenen“ bzw. „eigentlichen“ Applikationen mehr gibt und drittens, ohnehin bereits verschiedene Unabhängigkeit realisierende Schichten bestehen (Domänen-Modell, Applikations-Modell, GUI-Schicht), erscheint es als eine probatere Lösung, die benötigten Modifikationen in allen abhängigen Klassen vorzunehmen als noch weitere Schichten zu pflegen und in einem Objektmodell parallel die Diskurswelt abzubilden und alle Änderungen nach außen abzufangen.

Die eigentlichen Probleme, sehr viele einzelne Client-Applikationen anpassen zu müssen und die zu ändernden Klassen zu finden und zu ändern, kann inzwischen einfacher bewerkstelligt werden. Ersteres kann inzwischen durch Lösungen wie automatische Code-Propagierung oder eigene Applikationsserver angemessen

gelöst werden, letzteres durch komfortable Entwicklungsumgebungen[127] bzw. CASE-Werkzeuge.

Sichten, die bestimmte Zustände des Schemas, d. h. Versionen des Schemas repräsentieren — was insbesondere für Endanwender sinnvoll ist — helfen hingegen, viele der beschriebenen Schwierigkeiten zu lösen.

Für rein lesende Zugriffe stellt das Sicht-Konzept in hohem Maße nutzbringende Lösungen bereit. Beim Object Warehouse können Sichten deshalb beispielsweise dazu genutzt werden, Ausschnitte oder sogar andere Strukturen (z. B. die Abbildung von Dimensionen und Fakten in Form einer Art Snowflake-Schema) für optimierte Anfragen zu erzeugen, die dann sogar problemlos materialisiert werden können.

Für vollständige logische Datenunabhängigkeit fordert *Cattell* „invertible Mappings", d. h. die Möglichkeit nicht nur lesend sondern auch schreibend auf Sichten zugreifen zu können. Dies ist jedoch prinzipiell in hohem Maße problematisch. [Date00, S. 289 ff., Embl98, S. 643 ff.]

3.4.3.5 Persistenzmodelle

Das im Gegensatz zum Relationen-Modell sehr komplexe Objektmodell und die sehr verschiedenen Objekt-Paradigmen der einzelnen OBMS führen auch zu entsprechend unterschiedlichen Modellen zur Realisierung von Persistenz. *Saake*, *Schmitt* und *Türker* klassifizieren Persistenzmodelle anhand von vier Konzepten [SST97, S. 162 ff.]:

1. Persistenzdefinition
   - ◊ typabhängig
     - durch Vererbung (Persistence by Inheritance) [Schm91, S. 53 ff.]
     - durch spezielles Programmiersprachenkonstrukt (Schlüsselwort)
   - ◊ typorthogonal
2. Persistente Extensionen
   - ◊ implizit (alle Instanzen einer bestimmten Klasse)
   - ◊ explizit (z. B. durch Schlüsselwort)
   - ◊ keine Unterstützung

[127] VisualWorks verfügt über einen solchen Mechanismus. [PaDi95a, S. 115 ff.]

3. Persistentmachung
   - ◊ statisch (bei Objekterzeugung, z. B. mit speziellem new-Operator)
   - ◊ dynamisch (zur Laufzeit, z. B. mit einer Funktion)
4. Persistenzpropagierung
   - ◊ implizit (Persistance by Reachability, d. h. Persistenz der gesamten transitiven Hülle)
   - ◊ explizit

Aus den möglichen Kombinationen ergeben sich theoretisch 24 Kategorien von Persistenzmodellen. Allerdings sind nur einige wenige davon von praktischer Relevanz. Zwei Beispiele sind das Modell der ODMG und das GemStone-Modell bzw. O_2-Modell.

Das Modell der ODMG [CaBa98, S. 11 ff.] zeichnet sich aus durch:

1. typabhängige Persistenz (Klasse)
2. explizite persistente Extensionen durch das Schlüsselwort „extent“ bei der Klassendefinition
3. statische Persistentmachung
4. explizite Persistenzpropagierung, d. h. die transitive Hülle des Objekts wird nicht implizit (automatisch auch) persistent

Die Persistenzmodelle von GemStone [Gems96b, S. 2-1 ff., Gems96c, S. 3-1 ff.] und O_2 [Gepp97, S. 36 f.] realisieren:

1. typorthogonale Persistenz
2. keine Klassen-Extensionen
3. dynamische Persistentmachung
4. implizite Persistenzpropagierung

Obwohl das GemStone-Modell völlig anders als das ODMG-Modell ist, entspricht es doch dem ODMG-Standard. Das GemStone-Modell ist ein reines Smalltalk-Modell und entspricht folglich auch den Smalltalk-Bindings, die Persistence by Reachability mit Persistence by Naming (durch Benennung) und den Verzicht auf explizite Klassen-Extensionen empfiehlt.

Persistence by Naming bedeutet, dass ein Eintrag in einem SystemDictionary von GemStone erzeugt wird, das Wurzel-Objekt (Root oder Named Object). Dessen transitive Hülle wird durch Pointer Chasing automatisch persistent (Persistence by Reachability).

Praktisch wird im Smalltalk-Client ebenfalls ein Eintrag im SystemDictionary vorgenommen (globale Variable) und mit einem Eintrag des Dictionary in Gem-

Stone gemappt. Die transitive Hülle dieser globalen Variablen wird für den Benutzer transparent nach GemStone geschrieben (repliziert oder direkt über einen Forwarder) und gespeichert. [Gems96c, S. 4-1 ff.]

Die wesentlichen Funktionen eines Persistenzmodells sind das Erzeugen, Ändern und Löschen eines Objekts, wie auch der Zugriff auf die Objekte. Während Persistenz bei RDBMS impliziter Bestandteil jeder Relation ist, muss diese beim OBMS explizit erfolgen. [Heue97, S. 266 ff.]

Durch die Aufhebung der Grenze zwischen DBMS und Applikation beim Objekt-Paradigma [SST97, S. 176 ff.] kann es transiente und persistente Objekte geben, die sich sonst nicht weiter unterscheiden und auch identisch behandelt werden, was gerade dem Wunsch nach der Überwindung des Impedance Mismatch entspricht (siehe Abschnitt 3.4.3.1).

Dementsprechend wichtig ist auch, ob ein Objekt dynamisch und typorthogonal persistent gemacht werden kann. Von entscheidender Bedeutung ist ebenso die Erzeugung von Klassen-Extensionen, da SQL-artige Anfragesprachen im Wesentlichen darauf beruhen, dass Relationen auch alle Elemente beinhalten (siehe die Abschnitte 3.4.3.6 und 3.4.3.7).

3.4.3.6 Typen, Klassen und Vererbung

Bisher sind Klassen als Modellierungskonzept, d. h. Abstraktionen von konkreten Phänomenen (siehe die Abschnitte 3.3.2 und 3.3.4.6), als Implementierungskonzept von abstrakten Datentypen oder als Schablone für ein Objekt bzw. als Objektfabrik (siehe Abschnitt 3.3.4.5) betrachtet worden. Dabei besteht eine Klasse aus bzw. wird beschrieben anhand von Attributen und Methoden.[128] Diese Form der Betrachtung einer Klasse bezeichnen *Saake*, *Schmitt* und *Türker* als Klassen-Intension [SST97, S. 91 f.], da das Innere einer Klasse beschrieben wird.

Im Datenbankbereich wird jedoch von der Einzelbetrachtung eines Objekts abgewichen, die speziell bei der OOP im Mittelpunkt des Interesses steht. Wie *Heuer* meint:

> „Außerdem wird in objektorientierten Programmiersprachen nie eine Objektmenge betrachtet, die aus allen bisher betrachteten Elementen einer Klasse besteht, sondern nur alle Objekte einzeln.“ [Heue97, S. 299]

[128] Genaugenommen besteht eine Klasse aus Attributen, Methoden und Beziehungen. [CaBa98, S. 35 ff.] Der Unterschied zwischen Attributen und Beziehungen ist jedoch nicht eindeutig, da Beziehungen auch über Attribute realisiert werden. Jedes nicht atomare Attribut kann als Beziehung zu einem weiteren Objekt aufgefasst werden. [CaBa98, S. 36] Ein sehr wesentlicher Unterschied zwischen Attribut und Beziehung ist allerdings, dass Attribute vollständig gekapselt sind, während dies auf Beziehungen in dieser Form nicht zutrifft. [ElNa94, S. 672 ff.]

Im Datenbankbereich findet eine Konzentration auf Mengen von Objekten statt. Dementsprechend werden unter dem Begriff Klasse weniger die primär auf die Klassen-Intension gerichteten Konzepte verstanden, sondern die Menge der Objekte, die zu einer Klasse gehören. Die Menge der Objekte einer Klasse wird gemeinhin als die Extension einer Klasse oder Klassen-Extension bezeichnet. [CaBa98, S. 16, ElNa94, S. 675 ff., Heue97, S. 299, LaVo96, S. 31, SST97, S. 92 f.]

Heuer vertritt die Meinung, dass diese Auffassung über den Klassenbegriff auch dem umgangssprachlichen Begriff der Klasse entspricht. [Heue97, S. 298 ff.] *Liu* meint aus Sicht der OOP:

> „'Class' is a misleaning word because it has meanings that predate object-oriented programming. Schoolchildren are commonly taught to use 'class' as a synonym for 'set.' Mathematicians, on the other hand, use 'class' to refer to collections that were found nearly a century ago to be too large to be treated as sets. (An example of such a collection is the 'class of all sets.')
>
> Of course, both the schoolchild and the mathematician have in the back of their mind some notion of a collection of entities. For object-oriented developers, however, the better conceptual model for a 'class' is a factory rather than a collection." [Liu96, S. 13]

Entscheidend ist jedoch die Verwaltung von Objektmengen, also Sets durch ein OBMS, da Betrachtungen einzelner Objekte eine geringere Priorität besitzen. Dies entspricht auch dem Konzept von RDBMS, die Mengen durch Relationen bzw. durch Tabellen realisieren.[129] Dies ist auch eng verbunden mit der Realisation von Persistenz. Wie im entsprechenden Abschnitt bereits erläutert (siehe Abschnitt 3.4.3.5) gibt es dort folglich auch das Konzept der Extensionen (Extents), das auch durch die ODMG als Standard festgelegt ist [CaBa98, S. 16].

Elmasri und *Navathe* meinen:

> „Notice the difference between standard database models and OO databases in this respect. In a typical database model, such as the EER or the relational model, **all** objects are assumed to be persistent. Hence, when an entity type or subclass such as EMPLOYEE is defined in the EER model, it represents both the **type declaration** for EMPLOYEE and a **persistent set** of **all** EMPLOYEE objects. In the typical OO approach, a class declaration of EMPLOYEE only specifies the type and operations for a class of objects. The user must separately define a persistent object of type set(EMPLOYEE) or list(EMPLOYEE) whose value is the **collection of references** to all persistent EMPLOYEE objects […]

[129] *Heuer* meint in diesem Kontext: „Bei einem Schema im Datenbanksinne wird der Mengenkonstruktor also immer implizit gefordert und nicht mehr explizit hingeschrieben." [Heue97, S. 276]

> The main reason for this difference is that it allows transient and persistent objects to follow the same type and class declarations of the OODDL and the OO programming language. Because we are mainly interested in database applications here, we assume that, for every class declaration, the class name refers to both the **type** and the **operations** definitions, as well as to **the set of all persistent objects** of that class.“ [ElNa94, S. 674 f.]

Aus all dem wird deutlich, wie elementar im Datenbankbereich die Betrachtung und Realisierung von Klassen-Extensionen ist. *Heuer* meint sogar:

> „Außerdem ist das Zusammenfassen gleichartiger Objekte zu Klassen die wichtigste Grundlage für objektorientierte Datenbanken.“ [Heue97, S. 191]

Auch Datenbank-Anfragesprachen, die selbst wieder als notwendiger Bestandteil eines DBMS angesehen werden, basieren maßgeblich auf dem Konzept der Klassen-Extensionen (siehe Abschnitt 3.4.3.7). [LaVo96, S. 185 ff.]

Es ist hilfreich, nicht nur die abweichenden Formen des Klassen-Begriffs zu differenzieren, sondern auch von den verschiedenen Konzepten der Klassenhierarchien und diese wiederum von den unterschiedlichen Konzepten der Vererbung zu trennen (siehe auch die Abschnitte 3.3.4.5 und 3.3.4.6). Klassen, Klassenhierarchien und Vererbung sind zwar eng verflochten, sind jedoch in einer Reihe von Kombinationen möglich, die in der Folge zu sehr unterschiedlichen Konzepten führen. Hierin liegt sicherlich einer der Hauptunterschiede zwischen den einzelnen Objekt-Paradigmata — synonyme und vor allem homonyme Begriffsverwendungen bewirken ein übriges.

Klassen werden in Klassenhierarchien organisiert. Dies gilt analog für die Mengen-Betrachtung. Eine Unterklasse ist dann eine Untermenge. Die Menge der Studenten ist dann eine vollständige Untermenge von Personen, wenn Personen die Oberklasse von Studenten ist. Es gilt auch das Prinzip der Substituierbarkeit. Da jeder Student auch zur Klasse der Personen gehört, muss auch jede Person durch einen Studenten substituiert werden können.

Die Bildung einer Klassenhierarchie erfolgt im Datenbankbereich mithin wiederum nach anderen als den bisherigen Kriterien. Idealerweise sollte jedoch versucht werden, eine Klassenhierarchie derart zu bilden, dass gleichzeitig eine Typhierarchie und eine Klassenhierarchie im Sinne von Mengen realisiert wird. Nur wenn der intensionale und extensionale Charakter einer Klasse gemeinsam verwirklicht werden kann, ist echte Substituierbarkeit gegeben.[130]

Dies bedeutet, dass eine Unterklasse zumindest über alle Attribute und Methoden-Signaturen der Oberklasse verfügt[131] und die Extension der Oberklasse auch

---

[130] „Der Begriff der Substituierbarkeit verbindet beide Formen der Spezialisierung.“ [SST97, S. 134]

[131] *Saake*, *Schmitt* und *Türker* erheben lediglich die Forderung nach „vorhanden sein“ der Attribute und Methoden: „Alle Attribute und alle Methoden, die auf einer allgemeineren Klasse definiert sind, müssen auch in der

alle Instanzen der Unterklasse enthält. Verfügt eine Klasse beispielsweise über fünfzehn Oberklassen, muss eine Instanz dieser Klasse auch in die Extensionen aller fünfzehn Oberklassen eingefügt werden. Dabei wird zwischen der flachen Extension unterschieden, die nur die Instanzen der Klasse selbst enthält und der tiefen Extension, die zusätzich die Instanzen aller Unterklassen umfasst. [SST-97, S. 134 ff.]

Zur Implementierung der Klassenhierarchien kann für diese Anforderungen die bei den meisten OOPL übliche Form der Vererbung verwendet werden. Alle Attribute und Methoden werden vererbt. Die Methoden-Implementierungen können verändert werden, die Attributwerte sind instanzspezifisch[132] . Die Attributstrukturen und Methoden-Signaturen bleiben unverändert.[133]

Dadurch wird insgesamt klar, dass das OBMS-Paradigma bei der Betrachtung der Klassen deutlich von den Konzepten der OOP und OOD abweicht und diese in der Konsequenz grundsätzlich infrage stellt. Das grundlegende Konzept der OOP ist der ADT und dessen Kapselungseigenschaft. Die Konzepte der OOD basieren ebenfalls grundsätzlich auf ADT, d. h. auf einer Programmierung nur über die Schnittstelle. Der Typ einer Klasse wird ausschließlich über die Schnittstelle der Klasse definiert, d. h. das Protokoll bestehend aus den Signaturen der öffentlichen Methoden. Von der Attributstruktur, den Attributwerten und den Methoden-Implementierungen wird abstrahiert (siehe Abbildung 12).

Im Datenbankbereich wird diese strenge Form der Kapselung aufgegeben. Auch wenn die Kapselung in OBMS ein noch umstrittenes Thema ist, besteht doch weitgehend Einigkeit darüber, dass zusätzlich zu den Methoden-Signaturen die Attributstrukturen und deren Werte öffentlich gemacht werden müssen (siehe Abbildung 12). [Catt91, S. 128 f., ElNa94, S. 672, Heue97, S. 384 ff., SST97, S. 229 ff.] Datenbank-Einkapselung besteht somit lediglich in der Kapselung von Methoden-Implementierungen und speziellen privaten Bestandteilen.[134]

---

spezielleren Klasse vorhanden sein.“ [SST97, S. 134] Zu den weitergehenden Bedingungen siehe Abschnitt 3.3.4.5.

[132] *Heuer* spricht in diesem Zusammenhang von Wertvererbung und meint damit, dass im Gegensatz zum OOPL-Konzept nicht nur die Attribut-Strukturen, sondern auch deren Werte vererbt werden. [Heue97, S. 190 f.] Diese Vorstellung scheint jedoch aus einer speziellen Betrachtung der Vererbung herzurühren. Wird die Instanz einer Unterklasse lediglich als die zusätzliche Teilinstanz der dann nur virtuell vorhandenen Gesamtinstanz eben jener Unterklasse gesehen und mit den dazugehörigen Teilinstanzen der Oberklassen verbunden, so kann dies als eine Art Wertvererbung interpretiert werden. Wird die Instanz jedoch als vollständige Instanz erzeugt, dann erbt lediglich deren Klasse — nicht jedoch die Instanz — und zwar die Attributstruktur für die Erzeugungs-Schablone. In diesem Fall scheint der Begriff der Wertvererbung unangebracht und irreführend. Echte Wertvererbung gibt es dagegen beispielsweise bei Self, durch das Konstrukt der Klassenvariablen bei Smalltalk oder bei Eigenschaften von GUI-Elementen (z. B. Paradox für Windows [Fink93, S. 493 ff.]).

[133] Allerdings können die Attributtypen bzw. Methoden-Parameter eingeschränkt werden (Kovarianz/Kontravarianz). (Siehe Abschnitt 3.3.4.5.)

[134] Diese Aussage ist selbstverständlich zu differenzieren, da noch nichts darüber ausgesagt ist, wem gegenüber und in welcher konkreten Ausprägung diese stark reduzierte Form der Kapselung gilt. Die strenge Kapselung

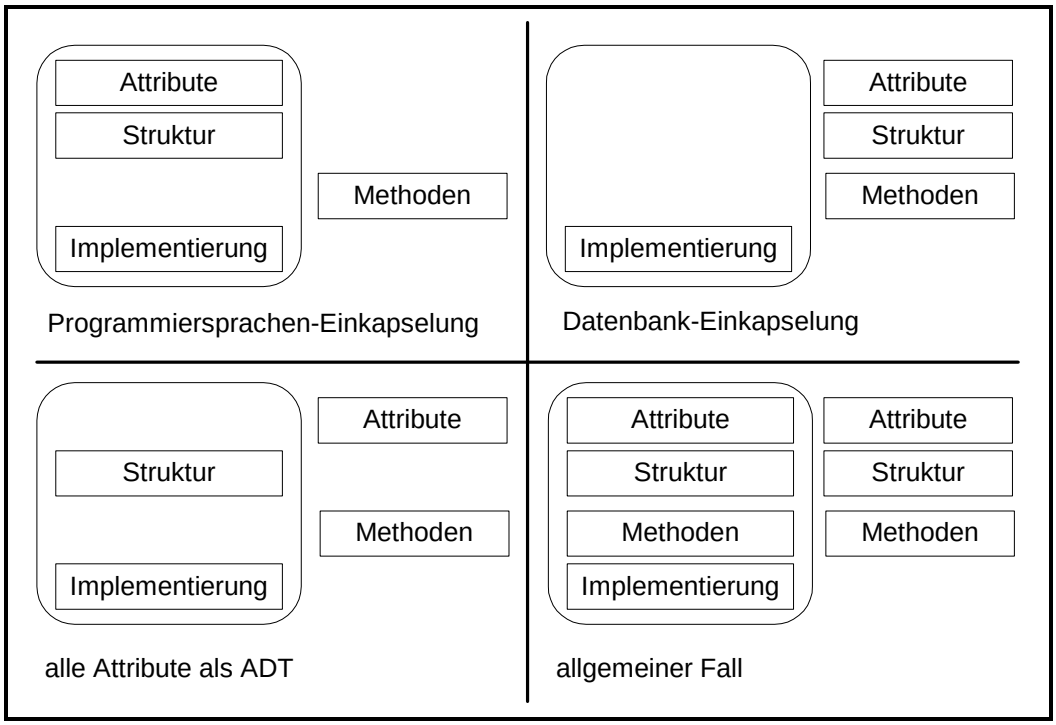


Abbildung 12: Programmiersprachen- und Datenbank-Einkapselung im Vergleich [Heue97, S. 286]

Private Bestandteile können dabei Methoden und Attribute sein, die auf jeden Fall vor dem Zugriff von außen geschützt werden sollen, da sie lediglich Implementierungsdetails darstellen. Beispielsweise bei C++ wird deshalb zwischen „public", „private" und „protected" unterschieden. [AhFi92, S. 34]

Die Betrachtung der Attribute ist deshalb auf zwei unterschiedlichen Ebenen elementar. Zum Ersten müssen Attribute beim Entwurf des Objektmodells nicht nur berücksichtigt werden, sondern bilden das zentrale Interesse der Modellierung. Anders ist die Bildung konsistenter redundanzfreier Objektmodelle überhaupt nicht zu erreichen.[135] Entwurfsmethoden, die von den Attributen abstra-

der OOP ist bei der Verwendung einer OOPL mit Sicherheit die beste Lösung. Und auch ein gutes objektorientiertes Design sollte weiterhin auf Schnittstellen-Programmierung basieren. Auch direkte Updates sind etwa durch die eventuelle Umgehung operationaler Constraints als problematisch zu beurteilen.

[135] Interessanterweise meinen verschiedene Autoren (z. B. [Balz96, S. 770]), dass beim Objekt-Paradigma ein mit der Normalisierung vergleichbarer Prozess überflüssig sei. Der Prozess ist tatsächlich anders. Etwa kann auf die Bildung der ersten Normalform aufgrund des List- bzw. Mengenkonstruktors (List, Bag, Set) verzichtet werden. Auch besteht eine verminderte Redundanz, da Werte in vielen Fällen nicht doppelt gespeichert werden müssen, sondern durch Referenzen abgebildet werden. Das ändert jedoch nichts am prinzipiellen Vorgehen der Normalisierung. Beispielsweise kann eine Rechnungssumme redundant in der Rechnungsklasse gespeichert werden, da sich diese aus den einzelnen Auftragspositionen et cetera ergibt. Dies ist zwar weniger tragisch, wird die strenge Kapselung der ADT verwendet, also ausschließlich über die Methoden zugegriffen, birgt jedoch weiterhin ein erhöhtes Inkonsistenzrisiko, weil z. B. auch die Formulierung und Wartung operationaler Constraints durchaus nicht unproblematisch sein kann. Ausserdem führt ein schlechter Entwurf zur redundanten Speicherung von Attributen in eventuell sehr unterschiedlichen Klassen. Auch werden beispielsweise Konstrukte wie inverse Beziehungen in der Regel durch zwei unterschiedliche Zeiger implementiert, die durchaus inkonsistent werden können. [RMK99]

hieren [BeCu89, S. 81 ff., Beck91, S. 61 ff., JCJÖ95, S. 69 ff., WWW90, S. 37 ff.], sind deshalb für Datenbank-Applikationen genauso unbrauchbar wie entsprechende Implementierungs-Methodiken [Beck00, S. 43 ff.]. Zum Zweiten ist zumindest der lesende Zugriff auf Attribute notwendig, um eine Ad-hoc-Anfragesprache zu ermöglichen (siehe Abschnitt 3.4.3.7).[136]

Ein Typ im OBMS-Bereich ist dann nicht mehr nur über die Schnittstelle im Sinne der OOP definiert, also die öffentlichen Methoden, sondern über die Schnittstelle im Sinne eines OBMS, also die Methoden-Signaturen (Methoden, deren Parameter und Parametertypen (Input/Output)) und die Attribute (Strukturen und Typen), (siehe Abbildung 12). Dementsprechend wird dann auch der Begriff der Typhierarchie verstanden. Die Klassenhierarchie wird dann als Obermengen-Untermengen-Hierarchie interpretiert. [SST97, S. 121 ff.]

Lausen und *Vossen* definieren den Typ eines Objekts sogar ausschließlich über die Attribute, nicht über deren Methoden. Den um die Methoden erweiterten Typ verstehen sie als Klasse. [LaVo96, S. 30 f.]

3.4.3.7 Object Query Language

Zunächst wird im Folgenden geklärt, inwieweit überhaupt die Notwendigkeit für eine Anfragesprache in Verbindung mit einem OBMS besteht, welchem Zweck eine solche Anfragesprache dient und wie sie sich von einer OOPL bzw. DPL, wie sie bisher erläutert wurde, unterscheidet. Die Diskussion, wie die Methoden einer Object Query Language (OQL) im Gegensatz zu einer Methode einer OOPL aussehen, wird für besonders wichtig erachtet und ist deshalb entsprechend detailliert.

Anschließend wird die Bedeutung der Deklarativität einer solchen Sprache erläutert und die Sprachmittel werden besprochen. Ein besonderes Problem stellen die Anfrageergebnisse einer OQL dar, wofür, nach einer geeigneten Klassifikation, erste mögliche Lösungsansätze aufgezeigt werden. Der Abschnitt schließt mit einer Zusammenfassung der unterschiedlichen geforderten Konzepte einer OQL und einem Fazit.

Notwendigkeit und Zielsetzung einer Ad-hoc-Anfragesprache

Die Möglichkeit, Ad-hoc-Anfragen an den Objektbestand zu richten, wird als weitere notwendige Fähigkeit eines OBMS gesehen. [ABD+89] Auf den Datenbe-

[136] Zumindest bis jetzt gibt es zu diesem Zweck keine andere Möglichkeit. Eine Alternative — ohne Berücksichtigung physischer Aspekte wie Optimierung — wäre für alle relevanten Attribute Zugriffsmethoden zu definieren (virtual Data [Catt91]), die dann von einer Anfragesprache verwendet würden. Dadurch wäre die Kapselung aber ebenfalls umgangen [Holu99], wenn auch in leicht geminderter Form.

stand von RDBMS wird grundsätzlich über eine deklarative Anfragesprache (z. B. SQL), eine dem entsprechende grafische Anfragesprache (z. B. QBE) oder sonstige grafische Werkzeuge für solche Zwecke (z. B. bestimmte MSS-Front-Ends), die in der Regel auch SQL-Code generieren, zugegriffen. Die Verwendung einer Programmiersprache mit Ad-hoc-Character ist bei relationalen Systemen mithin der Normalfall.

Beim Objekt-Paradigma ist dies grundsätzlich anders. Die fehlende Trennung von Daten und Funktionen, die nicht mehr klar zu ziehende Grenze zwischen Applikation und Datenbank bzw. Objektbank (siehe Abschnitt 3.4.1) und die weitgehende Aufhebung des Impedance Mismatch (siehe Abschnitt 3.4.3.1) führen gerade zur Überwindung der Trennung von APL und DPL. Die prinzipiell andere Vorgehensweise lässt sich gut an einem Beispiel verdeutlichen.

Für die Information des Gesamtumsatzes eines Kunden wird am besten eine entsprechende Methode, beispielsweise „Gesamtumsatz“ beim Kundenobjekt implementiert (siehe Abschnitt 3.3.3.2). Der Kunde wiederum bzw. die Methode „Gesamtumsatz“ benötigt dazu die Umsätze aller Aufträge. Dazu wird beim Auftrag ebenfalls eine entsprechende Methode implementiert, die auch „Gesamtumsatz“ benannt sein kann. Diese Methode wiederum benötigt alle Gesamtumsätze der Auftragspositionen. Eine entsprechende Methode der Auftragsposition benötigt den Preis, den es vom Artikel bekommt.

Da sich die Preise der Artikel gewöhnlich von Zeit zu Zeit ändern, wird die Methode des Artikels zum Beispiel das Datum des Auftrags benötigen, welches es (das Datum) dann vom Auftrag bekommt, wozu der Artikel von der Auftragsposition den entsprechenden Auftrag mitgeteilt bekommen muss oder die Auftragsposition teilt das Datum dem Artikel mit, welches (das Datum) die Auftragsposition dann vom Auftrag erfragt hat.

Wird dann zusätzlich eine Methode benötigt, die den Gesamtumsatz des Kunden eines bestimmten Jahres oder einer Woche oder eines speziellen Zeitraums errechnen soll, können ein Grossteil der Methoden wiederverwendet werden. Zum Beispiel könnte der Kunde eine Methode „Gesamtumsatzprojahr“ mit dem Parameter des Jahres implementieren. Dann könnte die Auftragsmenge des Kunden, sofern diese als eigenes Objekt[137] realisiert wird, eben nur jene Aufträge dieses Jahres aussortieren, so dass genau diese Aufträge ihren Gesamtumsatz an die Methode des Kunden übermitteln.

Diese Art der Vorgehensweise wirkt auf den ersten Blick umständlich, doch kann genau damit das Potenzial des Objekt-Paradigmas in Bezug auf Wiederverwendung, Erweiterbarkeit, Wartung etc. in vollem Umfang genutzt werden. Werden

[137] Dies ist beispielsweise bei GemSonde der übliche Weg. [Gems96b, S. 4-8]

mithin neue Anfragemethoden benötigt, ist die Implementierung sehr einfach und schnell durchzuführen.

Es sprechen allerdings einige Gründe dafür, dies nicht als einzige Methode zur Realisierung von Anfragen zu verwenden. Nicht jeder Benutzer soll Methoden der Objekte hinzufügen oder verändern dürfen. Ein großer Teil der Anfragen wird nur wenige Male benötigt, so dass die explizite Implementierung zu aufwendig ist, also ineffizient, mit zunehmender Code-Menge die Wartungs- und Ausführungseffizienz überproportional sinkt und diese Vorgehensweise für Ad-hoc-Anfragen von Endbenutzern überhaupt zu kompliziert ist.

Deshalb fordern *Atkinson et al.* eine deklarative, effiziente, applikations-unabhängige Ad-hoc-Anfragesprache, die sich auf der einen Seite durch ihre Einfachheit in der Bedienung und auf der anderen Seite durch ihre Mächtigkeit bei der Formulierung komplexer Anfragen auszeichnet. *Atkinson et al.* verlangen jedoch nicht unbedingt eine explizite Anfragesprache, sondern lediglich die Möglichkeit der Formulierung von Ad-hoc-Anfragen, was durchaus auch mithilfe eines grafischen Werkzeugs in ausreichendem Maße realisiert werden kann. [ABD+89]

Auch *Elmasri* und *Navathe* halten die Idee der Vorformulierung aller möglichen auch noch so simplen Anfragen durch Methoden für unhaltbar, so dass sich als Konsequenz die Forderung nach einer Ac-hoc-Anfragesprache mit besonderen Eigenschaften ergibt. [ElNa94, S. 672 ff.]

Die Vorformulierung widerspricht vor allem dem grundlegenden MSS-Gedanken. MSS haben zur Aufgabe, gerade die Ad-hoc-Formulierung aller möglichen Anfragen erstens zu ermöglichen und zweitens so einfach wie möglich zu machen, ohne dass Endanwender zur Erzeugung expliziten „Programm-Codes“ gezwungen sind.

Es hat sich im MSS-Umfeld sogar herausgestellt, dass selbst SQL, durch welches ein wesentlicher Schritt in diese Richtung realisiert wird, für die Zielsetzungen von MSS zu kompliziert sind. Aus diesem Grund haben auch MSS wie multidimensionale DBMS und OLAP-Front-Ends starke Verbreitung gefunden (siehe Kapitel 2) und verringern die Bedeutung fest programmierter Kennzahlen.

Methoden einer Ad-hoc-Anfragesprache

Obiges Beispiel wird dann auf völlig andere Weise gelöst. Anstelle der Aufteilung des Algorithmus zur Berechnung des Kundengesamtumsatzes auf die einzelnen jeweils zuständigen Objekte erfolgt die Berechnung vollständig innerhalb einer Methode, der Anfragemethode, die die dafür benötigten Daten der jeweiligen Objekte von diesen erfragt.

Dazu gibt es prinzipiell zwei Möglichkeiten. Entweder erfolgt der Zugriff auf die Daten über die jeweilige Schnittstelle der Objekte, d. h. über Methoden, was

dann der strengen Kapselung der OOP entspricht und die Ausnutzung bereits vorhandener Berechnungsfunktionen erlaubt oder direkt auf die Attribute, was dann bedeutet, dass die Kapselung, zumindest für die Anfragesprache, aufgegeben wird.

Der direkte Zugriff auf die Attribute wird von fast allen Autoren im OBMS-Bereich gefordert [CaBa98, S. 83 ff., Catt91, S. 128 f., Date00, S. 816 ff., ElNa94, S. 672 f., Heue97, S. 384 ff., Khos93, S. 140 ff., LaVo96, S. 173 f., Mano94, S. 222 ff., SST97, S. 229 ff., Voss94, S. 375 ff.], wodurch die Kapselung auf die Kapselung der Methoden-Implementierungen reduziert wird[138] (siehe auch Aschnitt 3.4.3.6, insbesondere Abbildung 12 auf Seite 165).

Werden ohnehin nur die Werte der Attribute abgefragt, handelt es sich für diesen Zweck um rein passive Objekte. Der Zugriff über (reine Zugriffs)-Methoden ist dann, zumindest unmittelbar, nur noch wenig sinnvoll — der direkte Zugriff auf die Attribute wesentlich effizienter.

Der primäre Grund für die Forderung, die Kapselung der Attribute für eine Adhoc-Anfragesprache aufzugeben, besteht in der notwendigen Optimierung der Anfragen durch das OBMS. Wird auf Attribute direkt zugegriffen, ist dies in jedem Fall effizienter als über Methoden. Die Verwendung von Methoden führt zu einer nicht einzuschätzenden Komplexität der Anfrage. Methoden können zu ungewollten Nebeneffekten führen. Methoden sind berechnungsvollständig und deshalb nicht sicher, so dass bei deren Verwendung in einer Anfrage die Sicherheit der Anfrage nicht mehr garantiert werden kann, da beispielsweise Probleme der Unentscheidbarkeit auftreten können [SST97, S. 230 ff.]. Vor allem ist jedoch die Optimierbarkeit einer Anfrage durch das DBMS nicht mehr vollständig möglich und deshalb insgesamt ganz erheblich eingeschränkt.[139]

Die Optimierung einer Anfrage allein auf den Daten, d. h. mit direktem Zugriff auf die Attribute ist aufgrund der wesentlich komplexeren Strukturen eines OBMS-Modells ohnehin wesentlich schwieriger zu bewerkstelligen als beim Relationen-Modell. Kommt allerdings die Aufgabe hinzu, in diese Anfrageoptimierung die Methoden der verwendeten Objekte mit einzubeziehen, die außerdem

---

[138] Erfolgt der Zugriff ausschließlich direkt auf den Attributen, ist die Kapselung der Methoden-Implementierungen allerdings sinnlos, da die Methoden ohnehin keine Verwendung finden. Von Kapselung kann dann nicht mehr gesprochen werden.

[139] *Saake*, *Schmitt* und *Türker* weisen zusätzlich darauf hin, dass die Verwendung von Methoden bei einem OBMS, welches selbst keine Methoden ausführen kann und deshalb alle von der Anfrage benötigten Objekte zuerst in den Client transferiert werden müssen, nicht effizient sein kann, insbesondere wenn es sich um Seitenserver handelt. [SST97, S. 474 ff.] Dies verdeutlicht allerdings mehr den Mangel solcher OBMS-Architekturen und sollte deshalb nicht weiter als Argument gegen die Verwendung von Methoden in Anfragen betrachtet werden. Zugegebenermaßen sind solche eingeschränkten OBMS bisher noch die Regel. [Loom95, S. 47 f., Mano94, S. 213 ff.] (Siehe insbesondere Abschnitt 3.4.3.8.)

noch über einen berechnungsvollständigen Befehlssatz verfügen, erscheint dies zumindest im Moment als eine „praktisch“ unlösbare Aufgabe.

Gelingt es allerdings, die Methoden der Objekte durch das OBMS analysierbar zu machen, mit in die Optimierung einzubeziehen, sicher zu machen und vergleichsweise effizient auszuführen, ist diese Vorgehensweise zu bevorzugen.

Auf jeden Fall erscheint es als wünschenswert, einen vollständigen Verzicht der Einbeziehung von Methoden zu vermeiden. [CaBa98, S. 83 f., LaVo96, S. 200 ff.] Hierbei könnte es beispielsweise sinnvoll sein, zwischen zwei Arten von Methoden zu unterscheiden.

Methoden die, wenn auch nur mittelbar, Veränderungen bzw. echte Aktionen bewirken, sollten prinzipiell nicht in Anfragen verwendet werden. Sie entsprechen nicht primär dem Gedanken des Information Retrieval und sind in ihren Auswirkungen kaum zu kontrollieren.

Anders verhält es sich mit Methoden, die aus einfachen Berechnungen bestehen, um etwa berechnete Attribute abbilden zu können (die z. B. aufgrund der dritten Normalform eliminiert werden müssen), wie beispielsweise das Alter (Alter = aktuelles Datum - Geburtsdatum), eine Rechnungssumme oder ob eine Person ein Rentner ist (Alter > 65 Jahre und Arbeit = nil)[140] .

Die Verwendung solcher Methoden wäre ein wesentlicher Schritt im Sinne des Objekt-Paradigmas, da der Endanwender weder die Berechnungsfunktion zu kennen braucht, noch wissen muss, dass dieses Attribut überhaupt berechnet wird. *Lausen* und *Vossen* empfehlen deshalb

> „… zwischen Attributen und parameterlosen Methoden nicht weiter zu unterscheiden und die Verwendung von Methodennamen überall dort zu gestatten, wo Attributnamen auftreten können.“ [LaVo96, S. 51]

Cattell schlägt in diesem Zusammenhang das Konzept virtueller Attribute vor, wodurch für den Benutzer nicht ersichtlich ist, ob es sich um echte oder berechnete Attribute handelt. [Catt91, S. 90 ff.] Auch die nachträgliche Änderung, ob es sich um ein echtes oder berechnetes Attribut handelt sollte möglich sein, ohne den Code der Anfragemethoden ändern zu müssen.[141]

Bertino und *Martino* empfehlen dafür die Verwendung von „Derived Attributes“, was dem Konzept berechneter Attribute entspricht und „Predicate Methods“, die

---

[140] Dieses Beispiel ist natürlich grob vereinfacht.

[141] Bei C++ besteht der Unterschied etwa darin, dass der Aufruf einer Methode durch abschließende leere Klammern gekennzeichnet wird (z. B. „d.alter()“), während Attribute ohne diese Klammern beschrieben werden (z. B. „d.alter“). Soll dann später eine Methode anstelle des direkten Attributs verwendet werden, müssen zumindest die Klammern in den Code eingefügt werden. [Catt91]

insofern eine Besonderheit darstellen, da sie Bool'sche Werte als Ergebnis haben. [BeMa94, S. 92 f.]

Allerdings führt auch dieses Vorgehen zu einer Erschwerung der Anfrageoptimierung und unter Umständen zu verminderter Ausführungssicherheit. Die Methoden könnten jedoch speziell gekennzeichnet sein, um sie von aktionsauslösenden Methoden zu unterscheiden und zur Gewährleistung der Sicherheit lediglich über einen eingeschränkten Befehlssatz verfügen, was zu diesem Zweck in der Regel ausreichend sein dürfte.

Zusätzlich fordert etwa *Heuer* — analog zu SQL — generische Anfragemethoden durch die Anfragesprache bereitzustellen. [Heue97, S. 491 ff.] Generische Methoden widersprechen in mindestens ebensolchem Maße dem Objekt-Paradigma, wie die Durchbrechung des Kapselungsprinzips, also die Aufhebung von ADT. Die erhöhte Effizienz, Optimierbarkeit (durch das OBMS) und Sicherheit generischer Methoden wie auch die Einfachheit für den Benutzer [SST97, S. 230 ff.] lassen sie allerdings als unverzichtbar erscheinen.

Im Gegensatz zum SQL-Ansatz gilt das Konzept für eine Ad-hoc-Anfragesprache eines OBMS lediglich für rein lesende Zugriffe. Updates oder gar Änderungen des Schemas bergen unkontrollierbare Auswirkungen. Allein die Tatsache, dass ein erheblicher Anteil der zur Integritätssicherung des Objektbestandes notwendigen Constraints in den Methoden der Objekte implementiert wird, verbietet den direkten schreibenden Zugriff.

Auch wenn es dennoch Ansätze für Update-Mechanismen in einer OQL gibt [LaVo96, S. 56 ff., Voss94, S. 375 ff.], sind diese nur in speziellen Fällen einzusetzen. Die Gefahr der Schaffung von Inkonsistenz ist allerdings kaum auszuschließen. Bei rein lesenden Zugriffen ist dies unproblematisch. Die ODMG erlaubt deshalb zwar auch Updates durch die Anfragesprache, jedoch ausschließlich unter Verwendung der Methoden eines Objekts. Dessen ungeachtet widersprechen Änderungen insgesamt der grundsätzlichen Zielsetzung einer Ad-hoc-Anfragesprache und sind insofern für diesen Zweck nicht geeignet.

Weiterhin ist für eine reine Anfragesprache, die primär dem Zweck von Anfragen mit Ad-hoc-Charakter dient, die Durchbrechung des Kapselungs-Prinzips unproblematisch, da bei Änderungen des konzeptionellen Schemas dann eben auch die Anfragen geändert werden müssen. Für diesen Zweck scheint der resultierende Aufwand jedoch vertretbar.

Für komplexe Anwendungen in diesem Bereich, etwa für MSS, müssen entsprechend flexible Möglichkeiten der Anpassung der MSS-Werkzeuge realisiert werden oder es sind alternativ geeignete Sichten erforderlich (siehe Abschnitt 3.4.3.4).

Die Aufhebung der Kapselung darf allerdings nur für eine Anfragesprache erfolgen. Applikationen müssen streng über die Schnittstellen der Objekte implemen-

tiert werden. Strenge Kapselung ist mithin nicht unbedingt für das OBMS selbst und die Endbenutzer, im speziellen Nutzer von Ad-hoc-Anfragen notwendig, aber auf jeden Fall für Applikationen.

Deklarativität

Bislang wurde der Unterschied einer Ad-hoc-Anfragesprache zu einer OOPL durch die Verwendung nur einer Methode für die Anfrage, die Möglichkeit des direkten Zugriffs auf die Attribute, alternativ den Gebrauch von Methoden der Objekte und durch den Einsatz generischer Operationen beschrieben. Weiterhin sollte in einer Methode der Anfragesprache ein nur eingeschränkter Befehlssatz verwendet werden, damit Sicherheit, Effizienz und Optimierbarkeit durch das OBMS gewährleistet werden können.

Dies führt zur Forderung, dass eine Anfragesprache deklarativ sein muss. Der Benutzer der Anfragesprache soll lediglich spezifizieren, was er aus dem Objektbestand (konzeptionelles Schema oder Sichten) anfragen möchte — die Anfrage „programmieren“ soll er hingegen nicht. Die Spezifikation muss für ihn möglichst einfach zu bewerkstelligen sein.

Der eigentliche Algorithmus für die Anfrage wird aus der Spezifikation des Benutzers unter Einbeziehung der speziellen technischen Gegenbenheiten des DBMS (physisches Design) vom DBMS optimal generiert. Die Wahrung der physischen Datenunabhängigkeit ist dadurch garantiert.

Gerade die Ausnutzung des physischen Designs, wie beispielsweise die Ausnutzung von Indizes, Plattenspiegelungen, parallelen Prozessoren, die Optimierung des Cache oder die Festlegung der Ausführungsreihenfolge (Ausführungsplan) kann auf diese Weise optimal durch das OBMS vorgenommen werden und bleibt so vor dem Benutzer verborgen. Die Anfragesprache soll also lediglich auf das konzeptionelle Schema zugreifen und somit physische Datenunabhängigkeit realisieren (siehe Abschnitt 3.4.3.8).

Deshalb scheinen insbesondere für eine deklarative Anfragesprache auch generische Operationen hilfreich zu sein, da sie vollständig der Kontrolle des OBMS unterliegen und also auch nicht vom Benutzer beeinflusst werden müssen oder können.

Sprachmittel

Das Relationen-Modell basiert auf Mengen. Dadurch sind auch die Sprachmittel mengen-orientiert (Set-at-a-time). Programmiersprachen sind dies in der Regel nicht (Record-at-a-time). (Siehe Abschnitt 3.4.3.1.) Da bei OBMS, wie bei allen DBMS Mengen von Objekten im Zentrum des Interesses stehen — und zumeist

nicht ein einzelnes Objekt — muss auch eine Ad-hoc-Anfragesprache eines OBMS Mengen-Orientierung unterstützen.

Weil Objektmodelle — im Gegensatz zum Relationen-Modell — nicht ausschließlich die Menge als Konstrukt für Objektstrukturen verwenden (siehe Fußnote 107 auf Seite 137), sondern mit verschiedenen, ausdrucksstärkeren Mitteln komplexe Objektstrukturen bilden, müssen diese auch in den Anfragen ausgenutzt werden können. Anstelle eines expliziten Verbundes (Join-Operation) über Wertvergleiche (in der Regel Fremd- und Primärschlüssel) können die vorhandenen Referenzen ausgenutzt werden.

Dies erleichtert nicht nur die Navigation in komplexen Objekten, die besonders bei semistrukturierten Objekten unverzichtbar sind [ChSt93, S. 201 f.], sondern erlaubt auch die Verwendung von Pfadausdrücken in mengen-orientierten Anfragen. *Lausen* und *Vossen* zeigen, dass gerade in diesem Konzept ein erhebliches Potenzial einer objektorientierten Anfragesprache liegt. [LaVo96, S. 61 ff.] Komplexe Anfragen werden dadurch erheblich vereinfacht. [Heue97, S. 297]

Teilweise wird die Verwendung von Referenzen als Aufgabe der logischen und physischen Datenunabhängigkeit und als Rückfall in das implementierungsnahe hierarchische Datenmodell oder Netzwerkmodell (CODASYL) gesehen. *Date* bezeichnet OBMS deshalb sogar als 3GL-Systeme. [Date00, S. 843] Diese Polemik ist allerdings nicht haltbar. Bei relationalen Systemen müssen die Strukturen genauso bekannt sein, sind allerdings nicht im Modell eines RDBMS vorhanden.

Vor allem aber können Zugriffe in Anfragesprachen von OBMS mengen-orientiert **und** über direkte Referenzen erfolgen, d. h. es ist durchaus möglich, Pfadausdrücke in einer deklarativen Anfragesprache zu verwenden und mithin von Implementierungsdetails zu abstrahieren. Außerdem ist mit ADT ein zusätzliches Abstraktions-Konzept vorhanden. Der explizite Verbund ist ebenfalls weiterhin möglich, da normalerweise nicht alle interessierenden Beziehungen in einem Objektmodell vordefiniert sind.[142]

In der Verwendung von Pfadausdrücken in deklarativen mengen-orientierten Zugriffen in Verbindung mit dem OID-Konzept liegt ein enormes Potenzial des Objektmodells. Gerade das Fehlen von Pfadausdrücken im Relationen-Modell stellt eine erhebliche Einschränkung dar — nicht nur für hoch komplexe Anwendungen.

Dates Kritik greift mithin nicht. *Hughes* meint deshalb auch:

---

[142] *Lausen* und *Vossen* geben das folgende Beispiel, wobei in ihrem Objektmodell keine entsprechende Beziehung vordefiniert ist [LaVo96, S. 50]:

```
select p, f
from p in Person, f in Firma
    where p.Name = f.Hersteller.Praesident.Name
```

„… kann der Kritk [von *Date*] am objektorientierten Modell nicht zugestimmt werden (Date, 1990), die behauptet, die derzeitigen Datenmanipulationstechniken im objektorientierten Modell führten zu einer satzweisen Verarbeitung und navigierenden Vorgehensweise wie in den alten Netzwerkdatenbanken. Man kann weder einen Datensatz mit einem komplexen Objekt, noch eine verzeigerte Liste mit einer Klassenhierarchie vergleichen. Tut man es dennoch, so ist dies unfair gegenüber dem objektorientierten Modell und seinen ausgeklügelten Datenabstraktionsmechanismen.“ [Hugh92, S. 148]

Insgesamt sollen alle Sprachmittel des jeweiligen Objektmodells adäquat unterstützt werden und auch Erweiterungen des Objektmodells unterstützen. Beispielsweise muss es auch möglich sein, eine Methode zum Zugriff auf ein nicht durch das Objektmodell strukturiertes Objekt, wie z. B. ein Video, einzubinden, um dadurch, als ein Beispiel, die jeweiligen Anfänge einer Video-Collection zu selektieren (siehe auch Abschnitt 3.4.3.2). [LaVo96, S. 52]

Khoshafian beispielsweise macht Vorschläge zur Verarbeitung von Textdaten und Imagedaten mit OQL-Konstrukten (Object-Oriented Intelligent SQL), um diese Daten zu typisieren (Struktur und Operationen) und in Anfragen ausnutzen zu können. [Khos93, S. 190 ff.][143]

Auch die Vererbung in Form von Klassenhierarchien (im Sinne des Mengenbegriffs) muss adäquat unterstützt werden, was sich im Konzept der Extensionen niederschlägt (siehe Abschnitt 3.3.4.5). [CaBa98, S. 16, Heue97, S. 299, SST97, S. 124 ff.] GemStone oder O_2, die dieses Konzept bisher nicht implizit unterstützen, erlauben hingegen die einfache Bildung von Multiklassifikations-Schemata.[144]

---

[143] Beispiel mit Textintegration [Khos93, S. 199]:

```
SELECT Author, Publisher, RANK (BookText)
FROM Books
WHERE
    [„Object Orientation“ AND „Databases“ OR „Semantic data models“]
          IN BookText
    AND Year <= 1986
```

Beispiel mit Imagedaten-Integration (geometrische Daten) [Khos93, S. 201]:

```
SELECT City, State, Population
FROM Cities
WHERE Location COVERED-BY (4 +/- 1, 9 +/- 1)
    AND Population > 1,000,000
```

[144] Sollen Extensionen implementiert werden, ist dies mit erheblichem Aufwand verbunden. Eine einfachere Möglichkeit wäre, die Instanzen einer Klasse in jeweils einer Collection zu sammeln und bei Anfragen, welche sich auf die tiefe Extension beziehen, eine Vereinigungsoperation entweder explizit in der Anfrage oder — was erheblich einfacher für die Anfrage wäre — durch Aufruf einer entsprechenden Methode in der Collection-Klasse zu implementieren. Multiklassifikations-Schemata bilden ein starkes zusätzliches Ausdrucksmittel eines OBMS, insbesondere bei dem Verzicht auf strenge Typisierung und fehlender Mehrfachvererbung. Bei GemStone etwa liegt der Hauptgrund ursprünglich darin, dass dadurch effizientere Indizes erzeugt werden können — nur die Instanzen der Collection werden indiziert und nicht alle Instanzen der Klasse — und der Zugriffschutz leichter zu realisieren ist. [MSOP90, S. 217 f.] Ein gemeinsamer Protokollteil sollte allerdings in der Regel vorhanden sein, da die gemeinsame Menge ansonsten lediglich der Klassifikation dient, nicht jedoch der gemeinsamen Verarbeitung. Ein erhebliches Problem in diesem Zusammenhang ergibt sich gleichwohl, wenn Objekte gelöscht werden sollen. Die Verwendung von Persistence by Reachability und

Anfrageergebnisse

Im Relationen-Modell ist das Ergbenis einer Anfrage auf Relationen wieder eine Relation. Dies ist unproblematisch. Die Abgeschlossenheit der Anfragesprache ist damit garantiert.[145] Beim Objektmodell ist die Einordnung allerdings ein Problem, das kaum auf befriedigende Weise lösbar scheint.

Manola unterscheidet vier Arten von Anfrageergebnissen [Mano94, S. 222]:

1. Eine Menge existierender Objekte.
2. Eine Menge neuer Objekte existierender Klassen[146].
3. Eine Menge von Objektwerten (ohne eigene OID).
4. Eine Menge neuer Objekte von neuen Klassen.

Die **erste Variante** ist unproblematisch, es handelt sich um objekterhaltende Anfragen. Alle Anfragen, die lediglich Selektionscharakter haben (Selektion), also eine Teilmenge erzeugen bzw. reine Mengenoperationen sind (Vereinigung, Schnitt und Differenz), sind objekterhaltende Anfragen. Diese Form der Anfrage ist beispielsweise die einzige, die von GemStone unterstützt wird.

Bei der **zweiten Variante** muss die Klasse und die Zuordnung der Ergebnisattribute zu den Instanzvariablen angegeben werden. Es wird allerdings vorausgesetzt, dass bereits eine entsprechende Klasse im Objektschema vorhanden und diese dem Anwender bekannt ist. Für Ad-hoc-Anfragen dürfte dies lediglich einen, wenn auch interessanten, Spezialfall darstellen.

Die **dritte Variante** ergibt sich, wenn die Semantik einer objekterhaltenden Operation nicht ausreichend ist, jedoch keine expliziten Objekte erzeugt werden sollen oder können.

Da es bei Smalltalk-basierten Systemen wie GemStone keine Werte gibt, die nicht auch Objekte sind, gibt es dort auch keine Werte im eigentlichen Sinne. Trotzdem gibt es auch in diesen Systemen die Anforderung ein Ergebnis zu erzeugen, das ohne die Zuordnung oder das Erzeugen einer expliziten Klasse auskommt. Als Lösung hierfür kann beispielsweise der Array-Konstruktor verwen-

---

Garbage Collection verhindert das Löschen von Objekten, wenn diese zur transitiven Hülle persistenter Objekte gehören. Um ein Objekt löschen zu können, muss deshalb bekannt sein, in welchen Collections das Objekt vorhanden ist und durch welche weiteren Objekte das Objekt referenziert wird.

[145] Auch wenn es zum Beispiel in früheren SQL-Versionen noch nicht die Möglichkeit gab, Anfragen beliebig ineinander zu verschachteln. Hinter dem FROM-Konstrukt konnten etwa keine eigenen Anfragen eingesetzt werden. Durch temporäre Tabellen kann dieses Problem allerdings gemindert werden.

[146] Mit Klasse wird hier (weiterhin) der intensionale Teil einer Klasse bezeichnet, also der Klassenbegriff im Sinne der OOP, sofern dies nicht explizit anders gekennzeichnet ist. Die Betrachtung der Typdefinition einzig über die Schnittstelle spielt im Datenbankbereich allerdings keine Rolle, da auch das Prinzip der Kapselung im Sinne eines ADT der OOP nicht aufrechtzuerhalten ist (siehe Abschnitt 3.4.3.6) — der Typ eines Objekts ergibt sich aus seinen Methoden und Attributen (nicht hingegen den Attributwerten).

det werden, wodurch auch das Problem der fehlenden Zugriffsmethoden umgangen werden kann. Diese Methode wird etwa bei objekterzeugenden SQL-Anfragen angewandt, findet die ObjektLens von VisualWorks Verwendung. [Parc94, S. 5 ff.]

Eine Spezialform von Anfragen, die komplexe Werte als Ergebnis liefern, bildet eine Anfragesemantik, die von *Saake*, *Schmitt* und *Türker* als „relationale Semantik" bezeichnet wird. [SST97, S. 247 f.] Einen entsprechenden Ansatz gibt es beispielsweise bei O_2. [Gepp97, S. 93 ff.] Diese Form ist relativ einfach, da Klassen- bzw. Objektkollektionen als Mengen komplexer Tupel interpretiert werden können, wobei die bekannte SQL-Semantik um Konzepte geschachtelter Relationen erweitert wird. [SST97, S. 248]

Diese dritte Variante ist insgesamt relativ einfach umzusetzen, da die Beschränkungen der ersten beiden Varianten überwunden, die Probleme der vierten Variante vermieden werden, ist jedoch im Sinne des Objekt-Paradigmas unbefriedigend. Die Ergebnisse sind nicht adäquat im Sinne des Objektmodells und verletzen insbesondere das Prinzip der Abgeschlossenheit.

Die **vierte Variante** erfordert einen entsprechenden Klassenkonstruktor. Auch eine implizite Klassengenerierung ist möglich und für eine benutzerfreundliche OQL wünschenswert. Allerdings ergibt sich hierbei das Problem der Einordnung des Ergebnisses in die Klassen- und Typhierarchie, wobei Klasse im intensionalen und extensionalen Sinne verstanden werden muss. Außerdem stellt sich die Frage, inwieweit durch Oberklassen zugeordnete (ererbte) Methoden noch Gültigkeit besitzen oder zusätzliche Methoden notwendig werden. [SST97, S. 248 ff.]

Diese Variante ist für Operationen des Verbundes, der Aggregation und der Gruppierung notwendig. *Saake*, *Schmitt* und *Türker* rechnen die einfache Projektion allerdings zu den objekterhaltenden Anfragen. Die Objektidentität bleibt erhalten und zusätzlich können, unter anderem, berechnete Attribute aufgenommen oder nicht benötigte Attribute ausgeblendet werden.[147] [SST97, S. 250 ff.]

Diese Auffassung ergibt sich allerdings durch die andere Verwendung des Klassenbegriffs, den sie im Sinne der Klassen-Extension verstehen. Die Klassen-In-

[147] *Lausen* und *Vossen* vertreten ebenfalls diese Auffassung, zeigen in ihrem Beispiel allerdings nur den Fall der Verringerung von bereits vorhandenen Attributen [LaVo96, S. 200 ff.], wodurch die neue Klasse (im Sinne des Typs, also der Klassen-Intension) klar als Oberklasse eingeordnet werden kann. Für diesen eingeschränkten Fall ist die Einordnung in die intensionale Klassenhierarchie zwar einfacher, doch sind die Auswirkungen bei der Ausführung eines Programms als höchst problematisch anzusehen. Beispielsweise kann ein Adressobjekt, bei welchem auf die PLZ verzichtet wurde, keine Methode mehr ausführen, die diese PLZ benötigt. Dies dann als dasselbe Objekt anzusehen kann leicht zu Programmfehlern (Typfehlern) führen („does not understand"), die nur schwer nachzuvollziehen sind. Dies gilt umso mehr, wenn Objekte als Ergebnis einer Anfrage mit anderen Objekten gemeinsam weiterverarbeitet werden. Diese Anforderung besteht. [Loom95, S. 173 ff.]

tension (der Typ) wird durchaus verändert. Je nach Objektmodell des OBMS ergeben sich hieraus sehr unterschiedliche Konsequenzen.

Wird dieses Konzept von *Saake*, *Schmitt* und *Türker* tatsächlich verwirklicht, gibt es dasselbe Objekt gleichzeitig zwei Mal mit zwei unterschiedlichen Typen bzw. als Instanzen unterschiedlicher Klassen[148]. Dieses Konzept erscheint deshalb nicht effizient verwirklichbar, weshalb gerade in dieser Arbeit Projektionen als typische objektgenerierende Anfragen verstanden werden. Beispielsweise muss bei GemStone eine neue Klasse mit einer neuen Instanz erzeugt werden, die dann natürlich auch über eine neue OID verfügt[149].

Weiterhin ergibt sich die Frage, warum *Saake*, *Schmitt* und *Türker* Ergebnisse einer Anfrage unbedingt in Klassen-Extensionen einordnen wollen, die Einordnung in die Hierarchie der Klassen-Intension allerdings als weniger gravierendes Problem betrachten. Gerade aus OOP-Sicht ist dies nicht nachzuvollziehen. Es bleibt offen, warum Objekte einer Ad-hoc-Anfragesprache überhaupt in eine Extension eingeordnet werden sollen. Mehr als eine temporäre Speicherung ist zumindest im Sinne einer Ad-hoc-Anfragesprache nicht sinnvoll.

Insgesamt sind die erste und vierte Variante von größter Bedeutung. Die erste Variante hat den Vorteil die Objekte zu erhalten, so dass weitere Anfragen — etwa im Sinne eines navigierenden Zugriffs — in vollem Umfang unterstützt werden können. Für die Anforderungen, die darüber hinaus gehen, muss auch die vierte Variante zur Verfügung stehen.[150]

---

[148] Es könnte sich anstelle einer Zuordnung zu einer Klasse auch um eine Kombination von Klassen handeln, werden Konzepte wie Mehrfachvererbung oder Rollen unterstützt. Allerdings genügen diese Konzepte nicht. Zusätzlich wären etwa virtuelle Sichtobjekte oder Objektbestandteile notwendig.

[149] Natürlicht ist es möglich, dass diese neue Instanz Referenzen auf das oder die Originalobjekte (also deren OID) enthält, um den Bezug für weitere Anfragen aufrechtzuerhalten.

[150] Es sei angemerkt, dass etwa bei einem Objektmodell wie GemStone, für diesen Fall mit wesentlich weniger Problemen zu rechnen ist. Neu erzeugte Attributwerte (z. B. Umsatz = Preis * Menge oder bei einer Gruppierung eines Kunden (GROUP BY Kunde) mit Summenbildung des Umsatzes (SUM (Umsatz))) führen im Ergebnisobjekt konsequenterweise auch zu neuen Objekten. Die Instanzvariable des Ergebnisobjekts referenziert mithin ein neues Objekt (das wiederum in eine Semantik der obigen vier Varianten fällt). Attributwerte, deren Identitäten erhalten bleiben, die also dieselben Objekte bleiben (z. B. der Kunde über den (im vorigen Beispiel) gruppiert wurde), sind im Ergebnisobjekt konsequenterweise auch die identischen Objekte. Nichts hat diese Objekte verändert, was (siehe oben) durch eine Ad-hoc-Anfragesprache auch prinzipiell nicht geschehen sollte. Die Instanzvariable des Ergebnisobjekts referenziert mithin genau dasselbe Objekt, mit der selben OID. Das Ergebnisobjekt des Beispiels enthält dann eine Instanzvariable mit einer Referenz auf ein neues Objekt, den Gesamtumsatz und eine Instanzvariable mit einer Referenz auf ein schon bestehendes Objekt, den Kunden (dessen Gesamtumsatz gruppiert wurde). Dies liegt daran, dass Smalltalk alle Elemente als Objekte behandelt, eine Klasse direkt immer nur den einfachen Tupel-Konstruktor realisiert (die Konstruktion komplexer Typen in einer Klasse ist also nicht möglich) und Attributwerte durch Referenzen in den Instanzvariablen realisiert werden. Außerdem ist die für die Variante vier notwendige dynamische Erzeugung von Klassen und Instanzen aus einer Methode, mittelbar etwa einer OQL-Spezifikation, mit Smalltalk möglich. Das Problem der Klassen-Extensions-Zuordnung ergibt sich nicht, da GemStone keine Klassen-Extensionen unterstützt.

Zusammenfassung der Eigenschaften

Auf der Grundlage der Begründung und der damit einhergehenden Zielsetzungen einer Anfragesprache ergeben sich vielfältige Anforderungen an eine OQL. Die Zusammenfassung der Anforderungen der verschiedenen Autoren ergibt sich wie folgt, wobei diese Konzepte konkurrierend sind und deshalb eine Abwägung der jeweiligen Prioritäten erfolgen muss — beispielsweise ist es nicht möglich, gleichzeitig eine eingeschränkte und berechnungsvollständige Sprache zu realisieren [BeMa94, S. 81 ff., CaBa98, S. 83 ff., Heue97, S. 489 ff., LaVo96, S. 45 ff., SST97, S. 230 ff., Voss94, S. 375 ff.]:

- **Abgeschlossenheit**: Ergebnisse einer Anfrage müssen selbst wieder als eine Eingabe einer neuen Anfrage dienen können, so dass verschachtelte Anfragen möglich sind. Die Ergebnisse müssen also konsistent im Objektmodell darstellbar sein.
- **Ad-hoc-Formulierung**: Anfragen müssen einfach zu formulieren sein, ohne dass dafür „programmiert“ werden muss.
- **Adäquatheit**: Die vorhandenen Konstrukte des jeweiligen Objektmodells sollen ausgenutzt werden können. Beispielsweise müssen die Einbindung von Methoden wie der direkte Zugriff auf die Attribute oder die Verwendung von Pfadausdrücken in Anfragen genauso unterstützt werden, wie Klassen- und Typhierarchien zu beachten sind.
- **Anwendungsunabhängigkeit**:[151] Die Anfragesprache ist unabhängig von bestimmten Applikationen, sie darf also nicht auf Konstrukte von applikations-spezifischen Klassen angewiesen sein, d. h. für einen bestimmten Anwendungsbereich entworfen werden und verwendet im Allgemeinen das konzeptionelle Objektschema.
- **Deskriptivität**: Die Anfragesprache ist rein deskriptiv, es wird also lediglich das gewünsche Ergebnis spezifiziert, nicht jedoch eine Ausführungsreihenfolge oder gar ein Algorithmus angegeben. Von Implementierungsdetails wird also abstrahiert. Insbesondere muss Mengen-Orientierung bei Zugriffen unterstützt werden. Die Spezifikation für Benutzer soll auf diese Weise mit möglichst einfachen Mitteln zu bewerkstelligen sein.
- **Effizienz**: Die Anfragesprache muss eine effiziente Ausführung gewährleisten. Die Sprache muss also effizient implementiert sein und zusätzlich bei-

[151] Damit ist im Wesentlichen gemeint, dass die Sprachkonstrukte nicht in den Klassen der Applikationen implementiert werden dürfen. Dies ist unproblematisch. Die Anfragen dürfen sich aber auch nicht nur auf bestimmte Applikationen beschränken. Beim Object Warehouse ist auch diese Forderung unproblematisch, da das Objektmodell eines Object Warehouse aus genau diesem Grund modelliert und implementiert wird. Die anwendungs-unabhängige Anfrage soll ermöglicht werden. Das konzeptuelle Schema ist deshalb der Reconciled Object Layer, externe Sichten bilden den Derived Object Layer (siehe Kapitel 4).

spielsweise Indizes oder gespiegelte Festplatten ausnutzen können. Endliche Ergebnisse von Anfragen sollten auch in endlicher Zeit berechenbar sein.

- **Eingeschränktheit**: Die Anfragesprache darf nicht berechnungsvollständig sein.
- **Erweiterbarkeit**: Bei Erweiterungen des Datenbankmodells, z. B. durch benutzer-spezifische Typen oder etwa multimediale Typen muss die Anfragesprache auch für diese Erweiterungen angemessene Sprachmittel zur Verfügung stellen.
- **Formale Semantik**: Die Anfrageoperationen sollen (einfach) formal definiert sein.
- **Generische Operationen**: Die Anfragesprache verfügt über einige wenige generische Operationen, die einzig in Abhängigkeit der Typstruktur auf die Werte anwendbar sind.
- **Mächtigkeit**: Die Sprache sollte berechnungsvollständig (Turing-vollständig) sein und Konzepte wie beispielsweise rekursives Traversieren unterstützen.
- **Mengen-Orientiertheit**: Die Anfragesprache muss, analog zu SQL, auch Operationen unterstützen, die auf Mengen von Objekten basieren.
- **Optimierbarkeit**: Anfragen werden vor der Ausführung durch das OBMS optimiert, wozu geeignete Regeln bzw. Strategien verfügbar sein müssen.
- **Orthogonalität**: Die Operationen müssen beliebig kombinierbar sein.
- **Sicherheit**: Jede Anfrage muss terminieren, ein endliches Ergebnis liefern und darf keine ungewollten Nebeneffekte haben.
- **Updates nur über Methoden**: Die Anfragesprache stellt keine expliziten Update-Operationen zur Verfügung, kann aber durch die Einbindung von Accessing-Methoden der Objekte Updates unterstützen.
- **Vollständigkeit**: Die Anfragesprache muss für jedes Konzept des Objektmodells ein geeignetes Verarbeitungskonzept besitzen.

Fazit

Eine Ad-hoc-Anfragesprache ist für ein OBMS genauso unverzichtbar wie für jedes andere DBMS. Eben aus diesem Grund war SQL ein Meilenstein in der DBMS-Entwicklung. SQL ermöglichte es erstmals, beliebige Anfragen an einen Datenbestand zu stellen. Relativ einfach und schnell und vor allem auch für Endbenutzer.

Gerade für den MSS-Bereich sind einfache und effiziente Ad-hoc-Anfragen die zentrale Zielsetzung (siehe vor allem Abschnitt 2.2). Festprogrammierte Kennzahlen decken immer nur einen Teilbereich des betriebswirtschaftlich notwendi-

gen Informationsbedarfs eines Unternehmens ab. Die Chance einer Ad-hoc-Überprüfung von Hypothesen oder noch besser: „das Spielen mit den Daten“ birgt das wesentliche Potenzial von MSS.

Eine Object Query Language bildet mithin eine zentrale Anforderung, sollen die Potenziale der Objekt-Technologie für MSS nutzbar gemacht werden. Und genau in einer Object Query Language liegt auch ein ganz wesentliches Potenzial für objektorientierte MSS (OMSS), denn eine OQL bietet für viele Bereiche eine sehr effiziente Anfragemöglichkeit, wesentlich einfacher zu realisierende Anfragen bei komplexen Zusammenhängen und eine OQL bietet außerordentlich mächtige Sprachkonstrukte [LaVo96, S. 45 ff.]. *Heuer* meint beispielsweise:

> „Während die Definition, Semantik und Implementierung von Anfragsprachen in OODMs wesentlich komplizierter wird, […] ist die Formulierung von komplizierten Anfragen wesentlich natürlicher.“ [Heue97, S. 297]

Außerdem zeichnen sich deklarative Anfragesprachen durch eine erheblich höhere Bearbeitungseffizienz aus, wird die Optimierung durch das OBMS vorgenommen (siehe oben). Es hat sich bei vergleichbaren relationalen Produkten (z. B. Oracle) gezeigt, dass deren Stärke wesentlich in der Fähigkeit liegt, Anfragen effizient bearbeiten zu können, wodurch auch deren Preis maßgeblich bestimmt wird.

Gerade bei der für MSS notwendigen Massenverarbeitung von Daten ist eine OQL deshalb einer expliziten Programmierung unter Verwendung einer konventionellen OOPL vorzuziehen.

3.4.3.8 Architektur

Die bisherigen Ausführungen in diesem Kapitel machen deutlich, wie unterschiedlich die Anforderungen und folglich auch Konzepte von OBMS sind. Dies führt in der Folge auch zu sehr unterschiedlichen OBMS-Architekturen. Zwei Extremformen sind, entweder die strikte Trennung von OBMS und Applikation vorzunehmen, wie dies bei konventionellen Systemen mit RDBMS und prozeduralen Sprachen der Fall ist oder eine vollständige Integration anzustreben, was insbesondere aus OOP-Sicht wünschenswert ist.

Relationale DBMS realisieren für die **erste Variante** eine Drei-Ebenen-Schema-Architektur (Drei-Ebenen-Architekur, Drei-Schichten-Architektur, 3-Level-Architecture), die nach ANSI/SPARC standardisiert ist (siehe Abbildung 13) und deren Realisierung auch für OBMS gefordert wird [Catt91, S. 50 ff., Date00, S. 33 ff., Khos93, S. 111 f., SST97, S. 2 ff.].

Die unterste Schicht ist die interne oder physische Datenbankschicht. Deren Design und Verwaltung wird allein vom Datenbankadministrator („Database Pro-

fessionals or DBAs“ [Khos93, S. 112]) durchgeführt. Primäres Ziel sind eine hohe Effizienz und Sicherheit des DBMS. Bestandteile des physischen Datenbank-Designs sind die Ausnutzung bzw. Optimierung von unter anderem Speicherstrukturen, Clusterung, Indizes, Links, Replikation und andere Formen kontrollierter Redundanz, parallele Prozessoren oder Rechner, Cache und Netzwerkverkehr.

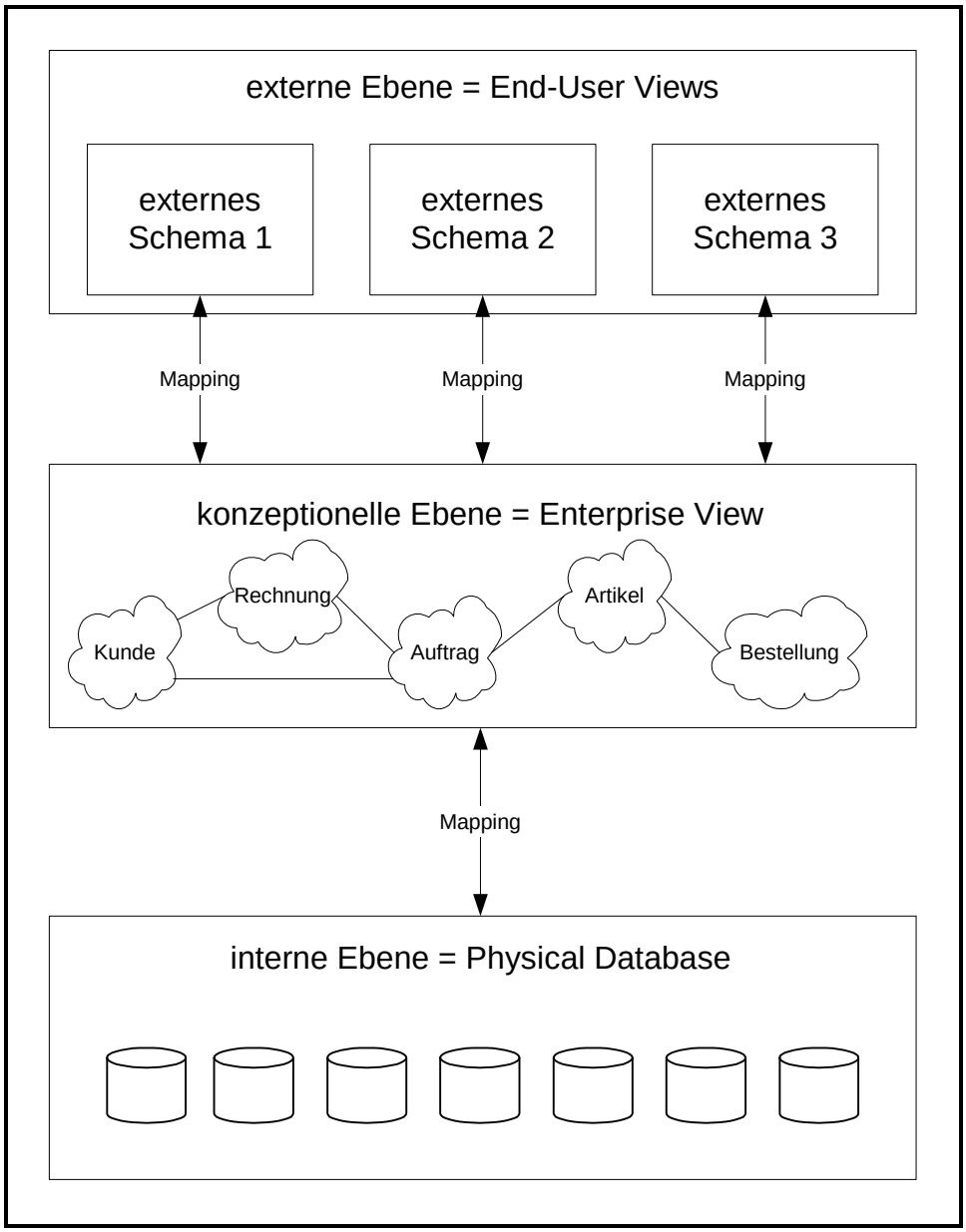


Abbildung 13: Drei-Ebenen-Schema-Architektur (ANSI/SPARC) für OBMS nach *Khoshafian* [Khos93, S. 111]

Die zweite Ebene bildet die konzeptionelle Datenbankschicht, die die „logische Gesamtsicht aller Daten und ihrer Beziehungen untereinander im konzeptionellen Schema (‘der Datenbank-Konzeption’) repräsentiert.“ [Voss94, S. 24] Diese

Schicht ist die typische Domäne des Wirtschaftsinformatikers. Sie bildet das unternehmensweite Objektmodell ab, das zentraler Gegenstand der OOA ist und aus den persistenten Objekten besteht. *Khoshafian* meint:

> „The conceptual design ist primarily concerned with the specification of the enterprise model." [Khos93, S. 111]

Das konzeptionelle Schema steht mithin auch im Zentrum dieser Arbeit, wie bereits mehrfach erläutert. Das externe Schema, die dritte Schicht der Drei-Ebenen-Schema-Architektur, ermöglicht es, spezielle Sichten für Benutzer oder Applikationen bereitzustellen (siehe Abschnitt 3.4.3.4).

Durch diese Dreiteilung in internes, konzeptionelles und externes Schema wird Datenunabhängigkeit erreicht. Dabei werden die internen Strukturen der jeweils zugrunde liegenden Schicht erstens nach obenhin verborgen („Hiding") und zweitens mit der darunterliegenden Schicht verknüpft („Mapping"). [Catt91, S. 127]

Durch die Trennung von konzeptionellem Schema und internem Schema wird die physische Datenunabhängigkeit realisiert; durch die Trennung von konzeptionellem Schema und externem Schema die logische Datenunabhängigkeit, jedenfalls zu einem hohen Grad. [ElNa94, S. 26 ff.]

Das konzeptionelle Schema wird teilweise auch als logisches Schema bezeichnet; dieser Begriff findet allerdings auch für das externe Schema Verwendung. [Catt91, S. 50 f.] Dies liegt daran, dass bei der Verwendung eines RDBMS vor dem konzeptionellen Datenbankschema, welches speziell für das zu verwendende RDBMS erstellt werden muss, ein konzeptionelles Modell gebildet wird, welches in der Regel semantisch reichhaltiger, produkt-unabhängig und — zumindest als Forderung [Voss94, S. 48 ff.] — datenbankmodell-unabhängig ist und in den meisten Fällen durch ein ERM realisiert wird. Die Modellierung des ERM wird dann als konzeptionelles Modell bezeichnet, das Modell für das RDBMS als logisches Modell.

Vossen unterscheidet sogar noch weitere Phasen, da er das auf das spezielle Relationen-Modell eines — bereits konkreten — RDBMS (z. B. für Oracle 7.3) übertragene Modell als logischen Entwurf und diesen dann normalisierten logischen Entwurf als Implementierungs-Entwurf bezeichnet, dem sich dann der physische Entwurf anschließt. [Voss94, S. 48 ff.]

Diese Differenzierung gibt es bei der Objekt-Technologie nicht. Das konzeptionelle Modell wird weitestgehend über alle Entwicklungsphasen durchgängig in seinen Strukturen erhalten und ohne semantischen Verlust in das OBMS implementiert. Es gibt also keinen Bruch in der Entwicklung wie bei der Verwendung von RDBMS. Es gibt mithin auch nur **ein** konzeptionelles Modell, das sich im konzeptionellen Schema des OBMS wiederfindet. *Bertino* und *Martino* meinen dementsprechend:

> „However, it should be noted that the logical design of an object-oriented database is simplified because, in contrast with relational DBMS, there is no need first to define a conceptual schema and then translate it into the DBMS model. In fact, the object-oriented model is of a sufficiently high level that it can be used to model the schema of the database directly from user requirements. This removes the distinction, which applies to conventional methodologies, between conceptual schemas and logical schemas.“ [BeMa94, S. 249 f.]

Die **zweite Variante** einer Extremform für eine OBMS-Architektur einer vollständigen Integration eliminiert alle drei Schema-Ebenen und verringert dadurch den Grad der Datenunabhängigkeit erheblich. *Saake*, *Schmitt* und *Türker* meinen:

> „Die C++-basierten ODBMS haben einen gegenläufigen Trend eingeleitet, indem sie keinen Unterschied der Darstellungen auf den drei Ebenen unterstützen: Die interne Darstellung entspricht direkt der Darstellung im Anwendungsprogramm.“ [SST97, S. 8]

In beiden Varianten liegt aber nicht zwingend ein Widerspruch. Beide Forderungen müssen für ein OBMS erfüllt sein. Es muss eine Integration stattfinden und es müssen drei Schema-Ebenen realisiert werden können.

Das **konzeptionelle Schema** bildet den Kern des OBMS und muss auch vollständig mit einer OOPL integriert werden, so dass Typsystem und Sprachparadigma (Computational Model) identisch sind.

Hierin liegt der Kern des Objekt-Paradigmas. Wird diese Forderung nicht erfüllt, bleibt der Impedance Mismatch erhalten und ein bedeutender Teil des Potenzials der Objektorientierung geht verloren. Eine Abbildung von Datenbankobjekten auf Anwendungsobjekte darf es nicht geben. Das konzeptionelle Modell muss direkt im OBMS abgebildet werden und ohne Bruch durch eine OOPL verwendet werden können.

Außerdem darf sich ein OBMS nicht auf die reine Speicherung von Daten beschränken, die dann bei Bedarf in den Client (der auch ein Applikationsserver sein kann) geladen werden. Ein OBMS kann erst als solches bezeichnet werden, wenn es vollständige Objekte speichert (siehe insbesondere das Zitat von *Manola* auf Seite 136) und auch in der Lage ist, deren Methoden auszuführen, also über eine eigene Laufzeitumgebung verfügt.[152] Derartige OBMS werden als aktive DBMS bezeichnet. [Loom95, S. 48]

[152] Dabei müssen die Methoden sowohl im Client als auch im OBMS ausführbar sein. Welche Lösung dann im jeweiligen Fall verwendet wird, ist im Wesentlichen eine Effizienzfrage.

Der Kern des Objekt-Paradigmas liegt in der Aufhebung der Trennung von Daten und Funktionen. Deshalb ist ein OBMS im Gegensatz zu einem RDBMS[153] kein Datenspeicher mehr, sondern ein vollständiges Laufzeitsystem. Werden Daten und Funktionen gemeinsam in Objekten strukturiert, um dann in einem weiteren Schritt wieder getrennt zu werden, ist nichts gewonnen.

Es gibt beim Objekt-Paradigma

- keinen Unterschied zwischen logischem und konzeptionellem Schema,
- keinen Unterschied zwischen Datenbankobjekten und Anwendungsobjekten, d. h. auch
  - ◊ eine weitgehende Transparenz (im Sinne von unsichtbar), ob auf Objekte im Hauptspeicher oder Sekundärspeicher zugegriffen wird und
  - ◊ eine Gleichbehandlung von transienten und persistenten Objekten, und
- keinen Unterschied zwischen Typsystem und Sprachparadigma (Computational Model) des OBMS und der OOPL.

APL und DPL müssen deshalb prinzipiell gleich sein.[154] Ein Beispiel für ein solches OBMS ist GemStone. [Gems95c]

Die Trennung von konzeptionellem und **physischem Schema** muss ebenso bei der OBMS-Architektur realisiert werden. Allerdings bieten die Möglichkeiten des Objektmodells durch die Betrachtung der Objekt- anstelle der Relationen-Ebene, also z. B. durch die Verwendung objektraum-übergreifender Referenzen (siehe oben), ein erheblich gesteigertes Optimierungs-Potenzial auf der einen Seite und eine nur schwer zu lösende Aufgabe auf der anderen Seite.[155]

Zwar bieten OOPL (auch als DPL) bereits weitreichende Abstraktionsmechanismen für physische Datenunabhängigkeit [Catt91, S. 129 ff.], doch ist die Trennung von physischem und konzeptionellem Schema gerade unter der Berücksichtigung des Software Engineering, speziell einem guten Design und der Erweiterbarkeit, essenziell. Smalltalk verfügt bereits über viele Abstraktionsmechanismen, um den Anwendungsprogrammierer vor der Notwendigkeit zu physi-

---

[153] Allerdings wird auch bei RDBMS inzwischen zunehmend von dieser klaren Trennung zwischen Programm und Daten abgewichen. Es erfolgt eine Annäherung an objektorientierte Konzepte, in dem hierfür beispielsweise Stored Procedures zur Verfügung stehen (z. B. Oracles PL/SQL). [KoLo95, S. 493 ff.]

[154] Dies bedeutet aber nicht, dass eine Client-OOPL (APL) überflüssig ist. Der Client verwaltet die transienten Objekte, d. h. die GUI- und Applikations-Schicht (siehe Abschnitt 3.4.1). Deshalb muss eine APL über einen erheblichen zusätzlichen Klassenumfang verfügen (z. B. zur Konstruktion von Oberflächen [PaDi95b], auch speziellen Datenbankmasken [Parc94, S. 12 ff.]), der von einer DPL nicht benötigt wird. Eine DPL hingegen benötigt zusätzliche Klassen für spezielle Datenbankfunktionen wie Transaktionen oder die Zugriffskontrolle.

[155] Werden beispielsweise Objekte zusammen geclustert, ist der gemeinsame Zugriff auf diese Objekte schneller, die anderen Anfragen verzögern sich aber dadurch. Werden die Objekte deshalb repliziert, um verschieden geclustert zu werden, birgt dies den Nachteil eines größeren Zeitfensters bei schreibenden Zugriffen.

schem Design zu bewahren, allerdings gibt es bisher keine ausreichenden Mechanismen, um das physische Design gesondert zu beeinflussen.

Das **externe Schema** in Form von Sichten spielt zur Realisierung von logischer Datenunabhängigkeit keine bedeutende Rolle, da es neben anderen Nachteilen (siehe Abschnitt 3.4.3.4) wieder eine Trennung von Datenbankobjekten und Anwendungsobjekten zum Ziel hätte.[156] Für Anfragen bieten Sichten jedoch Potenzial, insbesondere wenn sie effizient sind, was beispielsweise durch die direkten Referenzen des Objektmodells oder Materialisierung wie bei Object Marts (siehe Kapitel 4) erreicht werden kann.

In diesem Zusammenhang stellt sich die Frage, wie eine spezielle Anfragesprache (OQL) in eine OBMS-Architektur integriert werden kann. Da eine DPL im Kern mit der APL identisch ist, also einer vollständigen OOPL (General Purpose Programming Language) entspricht, können beide zwar das identische Typsystem haben, realisieren allerdings ein fundamental anderes Sprachparadigma.

Dies stellt in gewisser Weise ein Dilemma dar, da eine deklarative Anfragesprache eine essenzielle Anforderung ist, aber prinzipiell andere Anforderungen erfüllt bzw. Konzepte realisiert als eine OOPL (siehe Abschnitt 3.4.3.7).

Bisher wurde eine OQL allerdings ausschließlich für den Zweck von Ad-hoc-Zugriffen auf den Objektbestand betrachtet, weshalb die unterschiedlichen Konzepte, wie etwa die Durchbrechung des Kapselungs-Prinzips, zu diesem Zweck als unproblematisch angesehen werden können. Deshalb meint etwa *Cattell* im Einklang mit *Bancilhon*, am Beispiel der Kapselung:

> „The private implementation of an object is hidden from programs, but not from queries. This justification behind this approach is that the query language is for ad hoc end-user access, so logical data independence is less of an issue.“ [zitiert nach Catt91, S. 128]

Wird eine OQL mithin ausschließlich für Ad-hoc-Anfragen verwendet, stellt sich kein wesentliches Problem, wenn neben einer DPL zusätzlich noch eine OQL in das OBMS implementiert wird. Ad-hoc-Anfragesprache und DPL sind dann zwei getrennte Sprachen.[157] Nichts anderes fordert auch das *MANIFESTO* [ABD+89].

Allerdings bietet gerade die Kombination aus einer OOPL als General Purpose Programming Language und einer deklarativen OQL in manchen Fällen enormes Potenzial, z. B. bei sehr umfangreichen und komplexen Anfragen, bei denen

---

[156] Es zeigt sich auch, dass das Konzept von extenen Schemata bei RDBMS in der Praxis keine Verwendung findet. Zuviele Probleme, insbesondere das Update-Problem und geminderte Effizienz durch den Einsatz von Views, verhindern den Gebrauch von applikationsspezifischen Sichten.

[157] Wobei die Einbindung von DPL-Methoden in begrenztem Umfang (siehe Abschnitt 3.4.3.7) möglich sein muss.

sehr effiziente Zugriffe und zusätzlich mächtige Kontrollstrukturen benötigt werden.

Deshalb fordert die ODMG die Möglichkeit, dass nicht nur die Methoden der OOPL durch die OQL, sondern auch die OQL-Sprachkonstrukte in die OOPL eingebunden werden können müssen.

> „Based on the same type system, OQL can be invoked from within programming languages for which an ODMG binding is defined. Conversely, OQL can invoke operations in these languages." [CaBa98, S. 84]

Für die Kombination von APL und OQL sehen *Saake*, *Schmitt* und *Türker* vier unterschiedliche Ansätze [SST97, S. 269 ff.]:

1. Die Anfrage wird als Parameter einer Query-Funktion in Form eines Strings aus einer APL-Methode an das OBMS übergeben. [Gems96c, S. 10-8][158]
2. Die APL wird um Anfrage-Konstrukte erweitert, was einen eigenen Compiler erfordert. [Gepp97, S. 93 ff.]
3. Analog zum klassischen Embedded-SQL werden OQL-Konstrukte in den APL-Code eingebunden und durch einen Precompiler behandelt.
4. Die OQL des OBMS wird um spezielle Sprachkonstrukte erweitert, wie dies beispielsweise bei Oracles PL/SQL der Fall ist.

Saake, *Schmitt* und *Türker* sehen den Zweck einer OQL allerdings nicht auf Anfragen mit Ad-hoc-Charakter beschränkt, weshalb sie auch keine Unterscheidung einer OQL von einer DPL vornehmen, was wiederum in ihrem Festhalten an der prinzipiellen Trennung zwischen Daten und Anwendungen begründet ist.

Die Kopplung der OQL sollte stattdessen zu APL und DPL vorgenommen werden können. Da transiente und persistente Objekte in der Behandlung durch die Programmiersprache gleichbehandelt werden können müssen, sollte auch eine OQL diese Anforderung erfüllen. Dies ist allerdings nicht nur aus technischen Gründen äußerst schwer zu realisieren, sondern scheitert in der Regel daran, dass die DPL des OBMS und die APL-OOPL von unterschiedlichen Herstellern kommen.

Doch wie auch immer, die Kopplung von APL und DPL zur OQL erzeugt in allen diesen Fällen genau wieder den klassischen Impedance Mismatch, wobei bei allerdings zumindest die Typsysteme von OQL und APL bzw. DPL gleich sein kön-

---

[158] Bei GemStone steht aus der APL heraus nur diese erste Variante zur Verfügung, was allerdings sehr unbefriedigend ist. GemStone-Methoden können hingegen direkt die erweiterten Möglichkeiten der GemStone-OQL (die allerdings in die Collection-Klassen implementiert ist und sich auf objekt-erhaltende Anfragen beschränkt) verwenden, was dann insoweit der zweiten Variante entspricht, da die DPL erweitert wird und über einen eigenen Compiler verfügt. Diese GemStone-Methoden können wiederum auch aus der APL heraus aufgerufen werden. Für Ad-hoc-Anfragen besteht in der Client-Entwicklungsumgebung eine Möglichkeit die GemStone-OQL direkt zu verwenden. Diese Möglichkeit kann auch als Bestandteil einer Applikation realisiert werden.

nen. Trotzdem bleibt die Lösung unbefriedigend, da auch alle Besonderheiten, die für eine OQL im Gegensatz zu einer OOPL gelten, von der OOPL indirekt genutzt werden können. Beispielsweise wird die Kapselung dadurch mittelbar umgangen. Das ist nicht akzeptabel.

Andere Vorschläge bestehen darin, Elemente einer deklarativen Anfragesprache in die OOPL, d. h. in APL und DPL zu verwirklichen. Dieser Ansatz wird beispielsweise von Smalltalk verfolgt. Dort werden deklarative Sprachkonstrukte wie „select“ und „detect“ in die Collection-Klassen implementiert. Dies ist zwar bisher nicht hinreichend, doch zeigt etwa *Howard*, dass guter objektorientierter Programmierstil in Smalltalk darin besteht, Methoden mit einem deklarativen Charakter zu formulieren. [Howa95, S. 295 ff.[159]]

Methoden werden dann derart formuliert, dass deren Bezeichungen in hohem Maß aussagekräftig sind und deren Implementierungen für den diese Methoden verwendenden Entwickler deshalb transparent (im Sinne von unsichtbar) bleiben können. Auf der konzeptionellen Ebene ist damit ein wichtiger Schritt in Richtung Deklarativität bereits realisiert, da bei neuen Anwendungen zunehmend das „wie“ zugunsten des „was“ in den Hintergrund tritt.

Cattell meint beispielsweise, dass es vielleicht durch zukünftige Forschung möglich sein wird, eine parallele Optimierung imperativer und deklarativer Sprachkonstrukte in Methoden durchzuführen, etwa durch die Kombination von traditioneller Programmiersprachen-Optimierung und Anfragesprachen-Optimierung und dadurch auch einen höheren Grad an physischer Datenunabhängigkeit erreichen zu können. [Catt91, S. 130]

Auch die ODMG formuliert den Integrationsgedanken der Funktionalität einer Anfragesprache in eine OOPL:

> „It is important to define the scope of our efforts, since ODBMSs provide an architecture that is significantly different than other DBMSs — they are revolutionary rather than an evolutionary development. Rather than providing only a high-level language such as SQL for data manipulation, an ODBMS or ODM transparently integrates database capability with the application programming language.“ [CaBa98, S. 2]

Zusammenfassend lässt sich zur Integration einer OQL sagen, dass

- eine OQL für reine Ad-hoc-Anfragen vorhanden sein muss,
- eine OQL Methoden der OOPL in eingeschränkter Form verwenden können muss (siehe Abschnitt 3.4.3.7),
- eine Erweiterung der DPL und, soweit möglich, auch der APL um Konstrukte einer deklarativen Anfragsprache, die dem Objekt-Paradigma einer OOPL

[159] Insbesondere im mitgelieferten Lernprogramm.

nicht entgegenstehen und prinzipiell integriert werden können, angestrebt werden sollte und

- die Einbindung einer echten OQL in eine Applikation, die nicht Ad-hoc-Charakter hat, vermieden werden sollte.

3.4.3.9 Begriff

Üblicherweise erfolgt die Definiton eines Begriffs vor der Diskussion über einen Gegenstand. Im Zusammenhang mit dem Paradigma von OBMS erscheint es allerdings sinnvoll, dies erst nach der Diskussion der einzelnen Konzepte zu tun, da die Konzepte — wie gezeigt — doch außergewöhnlich unterschiedlich sind. Der Begriff sollte aber gerade diese Konzepte mit umfassen. Dementsprechend sind auch die in der Literatur zu findenden Bezeichnungen unterschiedlich. Üblich sind etwa object-oriented Database Management System (OODBMS), Object Database Mangagement System (ODBMS), Objectbase (OB), objektorientierte Datenbanken (OODB), Objekt-Datenbanken (ODB) oder Objektbanksystem (OBS).

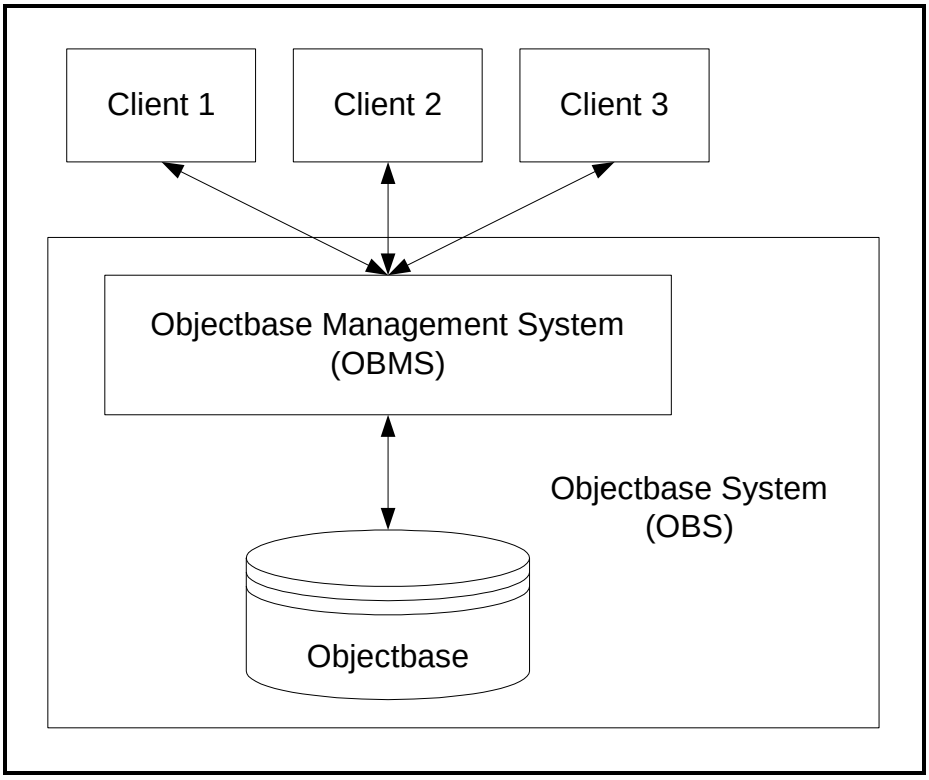


Abbildung 14: OBMS-Definition

In dieser Arbeit findet der Begriff Objectbase Management System (OBMS) Verwendung. Im Gegensatz zu einem DBMS managt ein OBMS keine eigentliche

Datenbank — es werden Objekte gespeichert und verwaltet. Es wird mithin davon ausgegangen, dass ein OBMS Objekte mit Attributen und Beziehungen, also den Daten im engeren Sinne und Methoden speichert, wobei die Methoden nicht nur abgelegt, sondern auch ausgeführt werden können. Der aktive Charakter eines OBMS wird damit hervorgehoben. Damit wird außerdem verdeutlicht, dass die bisherige Trennung von Daten und Funktionen auch auf der Ebene von Programm und DBMS aufgehoben ist.

Dabei stellt das Objectbase Management System (Objektbank-Verwaltungssystem) die eigentliche Anwendung zur Verwaltung der Objekte dar (siehe Abbildung 14). Die Objectbase (OB, Objektbank) ist der Datenspeicher mit den persistenten Objekten. Die OB (bei GemStone Repository genannt [Gems96d, S. 7-1 ff.]) besteht in der Regel aus mehreren physischen Objektbanken (bei GemStone Extents genannt [Gems96d, S. 7-8 ff.]). Das OBMS und die Objektbanken zusammen bilden das Objectbase System (OBS, Objektbanksystem).

3.4.4 Applikations-Schicht

Eine objektorientierte Applikation besteht aus drei Schichten (siehe Abschnitt 3.4.1). Die oberste Schicht ist die Benutzeroberfläche (siehe Abschnitt 3.4.2). Die unterste Schicht ist das Objectbase Management System (siehe Abschnitt 3.4.3). Um diese beiden Schichten zu verbinden bedarf es der dritten Schicht, der Applikations-Schicht (siehe Abbildung 15).

Dabei besteht die Applikations-Schicht prinzipiell mit der Zielsetzung, die GUI-Objekte mit den OBMS-Objekten, also den Domänen-Objekten zu verbinden, d. h. die Kommunikation zwischen diesen Objekten zu ermöglichen. Das primäre Ziel dabei ist es, eine hohe Unabhängigkeit zwischen GUI und OBMS zu erreichen. [Howa95, S. 375 ff.]

Wird ein relationales DBMS anstelle eines OBMS verwendet, benötigt die Applikation eine zusätzliche Schicht zur Transformation der Domänen-Objekte in Relationen bzw. umgekehrt der Tupel in Instanzen. [Parc94, S. 5 ff.] Die Domänen-Objekte müssen in diesem Fall vollständig von der OOPL verwaltet werden, d. h. sind APL-Objekte. Die Daten der Objekte liegen im RDBMS.

Obwohl dieser Ansatz in der Praxis häufig Verwendung findet, ist der Aufwand für die Verbindung von RDBMS und OOPL selbst dann noch enorm [Loom95, S. 75 ff.], wenn es eine Verbindungsschicht, wie etwa die ObjectLens bei VisualWorks gibt [Parc94, S. 5 ff.]. Vor allem kann aber das Potenzial des Objekt-Paradigmas nicht genutzt werden, denn das liegt im Wesentlichen in den Fähigkeiten des OBMS und der bruchlosen Verbindung zu OOPL und GUI bzw. OOA und OOD.

Dass dieser Ansatz trotzdem in vielen Fällen angewandt wird, liegt daran, dass einerseits die vorhandene Datenbanktechnologie weiter genutzt werden soll und andererseits die Stärken von objektorientierten GUI (siehe Abschnitt 3.4.2) genutzt werden sollen.

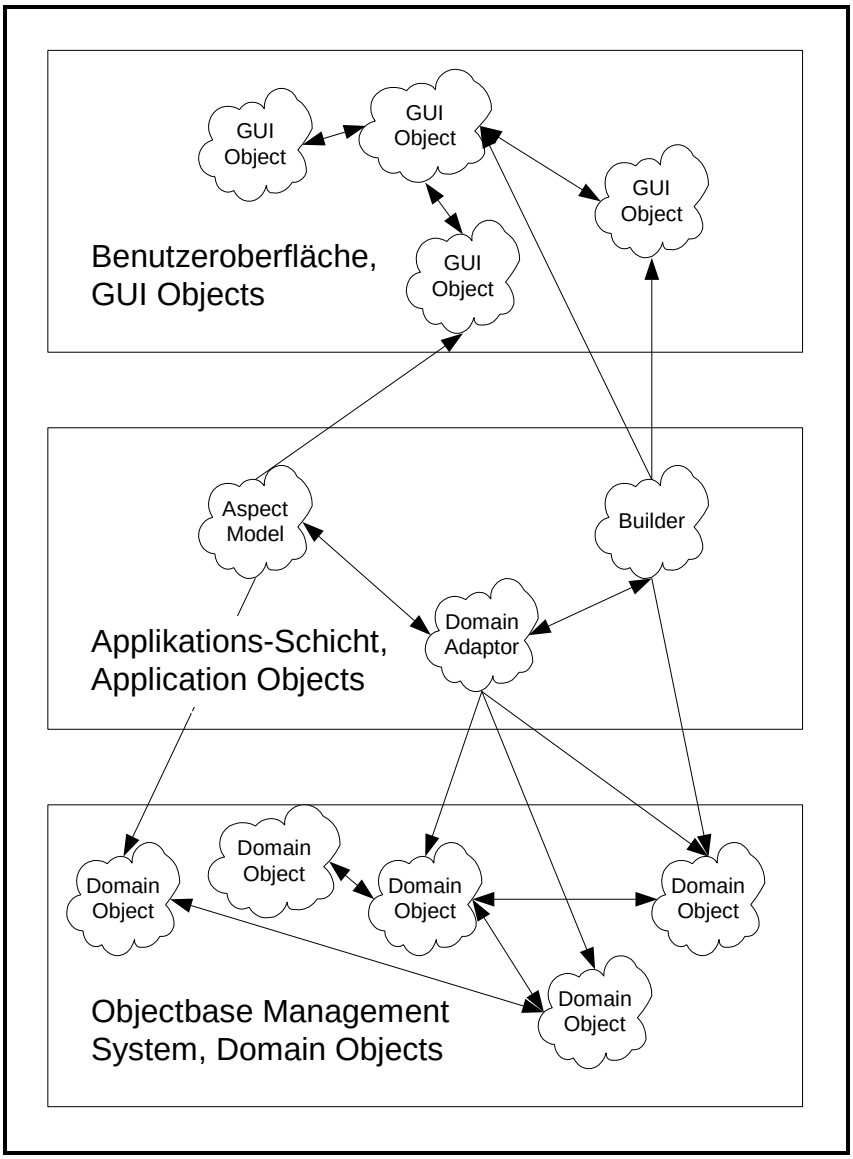


Abbildung 15: Drei-Schichten-Architektur einer objektorientierten Applikation

Die Applikations-Schicht selbst verfügt prinzipiell über keinen Applikations-Code, da dieser in die Domänen-Objekte gehört. Allerdings ist durch die Trennung von GUI-Objekten und Domänen-Objekten, wie beim Model-View-Controller-Konzept die Steuerung der Benutzeroberfläche in die Applikations-Schicht zu implementieren. Deshalb enthält die Applikations-Schicht zwangsläufig auch jenen Teil der Applikationslogik, der für die Benutzeroberfläche notwendig ist (siehe die Beispiele auf Seite 110 und Seite 133). [Howa95, S. 121 ff.]

3.5 Weitere Objekt-Paradigmen

Weitere Gebiete, die von der Objektorientierung betroffen sind, sind die Künstliche Intelligenz und die Unternehmens-Organisation. Beide Gebiete verwirklichen zum einen Objekt-Paradigmen und sind zum anderen eng mit den bisher diskutierten Objekt-Paradigmen verbunden, d. h. es bestehen vielfältige Abhängigkeiten.

3.5.1 Künstliche Intelligenz

Ein wesentlicher Bestandteil der Forschung der Künstlichen Intelligenz (KI) ist die Wissensrepräsentation und Wissensverarbeitung. [GKP92, S. 72] Es gibt auch einen Trend, für diesen Zweck geeignete Datenbanken, intelligente Datenbanken, zu entwickeln. Das Ziel besteht in einer natürlichen und benutzerfreundlichen Verwaltung der Informationen, so dass die Informationen leicht speicherbar, zugreifbar und verwendbar sind [Hugh92, S. 221] oder wie *Gray*, *Kulkarni* und *Paton* es ausdrücken [GKP92, S. 74]:

1. „Expressive power: Can experts communicate their knowledge effectively to the system?
2. Understandability: Can experts understand what the system knows?
3. Accessebility: Can the system use what it has been given?“

Dabei lassen sich bei den intelligenten Datenbanken zahlreiche Aspekte objektorientierter Datenmodellierung erkennen. Es zeigt sich jedoch, dass die Anwendungen der KI in realen Anwendungen zumindest bisher wesentlich unausgereifter sind als die der Datenbanktechnik. [Hugh92, S. 221] *Hughes* unterscheidet vier Kategorien von Wissensrepräsentationsverfahren [Hugh92, S. 223 f.]:

1. Bei **logischen Wissensrepräsentationsverfahren** wird die Wissensbasis durch Ausdrücke der formalen Logik repräsentiert. Dabei wenden Inferenzregeln dieses Wissen auf die Lösung spezifischer Probleme an. In der Regel findet das Prädikatenkalkül 1. Ordnung Verwendung, z. B. bei Prolog, das deshalb für diese Art von Wissensbasen gut geeignet ist.
2. Bei **Netzwerk-Wissensrepräsentationsverfahren** wird das Wissen durch einen Graphen dargestellt. Knoten stellen die Objekte und Konzepte dar, Kanten die Beziehungen. Die bekannteste Form sind die semantischen Netze.
3. Bei **prozeduralen Wissensrepräsentationsverfahren** wird das Wissen durch Prozeduren oder Anwendungsfolgen für die Lösung spezifischer Probleme dargestellt. Wissensbasen stellen dabei eine Ansammlung von aktiven Benutzerarbeitseinheiten und Prozessen dar. Lisp spielt in diesem Zusammenhang eine wichtige Rolle.

4. **Strukturierte Wissensrepräsentationsverfahren** können als Erweiterung der Netzwerkverfahren angesehen werden, wobei die Knoten zu komplexen Datenstrukturen wurden. Diese komplexen Datenstrukturen bestehen aus benannten Slots und dazugehörigen Werten, wie einfachen Werten und Wertmengen, Zeigern auf andere komplexe Strukturen oder Prozeduren. Frame-basierte Verfahren sind typische Vertreter.

Durch die Trennung der strukturellen Objektinformation und dem Objektverhalten, wie die Objekte agieren und wie die Objekte untereinander agieren, scheiden Logikmodelle als praktisch anwendbare Methode für die Darstellung komplexer Probleme aus. [Hugh92, S. 224] Als bedeutendste Vertreter der Methoden zur Wissensrepräsentation, werden die semantischen Netze und Frame-basierten Modelle angesehen. [Heue97, S. 149 f., HuKi90, S. 1 f.]

Semantische Netze unterstützen die Konzepte [HuKi90, S. 13]:

- IS-INSTANCE-OF
- IS-PART-OF (der Aggregation sehr ähnlich)
- IS-A

Allerdings gibt es bei den semantischen Netzwerken keine klare Trennung zwischen den Daten und dem Schema, es werden also keine Mechanismen zur Abstraktion der Datenstrukturen von den Daten selbst zur Verfügung gestellt. [HuKi90, S. 13] Dieses Konzept entspricht dem Prototypenansatz (siehe Abschnitt 3.3.4.5), da es bei den semantischen Netzwerken kein Klassenkonzept gibt, das die Eigenschaften einer Menge von Objekten beschreibt. [GKP92, S. 72 ff.] Als Folge werden Eigenschaften nicht von Klassen geerbt, sondern über Delegation von anderen Objekten (Instanzen) „geerbt“.

Gray, *Kulkarni* und *Paton* sehen bei dieser Form von System allerdings die Schwierigkeiten, Datenunabhängigkeit, Integrität, Persistenz und Abstraktion angemessen verwirklichen bzw. gewährleisten zu können, wodurch etwa Datenbankzugriffe und die Definition von Integritätsregeln nicht effizient zu realisieren sind, kein Data Dictionary erzeugt und eine deklarative OQL nicht implementiert werden kann und klassische Abstraktionsmechanismen von semantischen Datenmodellen nicht existieren. Die Idee der Klassifikation weicht mithin von der in Datenbankanwendungen ab. [GKP92, S. 75]

Die Kontrolle über eine effiziente und redundanzfreie Speicherung der Instanzen ist mithin nicht mehr gegeben, insbesondere die Handbabung umfangreicher Datenbestände kann auf diese Weise nicht sinnvoll realisiert werden. [Heue97, S. 149 f.]

Ein weiterer wesentlicher Nachteil ist, dass semantische Netze nur über einen Strukturteil und keine Abbildung der Dynamik des Systems, also der Zuordnung von Verhalten zu den Anwendungsobjekten verfügen. [LaLo95, S. 356 f.]

Die Frame-basierten Modelle beheben einige dieser Mängel. Frames entsprechen den Typen (Klassen) beim Objektmodell und haben Eigenschaften für die Menge der Instanzen (Meta-Abttribute) und für die Instanzen selbst, die Slots. [Heue97, S. 149 f.]

Slots sind Attribute mit Werten oder Zeigern oder Slots sind Prozeduren (Methoden). Prozeduren dienen dem Zweck, abgeleitete Attribute [Heue97, S. 149] zu erzeugen, sogenannte aktive Attribute [GKP92, S. 77] (Derived Attributes, siehe Abschnitt 3.4.3.7). Außerdem verfügen Slots über Facets (Subframes [GKP92, S. 77]), die zusätzliche Eigenschaften der Slots abbilden, wie Typspezifikationen, Integritätsbedingungen oder Demons, die dem Triggerkonzept entsprechen. [Hugh92, S. 228]

Echtes Verhalten lässt sich bei Frames allerdings nicht abbilden. Wie *Hughes* es ausdrückt:

> „Die Frames sind demnach passive Datenstrukturen, die nur durch Prozeduren außerhalb der Struktur manipuliert werden können. Diese Prozeduren müssen von einer beliebigen Benutzerarbeitseinheit — nicht von einem Frame — selektiert und angestoßen werden.“ [Hugh92, S. 230]

Weiterhin unterstützen Frames keine Aggregation. [Heue97, S. 149, HuKi90, S. 14] Außerdem sind bei Vererbungsbeziehungen Ausnahmen möglich. Etwa kann — übertragen auf das Objekt-Paradigma — eine Instanz zur Klasse Elefant gehören, die als Eigenschaft vier Beine hat, obwohl dieses Elefanten-Exemplar nur drei Beine hat. [HuKi90, S. 13] IS-A-Beziehungen und Vererbung können unabhängig realisiert werden. [Heue97, S. 140]

Gray, *Kulkarni* und *Paton* kritisieren, dass, obwohl Frame-basierte Systeme das Klassenkonzept unterstützen, keine klare Trennung von Prototypen und Klassen vorgenommen wird, wodurch diese Konzepte beliebig vermischt werden können. Sie sehen diese Systeme deshalb im Bereich der Prototypenerstellung von wissensbasierten Systemen, auf keinen Fall aber als Grundlage von Objektmodellen, die Struktur in ein OBMS bringen wollen. [GKP92, S. 76 ff.]

Ein wesentlicher Vorteil von Wissensbasen besteht allerdings darin, dass diese in der Lage sind, generelle Informationen zu speichern, was mit konventioneller Datenbanktechnik wie RDBMS oder OBMS nicht in dieser Form möglich ist. Wissensbasen erlauben die explizite Speicherung von regelbasierten Informationen und verfügen über eine dafür vorgesehene Verarbeitungseinrichtung, die Inferenzmaschine, die diese Informationen handhabbar und interpretierbar macht und diese Regeln dynamisch verketten kann (backward chaining, forward chaining). Beispielsweise müsste bei einem konventionellen DBMS die Regel Frauen

können Kinder gebären explizit für jede Instanz[160] bzw. jedes Tupel extra abgebildet werden. [Hugh92, S. 222]

Trotzdem besteht eine gemeinsame Zielsetzung insbesondere zwischen den Paradigmata von OBMS und Wissensrepräsentationsverfahren, speziell den strukturierten Ansätzen. Letztere sind für erstere allerdings nicht in dieser Form zu verwenden. Deshalb auf diese Konzepte zu verzichten, wäre jedoch ein Fehler. KI-Methoden zur Wissensrepräsentation und semantische Modelle haben dieselbe ursprüngliche Zielsetzung. [King89, S. 24] Und semantische Modelle wiederum bilden den Kern des Strukturteils des Objektmodells. [Heue97, S. 127 ff.]

Gerade für den MSS-Bereich sind entsprechende KI-Methoden und deren Kombination mit konventionellen MSS-Methoden bzw. Integration mit konventionellen MSS-Werkzeugen seit langem Bestandteil der Forschung. [Rieg93] Die Integration in eine Object-Warehouse-Konzeption ist deshalb folgerichtig, scheint aufgrund der bereits in hohem Maße kompatiblen Konzepte unschwer erreichbar zu sein und bietet vor allem ein enormes Potenzial für das MSS-Gebiet, welches es nutzbar zu machen gilt.

3.5.2 Organisation

In der Organisationstheorie ist der Begriff der Objektorientierung seit langem ein Thema, wobei objektorientierte Organisationsformen, wie bei der OOP von den funktionalen Organisationsformen abgegrenzt werden. *Klotz* definiert die objektorientierten Ansätze wie folgt:

> „Die objektorientierten Ansätze sind durch eine Zusammenfassung unterschiedlicher Verrichtungen, die bei der Bearbeitung eines Objektes anfallen, gekennzeichnet (sog. Objektzentralisation). 'Hierbei wird eine Folge ungleichartiger Verrichtungen an gleichartigen Objekten auf eine Aktionseinheit übertragen.' [Grochla 72[161]] Als Objekte kommen z. B. einzelne Produkte bzw. Produktbereiche, Regionen oder auch Kunden bzw. Kundengruppen in Betracht." [Klot93, S. 78]

Klotz kann zwar keine grundsätzliche Entscheidung zugunsten des objektorientierten Ansatzes treffen, sieht aber folgende Vorteile [Klot93, S. 79]:

- „**Koordinationsvorteile**: Zielausrichtung und Steuerungsmaßnahmen auf jeweils einzelne Organisationseinheiten beschränkt, die gegenüber dem Versagen anderer unanfällig sind. Möglichkeit der abteilungsweisen Erfolgsermittlung.

---

[160] Allerdings könnte diese Eigenschaft bei einem OBMS dadurch abgebildet werden, indem anstelle eines Attributs nur eine entsprechende Methode implementiert, da Methoden auf Klassenebene gültig sind. Für die Verkettung von Regeln ist diese Form der Abbildung jedoch unzureichend.

[161] Grochla, E.: Unternehmensorganisation - Neuere Ansätze und Konzeptionen, Rowohlt, 1972

- **Kommunikationsvorteil**: Aufgrund ihrer Unabhängigkeit voneinander geringe Kommunikationsintensität zwischen den objektzentralisierten Einheiten.
- **Transportvorteil**: Tritt allerdings nur ein, wenn die objektzentralisierten Stellen auch räumlich zusammengefasst sind.
- **Kostenvorteil**: Vermeidung von Mehrfacheinarbeitung durch verschiedene Stellen bzw. Spezialisten.
- **Soziologisch-psychologischer Vorteil**: Enge Verbindung des Einzelnen mit dem Objekt und den übrigen daran Beteiligten. Vermeidung von Arbeitsmonotonie."

Auch *Kilberth*, *Gryczan* und *Züllighoven* sehen in einer objektorientierten Organisation bedeutende Vorteile, da durch diese ein schnelles Reagieren auf veränderte Marktbedürfnisse unterstützt wird. „Management of Change" für die Aufbau- und Ablauforganisation, die Gestaltung der Führungs- und Kompetenzstrukturen, die Produktpalette, die Ausgestaltung der Informationsverarbeitung und die Nutzung moderner Hard- und Softwaretechnologien sehen sie als kritischen Erfolgsfaktor für die Zukunft. [KGZ94, S. 139]

Um dieses Ziel besser erreichen zu können fordern sie, schnell und angemessen reagieren zu können, die Umsetzung und Durchführung von Entscheidungen vor Ort und dazu eine Organisation in Form von schlagkräftigen, weitgehend autonomen Einheiten. Die Abhängigkeiten müssen reduziert und die Mitarbeiter ein hohes Maß an Eigeninitiative und Verantwortlichkeit entwickeln. Sie meinen:

> „Wir meinen jedenfalls, dass tradierte, hierarchische Organisationen mit ihrem *taylori-stischen* Gestaltungsprinzip sowie einer zentralistischen Entscheidungsstruktur den skizzierten Anforderungen nicht mehr gerecht werden." [KGZ94, S. 139]

Es zeigt sich also, dass es tatsächlich eine Art Objekt-Paradigma in der Organisationslehre gibt, nämlich eine Form, bei der die Arbeitsfunktionen nicht *taylori-stisch* gegliedert sind, sondern ganzheitlich am Objekt ausgerichtet — nach welchen Kriterien diese Objektstrukturen im einzelnen Fall auch immer gebildet werden. *Kilberth*, *Gryczan* und *Züllighoven* gehen sogar soweit, Analogien zum OOP-Paradigma herzustellen und „befinden" sich auf „einem direkten Weg zu einer objektorientierten Interpretation einer Unternehmensorganisation". [KGZ94, S. 140]

Wie weit auch immer eine solche Analogie sinnvoll gebildet werden kann, so zeigt dies doch auf jeden Fall, dass es auch einen Zusammenhang zwischen den Paradigmen gibt. Tatsächlich begünstigt etwa die Informations-Technologie im Allgemeinen und die Client/Server-Technologie im Besonderen die Schaffung objektorientierter Organisationsgestaltungen. Beispielsweise ist in den letzten

Jahren ein verstärkter Trend bei Verwaltungen festzustellen, die strenge *taylori-stische* Organisation mithilfe von PC umzugestalten. [Post00]

Außerdem gibt es einen klaren Zusammenhang zwischen Software-Design und Organisationsgestaltung (siehe Abschnitt 3.2.1). Software und Organisation sind Bestandteile eines Unternehmenssystems, die in diesem System eng verzahnt sind. [Kral94, S. 5 ff.]

Doch gerade der MSS-Bereich, insbesondere die Entwicklung des Data-Ware-house-Konzepts (siehe Abschnitt 2.2) zeigt das Potenzial solcher organisatori-schen Umgestaltungen (siehe Abschnitt 2.1.6). [Rieg99, S. 283 ff.] Das Objekt-Paradigma begünstigt dabei insbesondere auch die Bildung objektorientierter Organisationsformen, und zwar zum Ersten durch die erhöhte Flexibilität des Software-Designs und zum Zweiten durch die ähnlichen Strukturen der Para-digmen.

KAPITEL 4

OBJECT WAREHOUSE

In Kapitel 2 sind die konkreten Anforderungen an eine Data-Warehouse-Konzeption und -Architektur entwickelt worden. In Kapitel 3 wurden Prinzipien und eine durchgängige Methodik zur Entwicklung von objektorientierten Informationssystemen erarbeitet — speziell eine OBMS-Architektur, die den zentralen Kern eines objektorientierten Informationssystems bildet.

In diesem Kapitel erfolgt die Vereinigung dieser beiden Konzeptionen zu einer integrativen Konzeption für ein objektorientiertes Analogon des Data Warehouse, das Object Warehouse. Dabei besteht das Bestreben darin, durch die Konzeption des Object Warehouse sowohl die beschriebenen Mängel an zur Zeit vorherrschenden Data-Warehouse-Konzeptionen (mangelhafte Modellierung, fehlendes Schichtenkonzept, Einsatz von Star-Schema-Varianten etc.) wie auch an der Konzeption des „Data“ Warehouse an sich, d. h. des auf relationaler Technologie basierenden Data Warehouse, zumindest teilweise zu beseitigen.

Während also bis zu dieser Stelle der Arbeit das Ziel darin bestand, eine Betrachtung von Data-Warehouse-Konzeption auf der einen Seite und Objekt-Paradigma auf der anderen, weitgehend unabhängig voneinander durchzuführen, um nicht bereits im Vorfeld der Untersuchung das Untersuchungsgebiet bzw. den Untersuchungsgegenstand derart zu beschränken, dass die Betrachtung entscheidender Aspekte unterbleibt oder sogar die Konzeption insgesamt bereits im Vorhinein fälschlicherweise als untauglich verworfen wird, besteht im Folgenden die Zielsetzung, durch die geeignete Anwendung des Objekt-Paradigmas auf die Data-Warehouse-Technologie konkret neue Potenziale aufzuzeigen.[162]

[162] Aus diesen Gründen werden die Kenntnisse über die Ergebnisse des Vorangegangenen vorausgesetzt. Deshalb wird im Folgenden auf die detaillierte Begründung der bereits erarbeiteten Konzepte nicht erneut eingegangen.

Mithin werden mögliche sinnvolle Einsatzgebiete der Objekt-Technologie für die Data-Warehouse-Konzeption zusammen mit etwaigen daraus resultierenden neuen Problemen aufgezeigt. Dazu wird zunächst die grundlegende Idee der Object-Warehouse-Konzeption erläutert. Dem schließt sich die Betrachtung der Object-Warehouse-Architektur an, die durch eine Vertiefung bezüglich der Quell-Schichten, des Object-Warehouse-Kerns, respektive des Reconciled Object Layers und der Anwendungs-orientierten Schichten ergänzt wird.

4.1 Grundidee

Der Grundgedanke des Object Warehouse besteht, analog zum Data Warehouse, in der Bereitstellung aller MSS-relevanten Daten für Management Support Systems jedweder Kategorie und der zur Erfüllung dieser Aufgabenstellung benötigten Funktionen (ETL, Domänen-Funktionen etc.). Dies bedeutet, dass zum einen die MSS-relevanten Datenstrukturen als Abstraktion der Diskurswelt und Abbild der verfügbaren MSS-relevanten Daten modelliert werden müssen und zum anderen die vorhandenen Daten (intern/extern, operativ/nicht-operativ, strukturiert/semistrukturiert, qualitativ/quantitativ) im Rahmen des entstandenen Modells implementiert und für die im Einsatz befindlichen MSS nutzbar gemacht werden müssen.

Die Kernidee des Object Warehouse ist deshalb die Schaffung einer integrativen, einheitlichen Objektbasis, welche die MSS-relevanten Daten inklusive der MSS-relevanten Metadaten beinhaltet. Das Kernstück des Object Warehouse ist somit das MSS-Unternehmens-Objektmodell (MSS-UOM, unternehmensweites MSS-Objektmodell), das genau diese Daten abbilden muss und dessen Umsetzung bzw. Implementierung als Reconciled Object Layer (ROL) in einem Objectbase Management System.

Die grundlegende These der Object-Warehouse-Konzeption ist mithin, dass der bedeutendste kritische Erfolgsfaktor durch die Modellierung des MSS-UOM und dessen Umsetzung im ROL bestimmt wird. Von den modellierten Objektstrukturen — d. h. deren semantischen Ausdruckskraft (Objektmodell-Qualität), die sich über den Nutzen für MSS-Applikationen und MSS-Nutzer definiert — und deren Dateninhalten (Datenqualität) hängt die Qualtiät des Object Warehouse und damit auch das MSS-Potenzial ab.

Genau zu diesem Zweck dient die Objekt-Technoloige. Die Stärken des objektorientierten Ansatzes liegen in der angemessenen Modellierung, die die Grundlage des MSS-UOM und des ROL bilden. Die Potenziale für das gesamte Systems Engineering und für den durchgängigen Ansatz — von OOA bis OOP bzw. von GUI bis OBMS — und die vielen anderen aufgezeigten Chancen, die das Objekt-Paradigma bietet, können auch für das Object Warehouse genutzt werden. Das

Hauptpotenzial für das Object Warehouse liegt in der Stärke des Modellierungsansatzes des entwickelten Objektmodells.

Dabei besteht grundsätzlich die These, dass zumindest konzeptionell ein integriertes Unternehmens-Informationssystems verwirklicht werden sollte, bestehend aus sowohl den operativen wie auch den analytischen Informationssystemen, d. h. den Administrations- und Dispositionssystemen sowie den Management Support Systems.

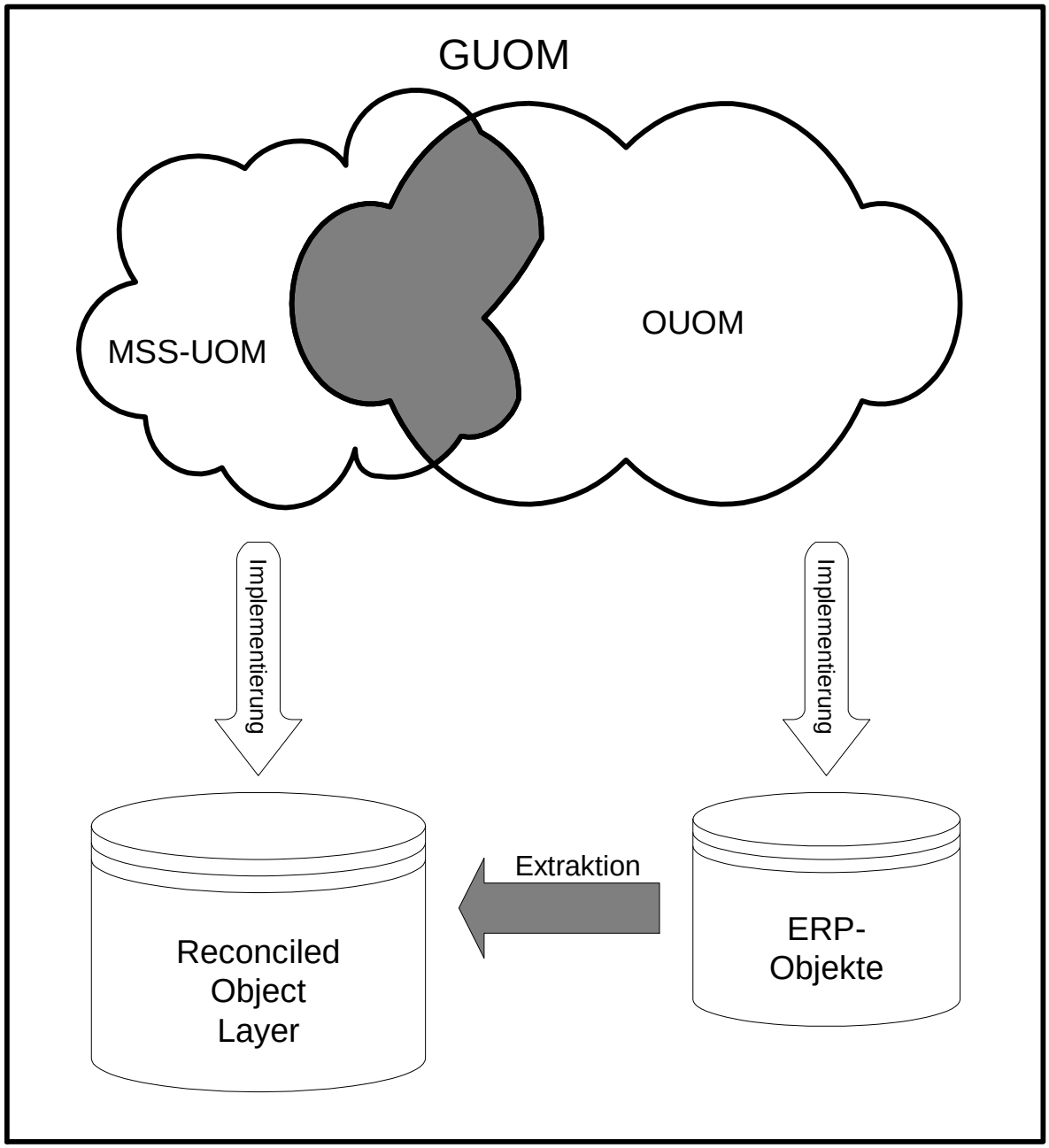


Abbildung 16: Generisches Unternehmens-Objektmodell (GUOM)

Aufgrund der Potenziale des objektorientierten Ansatzes sowohl für die operativen wie auch die analytischen Informationssysteme, besteht die Idealvorstellung in der Umsetzung eines integrativen objektorientierten Unternehmens-Informationssystems, in welcher auch ein integratives konzeptionelles Objektmodell, das generische Unternehmens-Objektmodell (GUOM, generisches UOM), aus den operativen Domänen-Objekten und den MSS-Domänen-Objekten modelliert und in einer gemeinsamen Unternehmens-Objektbasis eines Objectbase Management Systems implementiert wird.

Die Unternehmenspraxis verdeutlicht jedoch, wie weit entfernt von dieser Idealvorstellung die Informationstechnologie-Landschaften der Unternehmen tatsächlich sind und auch auf absehbare Zeit bleiben werden.[163]

Da jedoch operative und analytische Informationssysteme für heutige Unternehmen erfolgskritisch sind, weshalb die Realisierung eines Object Warehouse das Problem der fehlenden Integration der MSS-relevanten Objekte zu lösen hilft und in der Folge die Modellierung eines MSS-Unternehmens-Objektmodells wie auch eines operativen Unternehmens-Objektmodells und deren jeweilige Umsetzung in einem Reconciled Object Layer bzw. einer integrierten operativen Applikation, wie zum Beispiel einem Enterprise Resource Planning System, unverzichtbar sind (siehe Abbildung 16).

Es gibt also, je nach Sichtweise, ein generisches UOM, das sich aus MSS-UOM und OUOM zusammensetzt oder ein MSS-UOM und OUOM, die bezüglich ihrer gemeinsamen Daten[164] verknüpft (gemappt) sind. Gemeinsame Daten sind beispielsweise ein Kundenobjekt mit Attributwerten wie Name, Geburtsdatum und Adresse. Ändert sich aber zum Beispiel aufgrund einer Eheschließung der Name des Kunden von „Krebel“ nach „Muntrep“, wird dieser Attributwert des Kunden in der operativen Objektbasis mit „Muntrep“ überschrieben, während im Reconciled Object Layer diese Namensänderung abgebildet wird, d. h. sowohl der alte wie auch der neue Name des Kunden gespeichert sind.

4.2 Architektur

Die Object-Warehouse-Architektur muss als ganzheitliche MSS-Architektur begriffen werden. Während im letzten Abschnitt der Betrachtungsgegenstand die

[163] Es ist aufgrund der Komplexität solcher Unternehmens-Informationssysteme, sowohl der operativen wie auch der analytischen, äußerst fraglich, ob diese Idealvorstellung eines integrativen objektorientierten Unternehmens-Informationssystems überhaupt unter den gegebenen ökonomischen Restriktionen realisierbar sein kann.

[164] Daten wird hier im Sinne von Attributen und Methoden verstanden, was jeweils aus dem Kontext zu entnehmen ist.

konzeptionelle Ebene[165] war, beschreibt dieser Abschnitt über die Architektur, die konkrete Umsetzung des Object Warehouse, von den Quelldaten und -objekten bis hin zu den Informationsobjekten der MSS-Front-Ends (siehe Abbildung 17). Dabei ist der letztendliche Zweck von allen Schichten und Prozessen dieser Architektur die Unterstützung der eigentlichen analytischen Arbeit, d. h. der letzten Schicht der MSS-Front-Ends bzw. der Anwender dieser MSS-Front-Ends.

Die unternehmens-relevanten Daten sind als Geschäftsdaten definiert worden, die sich aus den MSS-relevanten Daten, den MSS-Daten und den operativen Daten zusammensetzen. Die MSS-Objekte, die auch als MSS-Domänen-Objekte oder MSS-Geschäftsobjekte bezeichnet werden, sind dann die objektorientiert modellierten — und also auch um die dazugehörigen Funktionen angereicherten — MSS-Daten. Entsprechendes gilt für die operativen Daten, die dann als operative Geschäftsobjekte bezeichnet werden. Die Geschäftsobjekte werden im generischen UOM abgebildet, die MSS-Objekte im MSS-UOM und die operativen Geschäftsobjekte im OUOM.

In der Object-Warehouse-Architektur wird das konzeptionelle Modell des MSS-UOM im Reconciled Object Layer umgesetzt, der den Kern des Object Warehouse bildet. Dort werden mithin die MSS-Objekte implementiert.

Die Objekte des ROL speisen sich aus den Quell-Schichten, die sich aus unterschiedlichen Daten- oder Objektquellen zusammensetzen. Die Quell-Schichten bestehen mithin aus den MSS-relevanten operativen Geschäftsdaten und Geschäftsobjekten bzw. den MSS-relevanten Teilen dieser Daten bzw. Objekte sowie den sonstigen MSS-relevanten Daten bzw. Objekten, d. h. den MSS-Daten bzw. MSS-Objekten. Um die relevanten Quelldaten bzw. Quellobjekte in den ROL zu überführen, werden diese extrahiert, geprüft, bereinigt, vereinheitlicht, transformiert und in den ROL geladen.

Der Reconciled Object Layer besteht im Wesentlichen aus den MSS-Objekten selbst sowie einem Reconciled Meta-Object Layer (RMOL), der auch die MSS-relevanten Metadaten integriert, die aus den Metadaten des ROL selbst, Metadaten über die Quelldaten, Metadaten über den Derived Object Layer und Metadaten der MSS-Front-Ends besteht (siehe Abbildung 17).

Aus dem ROL inklusive des RMOL wird der Derived Object Layer abgeleitet. Der Derived Object Layer besteht aus Object Marts oder eventuell auch konventionellen Data Marts, die je nach Bedarf einzelner Abteilungen, einzelner Funktionsbereiche oder Applikationen, d. h. applikations-spezifisch erzeugt werden.

---

[165] Der Begriff der konzeptionellen Ebene bezieht sich auf die Unternehmensmodelle GUOM, MSS-UOM und OUOM, nicht dagegen auf die konzeptionelle Ebene in der Architektur eines OBMS. Das MSS-UOM wird als konzeptionelles Schema (konzeptionelle Ebene) des OBMS umgesetzt.

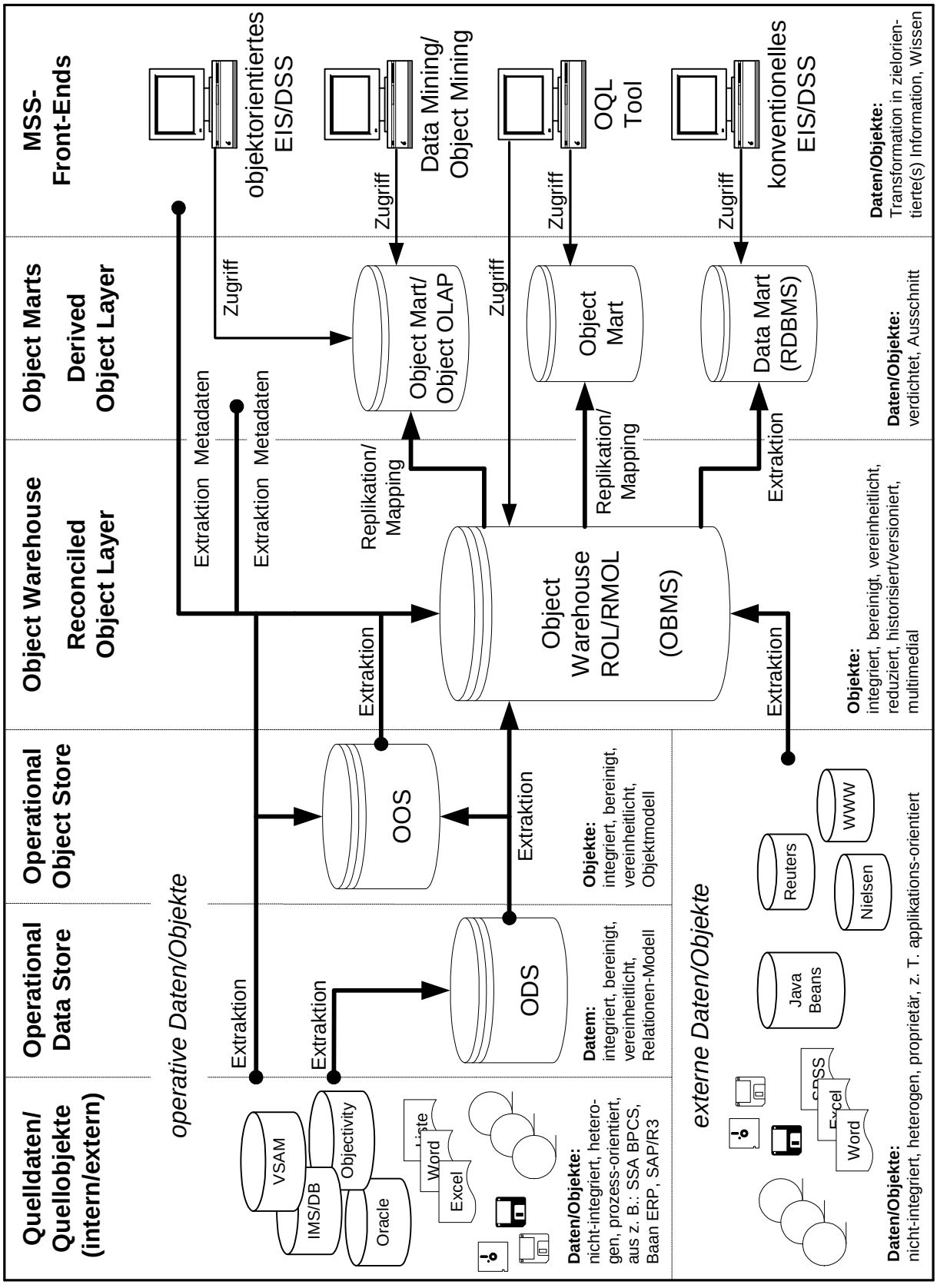


Abbildung 17: Object-Warehouse-Architektur

Die MSS-Front-Ends greifen entweder unmittelbar auf den Reconciled Object Layer zu oder mittelbar über den Derived Object Layer, um die notwendigen Daten bzw. Objekte zu selektieren oder eventuell auch zu importieren.

Mit dem Begriff Informationsobjekte werden alle MSS-relevanten Objekte bezeichnet, d. h. die Objekte des ROL, des DOL und der MSS-Front-Ends. Dabei verfügen die Informationsobjekte über ihre eigenen Metadaten, entweder direkt in Form von Methoden und Attributen oder über Referenzen auf die entsprechenden Meta-Objekte, was ersterem entspricht.

Die Object-Warehouse-Architektur muss nicht zwingend über alle dargestellten Quell-Schichten verfügen. Im Idealfall gibt es nur eine einzige operative Daten- bzw. Objektquelle, etwa in Form einer Datenbank eines RDBMS oder einer Objektbank eines OBMS eines ERP und nur wenige, bereits rekonziliierte, sonstige MSS-relevante Daten und Objekte, die ebenfalls in relationaler oder objektorientierter Form vorliegen.

4.3 Quell-Schichten

Die Quell-Schichten bzw. Quell-Daten sind beim Object Warehouse prinzipiell dieselben wie beim Data Warehouse. Den Kern bilden die MSS-relevanten operativen Geschäftsdaten bzw. Geschäftsobjekte, die intern, strukturiert und quantitativ sind. Diese werden um die sonstigen operativen und nicht-operativen, strukturierten und semistrukturierten, qualitativen und quantitativen MSS-Daten bzw. MSS-Objekte ergänzt, soweit deren Integration ökonomisch sinnvoll ist. Importiert werden allerdings nicht nur die reinen Geschäftsdaten, sondern auch die Metadaten der Geschäftsdaten bzw. Geschäftsobjekte, soweit sie in den Quell-Systemen (überhaupt) vorhanden und verfügbar zu machen sind.[166]

Ein Unterschied zum konventionellen Data Warehouse besteht darin, dass auch die bereits in Objekt-Form vorliegenden Daten, also Objekte, auf einfache Weise integriert werden können und deren — gegenüber konventionellen Daten — höhere Semantik nicht verloren geht, sondern mit in das Object Warehouse integriert werden kann. Die Transformation komplexer Objekte, d. h. von Objekten, die die Modellierungs-Eigenschaften des in dieser Arbeit entwickelten Objektmodells ausnutzen, in flache Tabellenstrukturen eines relationalen Data Warehouse, ist mit erheblichem Aufwand verbunden, verlangt den Verzicht auf bedeu-

[166] Wie die Analysen der Projekte gezeigt haben, sind in den meisten Fällen nicht nur die Quell-Daten von minderwertiger Datenqualität (Datenhygiene), auch wird fast immer auf die Definition von Metadaten verzichtet. Beispielsweise liegen die (überhaupt) vorhandenen Datenmodelle in nicht-elektronischer Form oder als Grafik (z. B. JPEG) eines Grafikprogramms (z. B. Corel-Draw) vor [Rusk00, S. 166]; Constraints werden durch ERP-Systeme in den die Daten verwaltenden RDBMS nur selten definiert, bei Baan beispielsweise überhaupt nicht.

tende Bestandteile der Semantik und muss insgesamt als nicht-sinnvoller Ansatz bewertet werden.

Im Zuge eines beobachtbaren verstärkten Wandels hin zu objektorientierten Quell-Daten, also Quell-Objekten, ist dies von hoher Bedeutung, insbesondere in Anbetracht einer zunehmenden Standardisierung (OMG, ODMG) für die Kooperation und den Austausch von Objekten unterschiedlicher Systeme (Hardware, Betriebssysteme, OBMS, OOPL etc.).

Konsequenterweise wird deshalb für eine mögliche Integration der operativen Alt-Daten aus den Legacy-Systems auch kein Operational Data Store (ODS), sondern ein Operational Object Store (OOS) implementiert. Ein OOS bietet die Chance, die Alt-Systeme sukzessive als objektorientierte Applikation neu zu implementieren, d. h. als operative Geschäftsobjekte, um die Legacy-Systems auf diese Weise schrittweise durch eine integrierte objektorientierte Applikation (z. B. ein objektorientiertes ERP) zu ersetzen.

Zusätzlich zu den Potenzialen eines ODS und den Vorteilen, die ein solches objektorientiertes Administrations- und Dispositionssystem bietet, könnten auf diese Weise die Stärken des objektorientierten Ansatzes bereits bei den Quell-Systemen genutzt werden, so dass wichtige Daten und Metadaten, die mithilfe des Objektmodells abgebildet bzw. mithilfe eines OBMS implementiert würden, auch genutzt werden könnten und nicht nachträglich „von Hand“ in das Object Warehouse eingfügt werden müssten, soweit dies überhaupt möglich ist.

Da allerdings in vielen Fällen der Schritt von einer Alt-Applikation (z. B. einem COBOL-Programm mit einer ISAM-Datenbasis) hin zu einer objektorientierten Anwendung, die ein hohes Maß der in dieser Arbeit aufgezeigten objektorientierten Prinzipien verwirklicht, nur unter hohem Aufwand zu verwirklichen ist, kann die Implementierung eines Operational Data Stores als vorübergehende Zwischenstufe eine sinnvolle Lösung sein (siehe Abbildung 17).

Zusätzlich zu den bisher behandelten Quell-Daten werden Metadaten bzw. Meta-Objekte des Derived Object Layers und der MSS-Front-Ends in das Object Warehouse integriert, weshalb auch diese beiden, dem Object Warehouse nachgelagerten Schichten, als Quell-Schichten dienen.

4.4 Object-Warehouse-Kern

Der Object-Warehouse-Kern ist der Reconciled Object Layer, der das MSS-Unternehmens-Objektmodell implementiert. Der ROL lässt sich in weitere „virtuelle“ Schichten untergliedern, die — tatsächlich — weder physisch noch logisch abgegrenzt werden können, was darin begründet ist, dass die grundlegende Idee des ROL gerade die Integration aller MSS-relevanter Daten ist. Für ein besseres

Verständnis des ROL werden jedoch im Folgenden zum Ersten die zu integrierenden Daten in drei Kategorien eingeteilt, die sequenziell integriert werden und zum Zweiten der ROL im engeren Sinne und der Reconciled Meta-Object Layer (RMOL) unterschieden.

Im ersten Schritt werden die internen, strukturierten, quantitativen, operativen Daten entweder direkt aus den operativen Systemen oder dem ODS bzw. OOS importiert. Alle nicht MSS-relevanten Daten respektive Objekte bzw. Attribute werden dabei entfernt bzw. nicht mit importiert. Dabei werden die Daten extrahiert, geprüft, bereinigt, vereinheitlicht, transformiert und in das Objektmodell des Reconciled Object Layers, d. h. das Objectbase Management System geladen.

Dabei zeigt sich, dass die Datenmodelle der operativen Systeme ganz erheblich von den Objektmodellen der ROL abweichen, zum einen bezüglich der Auswahl der MSS-relevanten Daten und zum anderen bezüglich der Modellierung der Datenmodelle, da reale ERP-Systeme in der Regel nicht einmal die Grundsätze eines relationalen Datenschemas umsetzen und die Tabellen stattdessen prozess- oder sogar masken-orintiert organisiert sind und in der Folge unzählige Redundanzen aufweisen.[167]

Als ein Beispiel für ersteres seien die Attribute lediglich einer operativen Tabelle (siehe Tabelle 1) den Attributen des entsprechenden Objekts des ROL (siehe Abbildung 18) gegenübergestellt.[168] Wie zu sehen ist, wurden in diesem Fall lediglich 10 der 127 Attribute als MSS-relevant eingestuft und folglich in das Object Warehouse aufgenommen. Dieses Beispiel lässt jedoch auch einen Eindruck für zweiteres gewinnen, nämlich die mangelhafte Modellierung und die erheblichen Redundanzen. Die in den anderen Tabellen benötigten Attibutwerte müssen bei jeder Änderung dorthin kopiert werden.[169]

Die erzeugten Objekte werden sämtlich, unter Ausnutzung der objektorientierten Versionierungskonzepte, historisiert bzw. versioniert, was sowohl für die Datenwerte selbst, wie auch deren Strukturen, d. h. das Objekt-Schema erfolgen muss. Dies ist ein Schlüsselkonzept des Object Warehouse und stellt insofern eine notwendige Bedingung dar.

---

[167] Dies konnte bei der Untersuchung des ERP Baan deutlich bestätigt werden.

[168] Dieses Beispiel stammt aus einem realen Projekt mit einem Landmaschinenhersteller, bei dem das ERP Baan als grundlegendes operatives System eingesetzt wurde. Auf dieses Projekt wird in Teil II (Kapitel 5 und 6) detaillierter eingegangen werden.

[169] Die redundanten Attribute der Tabelle sind die Nummern (siehe Tabelle 1): 10-18, 32-34, 62, 88-92, 97, 100-104, 110-113, 120, 124.

Nr.	Name	Bezeichnung
1	item	Artikelcode
2	dsca	Bezeichnung
3	dscb	Material
4	dscc	Größe
5	dscd	Standard
6	wght	Gewicht
7	seak	Suchbegriff I
8	seab	Suchbegriff II
9	kitm	Artikelart
10	citg	Artikelgruppe
11	ctyp	Produktart
12	csel	Selektionscode
13	csig	Signalcode
14	cvat	USt-Code
15	txta	Artikeltext
16	cuni	Vorratseinheit
17	stgu	Einlagerungseinheit
18	cwar	Lager
19	kltc	Chargenverwaltung
20	obpr	Auslagerungspriorität
21	kpsl	Haltbarkeitsperiode
22	npsl	Haltbarkeit in Perioden
23	pics	Greifvorrat
24	abcc	ABC-Code
25	lcod	Letztes Zähldatum
26	uscu	Kumulierter Verbrauch
27	usab	Erwarteter Jahresverbrauch
28	slmp	Unverkäuflichkeitsprozentsatz
29	serv	Servicegrad
30	sfst	Sicherheitsbestand
31	maxs	Maximalbestand
32	csps	Saisonale Schwankung des Sicherheit
33	cspd	Saisonale Bedarfsschwankung
34	cfmd	Prognoseverfahren
35	scst	Lagerkosten p.a.
36	stoc	Vorhandener Bestand
37	blck	Gesperrter Bestand
38	ordr	In Produktion/Bestellt
39	allo	Reservierter Bestand
40	quot	Reserviert für Angebote
41	ltdt	Letzte Vorratsänderung
42	opol	Bestellverfahren
43	osys	Bestellsystem
44	omth	Bestellmethode
45	oqmf	Bestellmengenfaktor
46	mioq	Mindestbestellmenge
47	maoq	Maximalbestellmenge
48	umer	Aktualisierungsverfahren K-Artikelb
49	fioq	Feste Bestellmenge
50	ecoq	Wirtschaftliche Bestellmenge
51	reop	Meldebestand
52	oint	Bestellintervall
53	ddfq	Direktlieferung ab Menge
54	oltm	Wiederbeschaffungszeit
55	sftm	Sicherheitszeit
56	fodt	Frühestes Bestelldatum
57	ocst	Bestellkosten
58	auso	Bestell-/Auftragsdaten aktualisiere
59	cpha	Fiktiver Artikel
60	oqdr	Mengengesteuerte Arbeitsplanung
61	repi	RPT-Artikel
62	scdl	Programmcode (tirpt001.scdl)
63	orip	RPT-Artikelauftrag pro
64	pcrp	Prozentsatz für Planung

Nr.	Name	Bezeichnung
65	stpm	Startdatum für MPS-Auftragsvorschlä
66	mrpc	Minimaler Restprozentsatz
68	plmm	MPS-Auftragsplanverfahren
69	mrpo	MPS/MRP-Bestellsystem RPT-Artikel
70	eitm	Revisionsgesteuert
71	bfcp	Stüliabbuchung Komponente
72	bfep	Stüliabbuchung Endprodukt
73	bfhr	Stüliabbuchung Stunden
74	ndrp	Nett Change DRP-Status
75	nnts	Anzahl Pendelka (tigen.nnts)
76	qpnt	Menge pro Pende (tigen.qpnt)
77	unom	Stücklisteneinheit
78	runi	Arbeitsplaneinh (tigen.rout.un)
79	scpf	Ausschußfaktor
80	crmp	Kritisch in MPS
81	tmfc	Planungshorizont
82	roun	Runden in MPS
83	ncst	Net Change MRP
84	llcd	Low Level Code
85	llci	Indikator für Low Level Code
86	mrpi	MRP-Indikator
87	stmr	Vorrat (beim MRP-Lauf)
88	cuqp	Mengeneinheit
89	cupp	EK-Preiseinheit
90	cpgp	EK-Preisgruppe
91	csgp	EK-Statistikgruppe
92	ccur	Währung
93	prip	EK-Preis
94	ltpp	Datum letzte EK-Preisänderung
95	avpr	Durchschnittlicher EK-Preis
96	ltpr	Letzter EK-Preis
97	suno	Lieferant
98	qual	Kontrolle
99	purc	Kumulierte Wareneingänge
100	txtp	EK-Text
101	cuqs	VK-Einheit
102	cups	VK-Preiseinheit
103	cpgs	VK-Preisgruppe
104	csgs	VK-Statistikgruppe
105	pris	VK-Preis
106	ltsp	Datum letzte VK-Preisänderung
107	prir	Empfohlener VK-Preis
108	umsp	Obermarge
109	lmsp	Untermarge
110	ccde	Warencode (tcmcs028.ccde)
111	ctyo	Herkunftsland
112	txts	VK-Text
113	cpcp	Herstellkostenkomponente
114	copr	Herstellkosten
115	matc	Materialkosten
116	oprc	Bearbeitungskosten
117	cuid	Herstellkostenindikator
118	ltcp	Datum letzte Herstellkostenänderung
119	stva	Bewertungsverfahren
120	cplb	Einkäufer/Planer
121	cppp	Preispolitik
122	ccit	Überwachungscode
123	ccfu	Überwachungsfunktion
124	ccco	Kostenkomponent (tppdm100.ccco)
125	prre	Zwischenergebni (tppdm.prre)
126	copt	H.K.-Art TP
127	cprp	Herstellkosten manuell
128	itmt	Artikelart (Projekt)

Tabelle 1: Die Attribute der Artikel-Tabelle „tiitm001“ des Landmaschinenherstellers im ERP Baan

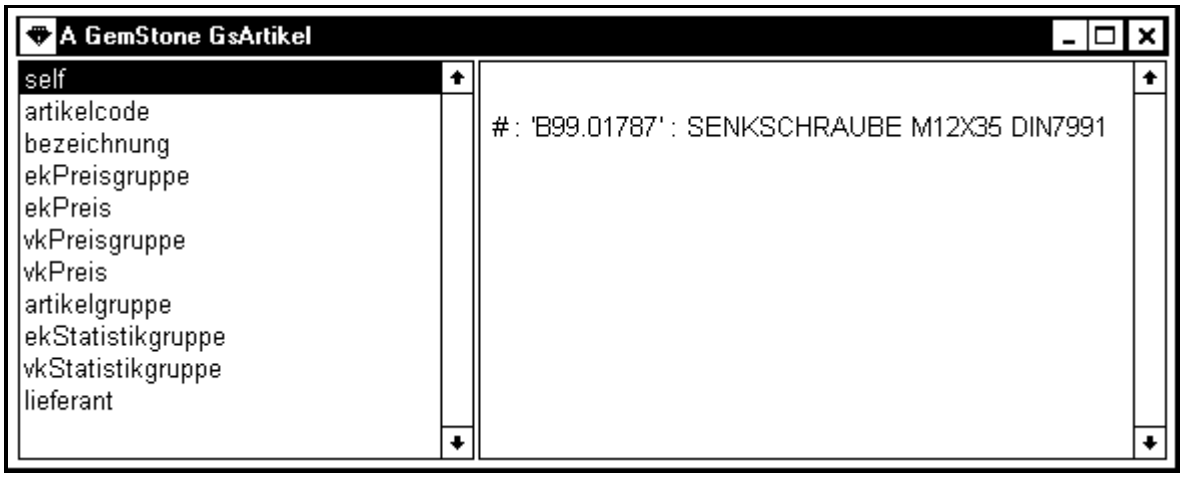


Abbildung 18: Die MSS-relevanten Artikel-Attribute der Ursprungstabelle „tiitm001“ am Beispiel eines Artikel-Objekts im Object Warehouse des Landmaschinenherstellers

Im zweiten Schritt werden die sonstigen strukturierten internen und externen Daten bzw. Objekte integriert, wofür sie dieselben Prozesse durchlaufen müssen. Die Daten müssen mithin extrahiert, geprüft, bereinigt, vereinheitlicht, transformiert, geladen und in das im ersten Schritt erstellte Objektmodell integriert werden. Auch diese Objekte müssen historisiert bzw. versioniert werden.

Im dritten Schritt wird wiederum derselbe Prozess für die semistrukturierten Daten bzw. Objekte durchgeführt. Die Prozess-Reihenfolge ist grundsätzlich veränderbar, scheint jedoch auf die vorgeschlagene Weise am einfachsten durchführbar zu sein, da die Daten mit höherem Strukturiertheitsgrad die Grundstruktur für das Objektmodell vorgeben.

Parallel zum Import der jeweiligen Daten werden auch die entsprechenden, d. h. mit diesen Daten in Zusammenhang stehenden Metadaten (Built-Time-, Control- und Usage-Metadaten) in den ROL überführt. An sich ist die Integration der Built-Time-Metadaten jeweils vor der Integration der „gewöhnlichen“ Daten notwendig, da die Built-Time-Metadaten die Strukturen des Objektmodells selbst, d. h. das Objekt-Schema festlegen bzw. verändern. Da allerdings das Laden des ROL keinen einmaligen Vorgang darstellt, sondern nach einem anfänglichen Grundladevorgang des Object Warehouse (Initial Load) in periodischen Zyklen erfolgt, beispielsweise täglich, müssen gegebenenfalls auch die Built-Time-Metadaten in eben solchen Zyklen aktualisiert werden, so dass sich das Objekt-Schema eventuell ebenfalls täglich ändert, wenn auch nur marginal.

Aufgrund der beim Objekt-Paradigma, insbesondere bei dem in dieser Arbeit entwickelten Objektmodell, fließenden Grenzen, erstens zwischen den unterschiedlichen Metadaten-Arten und vor allem zweitens zwischen den Metadaten und den eigentlichen Daten, ist eine exakte Abgrenzung dieser Daten-Arten kaum zu erreichen.

Wesentliche Teile, etwa der Metadaten eines Relationen-Modells, die beispielsweise als Constraints definiert werden müssen oder überhaupt nicht bzw. nicht im RDBMS vorhanden sind, sind impliziter Bestandteil des Objektmodells des OBMS. Eine logische oder gar physische Trennung von Daten und Metadaten, d. h. in MSS-Objekte und MSS-Meta-Objekte, zum Beispiel mit einem separaten MDBMS, sei dies auch objektorientiert realisiert, steht dem Konzept einer rekonziliierten Datenbasis, respektive Objektbasis fundamental entgegen. Eine solche Lösung würde einen erneuten Rekonziliierungs-Prozess erfordern und zu einer redundaten Objektbasis führen, deren Konsistenz nicht ökonomisch zu sichern sein könnte.

Auch das Objektmodell selbst wird deshalb in den ROL implementiert, ist mithin ebenfalls in Form von Objekten im ROL integriert und insofern selbst wiederum Bestandteil des Objektmodells und deshalb auch vollständig zugreif- bzw. anfragbar.

Die konkrete vorliegende Object-Warehouse-Konzeption, auf der Basis des OBMS GemStone, realisiert genau diesen Ansatz. In GemStone sind die Klassen als impliziter Systembestandteil vorhanden, sind selbst auch Objekte und stellen insofern Meta-Objekte dar, die sich ansonsten genauso wie gewöhnliche Instanzen verhalten bzw. behandeln lassen, denn auch die Klassen wiederum verfügen über eigene Meta-Objekte, die Meta-Klassen.

Konkret bedeutet dies, dass jedes Objekt im Sinne der diskutierten objektorientierten „Intelligenzverteilung“ und des objektorientierten „Zuständigkeits-Konzepts“ selbst seine Metadaten bzw. Meta-Objekte verwaltet, d. h. kennt und pflegt. Es gibt also keine von den Objekten „losgelösten“ Metadaten bzw. Meta-Objekte.

Gerade in Anbetracht des grundlegenden Objektgedankens, dass nämlich Objekte „aktive autonome“ Einheiten darstellen, also nicht wie Relationen-Tupel von irgendwelchen Funktionen manipuliert werden, entfaltet dieses Konzept die Potenziale. Objekte und Meta-Objekte, soweit diese Unterscheidung zu treffen ist, sind untrennbar im ROL verbunden und verfügen über die Fähigkeit zu einer beidseitigen aktiven Kommunikation.

Beispielsweise hat eine Instanz das „Wissen“ darüber, welche Methoden, welche Attribute, welche Beziehungen, welche Constraints oder welche Klasse sie hat und damit auch mittelbar, welche Klasse diese Klasse wiederum hat, et cetera, so dass deutlich wird, welche Potenziale das in dieser Arbeit entwickelte Objektmodell für den ROL-Ansatz hat.

Auf diese Weise muss auch der Entwurf des konzeptionellen Objektmodells erfolgen, also des MSS-UOM. Beispielsweise ist der VisualDesigner von ArsNova [Arsn95] vollständig in Smalltalk implementiert und generiert aus dem Objektmodell-Entwurf die Klassen mit Beziehungen, Methoden, Attributen et cetera

(CASE) und umgekehrt aus den Domänen-Klassen, den Entwurf des Objektmodells.

Das Konzept des ROL besteht darin, alle MSS-Objekte und die dazugehörigen Meta-Objekte, in nur einer einzigen rekonziliierten redundanzfreien Spezifikation zu integrieren. Mithin sind auch das konzeptionelle Modell des MSS-UOM und wenn möglich auch des OUOM sowie das GUOM in derselben Objektbasis implementiert.

Die Verbindung der konzeptionellen Modelle in unterschiedlichen CASE-Tools ist ohnehin nur mit erheblichem Aufwand zu bewerkstelligen, umso mehr die Wartung. Die Trennung von CASE-Tool und Applikations-Code (in beiden Richtungen) wird die Wahrung der Konsistenz zwischen konzeptionellem Modell und Implementierung zumindest mittelfristig unmöglich machen. Deshalb ist die Integration des CASE-Tools in die Entwicklungs-Umgebung die beste Alternative.

Dabei muss es möglich sein, dass der Code oder das Modell verändert wird, ohne dass die Spezifikation des CASE-Tools und der Applikations-Code inkonsistent werden, was entweder durch Mechanismen zur automatischen Propagierung oder noch besser durch eine identische Basis für Code und Modell realisiert wird.[170]

Das in dieser Arbeit entworfene Objekt-Paradigma bietet Konzepte für diesen Ansatz. Es werden alle strukturierten und semistrukturierten Daten bzw. Objekte und alle strukturierten und semistrukturierten Metadaten bzw. Meta-Objekte in einem Objektmodell integriert und in einem OBMS persistent gehalten. Ein Ansatz, der nicht in gleicher Weise integrativ wirkt, wird dem Anliegen des Object-Warehouse-Gedankens nicht gerecht.

Dies bedeutet, dass auch die Metadaten aus den Quell-Schichten extrahiert, geprüft, bereinigt, vereinheitlicht, transformiert sowie geladen und ansschließend versioniert werden müssen. Die Metadaten werden also grundsätzlich genauso behandelt wie auch die „gewöhnlichen" Daten. Wird dieser Ansatz nicht in dieser Form vollzogen, muss zumindest ein expliziter RMOL implementiert werden, für den dann jedoch dieselben Prinzipien und Vorgehensweisen wie für die MSS-Objekte bzw. den ROL gelten.

Neben den Built-Time- und Control-Metadaten kommt insbesondere für die unmittelbare Verwendung durch die End-Anwender den Usage-Metadaten eine besondere Bedeutung zu.

Für den End-Anwender selbst ist es im Gegensatz zum Object-Warehouse-Administrator in der Regel von geringem Interesse, woher genau beispielsweise die Daten für ein Objekt (z. B. einen Kunden) stammen (z. B. aus welchem ISAM-

[170] Der VisualDesigner verfolgt genau diesen Ansatz, erzeugt jedoch eine zusätzliche, also redundante Spezifikation des Objektmodells in Form von Klassenmethoden.

File welcher Attributwert) und welche exakten Bereinigungsprozesse und Transformationen die einzelnen Objektbestandteile durchlaufen haben. Auch für die Verwaltung der in der gewöhnlichen Unternehmenspraxis erforderlichen hochkomplexen Rechtestrukturen[171] (Virtual Security Layer) des ROL, wird sich der End-Anwender nicht im Detail interessieren.

Gerade bei RDBMS-Datenbanken zeigt sich, dass die Nutzer mit den vorhandenen „nackten" Datenbeständen nichts anzufangen wissen. Deshalb sind besonders die Usage-Metadaten von direktem Nutzen für den End-Anwender und damit ein essenzieller Bestandteil des ROL.

Zu diesen Daten, die die Bedeutung, d. h. den Geschäftskontext der Informationsobjekte verdeutlichen helfen, müssen deshalb zusätzlich auch die Metadaten verfügbar gemacht werden, die keine (unmittelbaren) Object-Warehouse-Metadaten sind.

Zum Beispiel muss der End-Anwender auch Daten darüber bekommen können, welche „Informationen" bereits in der MSS-Umgebung des Unternehmens vorliegen, also konkret welche Analysen und Berichte es bereits gibt, wo diese Auswertungen zu finden sind mit welchen Werkzeugen sie zu benutzen sind.

Und auch weitere Daten darüber, welche zusätzlichen Informationen, etwa Kommentare, wer der Ersteller ist (mit einem Verweis auf dessen Telefonnummer, E-Mail-Adresse etc.), die Aktualität der Daten, mögliche Fehler in den Daten oder der Analyse und Interpretationshilfen es dazu gibt, müssen verfügbar gemacht werden.

Auch diese Metadaten dürfen nicht losgelöst sein, sondern müssen in den RMOL integriert werden, d. h. werden in den ROL importiert und durchlaufen dafür ebenfalls einen zur Rekonziliierung der übrigen Daten analogen Prozess, der diese Daten „extrahiert", überprüft, bereinigt, vereinheitlich, transformiert und in das Object Warehouse lädt.

Allerdings weicht der Prozess der Usage-Metadaten-Extraktion von den bisherigen ab. Usage-Metadaten fallen kontinuierlich bei der Nutzung an und werden gewöhnlich nicht explizit in einer Datenbasis persistent gemacht. Deshalb können die Usage-Metadaten nicht einfach zu Fixterminen geladen werden. Als Konsequenz bietet sich zumindest deren temporäre Speicherung in einem Operational Usage-Metadata Store an, etwa in Form eines referenzierten Listservers.

---

[171] Dass diesem Punkt in vielen Unternehmen in nicht ausreichendem Maß Beachtung geschenkt wird, ändert nichts an der Tatsache, dass entsprechend komplexe Strukturen erforderlich und meistens auch erwünscht sind. Beispielsweise werden zur Zeit bei einem weiteren Projekt [Rieg99, S. 288 ff., Stif99] alleine für die aus Baan extrahierten und in das Data Warehouse überführten Daten aus der Finanzbuchhaltung, ca. 3000 Einzelberechtigungen implementiert. Die Existenz einer solchen Rechtestruktur stellt eine notwendige Bedingung für die Akzeptanz dar und damit auch für das Projekt insgesamt.

Wichtig für diesen Prozess ist, dass zum einen der Zugang für die End-Anwender geöffnet wird, damit diese die Möglichkeit bekommen diese Usage-Metadaten der MSS-Front-Ends und des DOL in das Object Warehouse zu überführen und zum anderen die Übernahme dieser Daten nicht dem sonstigen Maß an Datenqualität des ROL entgegensteht. [172] Die Daten müssen mithin auch den ökonomischen Restriktionen der Pflegbarkeit genügen. Dies ist vergleichbar mit einer Web-Site, die auf alte, nicht weiter funktionsfähige oder gar — was noch schlimmer ist — falsche Links verweist.

4.5 Anwendungs-orientierte Schichten

Es lassen sich zwei grundlegende Anwendungs-orientierte Schichten unterscheiden (siehe Abbildung 17):

- der Derived Object Layer (DOL) und
- die MSS-Front-Ends.

Der Derived Object Layer besteht aus unterschiedlichen Schichten oder Sichten, die jeweils auf dem Reconciled Object Layer aufsetzen. Dies bedeutet, dass alle Elemente des DOL aus dem ROL abgeleitet sind und entweder als eine Sicht (View) realisiert sind — etwa in Form einer Anfrage oder mithilfe einer strukturellen Sichtendefinition — oder als eine tatsächlich abgeleitete Schicht, die auf konzeptioneller und eventuell auch physischer Ebene eigenständig ist.

Insbesondere im Zusammenhang mit OBMS muss auch diesbezüglich zwischen dem konzeptionellen und dem physichen Design unterschieden werden. Physisches Design, beispielsweise die Replikation von Objekten, Objektmengen, Objekt-Clustern et cetera, sind übliche OBMS-Funktionalitäten und werden deshalb durch das OBMS verwaltet. Auch die Verteilung über unterschiedliche Orte, Plattformen et cetera ist dabei in der Regel problemlos möglich.

Solange die Object Marts durch dasselbe OBMS verwaltet werden, bleibt auch der Bezug zu den Original-Objekten erhalten, was in manchen Fällen von Interesse ist (Drill-Down). Werden allerdings losgelöste Kopien von den Objekten erstellt, gehen auch die Bezüge verloren, da anders als bei RDBMS keine expliziten Referenzen (Primärschlüssel) bestehen und die vom OBMS verwalteten Objektidentitäten (OID) beim tatsächlichen Kopieren verloren gehen.

Es spricht mithin alles dafür, auch die Object Marts innerhalb desselben Object Warehouse zu verwalten, was auch dem Integrations-Gedanken förderlich ist

[172] Vor allem müssen Anreize geschaffen werden damit eine ausreichende Motivation bei den End-Anwendern entsteht dies auch tatsächlich zu tun. Die Diskussion zu dieser Problematik wird insbesondere in der Knowledge Management Community geführt.

und insbesondere auch die gemeinsame Verwaltung bzw. den Austausch der Objekte bzw. der Daten und Metadaten erheblich vereinfacht. So brauchen anstelle von Kopien der Meta-Objekte lediglich die Referenzen auf diese Objekte kopiert werden.

Die diskutierte Sichten-Problematik ergibt sich bei diesem Ansatz nicht. Da die Sichten, d. h. die Object Marts, rein lesend verwendet werden, ergibt sich auch keine Update-Problematik. Für rein lesende Zugriffe sind die Views auch in OBMS auf unproblematische Weise zu verwenden.

Mögliche Object Marts sind Objekte auf höherer Aggregationssstufe bzw. Verdichtungen oder beliebige OQL-Anfragen. Dabei besteht die Möglichkeit zu einem verändertem bzw. ergänzenden physischen Design, wie unter anderem weiteren Indizes, Replikationen, zusätzlichen Objektmengen (Extensionen), die beispielsweise Ausschnitte, Vereinigungen oder vollständig neue Objektmengen sein können.

Im Bereich der MSS-Front-Ends kann mithilfe einer OQL entweder direkt auf den ROL oder auf den DOL zugegriffen werden. Der direkte Zugriff auf den ROL wird jedoch in vielen Fällen für den End-Anwender zu kompliziert sein, da der Umgang mit den ROL aufgrund der weitaus größeren Objektmenge bzw. des weitaus komplexeren Objektmodells, insbesondere auch aufgrund der Versisionierung, wesentlich schwieriger ist als über spezielle Benutzer-Sichten, d. h. die Object Marts zuzugreifen.

Neben der direkten OQL-Schnittstelle muss es jedoch vor allem spezielle MSS-Werkzeuge geben, die eine analoge Fuktionalität zu den konventionellen MSS-Werkzeugen (MSS-Generatoren) aufweisen. Der Hauptzweck solcher MSS-Werkzeuge liegt darin, die zusätzliche Semantik des Objektmodells, wie die implizite „Natürlichkeit“ oder auch die integrierten Usage-Metadaten für den End-Anwender tatsächlich auf einfache Weise nutzbar zumachen.[173]

Hierbei wird der Kreisschluss gezogen. Denn erst die OMSS-Werkzeuge rechtfertigen tatsächlich den Aufwand für das Object-Warehouse-Konzept. Hier kommen die objektorientierten GUI ins Spiel, die an dieser Stelle mit den Informationsobjekten des Object Warehouse verbunden werden können. Wie gezeigt, gibt es kein vergleichbares, derart leistungsstarkes GUI-Tool, wie das von echten objektorientierten Umgebungen, wie etwa das GUI-Framework von VisualWorks. Es gilt zukünftig, die Potenziale dieses Frameworks mit den Object-Warehouse-Objekten zu verbinden. Dasselbe gilt für die organisatorischen Konzepte.

Auch der Ansatz mehrdimensionaler Strukturen (OLAP) ist mit komplexen Objektstrukturen machbar und eröffnet die Möglichkeit, Object-OLAP-Marts zu im-

[173] Dies ist Gegenstand einer parallelen Forschungsarbeit am Fachgebiet BWL/Wirtschaftsinformatik II der Universität Osnabrück.

plementieren, bei welchen analog zum konventionellen OLAP die Objekte in Fakten und Dimensionen unterteilt werden, so dass eine zusätzliche Dimensions-Semantik ergänzt wird.

Beispielsweise schlägt *Ohlendorf* hierfür einen speziellen Modellierungs-Ansatz vor (siehe Abbildung 19 und Abbildung 20). *Ohlendorf* implementiert spezielle Dimensions-Klassen, die als Oberklasse eine spezielle Klasse „Dimension“ haben. Eine Instanz der Klasse „Würfelfeld“ setzt sich dann jeweils aus einer Kombination von Dimensions-Instanzen zusammen. Jede Instanz der Klasse „Würfelfeld“ kann dann zu verschiedenen Instanzen der Klasse „Kenngröße“ gehören, wodurch sich eine vollständige Identifizierung einer Instanz der Klasse „Atom-Datum“ ergibt. [Ohle98, S. 230 ff.]

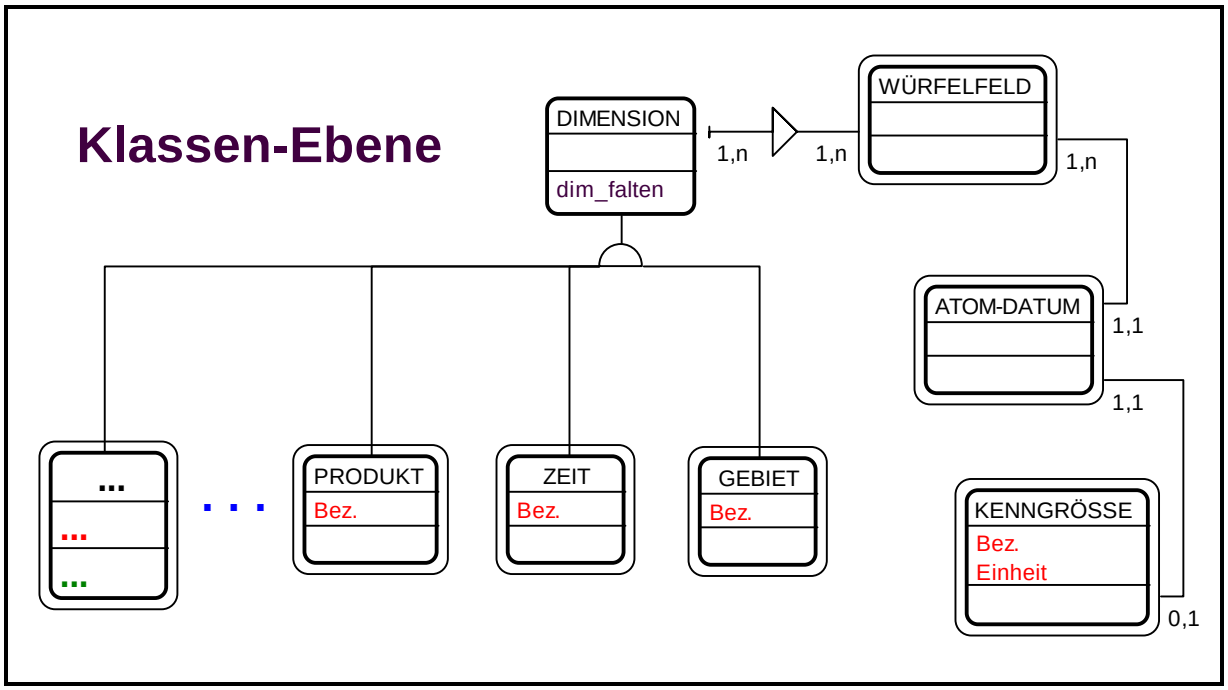


Abbildung 19: Klassendiagramm eines Object-OLAP-Marts [Ohle98, S. 232]

Außerdem ist die Möglichkeit gegeben, die Objekte auch für konventionelle MSS-Front-Ends nutzbar zu machen. Dazu müssen die komplexen Objekte in flache Strukturen überführt werden. Dies widerspricht zwar grundsätzlich dem Gedanken des Object Warehouse, da die Daten um wesentliche Bestandteile der Semantik reduziert werden müssen, bietet aber auf der anderen Seite die Möglichkeit, die zur Zeit verfügbaren konventionellen MSS-Werkzeuge zu nutzen. Sofern dies für Teilbereiche des Objektmodells durchgeführt wird, ist der Aufwand in vertretbarem Rahmen.

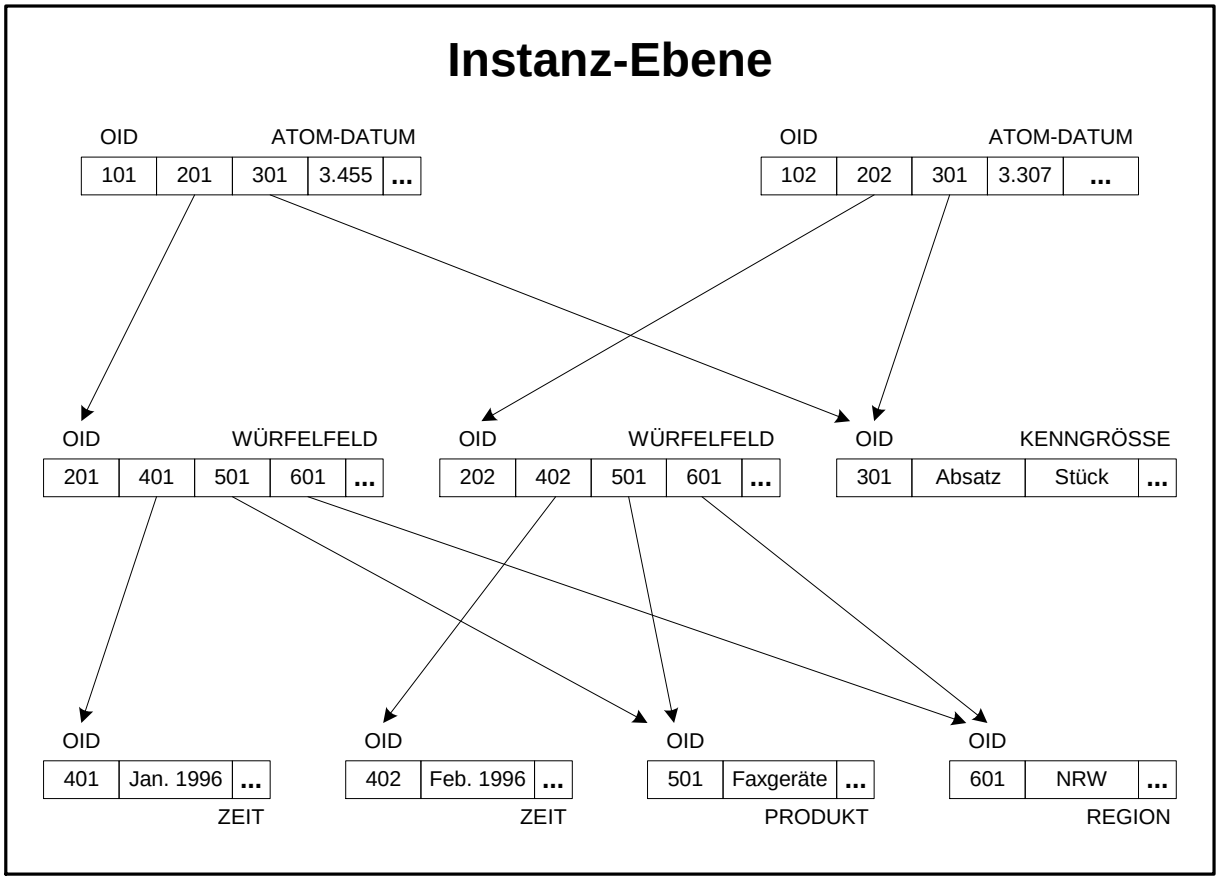
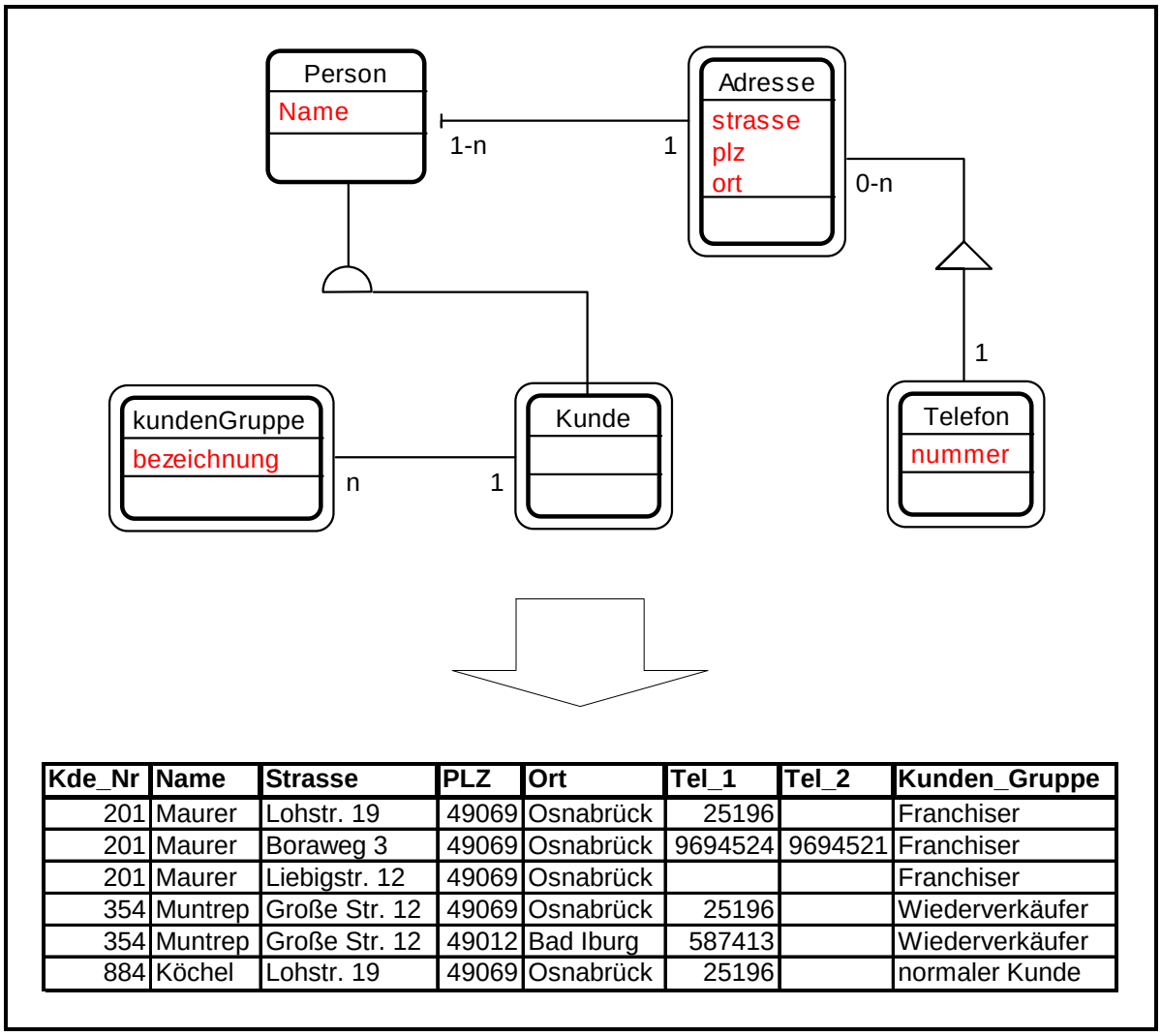


Abbildung 20: Darstellung eines möglichen Object-OLAP-Marts auf Instanz-Ebene [Ohle98, S. 233]

Beispielsweise könnten aus den Kunden-Objekten alle aktuellen Kunden-Attribute, d. h. der jeweils letzten Version (z. B. Name, Straße, PLZ, Ort, Telefonnummern etc.) in Tupel einer Tabelle eines RDBMS überführt und sogar mit dem ROL verknüpft werden, so dass auch hierfür automatisch periodische Aktualisierungen vorgenommen werden können (siehe Abbildung 21)[174] .

Auch für die Analyse müssen die Metadaten nutzbar gemacht werden. Beispielsweise muss außer dem bisher in der Regel auf nur einer einfachen Tabelle aufsetzenden Konzept des Data Mining, die komplexen Strukturen des Objek-Modells ausgenutzt werden können. Es müssen also auch die Metadaten der Objekte in die Analyse miteinbezogen werden. Damit verfolgt man zusätzlich auch den Weg von einem einfachen Data Mining zu einem komplexen Object Mining.

[174] Die Modellierung dieses Beispiels ist auf vielfältige alternative Weise durchzuführen. Im vorliegenden Fall findet eine Beschränkung auf ein mögliches Beispiel statt, wobei die Modellierung dem pragmatischen Modellierungs-Ansatz eines üblichen ROLAP-Data-Marts entspricht und folglich nicht angemessen modelliert ist. Letzteres wäre allerdings auf die gleiche Weise realisierbar.



Kde_Nr	Name	Strasse	PLZ	Ort	Tel_1	Tel_2	Kunden_Gruppe
201	Maurer	Lohstr. 19	49069	Osnabrück	25196		Franchiser
201	Maurer	Boraweg 3	49069	Osnabrück	9694524	9694521	Franchiser
201	Maurer	Liebigstr. 12	49069	Osnabrück			Franchiser
354	Muntrep	Große Str. 12	49069	Osnabrück	25196		Wiederverkäufer
354	Muntrep	Große Str. 12	49012	Bad Iburg	587413		Wiederverkäufer
884	Köchel	Lohstr. 19	49069	Osnabrück	25196		normaler Kunde

Abbildung 21: Überführung einer Objektstruktur des ROL in ein Relationen-Modell eines Data Mart am Beispiel eines einfachen Kunden-Objekts

Während also der Reconciled Object Layer den Kern des Object Warehouse darstellt, besteht der Derived Object Layer aus teilweise sehr unterschiedlichen Komponenten. Mit dieser Architektur gelingt es einerseits eine rekonziliierte Objektbasis bereitgestellt zu bekommen, die auch langfristig konsistent bleibt und den Anforderungen der Anwender gerecht wird wie auch andererseits beliebige Formen von MSS adäquat zu unterstützen, wofür der ROL als Daten- und Objektquelle zur Verfügung steht.

TEIL II

ENTWICKLUNG EINES OBJECT-WAREHOUSE-PROTOTYPEN

KAPITEL 5

PROTOTYP-KONZEPTION

Die Konzeption des Object Warehouse ist wesentlich durch praktische Erfahrungen beeinflusst. Dazu zählen drei konkrete Projekte, mehrere Diplomarbeiten und insbesondere auch die Erfahrungen, die aus der Durchführung von Lehrveranstaltungen und praktischen Übungen sowie der Implementierung verschiedener kleinerer Prototypen gewonnen werden konnten.

Nach der Vorstellung der Praxisprojekte, die dazu dient, die Aufgabenstellungen und den Umfang der Projekte einschätzen zu können, werden exemplarisch Ergebnisse der Informationsbedarfsanalysen und der Datenquellenanalysen erläutert. Schließlich wird das Domänen-Modell eines Prototypen vorgestellt, das als grundlegende Schicht des Reconciled Object Layers dient.

5.1 Praxisprojekte und praktische Erfahrungen

Die Erfahrungen für die Entwicklung und Überprüfung von einer Object-Warehouse-Konzeption stammen aus drei konkreten Projekten mit Unternehmen, die für diesen Zweck durchgeführt wurden. Die Projektgruppen setzten sich aus jeweils etwa acht Studenten zusammen, die sich zur Zeit der Projektdurchführung im fortgeschrittenen Haupstudium befanden. Die Projektleitung oblag dem Verfasser dieser Arbeit.

Ergebnisse sind umfangreiche Projektbeschreibungen (Pflichtenhefte und Präsentationsunterlagen) und objektorientierte MSS-Prototypen, wobei sich zeigt, dass der erste Schritt in der Regel darin besteht eine integrative Datenbasis zu schaffen, also ein Data Warehouse respektive Object Warehouse zu realisieren. Weiterhin sind gewonnene Erkenntisse aus Forschungsaktivitäten verzögerungsfrei in die Lehrkonzepte eingeflossen [Lehr00] — die dabei gewonnenen Erfahrungen wiederum in die Konzeption des Object Warehouse.

Das erste Projekt erfolgte 1996 in Zusammenarbeit mit einer führenden Drogeriemarktkette. [BHJ+96] Der damalige Jahresumsatz betrug 1,6 Mrd. DM. Das Unternehmen beschäftigt ungefähr 9000 Mitarbeiter. Bundesweit werden etwa 700 Filialen betrieben.

Die Zielsetzung des Projekts war die Analyse der an den Kassen anfallenden Scannerdaten. Mit einer Bonanalyse sollte ein erster Schritt zum Einstieg in die MSS-Technologie erfolgen. Vordringliches Anliegen war dabei, die Potenziale der MSS-Technologie aufzuzeigen, um die Geschäftsleitung von dem Nutzen einer Investition in diese Technologie zu überzeugen. Inzwischen wurde das bis dahin outgesourcte Warenwirtschaftssystem zugunsten von SAP/R3 ausgetauscht. Ein umfangreiches Data-Warehouse-Projekt wird in circa zwei Jahren abgeschlossen sein.

Das zweite Projekt wurde 1997 mit einem Unternehmen für Fahrzeugkühlung, stationäre Kühlung (insbesondere im Lebensmittel-Einzelhandel) und Klimatechnik durchgeführt. [BBB+97] Das Unternehmen beschäftigt bundesweit ungefähr 120 Mitarbeiter.

Den Schwerpunkt dieses Projekts bildeten eine objektorientierte System- und Informationsbedarfsanalyse und die Konzeption eines rechnergestützten Controllings. Die Realisierung eines Prototypen war aufgrund unternehmensinterner Veränderungen im Laufe des Projekts nicht möglich.

Das dritte Projekt fand 1998 mit einem Landmaschinenhersteller statt. Der Jahresumsatz beträgt circa 140 Mio. DM. Das Unternehmen beschäftigt ungefähr 500 Mitarbeiter. Die Geschäftsprozesse reichen von der Konstruktion über die Fertigung bis zum direkten Verkauf an Kunden, wobei der Vertrieb bisher hauptsächlich durch Vertreter auf Provisionsbasis erfolgt. Es besteht eine hohe Fertigungstiefe. Als operatives System wird Baan eingesetzt.

Zielsetzung war die Schaffung einer integrierten Datenbasis für die vertriebsrelevanten Daten und deren Aufbereitung mithilfe eines MSS-Front-Ends. Projektbegründend war die Unzufriedenheit des Unternehmens mit den erheblich beschränkten Auswertungsmöglichkeiten der Baan-Software und der Wunsch, durch eine verbesserte Informationsversorgung mithilfe von MSS den Vertrieb verstärkt selbst durchführen zu können. Bei der sich anschließenden Realisierung wurden zwei alternative Ansätze parallel umgesetzt:

- Ein konventioneller Ansatz auf der Basis von Oracle 7.3, als erste Stufe eines Data Warehouse und dem Oracle Discoverer, als MSS-Front-End. [Blac99]
- Ein objektorientierter Ansatz mit GemStone, als erste Stufe eines Object Warehouse. [Nieb99, Weus99]

5.2 Informationsbedarfsanalyse

Nach einer Zusammenfassung der wichtigsten Ergebnisse der durchgeführten Informationsbedarfsanalysen und einem Überblick über das gewählte Vorgehen, werden beispielhaft einige von den Projektpartnern als wichtig angesehene Analyseberichte vorgestellt. Im Fazit werden die essenziellen gewonnenen Erkenntnisse kurz subsumiert.

5.2.1 Vorgehen und Ergebnisse

Die Informationsbedarsanalysen[175], die verstärkt in den ersten beiden Projekten erfolgten, wurden mit den üblichen Erhebungstechniken Unterlagenstudium, Fragebogen, Interview, Beobachtung und Selbstaufschreibung durchgeführt. Die hauptsächlichen Schwachstellen bei den Analysen waren heterogene DV-Landschaften mit unzähligen Medienbrüchen, Redundanzen und Inkonsistenzen sowie insbesondere organisatorische und psycho-soziale Probleme im Mitarbeiterbereich.

Es zeigte sich in vielen Fällen eine starke Ablehnung gegenüber einer angestrebten Informationstransparenz, da Informationsinseln — oftmals als Sicherung der persönlichen Stellung im Unternehmen — aufgebaut wurden und dieser Umstand auch nicht durch die Geschäftsleitungen zu ändern versucht wurde.

Bei allen Erhebungen wurde deutlich, dass der eigentliche Informationsbedarf durch die angewandten Techniken nur schwer zu erheben ist, da die Mitarbeiter vor Projektbeginn in der Regel nicht wissen, welche Informationen sie benötigen, dies nicht ausdrücken können und/oder kein Interesse an der Weitergabe ihres diesbezüglichen Wissensstandes haben.

Gute Ergebnisse konnten zu diesem Zweck mit Prototypen erreicht werden. Durch die Rückkopplung anhand der Prototypen entstanden immer neue Anforderungen. Allerdings war es aufgrund der begrenzten Projektrahmen nicht möglich, ausgedehnte Versionszyklen zu durchlaufen.

---

[175] *Walpoth* setzt die Begriffe Informationsbedarfsanalyse und Systemanalyse gleich. [Walp93, S. 5 ff.] In dieser Arbeit wird die Informationsbedarfsanalyse allerdings als eine spezielle Form der Systemanalyse verstanden. Die Zielsetzung der Informationsbedarfsanalyse besteht primär in der Ermittlung des Informationsbedarfs im Sinne des Management Supports, wie er in dieser Arbeit formuliert wird (siehe Kapitel 2.1). Die Ermittlung bezüglich Informations- und Kommunikationsstrukturen und -systemen des operativen Bereichs [Kral94] ist deshalb nur am Rande betroffen. Ebenfalls erfolgt eine Einschränkung bezüglich der organisatorischen und psycho-sozialen Dimension [Kral94] der Informationsbedarfsanalyse, die zwar wie alle Projekte gezeigt haben höhere Grade erfolgskritischer Faktoren aufweisen, jedoch kein Gegenstand der vorliegenden Arbeit sind.

5.2.2 Informationsbedarf der Drogeriemarktkette

Als erste Stufe des Informationsbedarfs der Drogeriemarktkette konnte folgendes ermittelt werden:

1. Findung einer statistischen Basis (Menge + Regional).
2. Stückanalyse: Wieviele Artikel werden von den Kunden (= Bons) gekauft?

   Beispiel:

Abruftag: Donnerstag, den 30. Mai 2000				
Filiale Nr. 001 in 12345 Bad Schnitzlach			Summe Scanner-Statistikfilialen	
Anzahl Artikel	Anzahl Bons	Anteil %	∅ %	Anzahl Bons
1 Artikel	98	4,1 %	3,8 %	3665
2 Artikel	124	5,1 %	6,2 %	5980
3 Artikel	187	7,8 %	8,3 %	8005
usw.				
10-15 Artikel	2	0,1 %	0,1 %	96
16-20 Artikel	1	0,0 %	0,0 %	3
21 -	0	0,0 %	0,0 %	1
SUMME	2410	100 %	100 %	96453

Tabelle 2: Stückanalyse im Projekt Drogeriemarkt

3. Gezielte Anfrage nach bestimmten Artikelnummern/EAN (z. B. Inseratskontrolle)

   Beispiel:

10466-120 Kukident aktiv 3, 210 04 Gebissreiniger			Anfragezeitraum: 30.5. - 01.06. 2000	
Anzahl Märkte	Anzahl Bons gesamt	Anzahl Bons mit IPNr. 10466	Anzahl Artikel nur IPNr. 10466	Anteil % Bons mit IPNr. 10466
40 von 40	285.963	24.592	25.876	8,6 %
10466-120 Kukident aktiv 3, 210 04 Gebissreiniger			Vergleichszeitraum: 23.05. - 25.05.2000	
38 von 40	265.530	12.937	13.013	4,8 %
10466-120 Kukident aktiv 3, 210 04 Gebissreiniger			Vergleichszeitraum: 06.06. - 08.06.2000	
32 von 40	255.735	9.462	9.489	3,7 %

Tabelle 3: Artikelanfrage nach Artikelnummern im Projekt Drogeriemarkt

   3.1 Welche/Wieviele Artikel sind aus variablen Artikelgruppen verkauft worden?

   Beispiel: Inserat Gebissreiniger - Auswirkungen auf nicht inserierte Artikel aus den Artikelgruppen 210 02 Mundspülung + 201 28

Mundwasser
(dto. Vorzeitraum - Aktionszeitraum - Nachzeitraum)

3.2 Welche Auswirkungen hatte ein Mundhygiene-Inserat auf die Warengruppen 100 Kosmetik diverse + 223 Schaum-Duschbäder + 225 Handcremes, Körperlotion

4. Bonwertanalyse

   Beispiel:

Bonwertanalyse - Donnerstag, den 30. Mai 2000		Mittwoch, den 29. Mai 2000
Bonwert	Anzahl Bons	Anzahl Bons
DM 0,41 - 1,00	123	105
DM 1,01 - 3,00	543	482
DM 3,01 - 5,00	728	598
DM 5,01 - 7,50	...	...
DM 7,51 - 10,00	...	...
DM 10,01 - 15,00	...	...
DM 15,01 - 20,00	...	...
DM 20,01 - 30,00	...	...
DM 30,01 - 40,00	...	...
DM 40,01 - 50,00	...	...
über DM 50,01	...	...
GESAMT:	123,456	112,345

Tabelle 4: Bonwertanalyse im Projekt Drogeriemarkt

   Vergleiche: - Woche/Vorwoche
   - Monat/Vormonat
   - Vorjahresvergleiche

5. Warengruppen-Entwicklungsbeobachtung nach Umbauten, internen Umstellungen, Warenplazierung.

5.2.3 Informationsbedarf des Landmaschinenherstellers

Als erste Stufe des Informationsbedarfs des Landmaschinenherstellers für das Vertriebsinformationssystem ergab sich das Folgende:

1. Gewinnermittlung im Bereich Handelsware (Verkaufspreis-/Einkaufspreis-Abgleich pro Projekt).
2. Permanente Überwachung der Gewinnmargen jeder einzelnen Maschine (kumulierte After-Sales-Kosten und Herstellkosten bzw. Selbstkosten gegen Umsatz). After-Sales-Kosten können z. B. Servicekosten sowie Kosten für Umbauten sein. Auswertung ebenfalls über Projekt bzw. Installation.

3. Errechnung von Deckungsbeiträgen pro Maschine und pro Baureihe.
4. Optionsauswertungen, um z. B. Prognosen zu bewerten.
5. Umfassende und schnell verfügbare Informationen über den Kunden, z. B. Umsatz, Maschinen im Einsatz, ausstehende Aufträge et cetera in einer Maske bzw. im Rahmen einer Anfrage. Die Daten sollten tagesaktuell sein.
6. Nettoumsatzliste Maschinenvertrieb pro Land, pro Region, pro Vertreter, pro Kunde, pro Projekt, pro Installation (Auftragsart M01-M99).
7. Nettoumsatzliste Pflanzmaschinen pro Land, pro Region, pro Vertreter, pro Kunde, pro Projekt, pro Installation (Auftragsart HG1-HG9).
8. Erlösliste Pflanzmaschinen pro Land, pro Region, pro Vertreter, pro Kunde, pro Projekt, pro Installation. Gemeint ist hiermit ein statistischer Abgleich zwischen den Einkaufs-Tabellen und dazugehörigen Verkaufs-Tabellen. (Auftragsart Verkauf HG1)
9. Nettoumsatzliste Ersatzteile pro Land, pro Region, pro Vertreter, pro Kunde, pro Preisgruppe, pro Artikel (Auftragsart E01-E99).
10. Gesamterlösliste Maschinen pro Land, pro Region, pro Vertreter, pro Kunde, pro Projekt, pro Installation. Dabei muss die Auswertung als jeweiligen Zielwert den aktuellen Gewinn der einzelnen Maschinen ausweisen:

	Bruttoumsatz
./.	Rabatte (Kunde/Maschine als Artikel vorhanden)
./.	gesamte Herstellkosten
./.	Servicekosten
Ergebnis =	Nettogewinn

Dabei bildete der letzte Punkt den Schwerpunkt des Projekts, da mit diesen Anforderungen zum einen das größte Potenzial, zum anderen der meiste Aufwand verbunden war.

5.2.4 Fazit

Diese Ergebnisse zeigen, dass der primäre Bedarf der untersuchten Unternehmen zumindest anfänglich in relativ einfachen, meist mehrdimensionalen Analysen bzw. Kennzahlen liegt. Welche konkreten Analysen dabei benötigt werden, ist, wie sich zeigt, für die Gestaltung des Systems (nur) deshalb von primärem Interesse, da nach diesen Erhebungen die auf der ersten Stufe relevanten Datenquellen und benötigten Fakt- und Dimensionsdaten selektiert bzw. modelliert werden können. Wie die konkrete Analyse gezeigt hat, bestehen bei den geforderten Anfragen viele Überschneidungen. Mithilfe einer entsprechenden MSS-Datenbasis-Front-End-Kombination können die entsprechenden Anfragen dann auf

relativ einfache Weise durch den Fachspezialisten erstellt werden, ohne das Warehouse daraufhin jedes Mal anpassen zu müssen.

Ein Grundverständnis bei den Fachspezialisten der Projektpartner für das konzeptionelle Modell mehrdimensionaler Strukturen, am besten in Form der Würfelmetapher, erleichtert die Kommunikation bei der Informationsbedarfsanalyse und leistet deshalb auch für die Ermittlung des Domänen-Modells einen wichtigen Beitrag.

Damit bestätigt sich die *burlesonsche* These, dass mehrdimensionale Analysen in der Regel die erste Stufe der Informationsbedürfnisse abdecken bevor tiefergehende Analysen, wie etwa in Formen des Data Minings gefordert werden. [Burl-97, S. 40 ff.] Dabei ist das primäre Ziel, die vorhandenen Daten zusammenzuführen und zu vereinheitlichen, damit eine gemeinsame Auswertung überhaupt möglich wird. Es konnte bei allen Projekten als Ist-Zustand beobachtet werden, dass zu diesem Zweck bereits manuelle Übernahmen der Daten in MS-Excel erfolgten, teilweise auch in MS-Access — in vielen Fällen durch das Abschreiben von Drucklisten. Eine Vereinheitlichung findet dadurch allerdings nicht statt. Vielmehr zeigt sich, dass die Redundanz und Inkonsistenz der Daten dadurch weiter verstärkt wird. Dies verdeutlicht auch die Datenquellenanalyse.

5.3 Datenquellenanalyse

Nach einer kurzen Zusammenfassung über die wichtigsten Ergebnisse der durchgeführten Datenquellenanalysen und einem Überblick über das gewählte Vorgehen, werden die elementaren Schritte am Beispiel der vertriebsrelevanten Baan-Daten des Landmaschinenherstellers und der Scannerdaten der Drogeriemarktkette vorgestellt.

5.3.1 Vorgehen und Ergebnisse

Die Datenquellenanalyse zeigte in allen Fällen eine in hohem Maße heterogene DV-Landschaft mit einer unüberschaubaren Anzahl zusätzlicher Listen und Reports — größtenteils mit MS-Office-Komponenten (MS-Excel, MS-Word, MS-Access) erstellt und gepflegt. Medienbrüche, Redundanzen, Inkonsistenzen und Informationsinseln bzw. -intransparenz waren die Regel. Proprietäre Datenhaltungsformate, schlechte Datenmodelle und minderwertige Datenqualität bildeten den Normalfall. Ein zentrales OLTP-System war bei allen Projekten vorhanden (WWS-Eigenentwicklung von VRG, Pro-Fakt, Baan), ein RDBMS bildete aller-

dings zum jeweiligen Projektzeitpunkt in keinem der Fälle die Basis des zentralen operativen Systems (Enterprise Resource Planning System).[176]

Die wesentlichen MSS-Datenquellen sind die operativen Systeme. Den Ausgangspunkt bildeten deshalb die ERP-Systeme — beim Projekt mit der Drogeriemarktkette das Kassensystem, da keine Kopplung zwischen Kassensystem und Warenwirtschaftssystem vorhanden war.

Sind die primären Datenquellen identifiziert, gilt es, diese detaillierter zu analysieren. Der erste Schritt besteht darin, ein Grundverständnis für das untersuchte Datenmodell und die betroffenen Geschäftsprozesse zu bekommen, d. h. die wesentlichen Objekte des Unternehmens müssen vom Projektteam identifiziert und verstanden werden. Zu diesem Zweck bedurfte es eines intensiven Unterlagenstudiums und einer großen Anzahl von Besprechungen mit den Fachspezialisten der operativen Systeme.

Bei der Standardsoftware musste ein Verständnis für die Nutzung der Standardfunktionen und notwendigen Veränderungen vom Standard (Anpassungen) gewonnen werden. Beispielsweise wird aufgrund der ungenügenden Möglichkeiten von Baan, die Rabatte angemessen abbilden zu können, beim Landmaschinenhersteller zusätzlich eine umfangreiche „Produktpalette“ von Rabatten eingeführt, d. h. Artikel, die negative Kosten verursachen.

Nachdem die wichtigsten Dateien bzw. Tabellen gefunden waren, konnten diese auf ihr Mengengerüst und mögliche Attributwertausprägungen untersucht werden. Anschließend wurden verschiedene Tests durchgeführt, wie Plausibilitätstests, Tests zur Prüfung der Datenkonsistenz et cetera.

5.3.2 Datenquellenanalyse am Beispiel der Baan-Daten

Die im ersten Schritt für das Vertriebsinformationssystem des Landmaschinenherstellers identifizierten relevanten Tabellen stammen aus den Paketen Distribution, Manufacturing, Service und Common (siehe Tabelle 5).

Für die weitere Analyse werden die relevanten Tabellen (siehe Tabelle 5) aus TBase extrahiert und in Oracle importiert. Baan stellt hierfür in der Version Baan/Trition 3.1 Funktionen des Moduls „Triton Exchange 6.1“ zur Verfügung (siehe Kapitel 6).

Die Datenquellenanalyse ist ein heuristischer Prozess, so dass es in dieser Phase darauf ankommt überhaupt eine erste Vorstellung über die Daten und das Da-

[176] Die Datenhaltung in Baan erfolgte bisher in der proprietären Datenhaltungskomponente TBase der Baan-Software. TBase erfüllt allerdings nicht die notwendigen Kriterien [Date00, S. 33 ff.], um als RDBMS eingestuft werden zu können. Inzwischen ist die Möglichkeit gegeben, die Datenhaltung auf Oracle zu übertragen.

tenmodell der relevanten Baan-Tabellen zu bekommen, weshalb die in Oracle importierten Tabellen (siehe Tabelle 5) auf deren Attribute, Typen, Werte, Mengen und mögliche Attributwertausprägungen untersucht wurden.

Paket	Modul	Tabelle	Inhalt
Distribution (td)	Purchase Control (pur)	tdpur040	Bestellung
		tdpur041	Bestellposition
		tdpur050	Bestellhistorie
		tdpur051	Bestellpositionshistorie
	Sales Control (sls)	tdsls040	Verkaufsauftrag
		tdsls041	Verkaufsauftragsposition
		tdsls050	Verkaufsauftragshistorie
		tdsls051	Historie für Verkaufsauftragsposition
Manufacturing (ti)	Project Control System (pcs)	tipcs020	Projekte
		tipcs021	kundenspezifische Artikel
	Item Control (itm)	tiitm001	Artikel
Service (ts)	Service and Maintenance (sma)	tssma102	Installationen
		tssma301	Serviceaufträge
		tssma310	Vor- und Nachkalkulation für Serviceaufträge
Common (tc)	System Tables (mcs)	tcmcs010	Länder
		tcmcs023	Artikelgruppe
		tcmcs031	Branchen
		tcmcs045	Regionen
	Common Data (com)	tccom010	Kunden
		tccom020	Lieferanten

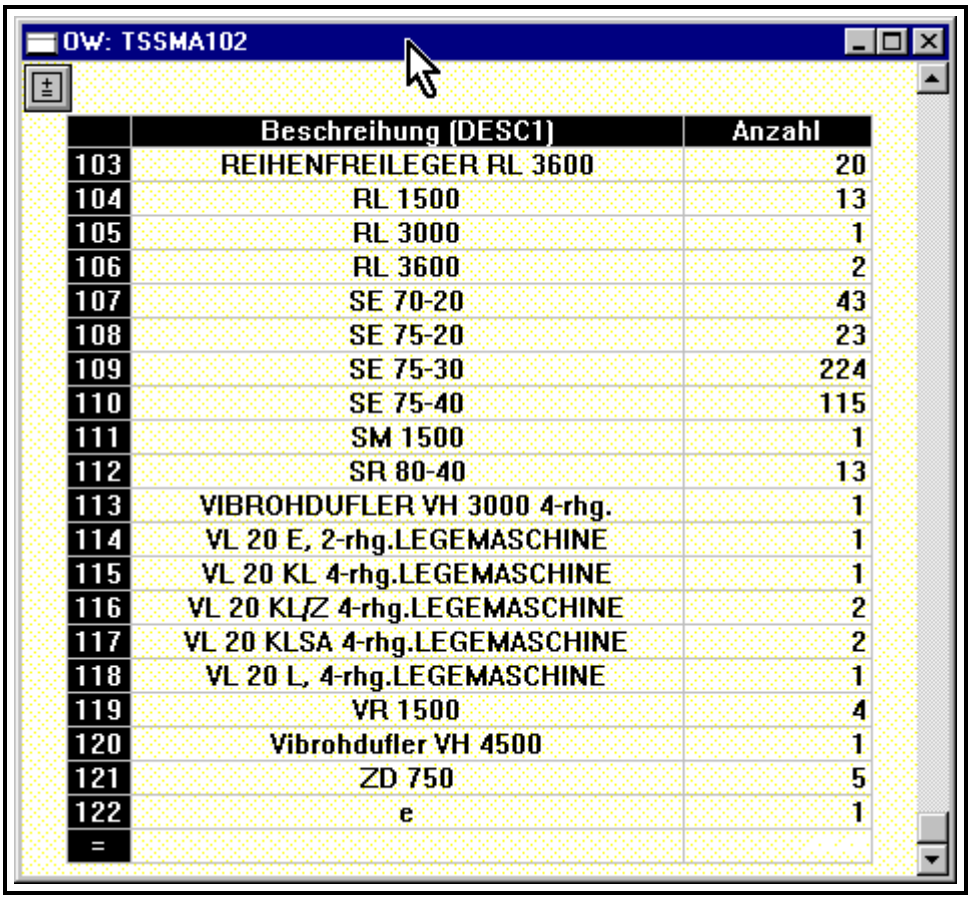
Tabelle 5: Relevante Baan-Tabellen des Vertriebsinformationssystems

Beispielsweise zeigt eine erste Überprüfung des Mengengerüsts, dass die Historietabelle der Verkaufsauftragspositionen „tdsls051“ 479.429 Datensätze enthält. Weiterhin wurde gezeigt, dass bei den insgesamt 4048 Installationen „tssma102“ 122 unterschiedliche Bezeichnungen „desc1“ vorhanden sind und wie oft die jeweilige Bezeichnung vorkommt („e“ gibt es z. B. einmal, siehe Abbildung 22).

Für die weiterführende Datenquellenanalyse wurden verschiedene Werkzeuge, wie Oracles SQL-Plus, der Oracle Navigator und MSS-Front-Ends eingesetzt, beispielsweise Forest & Trees (siehe Abbildung 22). Zum Beispiel erfolgten die Anfragen in Oracle, d. h. nur die Ergebnisse der Anfragen wurden mithilfe von Forest & Trees dargestellt, wodurch die Anfrage-Effizienz von Oracle und die Benzutzerfreundlichkeit der MSS-Front-Ends ausgenutzt werden konnten.

Beispielsweise mithilfe des Oracle Navigators, mit dem unter anderem die Tabellen, deren Strukturbeschreibungen und Inhalte analysiert wurden, konnte auch

das vielfach notwendige Ändern von Strukturen und Inhalten auf einfache Weise durchgeführt werden, da mit dem Oracle Navigator zum Beispiel direkt in die Tabellen geschrieben werden kann, um zumindest einige Datenfehler von Hand korrigieren zu können und auch Schemaänderungen sind direkt durchführbar.



	Beschreihung (DESC1)	Anzahl
103	REIHENFREILEGER RL 3600	20
104	RL 1500	13
105	RL 3000	1
106	RL 3600	2
107	SE 70-20	43
108	SE 75-20	23
109	SE 75-30	224
110	SE 75-40	115
111	SM 1500	1
112	SR 80-40	13
113	VIBROHDUFLER VH 3000 4-rhg.	1
114	VL 20 E, 2-rhg.LEGEMASCHINE	1
115	VL 20 KL 4-rhg.LEGEMASCHINE	1
116	VL 20 KL/Z 4-rhg.LEGEMASCHINE	2
117	VL 20 KLSA 4-rhg.LEGEMASCHINE	2
118	VL 20 L, 4-rhg.LEGEMASCHINE	1
119	VR 1500	4
120	Vibrohdufler VH 4500	1
121	ZD 750	5
122	e	1
=		

Abbildung 22: Eine erste Analyse möglicher Attibutwertausprägungen und deren Häufigkeit am Beispiel der Installationen-Tabelle „tssma102“ mit Forest & Trees

Nach weiteren Analysen zeigten sich Veränderungen bezüglich der relevanten Tabellen. Die Werte der aktuellen Verkaufstabellen Verkaufsauftrag „tdsls040“ und Verkaufsauftragsposition „tdsls041“ sowie die aktuellen Einkaufstabellen Bestellung „tdpur040“ und Bestellposition „tdpur041“ sind vollständig in den dazugehörigen Historietabellen abgebildet, so dass auf die Einbeziehung dieser Tabellen verzichtet werden konnte. Lediglich die Fremdschlüssel aus den aktuellen Tabellen mussten in die Historietabellen übernommen werden. Daraus ergibt sich das folgende Datenmodell (siehe Abbildung 23).

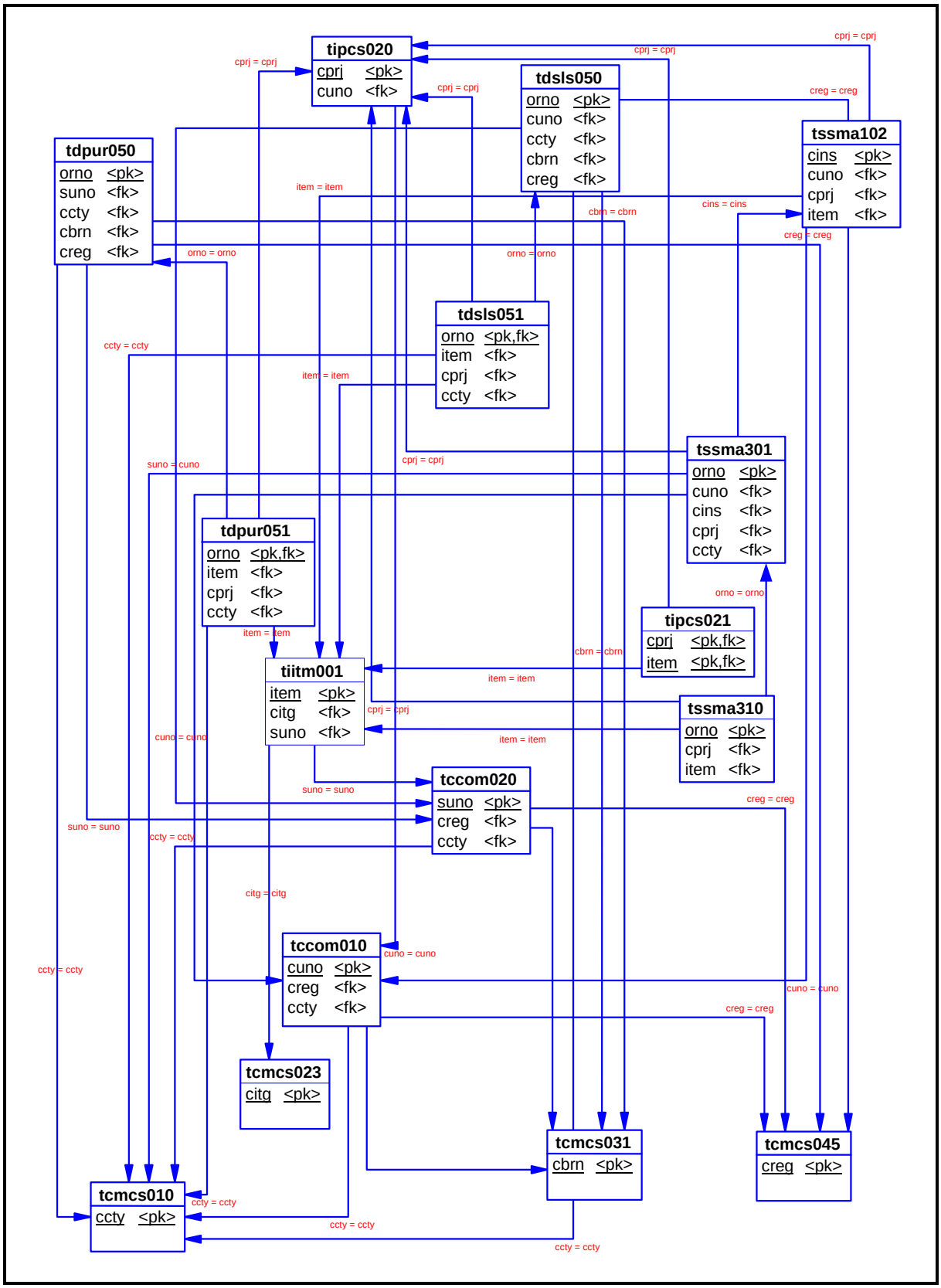


Abbildung 23: Relevantes Datenmodell aus TBase von Baan/Triton 3.1b

Wie mit der weiteren Analyse gezeigt werden konnte, weist das Baan-Datenmodell erhebliche Redundanzen auf, was bereits bei den Schlüsseln ersichtlich wird (siehe Abbildung 23).

Beispielsweise verfügen sowohl die Bestellhistorie „tdpur050“ wie auch die Bestellpositionshistorie „tdpur051“ über das Attribut Land „ccty“, das auf die Ländertabelle verweist. Bei einer Überprüfung der Primärschlüssel zeigte sich außerdem, dass diese in den Historietabellen nicht eindeutig sind, da Datensatzänderungen als jeweils eine negative und erneute positive Buchung mit dem selben Primärschlüssel durchgeführt werden.

5.3.3 Datenquellenanalyse am Beispiel der Scannerdaten

Die bei den Scannerkassen der einzelnen Filialen entstehenden Daten wurden als Zahlenkolonnen in einer ASCII-Datei gespeichert (siehe Abbildung 24) und über Nacht per Modem in die Zentrale übertragen[177]. Warum auch immer diese Form der Datenspeicherung vom Hersteller des Kassensystems (Höft & Wessels) gewählt wurde, sie verursachte das Problem, dass die Daten für eine Auswertung decodiert werden mussten. Entsprechende Funktionen des Herstellers wurden nicht bereitgestellt.

```
20258
60021513130049609163 00.00
70021513130049609165746
714003015006674
732.99
714003015006674
732.99
7130450
732.99
714056800114092
735.99
714056800114092
735.99
714084200401108
736.95
7427.90
75150.00
6102151313004960917
```

Abbildung 24: Ausschnitt aus einer Journaldatei der Scannerkassen des Drogeriemarktprojekts

[177] Diese weitgehend manuelle Übertragungsform verursachte bei den 128 Filialen, bei denen das Kassensystem zum damaligen Zeitpunkt bereits eingeführt worden war, erhebliche Probleme.

Es gibt verschiedene Satzarten (siehe Tabelle 6), die den Kontext der jeweils nachfolgenden Zahlen beschreiben. Der erste Schritt besteht darin, eine Vorstellung von der Häufigkeit der einzelnen Satzarten zu bekommen. Dazu wurden die Journaldateien in einem ersten Schritt einer Filiale und in einem zweiten Schritt aller 128 bis zu diesem Zeitpunkt angeschlossenen Filialen, jeweils von einer Woche in VisualWorks importiert und die Häufigkeit der Satzarten gezählt (siehe Tabelle 6). Es zeigte sich beispielsweise, dass mehr Kassiereranmeldungen als -abmeldungen verbucht wurden. Dadurch entstehen strukturelle Fehler in den Daten.

Satzart	Beschreibung der Satzart	Häufigkeit des Auftretens bei einer Journaldatei	Häufigkeit des Auftretens bei den Journaldateien einer Woche
20	Filialnummer	1	171
60	Kassiereranmeldung	134	18782
61	Kassiererabmeldung	134	18903
62	Kassiererabschluß	6	763
63	Filialabschluß	2	157
70	Kassenbonstatuszeile	1872	416590
71	Artikelnummer	5609	1294157
72	Menge	103	47327
73	Preis	5609	1293891
74	(Teil-)Summe	1914	431517
75	Zahlart/ Gegeben	1866	415091
76	Schecknummer/ Rechnungsnummer	0	175
80	Kontonummer	10	2152
81	Warengruppe/ Artikelnr.	0	24
82	Zahlart	9	1758
83	Betrag	10	2152

Tabelle 6: Satzarthäufigkeiten in 1 bzw. 128 Journaldatei(en)

Damit ein Decodierungs-Algorithmus entwickelt werden konnte, mussten Plausibilitätstests durchgeführt werden. Dazu wurde geprüft, welche Satzarten auf welche anderen Satzarten folgen bzw. welchen Satzarten vorausgehen (siehe Tabelle 7 und Abbildung 25).

Wie die Analyse verdeutlichen konnte, ist die Reihenfolge der Satzarten nicht eindeutig. Es konnte nicht in jedem Fall eine eindeutige Zuordnung der Satzarten erfolgen, so dass beispielsweise Bonbestandteile keinem Bon eindeutig zugeordnet werden konnten. Ein eindeutiger Decodierungs-Algorithmus ist deshalb nicht möglich.

Als hilfsweise Lösung wurden die nicht vollständig korrekten Bons aussortiert. Absolute Kennzahlen, wie etwa der Gesamtumsatz einer Filiale, sind deshalb nicht auf Grundlage der Journaldateien korrekt zu ermitteln. Doch auch andere Kennzahlen sind aufgrund dessen nur begrenzt aussagefähig.

SA	SA-Nachfolger	Häufigkeit
20	70	1
20	60	7
60	60	9
60	61	43
60	70	98
61	20	5
61	60	49
61	61	6
61	62	3
61	70	90
62	61	1
62	62	1
62	63	2
62	70	8
63	61	1
63	20	1
70	71	5.516
70	80	43
71	72	328
71	73	17.366
72	73	328

SA-Nachfolger	SA	Häufigkeit
73	74	5.688
73	71	12.006
74	74	37
74	71	172
74	75	5.516
75	60	84
75	61	99
75	62	8
75	63	1
75	75	1
75	76	3
75	70	5.321
75	20	1
76	70	2
76	75	1
80	82	40
80	83	3
82	83	40
83	70	39
83	60	1
83	61	3

Tabelle 7: Auswertung der Satzartfolgen einer Journaldatei

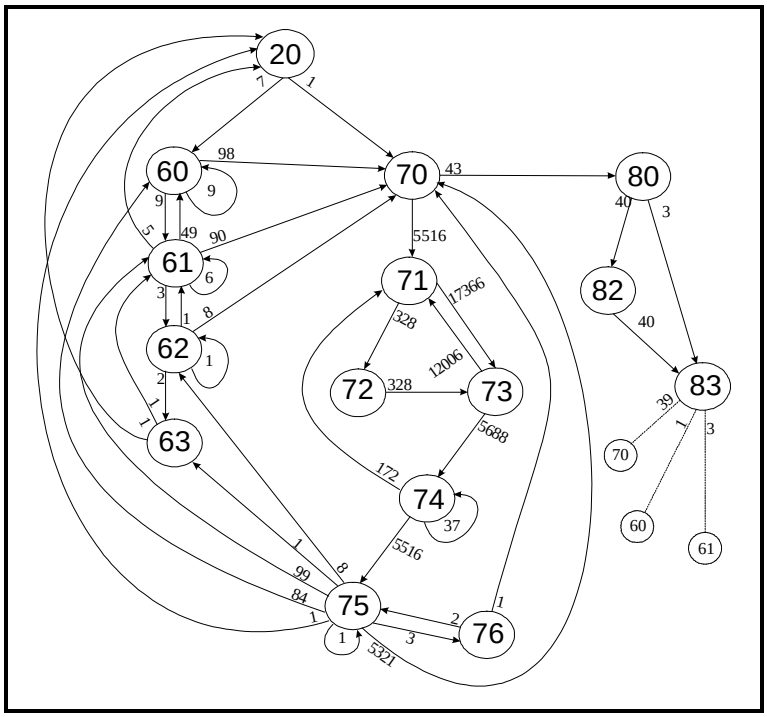


Abbildung 25: Grafische Auswertung der Satzartfolgen einer Journaldatei

5.4 Entwicklung des Domänen-Modells

Der Bedarf für einen Operational Data Store und/oder für einen Operational Object Store hat sich in keinem der Fälle ergeben. Der Fokus der durchgeführten Projekte wie auch der Lehre liegt außerhalb des operativen Bereichs, was dieses Ergebnis zusätzlich erklärt.

Zumindest im Drogeriemarktprojekt wäre kurz- bis mittelfristig die Erstellung eines ODS ein wichtiges Ziel gewesen, da die Daten dann zumindest unter Verwendung relationaler Technologie hätten vorgehalten werden können, was die Analysen mit gängigen MSS-Werkzeugen erlaubt und auch eine Kopplung mit dem Warenwirtschaftssystem ermöglicht hätte, beispielsweise zur Verbesserung des Bestellwesens (z. B. eine Vorschlagsliste für den Einkauf der Filiale).

Das Domänen-Modell für das Drogeriemarktprojekt wurde mit VisualWorks implementiert. Für die Objekt-Speicherung standen die Alternativen Smalltalk-Image und Oracle 7.3 zur Verfügung, wobei die erste Variante bereits aufgrund der anfallenden Datenmenge ausscheidet. Die durch dieses Projekt gewonnenen Erfahrungen führten zur Verlagerung des Untersuchunginteresses bezüglich der Datenhaltung für OMSS von RDBMS auf OBMS.

In der Folge wurden dann Objectivity und GemStone angeschafft und ihre enstprechende Eignung für Folgeprojekte untersucht. Für das Drogeriemarktprojekt

konnte allerdings kein OBMS mehr eingesetzt werden.[178] Deshalb wird die weitere Entwicklung des Domänen-Modells beispielhaft an dem Prototypen des Vertriebsinformationssystems für den Landmaschinenhersteller aufgezeigt (siehe Abbildung 26).

Da auf die Erstellung eines ODS und OOS verzichtet wird, werden die rein operativen Attribute aus dem Datenmodell (siehe Abbildung 23) entfernt. Weiterhin werden redundante Attribute eliminiert.

Die tiefergehende Analyse ergab zusätzliche Veränderungen. Auf die Branchentabelle „tcmcs031“ und diese Tabelle referenzierende Attribute konnte verzichtet werden.

Die Tabellen Auftragsarten „tcmcs042“ und Statistikgruppen „tcmcs044“ sowie entsprechende Referenzen mussten zusätzlich in das Datenmodell aufgenommen werden.

Aufgrund pragmatischer Erwägungen wurden die Auftragspositionen und Bestellpositionen der Historietabellen nach ihrer jeweiligen Auftragsart (gebucht (1), storniert (2), fakturiert (3)) in jeweils eigene Tabellen überführt.

Da aus keiner der drei Service-and-Maintenance-Tabellen („tssma102“, „tssma301“, „tssma310“) die Kosten für die Serviceaufträge zu ermitteln waren, wurden diese durch die Tabelle der Serviceauftragshistorie „tssma501“ ersetzt. Zusätzlich musste die Tabelle der Serviceauftragsarten „tssma303“ eingefügt werden.

Das Attribut tatsächliche Herstellkosten „cpra“ der Tabelle kundenspezifische Artikel „tipcs021“ enthält die Herstellkosten der Maschine inklusive der Materialkosten für Umbauten. Die benötigten Lohnkosten, Maschinenkosten und Gemeinkosten wie auch die Montagekosten, die durch die Umbauten anfallen, sind dort allerdings nicht vorhanden. Deshalb wurden weitere Tabellen benötigt.

Zur Ermittlung der Lohn- und Montagekosten und deren Gemeinkosten wurde die Tabelle Zeiterfassung pro Mitarbeiter „tihra100“ aufgenommen. Von Interesse sind allerdings lediglich jene Datensätze, deren Attribut Projekt „cprj“ einen Wert enthält, d. h. eine Referenz auf ein konkretes Projekt, womit gekennzeichnet wird, dass es sich bei einem solchen Datensatz um eine Zeiterfassung für den Umbau einer Maschine handelt.

Aufgrund der fehlenden Montagekosten, die den Herstellkosten einer Maschine, abhängig von der Maschinenbaureihe, pauschal zugeschlagen wurden, die bis dahin manuell in die Kalkulation eingefügt worden sind, wurde eine zusätzliche Tabelle Montagekosten pro Baureihe erzeugt. Die Montagekosten ergeben sich aus dem Produkt von Montagestunden und Stundensatz.

---

[178] Dass sich die Beschaffung um über ein Jahr verzögerte, ist begründet in der Lizenz- und Firmenpolitik von GemStone und dem damaligen Distributor *Georg Heeg*. [RMK98, S. 252 f.]

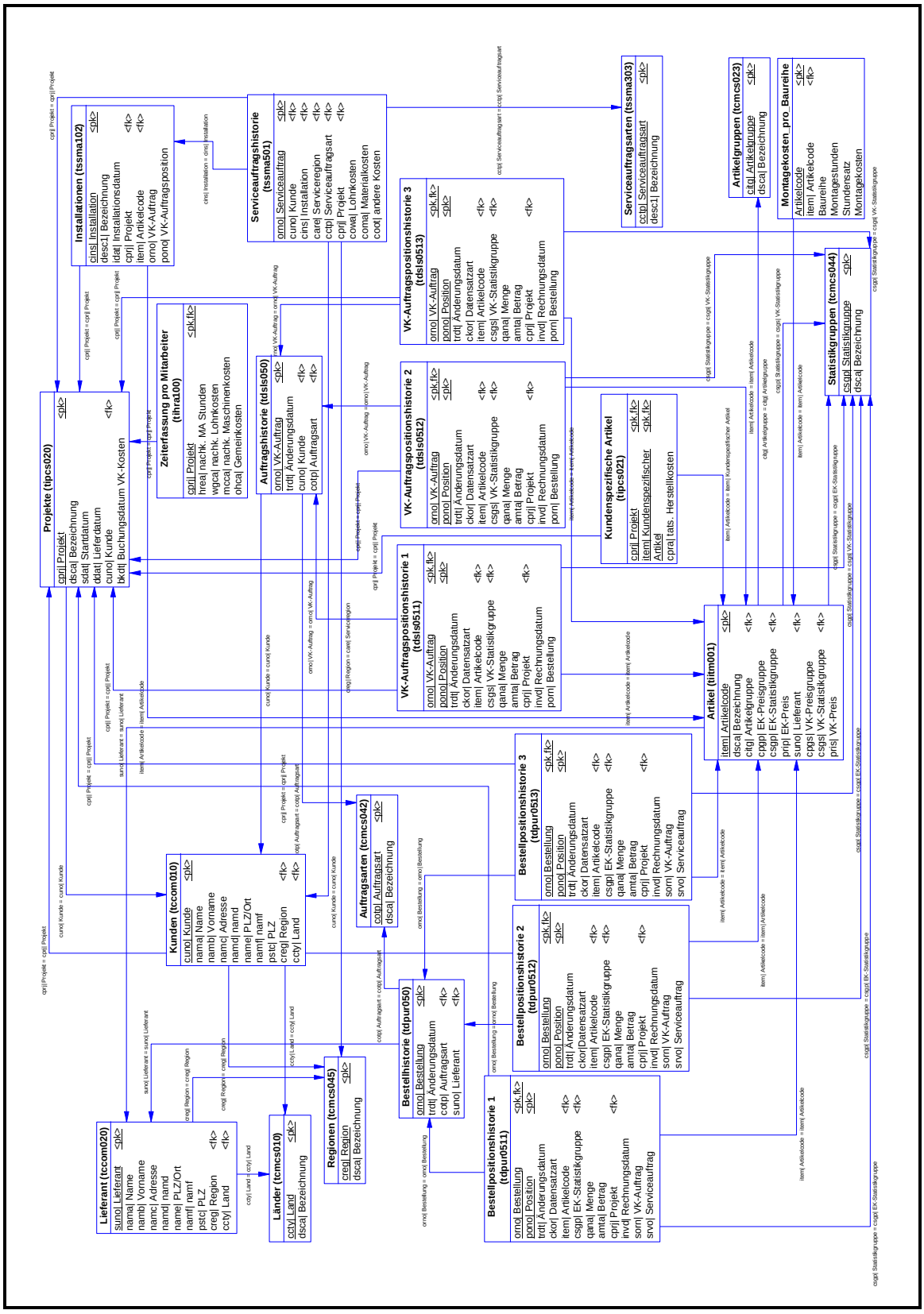


Abbildung 26: Datenmodell in Oracle mit den relevanten Attributen

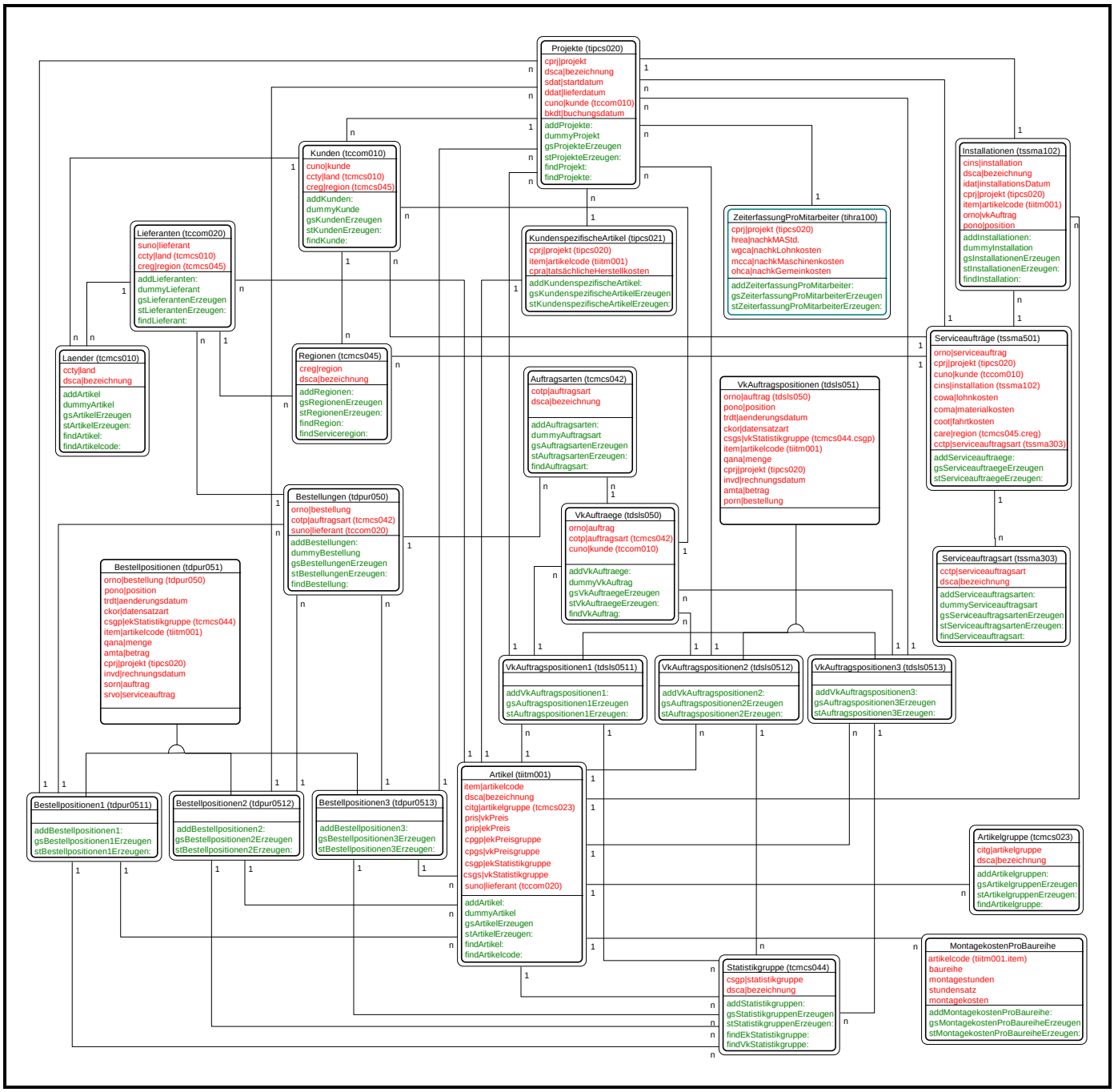


Abbildung 27: Grundmodell des Reconciled Object Layer mit den relevanten Attributen und Methoden

Dieses bereinigte und den MSS-Anforderungen auf dieser Stufe entsprechende relationale Datenmodell wird anschließend als Objektmodell umgesetzt. Da die meisten Bereinigungsprozesse bereits für das Oracle Datenmodell (siehe Abbildung 26) durchgeführt wurden, beziehen sich die Änderungen für das Grundmodell des Reconciled Object Layer im Wesentlichen auf die Bereinigung der Fremd- und Primärschlüssel sowie auf die Klassifikation anhand des Vererbungskonzepts (siehe Abbildung 27).

KAPITEL 6

PROTOTYP-IMPLEMENTIERUNG

Nachdem im letzten Kapitel (siehe Kapitel 5) die Projekte vorgestellt und die Konzeption der Prototypen beschrieben wurde, erfolgt an dieser Stelle die Erläuterung der konkreten Umsetzung. Schwerpunkt der Prototypenentwicklung waren außer der Analyse der Datenquellen, der Ermittlung des Informationsbedarfs und der Entwicklung des Domänen-Modells des Object Warehouse die Transformationsprozesse, mit welchen die MSS-relevanten operativen Daten in die Object-Warehouse-Datenbasis übernommen werden konnten.

Die folgende Diskussion konzentriert sich dabei auf die Entwicklung des Vertriebsinformationssystem-Prototypen in Zusammenarbeit mit dem Landmaschinenhersteller, da der Import der operativen Daten aus einem Standard-ERP, als grundlegende Schicht des Reconciled Object Layer, den Regelfall darstellt.

Nach einem Überblick über die Alternativen bei der Implementierung des Prototypen erfolgt die konkrete Beschreibung des Vorgehens der Protoyp-Erstellung. Dazu werden die Daten aus der Baan-Datenbasis TBase als ASCII-Dateien exportiert, um dann mithilfe des SQL-Loaders in Oracle importiert werden zu können. Anschließend wird ein Objektmodell in VisualWorks implementiert, das die Strukturen des Oracle-Datenmodells umsetzt, um auf diese Weise die Oracle-Daten automatisiert in VisualWorks importieren zu können. Abschließend wird ausführlicher auf die Implementierung des GemStone-Objektmodells eingegangen und die Transformation und Übertragung der Objekte erläutert.

6.1 Vorgehen

Wie in Kapitel 5 erwähnt, wurde als grundlegendes operatives OLTP-System des Landmaschinenherstellers das ERP Baan/Triton 3.1b eingesetzt. Die Datenhal-

tung erfolgte mithilfe des Baan-eigenen Dateiverwaltungssystems TBase[179] . Für die Überführung der Daten aus TBase nach GemStone sind verschiedene Alternativen realisierbar (siehe Abbildung 28).

Der direkteste Weg ist ein ASCII-Export der Daten aus TBase und der Import der resultierenden ASCII-Dateien in GemStone. Eine zweite Möglichkeit, die exportierten ASCII-Dateien nach GemStone zu übertragen, besteht darin, die exportierten ASCII-Dateien mit VisualWorks zu importieren und von dort aus über den GemBuilder nach GemStone zu schreiben. Die dritte Variante besteht in dem Import der ASCII-Dateien mithilfe des SQL-Loaders nach Oracle. Sind die Daten in Oracle, können sie direkt über die GemConnect-Schnittstelle, die GemStone mit Oracle verbindet, nach GemStone übertragen werden oder über die ObjectLens-Schnittstelle, die VisualWorks mit Oracle verbindet, nach VisualWorks und von dort wiederum über die GemBuilder-Schnittstelle nach GemStone.

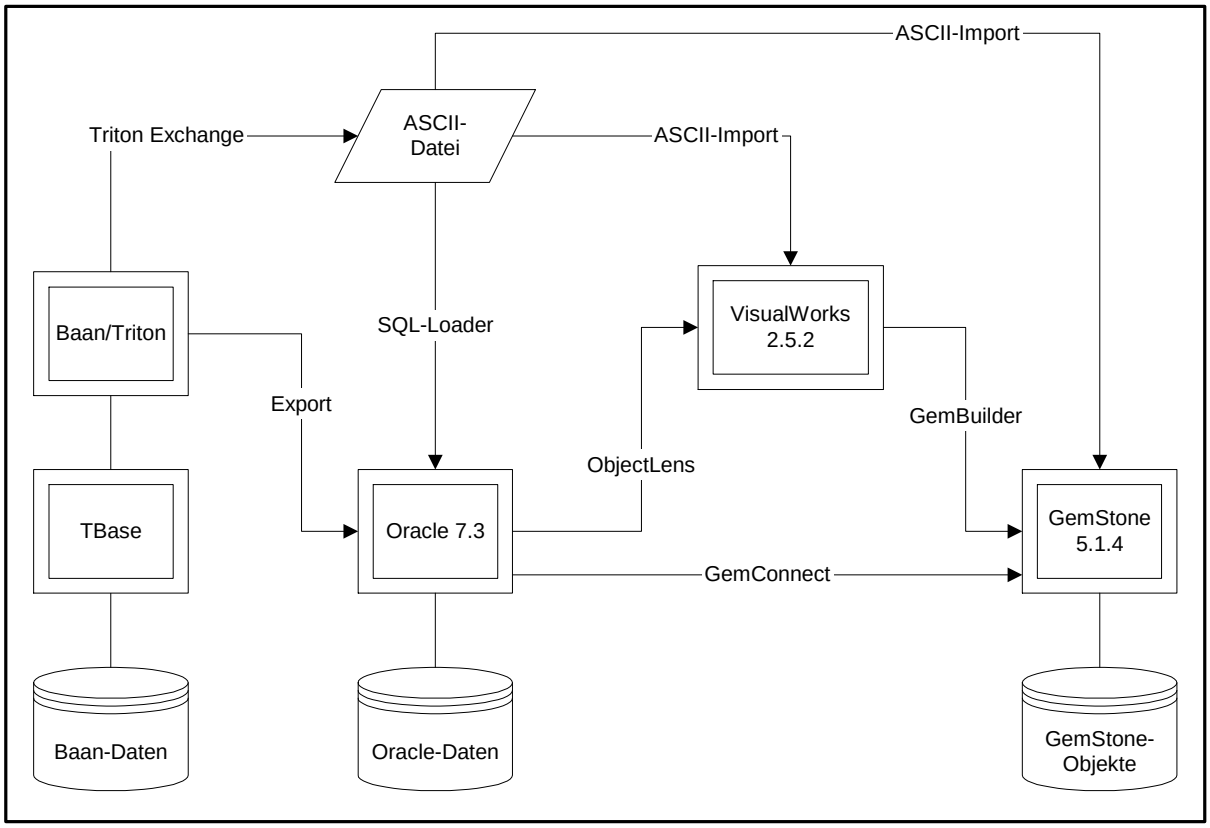


Abbildung 28: Alternative Transformations-Wege der TBase-Daten nach GemStone

[179] TBase setzt grundsätzlich das Relationen-Modell um und verfügt über viele RDBMS-Eigenschaften. Baan definiert TBase folglich auch als RDBMS. Grundlegende Anforderungen eines RDBMS [Date00, S. 33, Voss94] werden allerdings nicht erfüllt.

Eine weitere Möglichkeit besteht darin, die Daten durch eine Baan-Funktion von TBase nach Oracle kopieren zu lassen. Dabei können allerdings nur ganze Tabellen oder Module selektiert werden — die Auswahl von Attributen ist nicht möglich. Der Vorteil ist jedoch, dass bei dieser Methode auch Metadaten mit übertragen werden, wie beispielsweise Feldbeschreibungen und Datentypspezifikationen. Deshalb wurde diese Möglichkeit für die anfängliche Analyse der Daten verwendet, für die regelmäßige Transformation ist dies nicht geeignet: Der Job kann nicht automatisiert angestoßen werden, es ist nur ein vollständiges Laden der Daten möglich und die Beschränkung auf diejenigen Attribute, die auch Verwendung finden, ist nicht realisierbar.

Der direkte Daten-Export von TBase nach Oracle wurde deshalb nicht weiter betrachtet. Da zum damaligen Zeitpunkt bereits eine zukünftige Umstellung der Datenbasis von TBase nach Oracle absehbar erschien, die mit dem nächsten Release-Wechsel zu Baan-ERP (Version 5) erfolgen sollte, wurde auf einen direkten Import der ASCII-Dateien in VisualWorks oder GemStone verzichtet[180]. Stattdessen wurden die Import-Varianten von Oracle nach GemStone untersucht.

Als Übergangslösung bis zur Umstellung auf die neue Baan-Version wurde (deshalb) mithilfe des Moduls Triton Exchange ein ASCII-Export der relevanten Tabellen und Attribute durchgeführt, der dann mit dem SQL-Loader in Oracle geladen werden konnte. Für diese vorübergehende Lösung spricht auch, dass der SQL-Loader ein sehr leistungsstarkes Werkzeug ist, die Export- und Importprozesse mit Shell-Skripten automatisiert werden konnten und die parallele Projektanforderung bestand, auch einen konventionellen Prototypen zu entwickeln, der vom Landmaschinenhersteller praktisch eingesetzt werden können sollte, weswegen die Entscheidung ebenfalls zugunsten des RDBMS Oracle 7.3 als Datenhaltungskomponente, respektive Data Warehouse fiel.

Die Untersuchungsvariante der direkten Datenübertragung von Oracle nach GemStone über GemStones GemConnect-Schnittstelle entfiel, da es nicht gelingen konnte dieses Produkt zur Verfügung gestellt zu bekommen.[181]

6.2 Export der ASCII-Dateien aus TBase

Für den Export der ASCII-Dateien wird von Baan/Triton 3.1b das Modul „Triton Exchange 6.1“ bereitgestellt, mit welchem ein Export-Schema erzeugt werden

[180] Beim Projekt mit der Drogeriemarktkette wurden die ASCII-Daten der Kassen-Journal-Dateien mit VisualWorks eingelesen, transformiert und über die ObjectLens nach Oracle übertragen. GemStone stand für dieses Projekt noch nicht zur Verfügung (siehe Kapitel 5).

[181] GemStone Deutschland ist bis heute nicht bereit, dem Fachgebiet BWL/Wirtschaftsinformatik II der Universität Osnabrück diese Schnittstelle zur Verfügung zu stellen.

kann. Entscheidend sind die Untermodule „Master Data“ zur Definition der Export-Skripte und „Export Module“ für die Ausführung des ASCII-Exports.

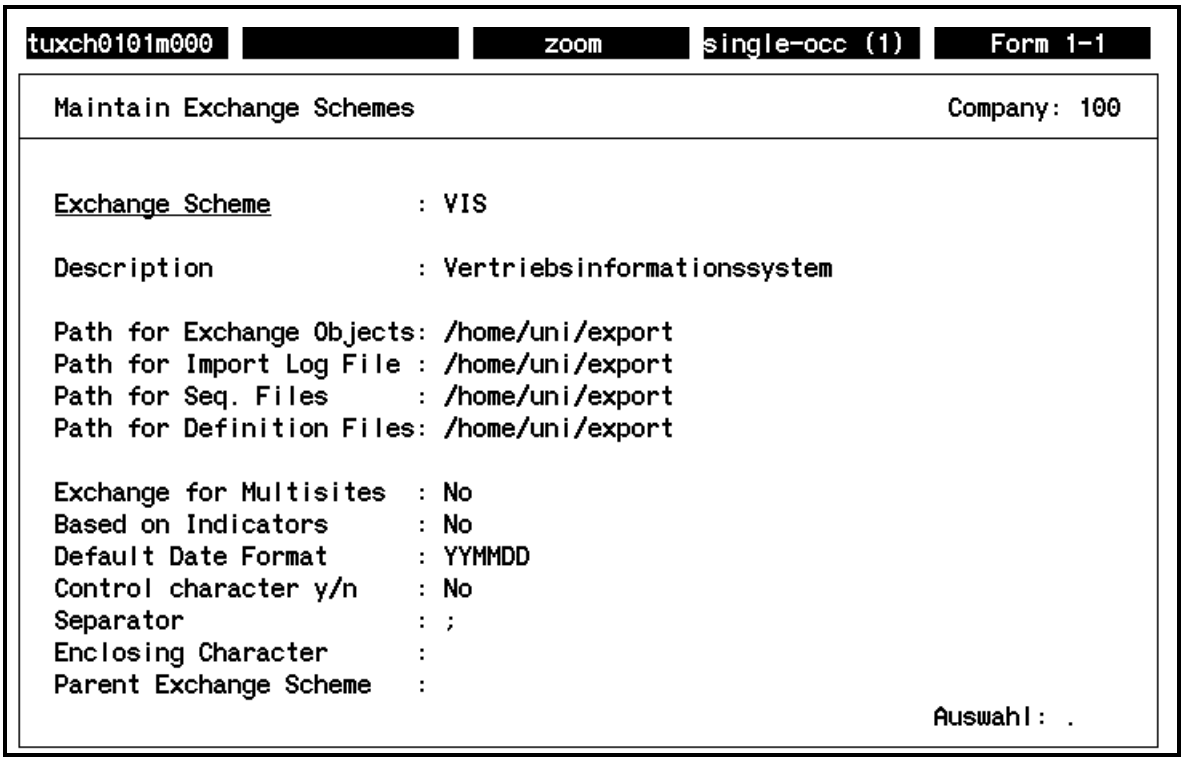


Abbildung 29: Maintain Exchange Schemes

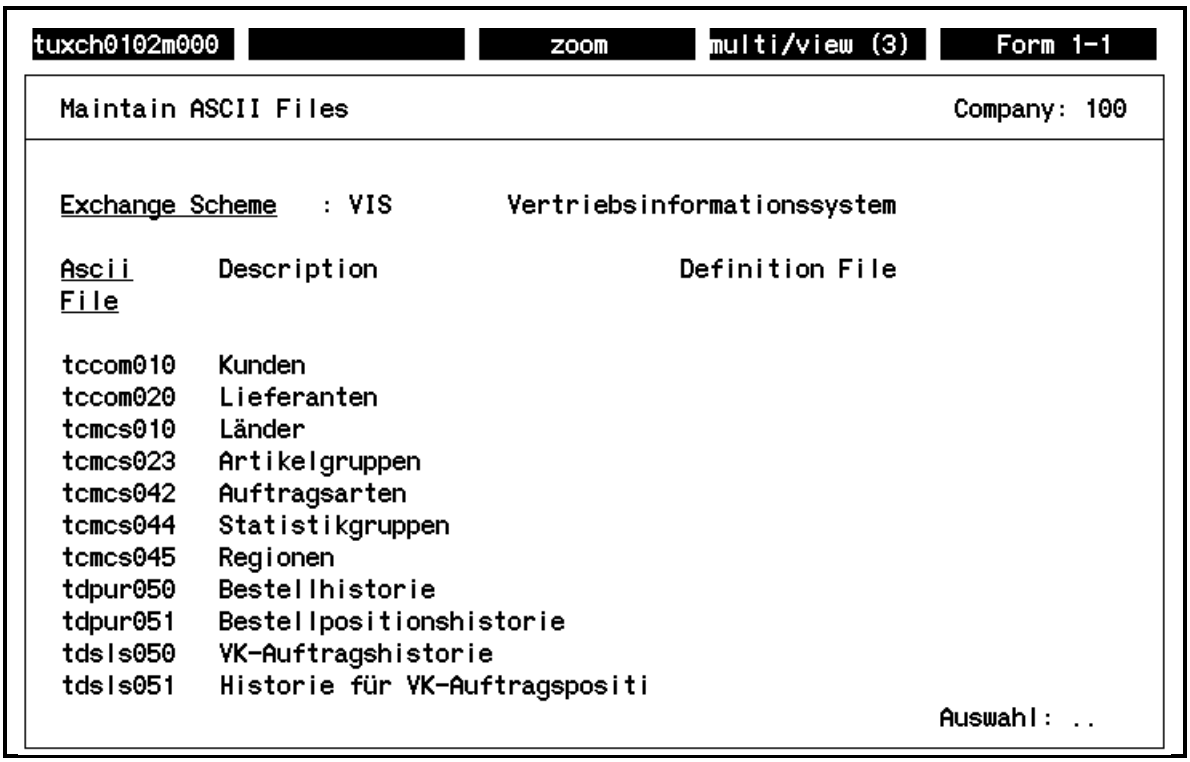


Abbildung 30: Maintain ASCII Files Teil 1

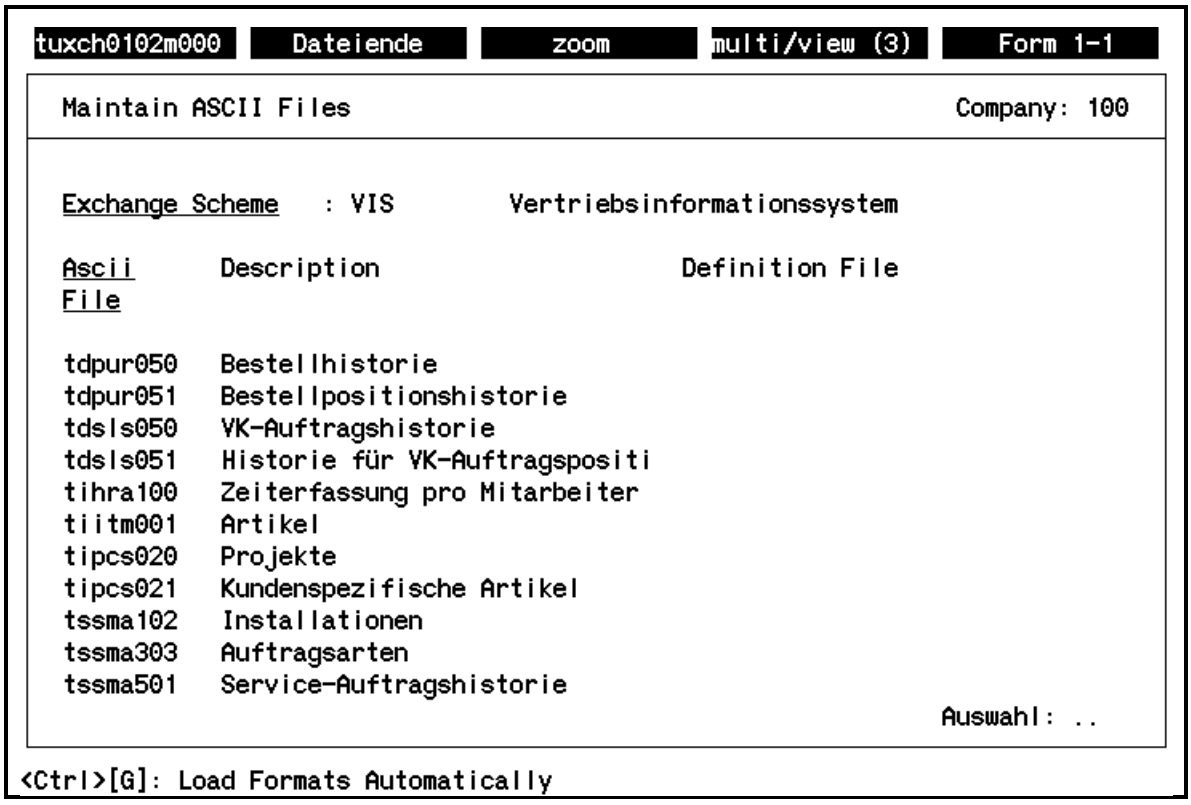


Abbildung 31: Maintain ASCII Files Teil 2

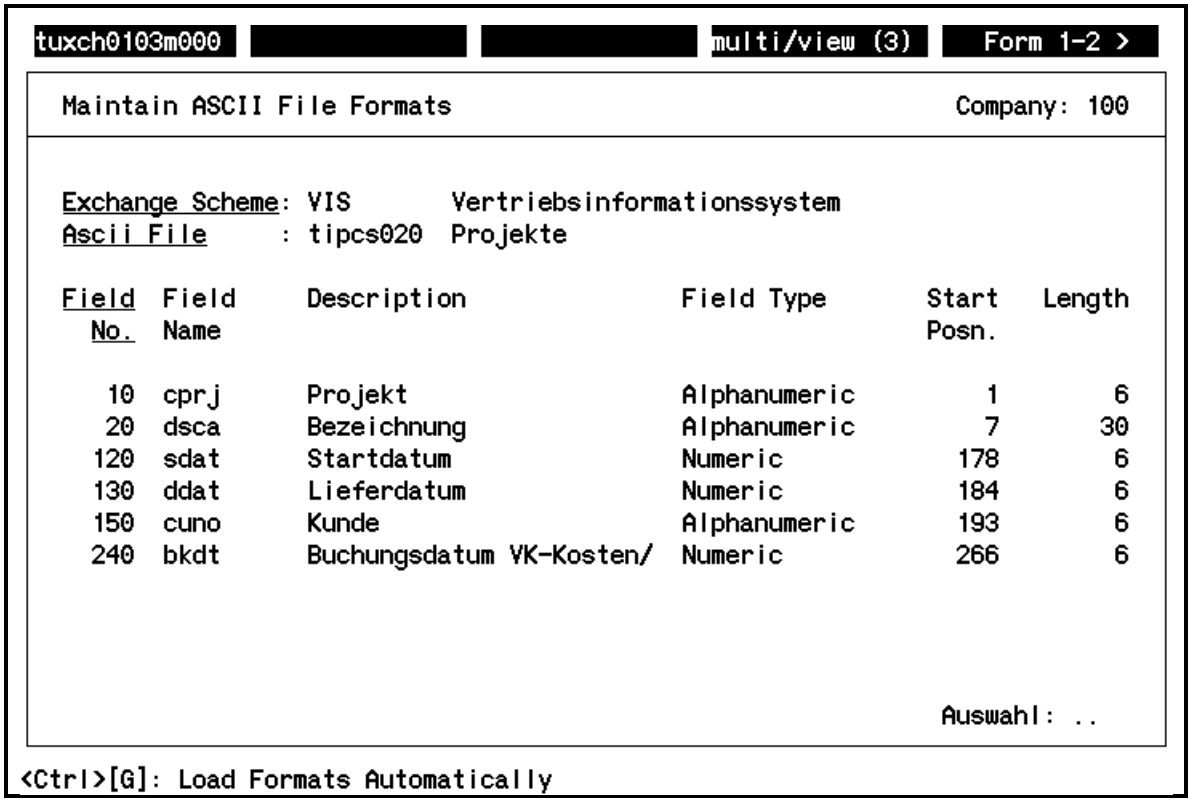


Abbildung 32: Maintain ASCII File Formats

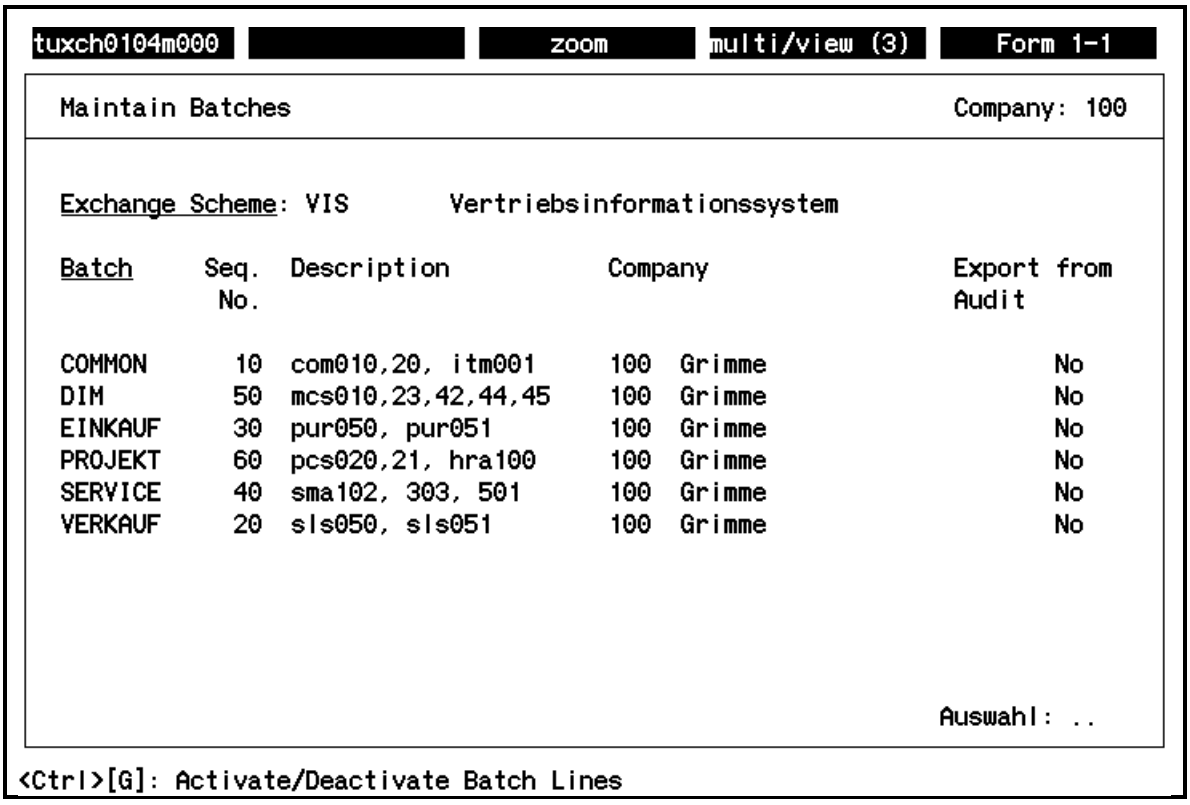


Abbildung 33: Maintain Batches

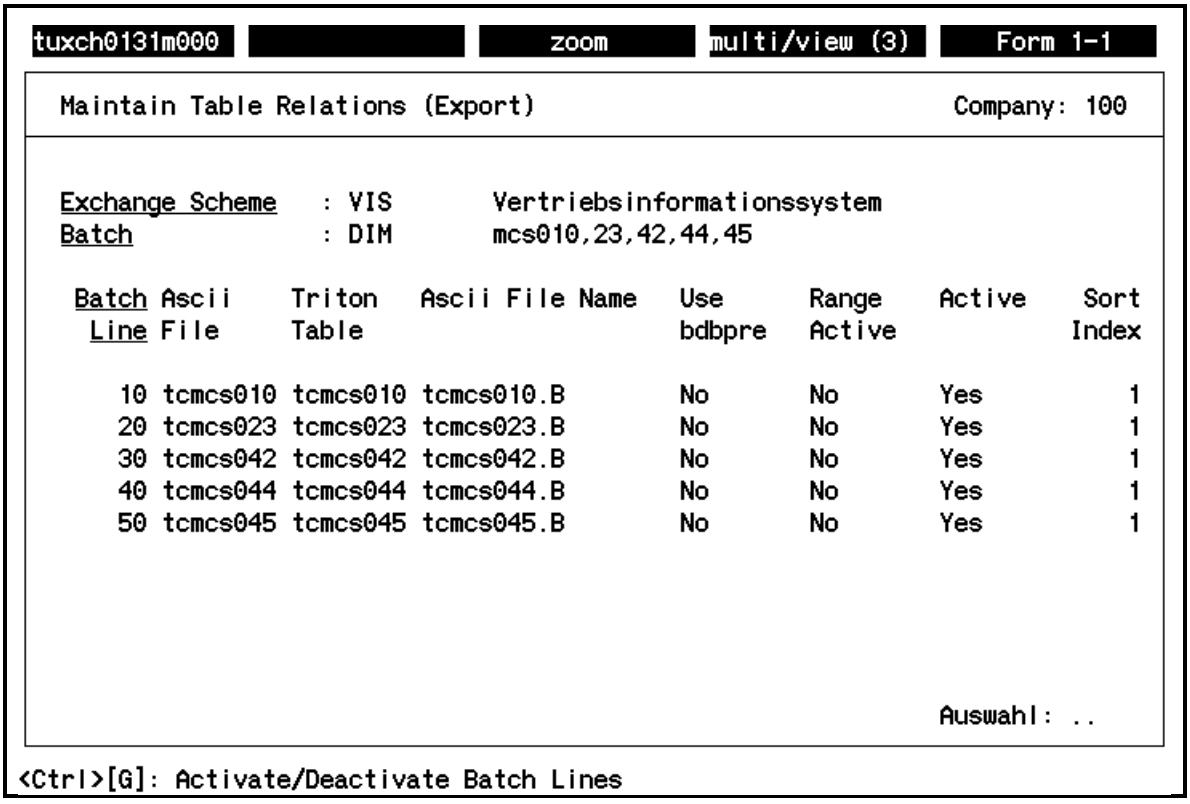


Abbildung 34: Maintain Table Relations (Export) am Beispiel des Batch DIM

Im ersten Schritt wurden die für die Erzeugung des Export-Schemas allgemeinen Angaben wie Trennzeichen, Dateipfade, Zieldatenformat et cetera definiert (siehe Abbildung 29). Anschließend erfolgte die Benennung der ausgewählten ASCII-Dateien (siehe Abbildung 30 und Abbildung 31) und die Auswahl der relevanten Tabellenfelder mit Angabe der Feldnamen, der Feldbeschreibungen, der Typen und der Feldlängen — beispielhaft gezeigt für die Tabelle Projekte „tipcs020“ (siehe Abbildung 32). Der Export der einzelnen Tabellen wurde in unterschiedliche Batches aufgeteilt, im konkreten Fall (siehe Abbildung 33) in die Bereiche Common (Stammdaten), Dim (Dimensiondaten), Projekt (Projekte, kundenspezifische Artikel und Zeiterfassung pro Mitarbeiter), Einkauf (Bestellhistorietabellen), Service (Servicetabellen) und Verkauf (Auftragshistorietabellen).

Anschließend mussten die erstellten ASCII-Export-Dateien und Batch-Export-Dateien mithilfe der Funktion „Maintain Table Relations (Export)“ den Baan-Tabellen zugeordnet werden — gezeigt am Beispiel des Batch DIM (siehe Abbildung 34). Schließlich wurden die Export-Skripte mit der Funktion „Create Export Scripts“ generiert und der Export angestoßen.

6.3 Import der ASCII-Dateien in Oracle

Das Oracle-Datenschema (siehe Kapitel 5) wurde mit SQL-Skripten in Oracles SQL-Plus angelegt. Dabei wurden die Tabellen mit ihren Attributen in Oracle erzeugt und alle notwendigen Constraints definiert. Es wurden zwei unterschiedliche Datenbank-Schemata angelegt. Die aus Baan exportierten ASCII-Dateien wurden mithilfe des Oracle SQL-Loaders in ein temporäres Datenbankschema geladen, da Schemaänderungen bei RDBMS in nicht ausreichendem Maße zu realisieren sind. Im Anschluss wurden die notwendigen Schema- und Datenänderungen vorgenommen. Schließlich wurden die geänderten temporären Tabellen in das Zieldatenschema übernommen.[182]

Nach dem Import der ASCII-Dateien wurde die Tabelle Zeiterfassung pro Mitarbeiter „tihra100“ auf die relevanten Datensätze reduziert. Es wurden alle Tupel, die dasselbe Projekt referenzieren, gruppiert, die Nicht-Schlüssel-Attribute summiert. Die Attribute dieser Tabelle sind nur pro Projekt von Interesse, weshalb auch das Attribut Projekt „cprj“ als Primärschlüssel angelegt werden konnte. Die Gruppierung wurde notwendig, da der Primärschlüssel mehrfach vorkam. Es gab mehrere Tupel die dasselbe Projekt referenzierten, was darin begründet ist, dass

[182] Es sei angemerkt, dass die notwendigen Schritte zum Import der Daten aus den ASCII-Dateien effizienter mit einem ETL-Werkzeug durchgeführt werden hätten können, ein solches Werkzeug zum damaligen Zeitpunkt allerdings noch nicht zur Verfügung stand. Inzwischen können gängige ETL-Werkzeuge beispielsweise direkt auf SAP/R3-Tabellen aufsetzen, die notwendigen Daten- und Datenschema-Änderungen durchführen und in ein Ziel-RDBMS überführen. ETL-Werkzeuge mit Schnittstellen zu OBMS sind nicht verfügbar.

die Ursprungstabelle die Sätze auch pro Mitarbeiter speichert. Dieses Attribut wurde jedoch nicht mit aus Baan übernommen, da es nicht von Interesse ist.

Die Datensätze der Tabelle kundenspezifische Artikel „tipcs021“ wurden auf jene Datensätze reduziert, die auch in der Tabelle Montagekosten pro Baureihe vorhanden sind. Die Tabelle Montagekosten pro Baureihe wird aus einer MS-Excel-Datei importiert bzw. durch ein generiertes Insert-Skript eingefügt.

Die Tabellen Verkaufsauftragshistorie „tdsls050“ und Bestellhistorie „tdpur050“ wurden auf die relevanten Tupel reduziert. Der Primärschlüssel „orno“ war mehrfach vorhanden, weil aus buchführungstechnischen Gründen die stornierten Datensätze vollständig erhalten bleiben mussten, da diese statt gelöscht zu werden, gegen- und neugebucht wurden. Aus diesem Grund hätte der Primärschlüssel um die Attribute Änderungsdatum „trdt“ und Änderungszeit „trtm“ ergänzt werden müssen. Dies ist unerwünscht. Außerdem sind die stornierten Aufträge bzw. Bestellungen nicht von Interesse. Deshalb wurde pro Primärschlüssel „orno“ der Datensatz ausgewählt, der das höchste Änderungsdatum „trdt“ und bei Gleichheit auch des Datums, die höchste Änderungszeit „trtm“ und aufweist. Die anderen Datensätze wurden entfernt.

Dasselbe musste auch für die Bestellpositionshistorie „tdpur051“ und die Historie der Verkaufsauftragspositionen „tdsls051“ erfolgen. Bei diesen Tabellen kam der kombinierte Primärschlüssel aus Bestellung bzw. Verkaufsauftrag „orno“ und Bestellposition bzw. Verkaufsauftragsposition „pono“ mehrfach vor. Es mussten diejenigen Tupel ausgewählt werden, deren Änderungsdatum „trdt“, Änderungszeit „trtm“ und Folgenummer „sern“ pro Primärschlüsselkombination am höchsten sind. Die anderen Datensätze wurden gelöscht.

6.4 Import der Oracle-Tabellen in VisualWorks

Für den Import der Daten aus den Oracle-Tabellen wurde eine Verbindung zwischen den relevanten Oracle-Tabellen und VisualWorks-Klassen hergestellt. Die einzelnen Tupel wurden dann in Smalltalk-Instanzen übertragen.

Zu diesem Zweck wurden mithilfe der VisualWorks-ObjectLens für Oracle Mappings zwischen dem Oracle-Datenmodell und einem Klassenmodell in VisualWorks erstellt.

Als erstes wurde in VisualWorks eine neue Datenbank-Applikation (LensMainApplication) erstellt. Anschließend konnten dann über eine Oracle-Session die Tabellen des Nutzers („Roder“) selektiert werden (siehe Abbildung 35).

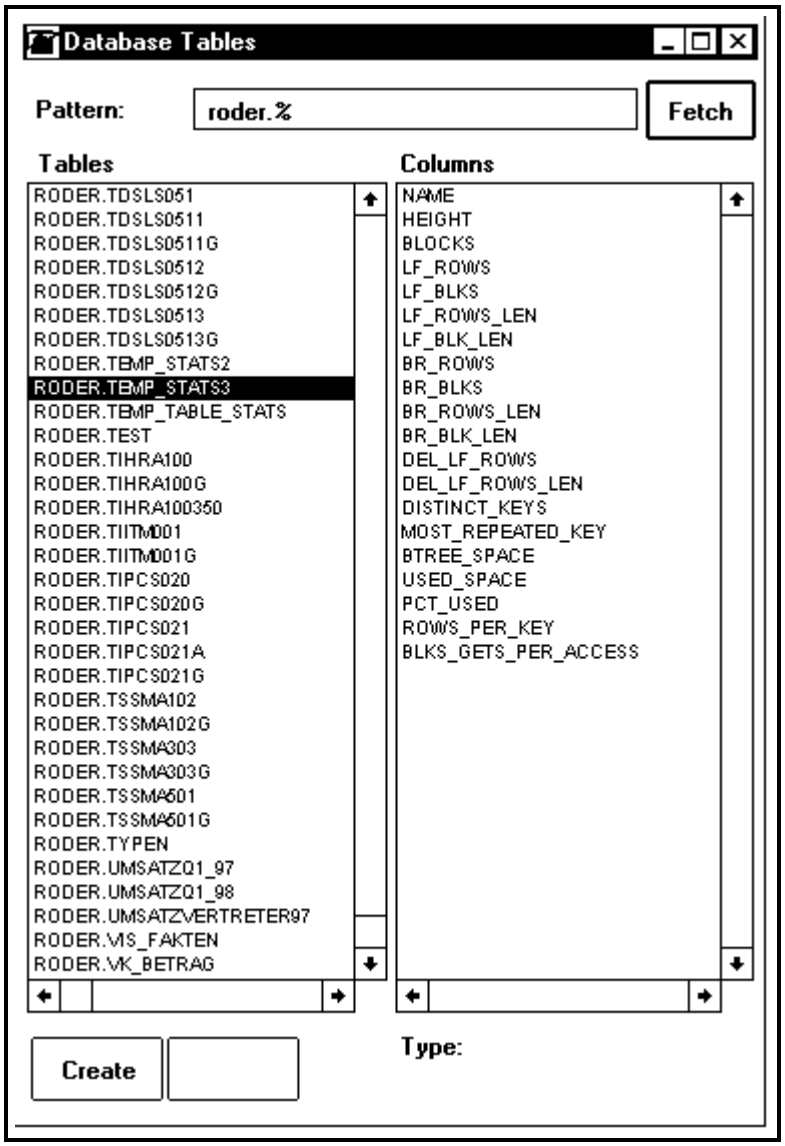


Abbildung 35: Generierung der Klassendefinition aus Oracle-Tabellen

Aus jeder Tabellendefinition kann eine Klassendefinition generiert werden, was für die Tabellen des Oracle-Modells durchgeführt wurde. Danach wurden dem erzeugten Objektmodell die Referenzen hinzugefügt (siehe Abbildung 36). Dabei wurden die Fremd- und Primärschlüssel-Constraints aus Oracle ausgelesen und in Instanzverbindungen (Assoziationen) der Smalltalk-Objekte umgesetzt.

Aufgrund des Impedance Mismatch zwischen Smalltalk und Oracle bzw. OOPL und RDBMS müssen die Typ-Mappings zwischen den Instanzvariablen und den Attributen definiert werden (siehe Abbildung 37). Anstelle des Typs „VARCHAR-2“ in Oracle kann besipielsweise die Klasse (Typ) „String“ in Smalltalk verwendet werden. Die ObjectLens übernimmt die jeweilige Konvertierung der Datentypen.

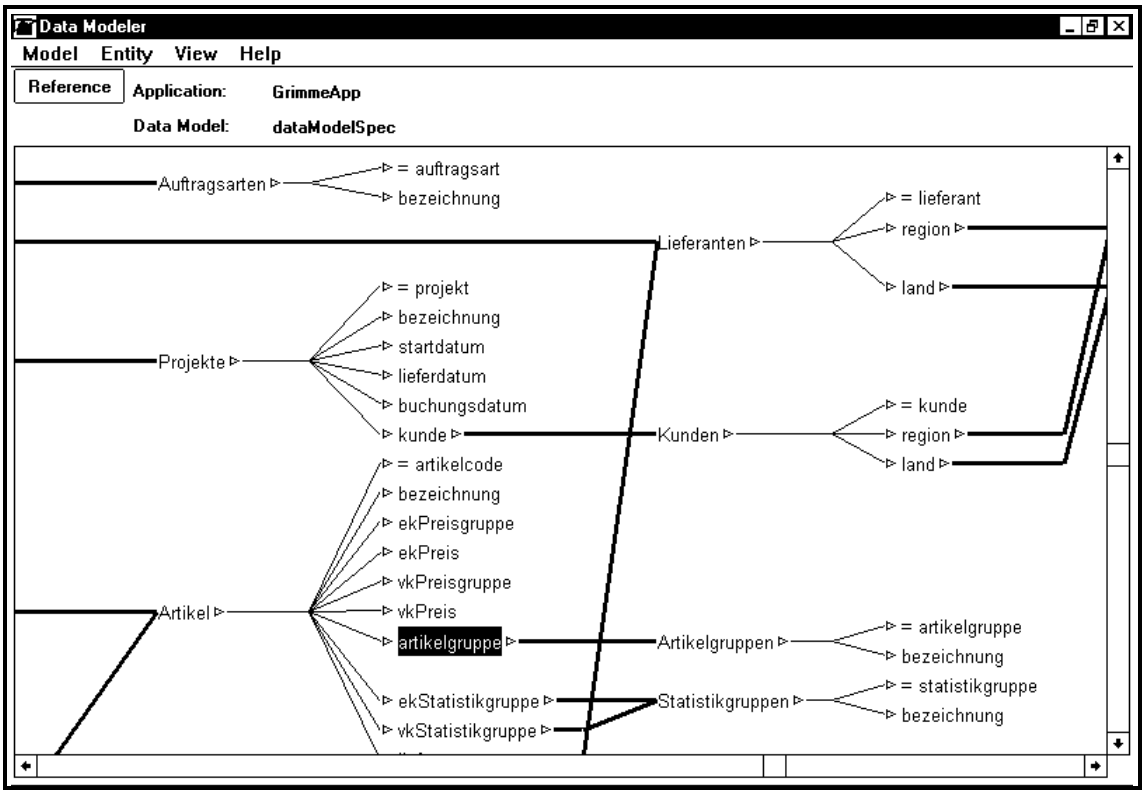


Abbildung 36: Aus dem Oracle-Datenmodell generiertes Smalltalk-Objektmodell (Ausschnitt)

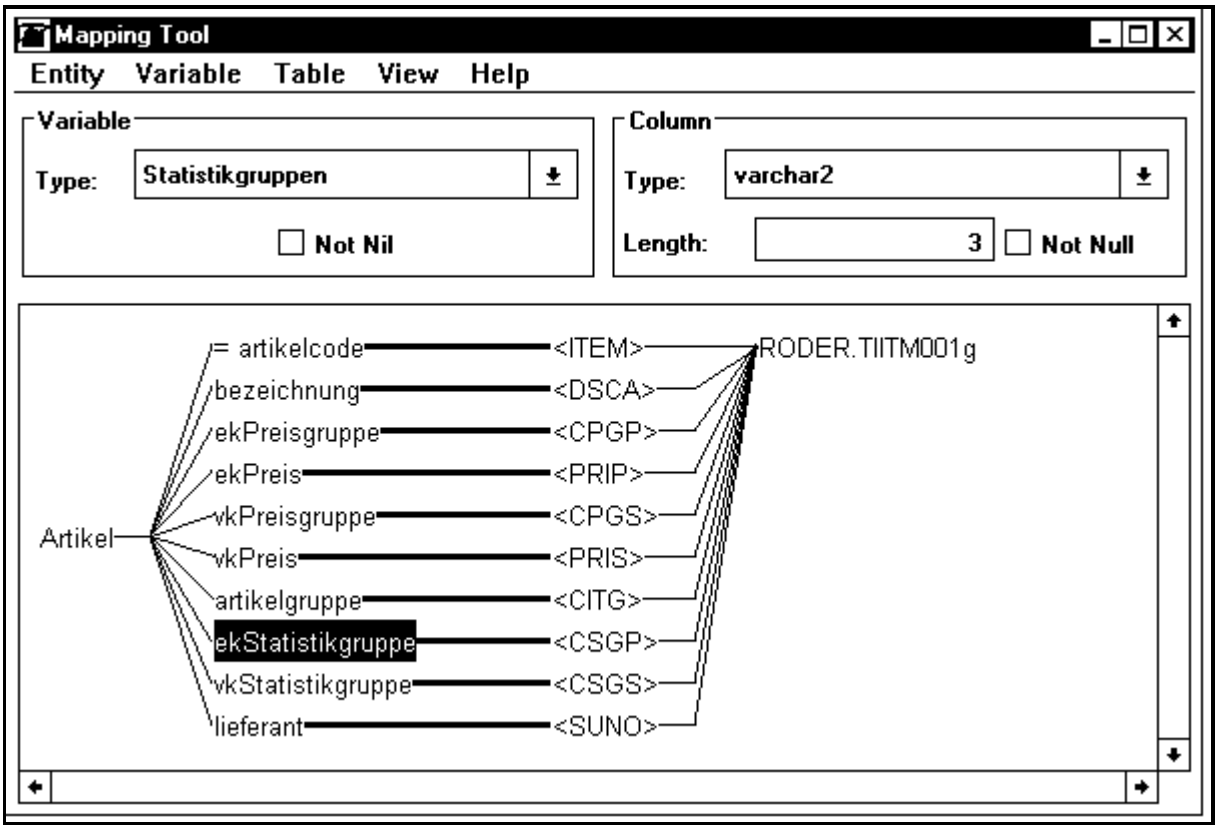


Abbildung 37: Mapping zwischen einer Oracle-Tabelle und der dazugeörigen Smalltalk-Klasse

Es ist aber auch möglich anstelle von einfachen Werten direkte Referenzen abzubilden (siehe Beispiel in Abbildung 37), die dann in Fremd/Primärschlüsselreferenzen umgesetzt werden. Mehrwertige (n:1)[183] und binäre (Inverse) Verzeigerungen sind allerdings nicht möglich, da sie in Oracle nicht abgebildet werden können.

Es müssen nicht alle Attribute zu Instanzvariablen und nicht alle Instanzvariablen zu den Attributen gemappt sein. An dieser Stelle wurden auch die Instanzvariablen umbenannt, damit diese bezeichnendere Namen bekommen konnten (siehe Abbildung 37).

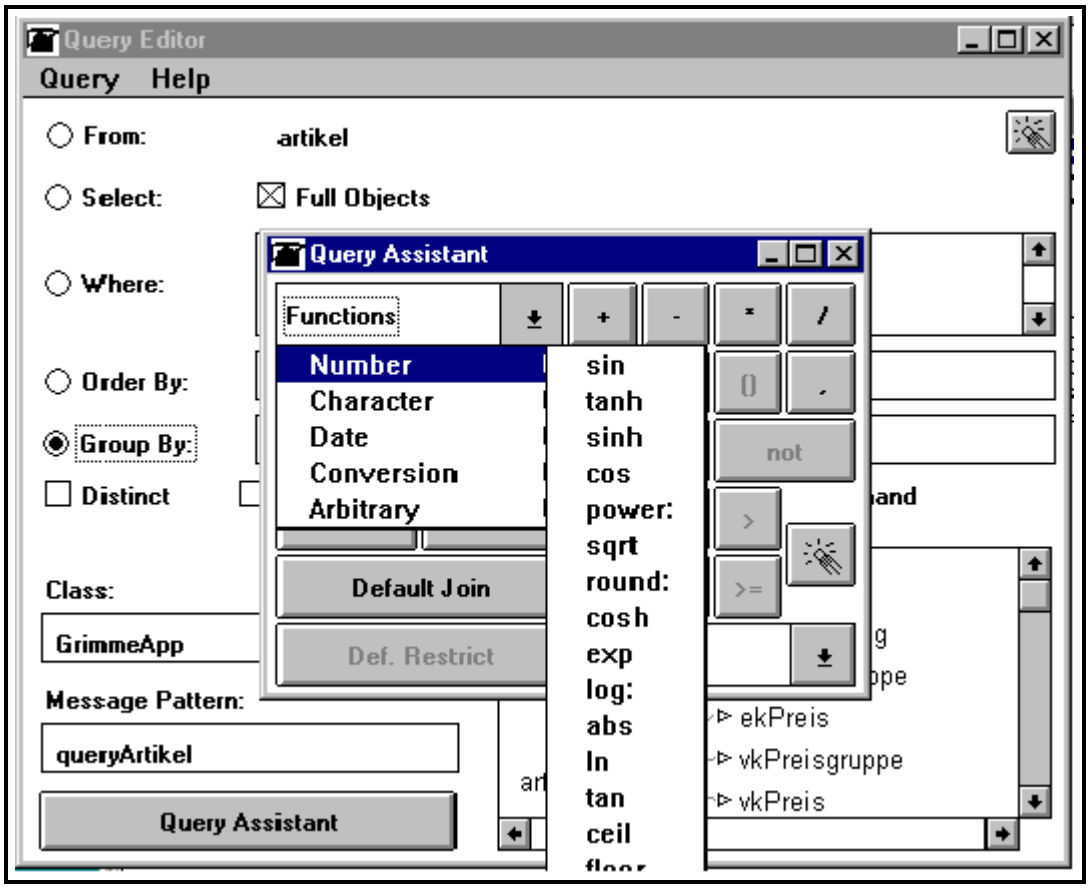


Abbildung 38: Generierung einer Oracle-Query

Die Definition des Objektmodells, das die Mappings zwischen Oracle-Datenmodell und Smalltalk Objektmodell beinhaltet, wurde als Klassenmethode in der „lensMainApplication“[184] gespeichert (dataModelSpec).

---

[183] Mehrwertig bedeutet, dass ein Objekt mehrere andere Objekte referenziert. Dies widerspricht im Relationenmodell der ersten Normalform, wird hingegen in Smalltalk über Mengen oder Listen verwirklicht.

[184] Im Folgenden werden analog zur Syntaxdefinition von Smalltalk alle Instanzen, Instanzvariablen und Methoden mit einem kleinen Anfangsbuchstaben geschrieben, sind dagegen Klassen gemeint, beginnen diese mit ei-

Mithilfe der implementierten Applikation (lensMainApplication) war es nun möglich, Anfragen an Oracle zu stellen. Dazu kann der Query Editor verwendet werden (siehe Abbildung 38). Bei komplexen Anfragen werden die Ergebnisse als Arrays zurückgegeben. Werden vollständige Instanzen selektiert, also Instanzen zu den implementierten Klassen der Applikation, werden aus den Tupeln der jeweiligen Tabelle Instanzen erzeugt. Die Fremdschlüssel werden als Instanzverbindungen umgesetzt. Die Referenzen sind allerdings nur für diejenigen Instanzen korrekt umsetzbar, die in derselben Oracle Session erzeugt werden.

Eine einfache Möglichkeit besteht darin, für jede Klasse bzw. Tabelle eine Query zu generieren, die alle Datensätze einliest. Diese Möglichkeit wurde genutzt. Anschließend wurden in einer Session alle Queries ausgeführt und die Ergebnis-Collections persistent gemacht. Dazu wurden die Ergebnis-Collections in ein Dictionary eingefügt, welches wiederum durch eine Instanzvariable der Applikations-Instanz referenziert wird. Solange die Applikation geöffnet ist, bleiben auch die eingelesenen Instanzen aus Oracle persistent.

6.5 GemStone-Objektmodell

Nachdem die Oracle-Daten in Smalltalk-Objekte (Client-Objekte) konvertiert worden sind, mussten diese Objekte in Smalltalk-Objekte des Ziel-Objektmodells transformiert werden, d. h. in das GemStone-Objektmodell (GemStone-Objekte), das als Grundlage des Reconciled Object Layers dient. Dazu musste zunächst das Ziel-Objektmodell in GemStone implementiert werden.

Analog zum üblichen Vorgehen in VisualWorks wurden die Klassen im Klassen-Browser angelegt (siehe Abbildung 39). Alternativ wurde das Objektmodell auch mit dem CASE-Tool VisualDesigner von ArsNova erzeugt, welches dann die Klassen mit Methodenrümpfen, Attributen, Zugriffsmethoden und Beziehungen, inklusive des Codes für die Verwaltung der Inversen-Semantik, generiert. Obwohl im Zusammenhang mit VisualWorks und der ObjectLens bzw. Oracle zufriedenstellende Ergebnisse erzielt werden konnten, ist diese Methode für GemStone nicht geeignet, da ein erheblicher Anteil der für den VisualDesigner benötigten Klassen und Methoden nur in VisualWorks, nicht dagegen in GemStone vorhanden sind. Außerdem werden die Besonderheiten von GemStone vom VisualDesigner missachtet. Mithin wurden die Klassen manuell implementiert.

Im Unterschied zu VisualWorks werden in GemStone für alle Attribute Constraints definiert. Dies erhöht die Typsicherheit und ermöglicht eine erhöhte Effizienz bei der Optimierung durch das OBMS. Constraints müssen allerdings

nem großen Buchstaben. Großbuchstaben im Wort werden in Smalltalk zur besseren Lesbarkeit verwendet und bilden die Alternative zu Binde- und Unterstichen.

nicht angegeben werden, so dass die Smalltalk-Flexibilität erhalten bleibt. Definierte Constraints gelten für die bezeichnete Klasse und alle Unterklassen. Wird beispielsweise ein Constraint „Integer“ für eine Instanzvariable definiert, sind sowohl Instanzen von Integer, wie auch von den Unterklassen „SmallInteger“, „LargePositiveInteger“ und „LargeNegativeInteger“ zulässig. Es wurden für alle Attribute Constraints definiert.

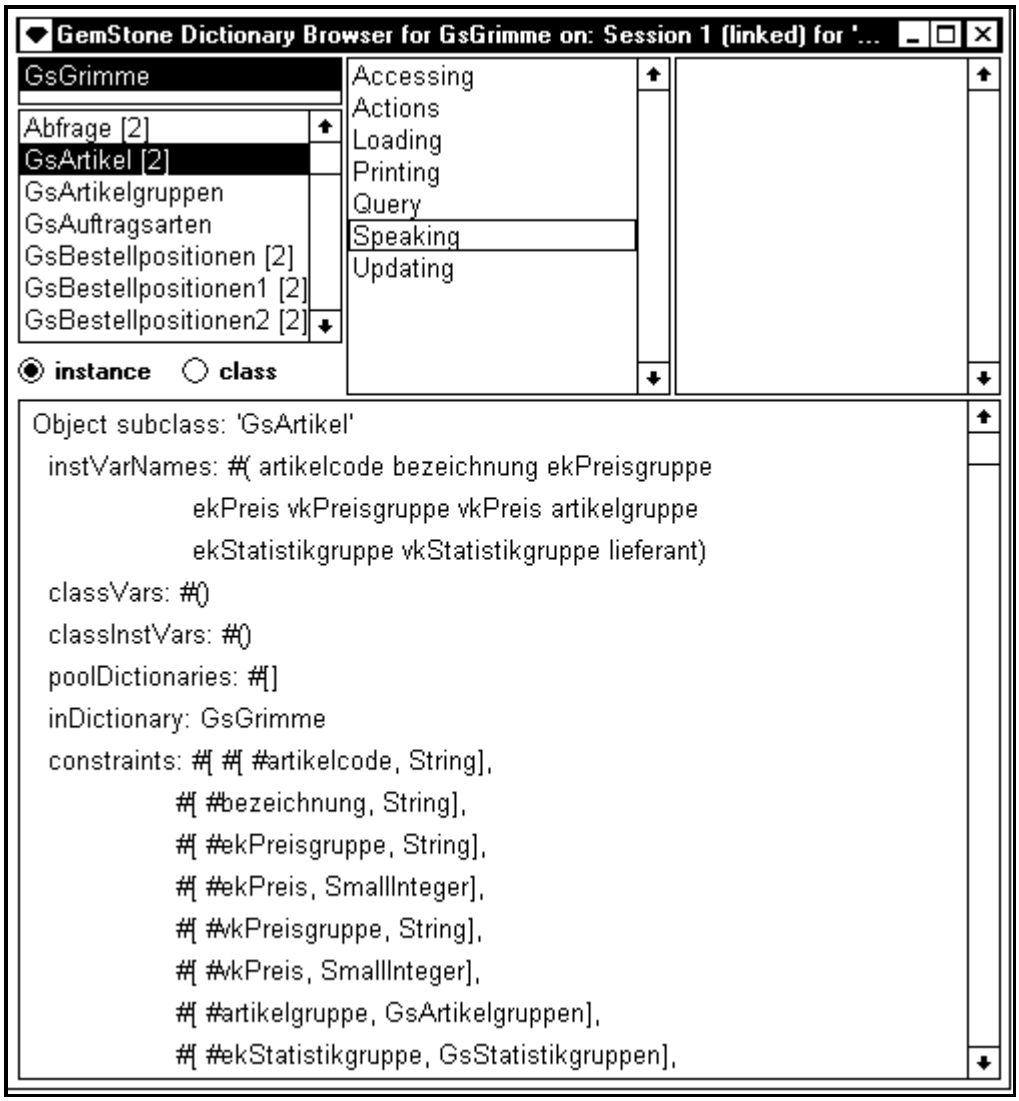


Abbildung 39: Klassendefinition des Objektmodells in GemStone

Aufgrund der besonderen Bedeutung von Mengen in OBMS, also der Betrachtung und Verarbeitung von Instanz-Mengen im Unterschied zur schwerpunktmäßigen Betrachtung von einzelnen Instanzen wie in OOPL, kommt der Menge („Set“, „IdentitySet“) eine zentrale Rolle zu.

Besonders hervorzuheben ist, dass für mengenwertige Attribute in den meisten Fällen eigene Klassen definiert wurden. Zu diesem Zweck sind Instanzen der

Klasse „IdentitySet“ verwendet worden, da diese besonders effizient sind und in OBMS, wie auch im vorliegenden Fall, meist nur einfache Mengen benötigt werden, also weder die Reihenfolge von Interesse ist, noch die mehrfache Speicherung von identischen Objekten. Im Client-Smalltalk werden dagegen für gewöhnlich Instanzen der Klasse „OrderedCollection“ genutzt. Mithin wurden jeweils Unterklassen von „IdentitySet“ gebildet und dann mit Constraints der jeweiligen Klasse versehen.

Die „bestellung“ hat beispielsweise mehrere „bestellPositionen“, weshalb die Klasse „Bestellung“ über eine Instanzvariable „bestellPositionen“ verfügt. Es wurde mithin eine Unterklasse „SetOfBestellPositionen“ von der Klasse „IdentitySet“ erzeugt und darauf der Constraint „all elements of 'bestellPosition'“[185] definiert. Das Attribut „bestellPositionen“ der Klasse „Bestellung“ erhielt deshalb den Constraint „SetOfBestellPositionen“.

Diese „IdentitySets“ mit den darauf definierten Constraints bilden einen zentralen Bestandteil des Objektmodells des OBMS, da sie auch für die Speicherung der Klassen-Extensionen, also der Instanzen einer Klasse verwendet werden.[186] Beispielsweise wurde eine weitere Instanz von „SetOfBestellPositionen“ erzeugt, in die dann alle Instanzen von „BestellPosition“ aufgenommen werden konnten. Effiziente Zugriffe auf die Instanzen der „Sets“ wurden durch die Kombination der Definition von Constraints und der Definition von Indizes möglich. Die Indizes wurden nach dem Einfügen der Instanzen definiert.

Nachdem die Klassen, Attribute, Accessing-Methoden und Constraints definiert wurden, mussten die Instanz-Beziehungen (Assoziationen[187]) implementiert werden. Beziehungen werden über Instanzvariablen realisiert. Es wurden ausschließlich binäre Referenzen implementiert, da dies dem ODMG-Standard entspricht und die Möglichkeit eröffnet, den Objektpfad von beiden Richtungen zu traversieren[188]. Beispielsweise enthält die „bestellPosition“ eine Instanzvariable „bestellung“, in welchem die zugehörige „bestellung“ referenziert wird, die „be-

---

[185] `IdentitySet subclass: 'SetOfBestellPositionen'`
`constraints: #["the elements" Bestellung]`

[186] Durch die Definition der Constraints auf den Sets können keine Instanzen von Klassen in die Sets aufgenommen werden, die nicht zur Klasse des Constraints gehören oder Unterklasse davon sind. Es ist aber nicht notwendig wie beim Extensionskonzept, dass alle Instanzen einer Klasse in einem Set sein müssen. Für Instanzen, auf die häufig zugegriffen wird, werden deshalb zusätzlich Sets mit nur diesen Instanzen definiert und indiziert, wodurch der Zugriff erheblich schneller wird.

[187] Für die Aggregation stehen in GemStone keine gesonderten Programmierkonstrukte zur Verfügung, weshalb alle Instanzbeziehungen grundsätzlich identisch implementiert werden.

[188] Es sind dann folgende beiden Pfade möglich:

1. „`bestellung.bestellPositionen.*.menge`“ (wobei der * den flatten-Operator realisiert)
2. „`bestellPosition.bestellung.lieferant`“

stellung“ verfügt über eine Instanzvariable „bestellPositionen“, in welcher alle dazugehörigen „bestellPositionen“ referenziert werden.[189]

Entscheidend ist, dass die definierten Klassen-Constraints die Referenzierung von Objekten verbieten, die außerhalb des Definitionsbereichs der jeweiligen Constraints liegen, nicht jedoch eine Referenzierung erzwingen. Dies bedeutet, dass auch „nil“ referenziert werden kann, also kein Objekt. Deshalb mussten, um Typfehler zu vermeiden, neue Instanzen initialisiert und die Löschung einer Referenz auf eine Instanz eingeschränkt werden.

Eine „bestellung“ muss zum Beispiel immer auch „bestellPositionen“ enthalten. Damit übehaupt „bestellPositionen“ eingefügt werden konnten, musste jedoch zuerst eine Instanz von „SetOfBestellPositionen“ in die Instanzvariable der neuen Instanz von „Bestellung“ eingefügt werden. Der Klassen-Constraint verlangt also eine Instanz von „SetOfBestellPositionen“ in der Instanzvariable „bestellPositionen“, die tatsächliche Initialisierung erfolgt dadurch allerdings nicht und musste deshalb durch eine eigene Methode (z. B. „initialize“) erfolgen. GemStone bietet nicht die Möglichkeit solche Strukturen zu definieren, wie dies in anderen OBMS der Fall ist. Deshalb wurde die Definition von „initialize“-Methoden für alle Klassen umgesetzt.

Die deklarative Definition von Inversen-Constraints ist in GemStone bisher nicht implementiert. Inversen-Constraints garantieren die binäre Konsistenz der Referenzen. Da Inverse nur über zwei unabhängige Zeiger implementiert werden können und auch keine Verwaltung durch das OBMS verfügbar ist, musste diese Semantik in die Zugriffsmethoden implementiert werden. Auch deshalb muss in der Folge sichergestellt werden, dass zumindest schreibende Zugriffe auf Instanzvariablen, die Beziehungen realisieren, ausschließlich über diese Zugriffsmethoden erfolgen.

Wird beispielsweise eine Referenz einer „bestellung“ auf eine „bestellPosition“ erzeugt, d. h. die „bestellung“ enthält eine „bestellPosition“[190], muss gleichzeitig auch eine Referenz der Instanzvariablen „bestellung“ der „bestellPosition“ auf eben jene „bestellung“ erzeugt werden. Wird einer „bestellPosition“ eine neue „bestellung“ zugeordnet, muss außer der Referenz von „bestellPosition“ zu der neuen „bestellung“ und der Referenz von der neuen „bestellung“ zur „bestellPosition“ zusätzlich dafür gesorgt werden, dass die „bestellPosition“ nicht weiter von der alten „bestellung“ referenziert wird.

---

[189] Implementiert werden diese mengenwertigen Referenzen (wie im obigen Beispiel beschrieben) durch spezielle Sets, hier mithin einer Instanz der Klasse „SetOfBestellPositionen“, die die Referenzen auf die dazugehörigen Bestellpositionen enthält und selber von der Instanzvariable „bestellPositionen“ der „Bestellung“ referenziert wird.

[190] Genauer: Die Instanzvariable „bestellPositionen“ der „bestellung“ enthält ein „setOfBestellPositionen“, das wiederum eine „bestellPosition“ enthält.

ODMG-konform werden Instanzvariablen, die einfache Instanzen referenzieren, wie etwa Instanzen von Integer, String oder Boolean, von Instanzvariablen unterschieden, die komplexe Objekte referenzieren, also Beziehungen darstellen (siehe oben). Erstere realisieren einfache Referenzen, letztere binäre Referenzen.

Für den Prototypen wurden lediglich zwei Kardinalitätsausprägungen umgesetzt, nämlich mehrwertige und einfache Referenzen. Beschränkungen auf eine Kardinalität von zum Beispiel mindestens zwei bis höchstens sieben, sind entsprechend dem ODMG-Standard nicht realisiert worden.[191] Zur Bennenung der öffentlichen Accessing-Methoden wurden ebenfalls die Vorschläge der ODMG übernommen.

Für die einfachen Instanzvariablen wurden die bei Smalltalk üblichen einfachen Accessing-Methoden realisiert, wobei diese teilweise um zusätzlich notwendige operationale Constraints erweitert wurden:

- „attributName“ (z. B. „bezeichnung“) für den lesenden Zugriff
- „attributName: neuesObjekt“ (z. B. „bezeichnung: #Reihenfreileger“) für den schreibenden Zugriff

Für den Zugriff auf Instanzvariablen, die Beziehungen realisieren, wurden folgende Zugriffsmethoden verwendet[192]:

- Beim Zugriff auf einwertige Attribute
  - ◊ „attributName“ (z. B. „bestellung“) für den lesenden Zugriff
  - ◊ „formAttributName: neuesObjekt“ (z. B. „formBestellung: bestellung“), um der Instanzvariablen ein neues Objekt zuzuweisen und die Referenz des bisher referenzierten Objekts beidseitig zu löschen
  - ◊ „dropAttributName: zuLoeschendesObjekt“ (z. B. dropBestellung: bestellung“), um das referenzierte Objekt der Instanzvariablen und die Rückreferenz zu löschen
- Beim Zugriff auf mehrwertige Attribute
  - ◊ „attributName“ (z. B. „bestellPositionen“), um die Menge der durch die Instanzvariable referenzierten Objekte (z. B. „bestellPositionen“) zu bekommen, d. h. das referenzierte „IdentitySet“ (z. B. „setOfBestellPositionen“)[193]

---

[191] Diese zusätzliche Einschränkung sollte allerdings berücksichtigt werden können und ist in der Konezption des folgenden zweiten Entwurfsmusters berücksichtigt.

[192] Die sonst üblichen Accessing-Methoden wie „attributName: objekt“ (z. B. „bestellung: bestellung“) wurden redefiniert mit „`^self shouldNotImplement`“ und erzeugen bei der Verwendung eine Fehlermeldung.

[193] Auf dieses Set darf allerdings nicht direkt schreibend zugegriffen werden. Aus diesem Grund dürfen auch nicht die direkten Accessing-Methoden wie beispielsweise „remove: objekt“, „removeAll: mengeVonObjekten“ oder „add: objekt“ verwendet werden. Deshalb wurden auch diese Methoden redefiniert und durch private Methoden mit der Bezeichnung „addAttributName: neuesObjekt“ (z. B: addBestellPosition: bestellPositi-

- ◊ „formAttributName: neuesObjekt“ (z. B. „formBestellPosition: bestellPosition“), um eine neue „bestellPosition“ der Menge der referenzierten „bestellPositionen“ hinzuzufügen
- ◊ „dropAttributName: zuLoeschendesObjekt“ (z. B. „dropBestellPosition: bestellPosition“), um die Referenz auf das zuLoeschendeObjekt und dessen Rückreferenz zu löschen

Anstelle der Implementierung dieser Semantik in jede einzelne Zugriffsmethode, die eine Beziehung realisiert, wurden zwei alternative Patterns entwickelt.

Es lassen sich vier Grundformen von Zugriffsmethoden für den schreibenden Zugriff auf Instanzvariablen, die Beziehungen realisieren, unterscheiden:

1. einwertige Beziehungen, hinzufügen („formAttribut:“)
2. einwertige Beziehungen, löschen („dropAttribut:“)
3. mehrwertige Beziehungen, hinzufügen („formAttribut:“)
4. mehrwertige Beziehungen, löschen („dropAttribut:“)

Beim ersten Entwurfsmuster wurden deshalb vier unterschiedliche Methoden implementiert. Diese vier Methoden sind privat und werden in die Oberklasse der Domänen-Objekte plaziert, in diesem Fall in die Klasse „DomainObject“. Alle Domänen-Objekte sind Unterklassen dieser Klasse. Diese vier Methoden generieren bei jedem Zugriff den notwendigen Code, der ansonsten in jeder einzelnen Zugriffsmethode hätte implementiert werden müssen. Sie werden von den öffentlichen Zugriffsmethoden der Domänen-Objekte aufgerufen. In den öffentlichen Zugriffsmethoden der jeweiligen Domänen-Objekte werden lediglich drei Werte benötigt, die an die von „DomainObject“ geerbten Methoden übergeben werden müssen:

1. der Parameter, d. h. das zu löschende bzw. hinzuzufügende Objekt
2. die Bezeichnung der betroffenen Instanzvariablen, d. h. in die das Objekt eingefügt bzw. von welchem das Objekt gelöscht werden soll
3. die Bezeichnung der betroffenen Instanzvariablen der inversen Referenz

Dies wird beispielhaft an der Methode gezeigt, bei welcher eine Instanz eine einwertige Referenz realisiert und durch eine andere Instanz ersetzt wird (siehe Abbildung 40[194]). Der für jede Methode zu implementierende Code konnte da-

---

on“) und „removeAttributName: zuLoeschendesObjekt“ (z. B. removeBestellPosition: bestellPosition) ersetzt. Der Zugriff muss allerdings über die öffentlichen Methoden des Haupt-Objekts (im Fall des Beispiels ist das Haupt-Objekt die „bestellung“) erfolgen und nicht direkt auf dem IdentitySet (im Beispiel „setOfBestellPositionen“).

[194] Der zum Teil kompliziert anmutende Code rührt daher, dass GemStone keine Erweiterung der Kernel-Klassen erlaubt.

durch bedeutend vereinfacht werden, da sich der Code-Umfang ganz erheblich reduziert und dieselbe Logik nicht immer wieder zu implementieren war. Auch bei Änderungen der Programmlogik oder Kardinalität sind diese an nur einer einzigen Stelle durchzuführen.

```
!BestellPosition methodsFor: 'Accessing-Relations'!
formBestellung: eineBestellung
  "1 Bestellung"

  self
    replace: #Bestellung
    with: eineBestellung
    inverse: #bestellPositionen

!DomainObject methodsFor: 'Accessing-Relations'!
  replace: attributeName
  with: newObject
  inverse: inverseAccessingMethodName
  "Methode für einfach-kardinale Beziehungen, um das alte Objekt im Attri-
  butslot durch das neue Objekt zu ersetzen. attributeName und inverseAc-
  cessingMethod sind Symbols; newObject ist das neue Objekt selbst. EvM"

  | oldObject iam |
  oldObject := self perform: attributeName.
  oldObject == newObject
    ifFalse:
      [ | aN |
      aN := attributeName asString.
      aN at: 1 put: (aN at: 1) asUppercase.
      self perform: ('set' , aN , ':') asSymbol with: newObject.
      iam := inverseAccessingMethodName asString.
      iam at: 1 put: (iam at: 1) asUppercase.
      oldObject isNil
        ifFalse:
          [oldObject perform: ('drop' , iam , ':') asSymbol with: self].
      newObject isNil
        ifFalse:
          [newObject perform: ('form' , iam , ':') asSymbol with: self]]
```

Abbildung 40: Methode zur Generierung einer Zugriffsmethode für eine einwertige binäre Referenz

Das zweite Entwurfsmuster stellt eine Weiterentwicklung der Idee des ersten Musters dar. Der Nachteil des ersten Entwurfsmusters ist, dass es eine immer noch redundante getrennte Spezifikation der Constraints erzwingt. Vor allem bestand jedoch das Problem, dass diese Form der Meta-Daten keinen einfachen Zugriff erlaubten. Deshalb wurden die Metadaten der Klassendefinition erweitert. Beispielsweise wurde eine Klassenvariable hinzugefügt, die ein „dictionary“ enthält, in welchem für jede Instanzvariable eine Metaspezifikation hinterlegt wurde, wie unter anderem die Bezeichnung der inversen und die Kardinalität der Beziehung. Deshalb wurde für alle Zugriffsmethoden für Beziehungen nur noch eine einzige, sehr einfache Form der Zugriffsmethode benötigt, die lediglich die Bezeichnung der Instanzvariablen und das neue Objekt als Paramter an die Ge-

nerierungs-Methode übergeben muss, aus der dann der jeweilige Code generiert wird (siehe Abbildung 41).

```
!BestellPosition methodsFor: 'Accessing-Relations'
formBestellung: aValue
  self set: #Bestellung with: aValue
```

Abbildung 41: Vereinfachte Methode zur Generierung einer Zugriffsmethode

Dieses Konzept bildet auch die Basis, um die verschiedenen Formen der Metainformationen im Objektmodell hinterlegen zu können und fehlende semantische Konzepte wie etwa die Aggregations-Semantik nachzubilden. [RMK99]

Anschließend wurden noch für die Domänen-Objekte zusätzlich notwendige Methoden, wie etwa das Printing-, Dirty-State-, Copying-, Comparing und Testing-Protokoll implementiert.

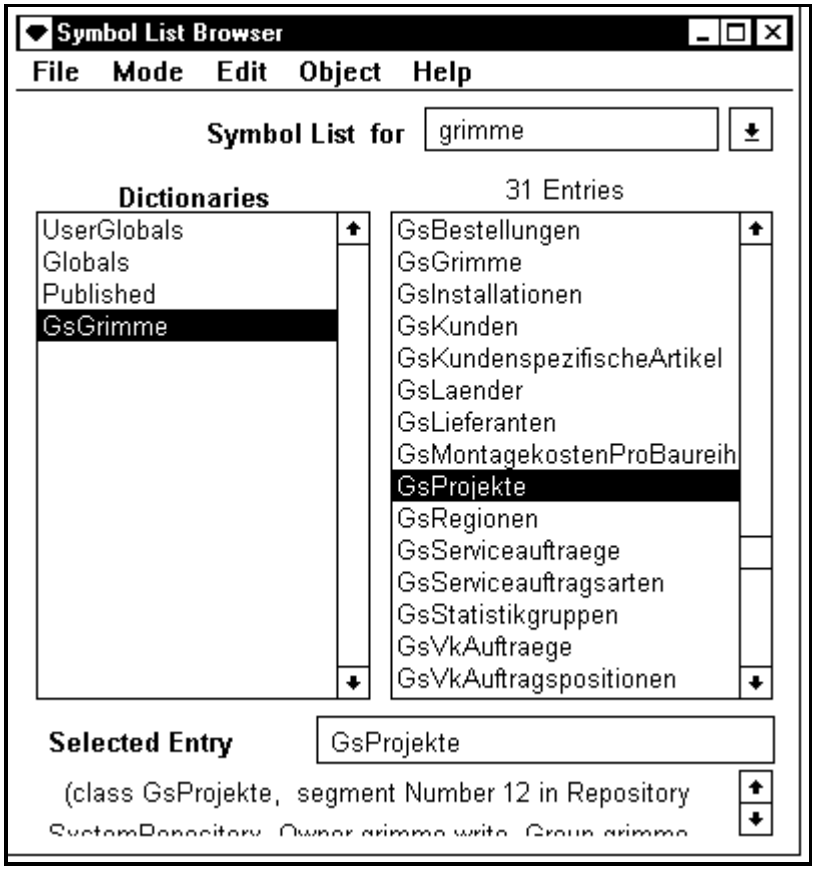


Abbildung 42: Symbol List Browser mit Dictionaries in GemStone

Nachdem das Objektmodell implementiert war, musste für die noch zu erstellenden Instanzen Persistenz erzeugt werden. GemStone implementiert ein Persistenzmodell durch Benennung (by Naming) und Erreichbarkeit (Reachability), das prinzipiell analog zur Image-Persistenz von Smalltalk ist. Während es bei

VisualWorks nur ein SystemDictionary („Smalltalk“) gibt, stellt GemStone die Möglichkeit zur Verfügung verschiedene benutzer-spezifische SymbolDictionaries zu definieren (siehe Abbildung 42), deren Priorität[195] durch die Reihenfolge im Symbol List Browser definiert wird.

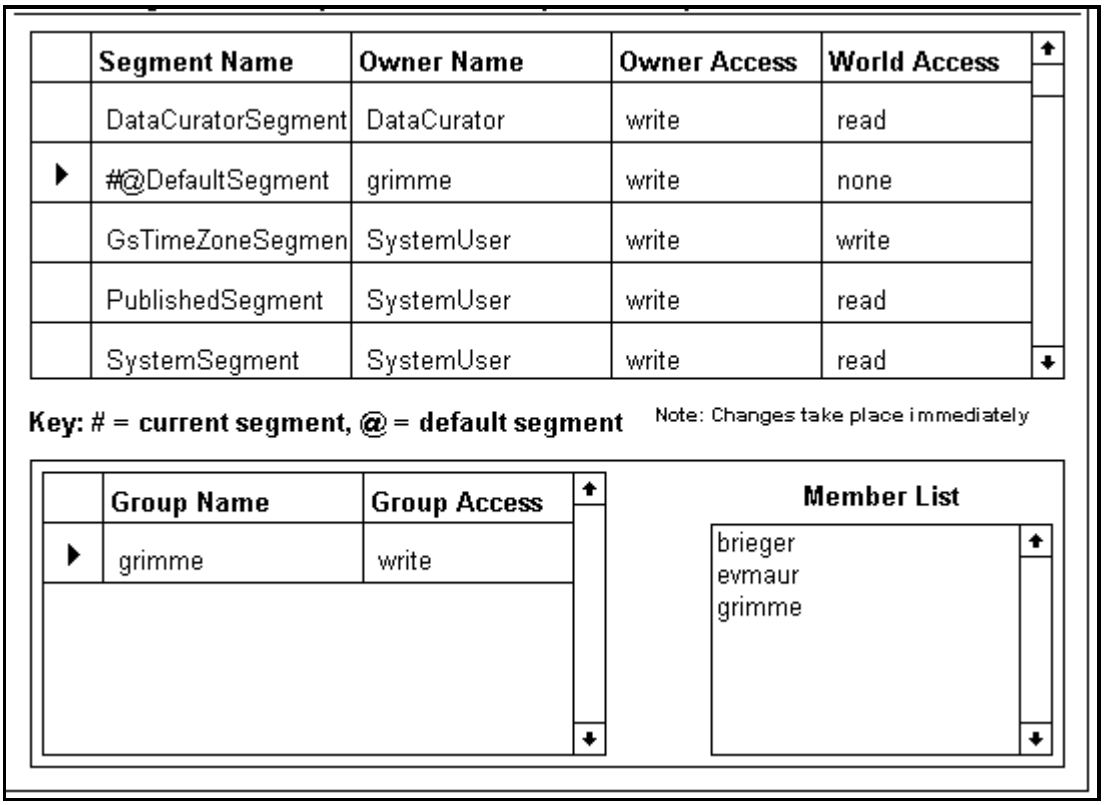


Abbildung 43: Segment Browser zur Definition der Objektrechte

Alle Objekte die persistent werden sollen, müssen entweder direkt von einem SymbolDictionary referenziert werden oder mittelbar zu dessen transitiven Hüllen gehören. Mithin werden auch alle Klassen durch ein SymbolDictionary referenziert. Zur Persistentmachung der Instanzen wurde ein Wurzelobjekt „GsBase“ definiert. Diesem Wurzelobjekt wurde eine Instanz von „Dictionary“ zugeordnet. In dieses Dictionary wiederum wurde für jede Extension bzw. Instanzmenge ein Eintrag erzeugt, der ein „IdentitySet“ referenziert. Für die Bestellungen wurde zum Beispiel ein Eintrag „#Bestellungen“ eingefügt, der eine Instanz von „SetOf-Bestellungen“ referenziert. In die einzelnen „IdentitySets“ wurden die jeweiligen Instanzen eingefügt, die dadurch Persistenz erlangen, da sie zur transitiven Hülle des Wurzelobjekts gehören.

Im letzten Schritt wurden die Zugriffsrechte für das Objektmodell definiert (siehe Abbildung 43). Die Rechte werden für Segmente vergeben. Dabei sind beliebi-

[195] Gibt es mehrere Klassen gleichen Namens, wird die Klasse mit höchster Priorität verwendet.

ge Segmente definierbar. Jedem Segment werden ein Eigentümer und verschiedene Gruppen zugeordnet. Mithin gibt es die drei Rechteebenen: Eigentümer (Owner), Gruppe (Group) und die restlichen GemStone-Benutzer (World). Diesen können jeweils drei alternative Rechteausprägungen zugeordnet werden, das Recht zu lesen und zu schreiben (write), nur zu lesen (read) oder keinerlei Zugriff (none). Für den Prototypen wurde lediglich ein Segment definiert. Der Eigentümer „grimme“ und die Mitglieder der Gruppe „grimme“ (brieger, evmaur und grimme) dürfen in diesem Segment lesen und schreiben, die anderen Benutzer haben keine Rechte (siehe Abbildung 43).

Den einzelnen Segmenten wiederum werden Objekte zugeordnet. Objekte sind unter anderem Instanzen und Klassen. Die Rechte werden mithin nicht für die Klassen und deren Extensionen vergeben, wie dies analog zu RDBMS bei den Relationen der Fall ist, sondern können beliebig auf jeder vorstellbaren Ebene definiert werden, da alles in GemStone ein Objekt ist und die Rechte auf Objektebene vergeben werden (siehe auch Abbildung 42).

6.6 Objekt-Transformation

Bis zu dieser Stelle wurden zwei Objektmodelle implementiert, eines im Client-Smalltalk, das im wesentlichen den Strukturen des relationalen Datenmodells in Oracle entspricht und ein zweites in GemStone, das die Grundlage des Reconciled Object Layers bildet und das Ziel-Objektmodell darstellt. Deshalb mussten die Objekte im Client-Smalltalk in das GemStone-Objektmodell transformiert werden.

Dazu lassen sich prinzipiell zwei alternative Vorgehensweisen unterscheiden. Bei der ersten Variante werden die Client-Objekte nach GemStone übertragen und dort in neue Objekte transformiert. Bei der zweiten Variante werden die Client-Objekte im Client in neue Objekte transformiert und anschließend nach GemStone übertragen. Im ersten Fall müssen die mithilfe der ObjectLens generierten Klassen auch in GemStone erzeugt bzw. generiert werden, da die Instanzen nicht ohne die zugehörigen Klassen übertragen werden können. Im zweiten Fall müssen aus demselben Grund die Klassen des GemStone-Objektmodells im Client-Smalltalk implementiert bzw. generiert werden.

Beide Varianten wurden getestet. Da die Vorgehensweisen prinzipiell gleich sind, die zweite Variante allerdings konzeptionell klarer ist, wird im Folgenden diese zweite Alternative beschrieben.

Für die Generierung der GemStone-Klassen in VisualWorks bietet GemStone eine Methode an, die es erlaubt, Klassen mit allen oder ausgewählten Methoden nach VisualWorks zu übertragen. Unter Verwendung dieser Methode wurden alle Klassen und Methoden auch in VisualWorks generiert. Anschließend wurden

die GemStone- und VisualWorks-Klassen gemappt, wobei angegeben werden kann, ob bei Veränderungen einer der beiden Klassen entweder die GemStone-Klasse oder die VisualWorks-Klasse angepasst werden soll (siehe Abbildung 44 auf Seite 258). Es ist außerdem möglich, einzelne Instanzvariablen nicht zu mappen oder mit Instanzvariablen zu verknüpfen, die eine andere Bezeichnung haben.

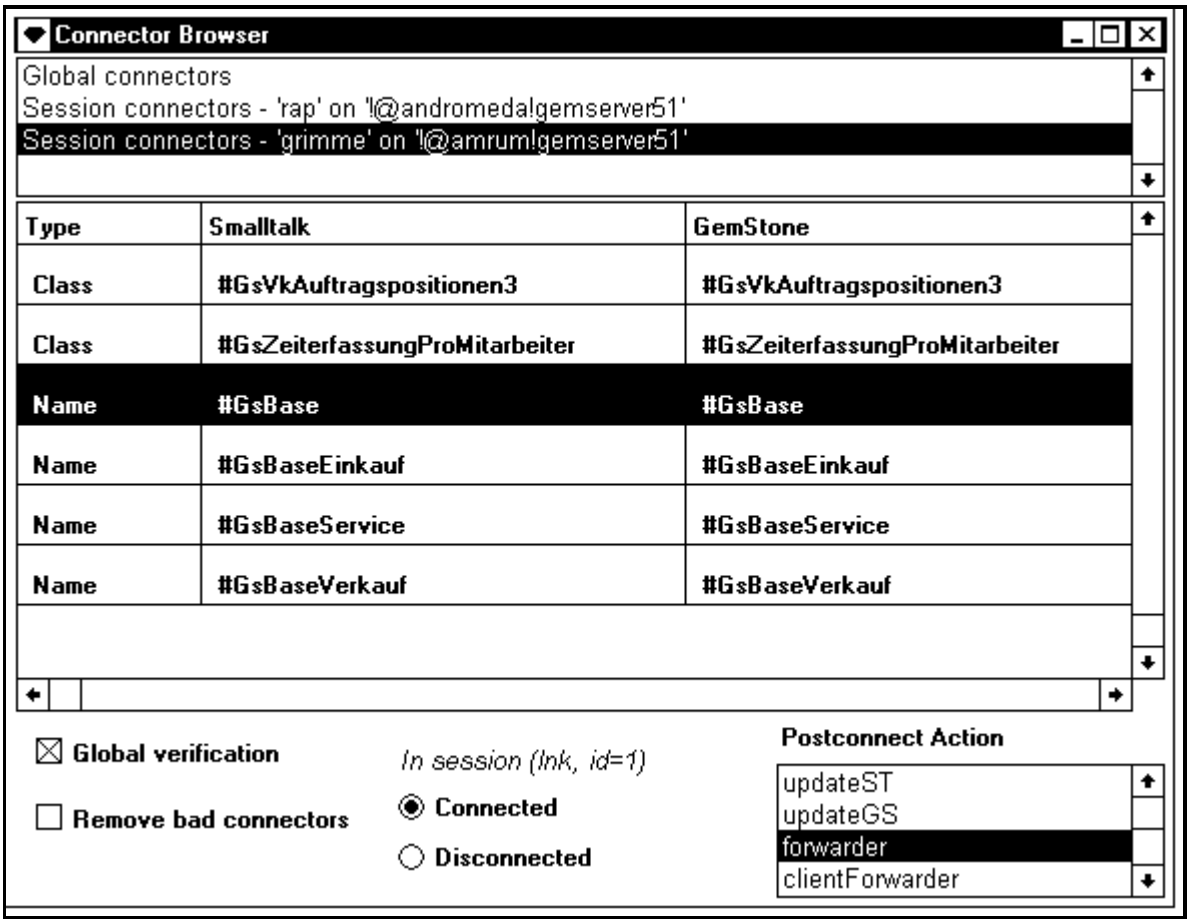


Abbildung 44: GemBuilder Connector Browser

Damit die neuen Instanzen von VisualWorks nach GemStone übertragen werden konnten, musste eine Verbindung zu GemStone hergestellt werden. Dies funktioniert im Prinzip auf die selbe Weise wie bei den Klassen-Mappings mithilfe eines Connectors (siehe Abbildung 44). Im Unterschied zu den Class-Connectors wird ein Name-Connector definiert. Hierzu erfolgte ein Eintrag im Smalltalk-SystemDictionary (globale Variable). Diese Variable wurde dann mit dem Wurzelobjekt in GemStone (Eintrag im SymbolDictionary von GemStone, siehe Abbildung 42) durch den Name-Connector verbunden. Für den Prototypen wurde beispielsweise „GsBase“ in das SystemDictionary von VisualWorks eingetragen und mit „GsBase“ verknüpft, das als Wurzelobjekt im SymbolDictionary von GemStone angelegt wurde.

Prinzipiell lassen sich nun drei Alternativen unterscheiden. Die durch das Wurzelobjekt referenzierten Objekte können jedes Mal von GemStone in den Client repliziert werden, die Objekte können vom Client nach GemStone repliziert werden oder die Objekte werden nicht repliziert, sondern verbleiben vollständig in GemStone. Aufgrund der Datenmengen des Prototypen ist eine Replikation in den Client nicht sinnvoll. Deshalb wurde die letzte Alternative gewählt. Hierzu wurde die Verknüpfung zwischen den Wurzelobjekten als „Forwarder" definiert (siehe Abbildung 44).

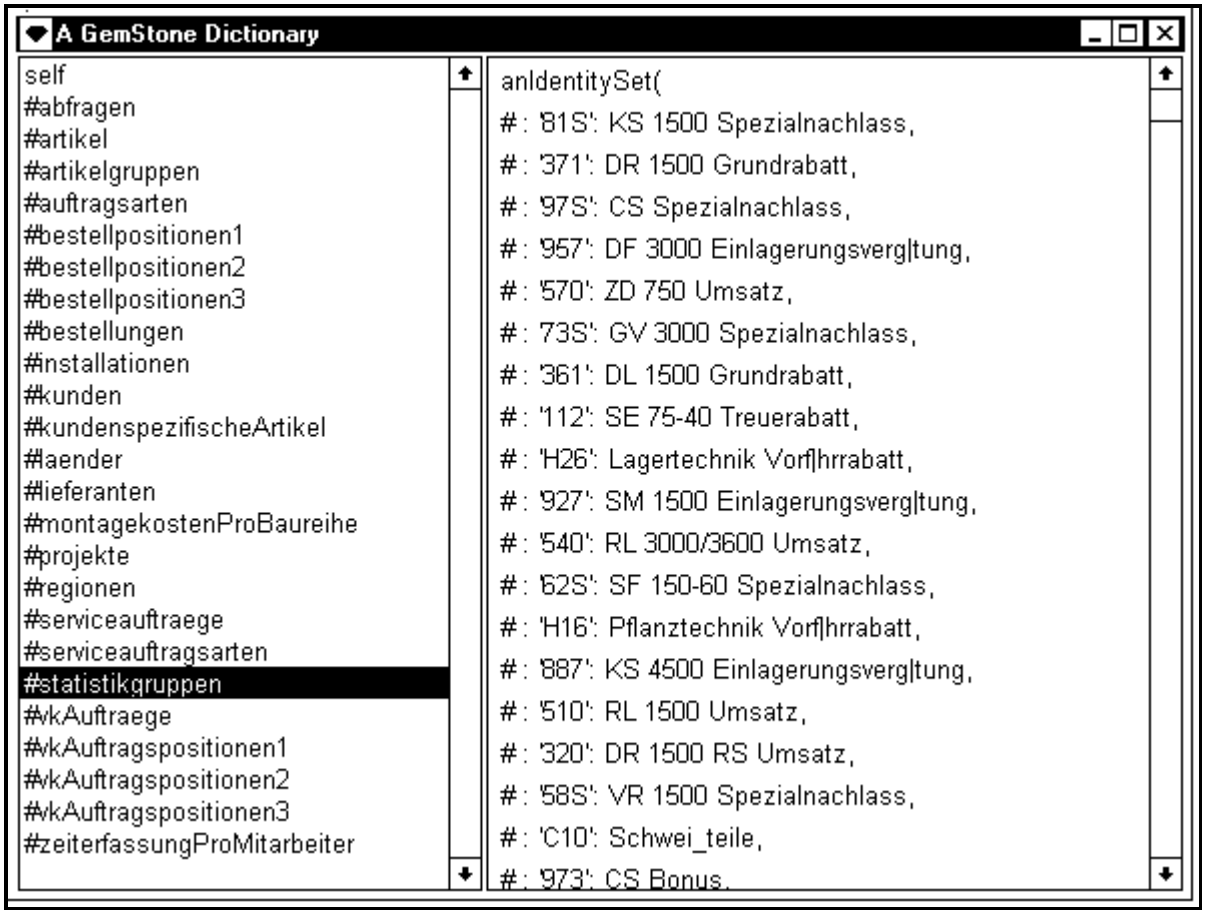


Abbildung 45: GemStone Inspector mit den transformierten GemStone-Objekten

Wird nun ein Objekt in das Client-Smalltalk-Wurzelobjekt eingefügt, d. h. gehört es zu dessen transitiven Hülle, wird dieses Objekt nach GemStone geschrieben und ist dort persistent. Solange keine Session zu GemStone besteht, ist das Wurzelobjekt im Client-Smalltalk („GsBase") mit „nil" belegt. Durch eine GemStone-Session werden die Wurzelobjekte verbunden. Dadurch wird in VisualWorks eine globale Variable simuliert, deren Objekte jedoch nur in GemStone existieren. Durch den Zugriff auf einzelne Objekte können diese in VisualWorks-Objekte umgewandelt werden. Dabei ist beliebig definierbar wieviele Referenzebenen übertragen werden. Außerdem kann beliebig bestimmt werden, ob Methoden in

GemStone oder in VisualWorks ausgeführt werden sollen und ob das Ergebnis VisualWorks-Objekte oder Forwarder auf GemStone-Objekte sein sollen.

Der letzte Schritt bestand darin, die aus Oracle generierten VisualWorks-Instanzen in GemStone-Objekte des Ziel-Objektmodells zu transformieren. Dieser Prozess war enorm aufwendig und erforderte den meisten Programmier-Aufwand, obwohl dies konzeptionell von geringem Interesse ist.[196] Jedes Attribut musste einzeln auf atomarer Ebene (z. B. Integer, String, Boolean) in die neuen Instanzen überführt werden.

Schließlich wurden alle Client-Objekte transformiert und nach GemStone übertragen (siehe Abbildung 45).

[196] Die Entwicklung eines entsprechenden Werkzeugs, wie dies in jüngster Zeit bei RDBMS verstärkt mit ETL-Tools der Fall ist, wäre in diesem Zusammenhang eine interessante Forschungsarbeit.

KAPITEL 7

KONKLUSION UND PERSPEKTIVEN

Die zentrale Forschungsfrage dieser Arbeit, wie eine mögliche Konzeption eines Object Warehouse prinzipiell aussehen kann und die daraus resultierende Kernhypothese, dass die bereits im Bereich der operativen Informationssysteme bewährten Potenziale des objektorientierten Ansatzes auch für das Data Warehouse, respektive Object Warehouse zu realisieren sein können, kann eindeutig positiv beantwortet werden.

Das Fundament der vorliegenden Arbeit bilden ausführliche Diskussionen zum Data-Warehouse-Konzept und zum Objekt-Paradigma, die jeweils in einer konkreten Konzeption resultieren, d. h. einer Data-Warehouse-Konzeption und einer Konzeption für ein durchgängiges Objekt-Paradigma. Bereits anhand dieser beiden grundlegenden Kapitel wurden die Potenziale für die Konzeption des Object Warehouse deutlich. Dabei bestehen bei der in dieser Arbeit entwickelten Object-Warehouse-Konzeption für alle aufgestellten Software-Qualitäts-Kriterien entscheidende Verbesserungs-Potenziale gegenüber der bisher üblichen konventionellen Data-Warehouse-Konzeption.

Die bedeutendsten Vorteile des Object Warehouse gegenüber seinem konventionellen Analogon liegen im Objektmodell und dessen Umsetzung in einem Objectbase Management System zur Verwirklichung des entwickelten Konzepts eines Reconciled Object Layers, dem integrativen Ansatz sowie dem durchgängigen Systems-Engineering-Prozess (OOA, OOD, OOP, Organisation).

Auch beim konventionellen Data Warehouse wurde die zentrale Bedeutung des Reconciled Data Layers herausgearbeitet, dessen kritische Erfolgsfaktoren erstens die Datenmodell-Qualität und zweitens die Qualität der Daten selbst sind.

Dabei besteht insbesondere die Zielsetzung, ein nicht nur kurz- bis mittelfristig sinnvoll zu verwendendes Datenmodell zu implementieren, wie dies bei den derzeit vorherrschenden Ansätzen der Fall ist, sondern das Datenmodell samt Daten derart zu modellieren, dass trotz struktureller Änderungen der Diskurswelt und

mithin auch des Datenschemas, der RDL den Anforderungen der MSS auch langfristig genügt.

Erst die Eigenschaften des entwickelten Objektmodells gestatten es jedoch, die Anforderungen an die zentrale rekonziliierte Datenbasis erfüllen zu können. Dabei wurden die essenziellen Unterschiede zwischen einer gewöhnlichen Datenbasis, heute in den meisten Fällen mithilfe von relationaler Technologie verwaltet und dem mächtigen Ansatz einer Objektbasis des in dieser Arbeit entwickelten Objectbase-Management-Systems-Konzepts aufgezeigt.

Die Anforderungen an das Object Warehouse gehen von den eigentlichen Management Support Systems bzw. deren Nutzer aus, die neben der angemessenen Modellierung und Integration, d. h. Rekonziliierung der auf verhältnismäßig einfache Weise verfügbaren operativen strukturierten quantitativen internen Daten bzw. Objekte, insbesondere die Forderung nach einer ebensolchen Rekonziliierung auch der semistrukturierten Daten, der nicht-operativen Daten, der qualitativen Daten und vor allem der Metadaten (Built-Time-, Control-, Usage-Metadaten) stellen, soweit dies den ökonomischen Restriktionen genügt und also die Ordnung bzw. Strukturierung der MSS-relevanten Daten der jeweils alternativen Suche aus Effizienz- und/oder Effektivitätsgründen vorzuziehen ist.

Die Umsetzung der Object-Warehouse-Konzeption wird allerdings zur Zeit durch fünf wesentliche Faktoren erschwert:

1. Der Ansatz ist hoch komplex, daher nur schwer um- bzw. einzusetzen und mit erheblichen Akzeptanzbarrieren versehen, wie die meisten komplizierten neuen Lösungsansätze.
2. Wesentliche Teile des Objekt-Paradigmas bedürfen weiterhin intensiver Forschungsarbeit.
3. Es fehlen immer noch benötigte Standards, was teilweise eine Folge des letzten Punktes ist, da für Konzepte die noch nicht ausreichend gefestigt sind auch keine Standards festgelegt werden sollten.
4. Die bereits entwickelten Konzepte sind nur unzureichend in den verfügbaren Werkzeugen umgesetzt.
5. Es fehlt die ausreichende Bereitschaft von Unternehmen, die einsetzbaren Konzepte und Werkzeuge im Feld zu testen.

Gerade Punkt vier und fünft werden in der vorliegenden Arbeit besonders deutlich. Während in Teil I Konzeptionen mit mächtigen Potenzialen entwickelt wurden, zeigt der Teil II ein ernüchterndes Bild. Alleine die zur Durchführung eines konventionellen MSS-Projekts notwendige Überzeugungsarbeit bei den Unternehmen ist in der Regel erheblich, dies zusätzlich mit innovativen[197] Technologi-

[197] Dabei wurden die grundlegenden Konzepte für das Objekt-Paradigma in den sechziger Jahren entwickelt!

en wie der Objektorientierung erreichen zu wollen, gelingt nur in äußerst seltenen Fällen.

Bei den dennoch durchgeführten Projekten zeigte sich leider, dass die Umsetzung der entwickelten Konzepte in einen Prototypen nur in Ansätzen realisierbar ist. Verantwortlich hierfür sind außer den bereits aufgeführten Aspekten in erster Linie die mangelhafte Software-Qualität der OLTP-Systeme, d. h. der heute erhältlichen ERP-Systeme und das fehlende Bewußtsein der Mitarbeiter für diese Problematik. Dabei sind auch die konventionellen Techniken und Werkzeuge in vielen Fällen durchaus sinnvoll einsetzbar, doch zeigt die Praxis, dass auch dies in der Regel nur in ausgesprochen unzureichender Weise geschieht, wie das Beispiel des ERP Baan[198] deutlich gezeigt hat (prozess- oder sogar maskenorientierte, minderwertige Datenmodellierung, kein Einsatz von Constraints, mangelhafte Datenqualität etc.).

Es besteht mithin die Hoffnung und Forderung, bessere kommerzielle OLTP-Systeme zu entwickeln und auch für diese OLTP-Systeme die Potenziale des objektorientierten Paradigmas zu nutzen. Dabei müssen die Anforderungen der MSS bereits in den OLTP-Systemen berücksichtigt werden können. Konkret besteht die Forderung nach einer Integration der operativen Daten, einer besseren Datenqualität bzw. Datenhygiene, der Berücksichtigung von MSS-relevanten Daten bereits durch die operativen Systeme, denn was nicht erhoben wird kann auch nicht analysiert werden und der Möglichkeit, die jeweils geänderten Daten auf einfache Weise extrahieren zu können, ohne die bisher notwendigen hochkomplexen Image-Vergleiche oder Auswertungen der Log-Files etc. durchführen zu müssen.

Die nächsten Schritte zur Evolution der entwickelten Object-Warehouse-Konzeption müssen darin bestehen, weitere Überzeugungsarbeit zu leisten:

- bei den Unternehmen, um weitere Prototypen realisieren zu können und deren praktischen Einsatz durch die End-Anwender zu testen,
- bei den Werkzeug-Herstellern, damit die entwickelten Konzepte in die Werkzeuge umgesetzt werden und
- in der OO-Community, speziell bei den OBMS-Forschern, damit diese die realen MSS-getriebenen Anforderungen in ihren Zielkatalog mit aufnehmen.

Parallel müssen die Forschungsaktivitäten sowohl auf dem Sektor der konventionellen MSS, speziell der relationalen Data-Warehouse-Technologie und des Objekt-Paradigmas weitergeführt werden:

---

[198] Dieses Resultat trifft in ebensolcher Weise auf alle untersuchten OLTP-Systeme zu, so dass der eventuell fälschlicherweise entstehende Eindruck, dass dies ein Baan-spezifisches Problem sei, zurückgewiesen werden muss. Dies bedeutet, dass es nicht etwa ein Ergbnis dieser Arbeit ist, dass Baan von schlechterer Qualität ist als beispielsweise SAP/R3.

- Für das konventionelle Data Warehouse ist die Entwicklung insbesondere in den Bereichen der Verwaltung und Nutzung von Metadaten [JJQ+00, S. 124], der Schema-Evolution und Versionierung im Relationen-Modell bzw. in RDBMS und im Bereich der ETL-Tools notwendig. All dies weist unter dem Gesichtspunkt der momentanen praktischen Relevanz einen hohen Dringlichkeitsgrad auf.
- Für das Objekt-Paradigma weist insbesondere die OBMS-Forschung und hier allem voran die weitere Entwicklung und dringlichst benötigte Implementierung von Object Query Languages in kommerzielle OBMS-Produkte einen hohen Prioriätsgrad auf. Ohne eine den entwickelten Konzepten genügende OQL ist der praktische Nutzen eines Object Warehouse erheblich eingeschränkt.

Die unmittelbaren Forschungsvisionen für das Object-Warehouse-Konzept sind die Untersuchung einer möglichen Integration der Ansätze zur Wissensrepräsentation, die mögliche Kopplung der Idee dessen was zur Zeit als „verteilte intelligente Agenenten“ [KnJo98, S. 2 ff.] und „Agenten-Orientierung“ [Shoh93, S. 51 ff., WJK99, S. 69 ff.] diskutiert wird und die Verbindung mit der WWW-Technologie, insbesondere des dynamischen OBMS-getriebenen Web-Publishing [RDM00], das deutliche Synergien mit dem vorgestellten Konzept aufweist. Dabei herrscht im Bereich der Agenten-Diskussion ein ähnliches Konzept- und Begriffs-Chaos, wie dies für das Objekt-Paradigma aufgezeigt wurde. [Flor00]

Agenten stellen vor allem deshalb eine weitere Perspektive für das Object-Warehouse-Konzept dar, weil deutliche Bezüge zum entwickelten objektorientierten Konzept bestehen — Agenten sind Objekte. Noch deutlicher als bei den objektorientierten Konzepten, steht die Autonomie der Objekte respektive Agenten und deren aktives Handeln im Vordergrund. Agenten verfügen jedoch im Unterschied zu Objekten nicht nur über „Intelligenz“, wie sie in dieser Arbeit definiert wurde, also einfache Methoden, sondern über eine weiterführende „Intelligenz“, wie sie im Bereich der Künstlichen-Intelligenz-Forschung definiert wird. [Flor00]

Außerdem wird der Aspekt der Verteilung noch stärker betont, was eine Kopplung mit Objekten anderer Systeme erlaubt und die Modellierung und Implementierung auf einer höheren Abstraktionsstufe unterstützt. Wie deutlich die Bezüge zur beschriebenen Object-Warehouse-Konzeption sind, zeigen die Umsetzungen intelligenter Agenten für verteilte Systeme von *Knapik* und *Johnson*, die mit Smalltalk implementiert wurden. [KnJo98, S. 265 ff.]

LITERATURVERZEICHNIS

[ABB+98] Anderson, A., Beattie, R., Beck, K., Byant, D., DeArment, M., Fowler, M., Fronczak, M., Garzaniti, R., Gore, D., Hacker, B., Hendrickson, D., Jeffries, R., Joppie, D., Kim, D., Kowalsky, P., Mueller, D., Murasky, T., Nutter, R., Pantea, A., Thomas, D.: Chrysler Goes to „Extremes“, 1998, http://xprogramming.com/publications/-dc9810cs.pdf, 11. 02. 2000

[ABD+89] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S.: The Object-Oriented Database System Manifesto, http://-www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/-htManifesto/Manifesto.html, (Original in: Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD) in Kyoto, 1989), 26. 04. 1995

[ABW98] Alpert, S., Brown, K., Woolf, B.: The Design Patterns Smalltalk Companion, Addison Wesley, 1998

[AhFi92] Ahrens, K., Fischer, J.: Objektorientierte Programmierung, Technik, 1992

[AIM77] Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language — Towns, Buildings, Construction, Oxford University, 1977

[Alex79] Alexander, C.: The Timeless Way of Buildings, Oxford University, 1979

[Appe95] Appelfeller, W.: Wiederverwendung im objektorientierten Softwareentwicklungsprozeß, dargestellt am Beispiel der Entwicklung eines Lagerlogistiksystems, Lang, 1995

[Arsn95] ArsNova: VisualDesigner 3.1 for VisualWorks 2.5 — Release Notes und Handbuch, Ars Nova, 1995

[BaEd97] Barquin, R., Edelstein, H. (Hrsg.): Planning and Designing the Data Warehouse, Prentice Hall, 1997

[Balz93a] Balzert, H.: CASE — Systeme und Werkzeuge, 5. Auflage, BI Wissenschaftsverlag, 1993

[Balz93b] Balzert, H.: Ein Überblick über die Methoden und Werkzeuglandschaft in: [Balz93a], S. 31-121

[Balz95] Balzert, H.: Methoden der objektorientierten Systemanalyse, BI Wissenschaftsverlag, 1995

[Balz96] Balzert, H.: Lehrbuch der Softwaretechnik — Software-Entwicklung, Spektrum Akademischer Verlag, 1996

[Balz98] Balzert, H.: Lehrbuch der Softwaretechnik — Software-Management, Software-Qualtitätssicherung, Unternehmensmodellierung, Spektrum Akademischer Verlag, 1998

[BaSe97a] Back, A., Seufert, A.: State of the Art des Management Supports — Teil 1: Klassische Management Support Ansätze, Universität St. Gallen, 1997

[BaSe97b] Back, A., Seufert, A.: State of the Art des Management Supports — Teil 2: Neuere Management Support Ansätze, Universität St. Gallen, 1997

[BaSe97c] Back, A., Seufert, A.: Executive Information Systems (EIS) in: [MBB+97], S. 157-158

[BBB+97] Beck, J., de Boer, T., Book, R., Cetinkaya, A., von Maur, E., Plöger, A., Türkeri, G., Weber, H., Zehrer, O.: Projektbericht Führungsinformationssysteme in der Praxis — Große Kracht GmbH & Co. KG, Universität Osnabrück, 1997, unveröffentlicht

[Beck91] Beck, K.: Think Like an Object, 1991 in: [Beck99b], S. 61-72 (Original in Unix Review, 10/1991)

[Beck93a] Beck, K.: Instance-Specific Behavior — How and Why, 1993 in: [Beck99b], S. 157-166 (Original in Smalltalk Report, 3-4/1993)

[Beck93b] Beck, K.: Instance-Specific Behavior — Digitalk Implementation and the Deeper Meaning of It All, 1993 in: [Beck99b], S. 167-174 (Original in Smalltalk Report, 5/1993)

[Beck93c] Beck, K.: CRC — Finding Objects the Easy Way, 1993 in: [Beck99b], S. 205-212 (Original in Object Magazine, 11-12/1993)

[Beck93d] Beck, K.: Inheritance — The Rest of the Story, 1993 in: [Beck99b], S. 181-187 (Original in Smalltalk Report, 7-8/1993)

[Beck93e] Beck, K.: Inheritance — The Rest of the Story (cont.), 1993 in: [Beck99b], S. 189-193 (Original in Smalltalk Report, 9/1993)

[Beck94a] Beck, K.: Where Do Objects Come From? in: [Beck99b], S. 219-223 (Original in Smalltalk Report, 3-4/1994)

[Beck94b] Beck, K.: Where Do Objects Come From? — From Variables and Methods in: [Beck99b], S. 239-243 (Original in Smalltalk Report, 5/1994)

[Beck94c] Beck, K.: Simple Smalltalk Testing, 1994 in: [Beck99b], S. 277-287 (Original in Smalltalk Report, 10/1994)

[Beck97] Beck, K.: Smalltalk — Best Practice Patterns, Prentice Hall, 1997

[Beck99a] Beck, K.: Simple Smalltalk Testing — With Patterns, http://-www.armaties.com/testfram.htm, 22. 2. 1999

[Beck99b] Beck, K. (Hrsg.): Kent Beck's Guide to Better Smalltalk — A Sorted Collection, Cambridge University/SIGS, 1999

[Beck00] Beck, K.: Extreme Programming Explained — Embrace Change, Addison Wesley, 2000

[BeCu89] Beck, K., Cunningham, W.: A Laboratory For Teaching Object-Oriented Thinking, 1989 in: [Beck99b], S. 51-61 (Original in Proceedings der OOPSLA '89)

[BEGR99] Becker, J., Ehrenberg, D., Griese, J., Roithmayr, F. (Hrsg.): Proceedings des Doktorandenseminars zur WI '99, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 1999, http://-wi99.iwi.uni-sb.de/de/doktorandenseminar.html

[Behm96] Behme, W.: Business-Intelligence als Baustein des Geschäftserfolgs in: [MuBe96], S. 27-45

[Behr91] Behrens, G.: Wissenschaftstheorie und Betriebswirtschaftslehre in: [WKK+], Spalte 4763-4772

[Bell97] Bellinger, G.: Knowledge Management — Bah Humbug, Knowledge Management Town Meeting at Documation '97, 27. 2. 1997, http://-www.knowledge-at-work.com/kmbh.htm, 28. 12. 1999

[BeMa94] Bertino, E., Martino, L.: Object-Oriented Database Systems — Concepts and Architectures, Addison Wesley, 1994

[BeSc96] Becker, J., Schütte, R.: Handelsinformationssysteme, Moderne Industrie, 1996

[BGL95] Bücker, M., Geidel, J., Lachmann, M.: Programmieren in Smalltalk mit VisualWorks — Smalltalk nicht nur für Anfänger, 2. Auflage, Springer, 1995

[BHJ+96] Brinker, S., Helms, D., Jarchau, D., Kalkstein, S., Krüger, D., Lönker, C., von Maur, E., Trentmann, J.: Projektbericht Führungsinformationssysteme in der Praxis — Ihr Platz GmbH & Co., Universität Osnabrück, 1996, unveröffentlicht

[BiAl97] Bischoff, J., Alexander, T. (Hrsg.): Data Warehouse — Practical Advice from the Experts, Prentice Hall, 1997

[BiHa97] Bissantz, N., Hagedorn, J.: Data Mining in: [MBB+97], S. 104-105

[Blac99] Blache, M.: Konzeption und prototypische Implementierung eines auf der Standardsoftware Baan aufsetzenden Vertriebsinformationssystems für einen Landmaschinenhersteller, Diplomarbeit an der Universität Osnabrück, 1999, unveröffentlicht

[BlPr98] Blaha, M., Premerlani, W.: Object-Oriented Modeling and Design for Database Applications, Prentice Hall, 1998

[BMO+89] Bretl, R., Maier, D., Otis, A., Penney, J., Schuchardt, B., Stein, J., Williams, E., Williams, M.: The GemStone Data Management System in: [KiLo89], S. 283-308

[Bode97] Bodendorf, F.: Methodenbanksystem in: [MBB+97], S. 262-263

[Booc96] Booch, G.: Objektorientierte Analyse und Design — Mit praktischen Anwendungsbeispielen, 2. Auflage, Addison Wesley, 1996

[BrBu92] Breutmann, B., Burkhardt, R.: Objektorientierte Systeme — Grundlagen, Werkzeuge, Einsatz, Hanser, 1992

[Bric98] Bricker, E.: Data Modeling for Data Warehouse in: Information Builders System Journal, 4/1998, S. 67-78

[Burl97] Burleson, D.: ORACLE Data Warehousing, Coriolis Group Books, 1997

[CaBa98] Cattell, R., Barry, D.: The Object Database Standard: ODMG 2.0, Morgan Kaufmann, 1998

[CáLe90a] Cárdenas, A., McLeod, D.: Research Foundations in Object-Oriented and Semantic Database Systems, Prentice Hall, 1990

[CáLe90b] Cárdenas, A., McLeod, D.: An Overview of Object-Oriented and Semantic Database Systems in: [CáLe90a], S. XVII-XXV

[Catt91] Cattell, R.: Object Data Management — Object-Oriented and Extended Relational Database Systems, Addison Wesley, 1991

[CCS93] Codd, E., Codd, S., Salley, C.: Providing OLAP (On-line Analytical Processing) to User-Analysts — An IT Mandate, White Paper, Arbor, 1993

[Cham97] Chamoni, P.: Online Analytical Processing (OLAP) in: [MBB+], S. 294-295

[ChGl99a] Chamoni, P., Gluchowsky, P. (Hrsg.): Analytische Informationssysteme, 2. Auflage, Springer, 1999

[ChGl99b] Chamoni, P., Gluchowsky, P.: Analytische Informationssysteme — Einordnung und Überblick in: [ChGl99a], S. 3-25

[ChGl99c] Chamoni, P., Gluchowsky, P.: Entwicklungslinien und Architekturkonzepte des On-Line Analytical Processing in: [ChGl99a], S. 261-280

[ChSt93] Chorafas, D., Steinmann, H.: Object-Oriented Databases, Prentice Hall, 1993

[ChZe96] Chamoni, P., Zeschau, D.: Management-Support-Systems und Data-Warehousing in: [MuBe96], S. 47-83

[Clau96] Claus, T.: Objektorientierte Simulation und Genetische Algorithmen zur Produktionsplanung und -steuerung, Lang, 1996

[CNM97] Coad, P., North, D., Mayfield, M.: Object Models — Strategies, Patterns, and Applications, 2. Auflage, Yourdon, 1997

[Coad92] Coad, P.: Object-Oriented Patterns in: Communications of the ACM, 9/1992, Volume 35, Nr. 9, S. 152-159

[Colo98] Colomb, R.: Deductive Databases and Their Applications, Taylor and Francis, 1998

[CoNi94] Coad, P., Nicola, J.: Objektorientierte Programmierung, Prentice Hall, 1994

[CoSc98] Coplien, J., Schmidt, D. (Hrsg.): Pattern Languages of Program Design, PLoP '94, 3. Auflage, Addison Wesley, 1998

[CoSh95] Conell, J., Shafer, L.: Object-Oriented Rapid Prototyping, Yourdon, 1995

[CoYo94] Coad, P., Yourdon, E.: Objektorientiertes Design, Prentice Hall, 1994

[CoYo96] Coad, P., Yourdon, E.: Objektorientierte Analyse, 2. Auflage, Prentice Hall, 1996

[Cw98] Warehouse-Vater Inmon nimmt Kritiker ins Visier in: Computerwoche 33/1998, S. 16

[Dahr97] Dahr, M.: Deductive Databases — Theory and Applications, International Thomson, 1997

[Date00] Date, C.: An Introduction to Database Systems, 7. Auflage, Addison Wesley, 2000

[Dbms96] DBMS-Magazin: Interview, 2/1996, http://www.dbmsmag.com/-int9602.html, 22. 4. 2000

[DeLi91] DeMarco, T., Lister, T.: Wien wartet auf dich — Der Faktor Mensch im DV-Management, Hanser, 1991

[Dema79] DeMarco, T.: Structured Analysis and System Specification, Prentice Hall, 1979

[DeMu88] Devlin, B., Murphy, P.: An Architecture For a Business and Information System in: IBM Systems Journal, Volume 27, 1/1988

[Devl97] Devlin, B.: Data Warehouse — From Architecture to Implementation, Addison Wesley, 1997

[Dobe98] Dobenecker, G.: Interactive Data Marts — Wege zum anwendungsorientierten Knowledge Management in: Tagungsband Informationsmanagement für Business Intelligence und Excellence, Universität Hamburg, http://mbaix2.mba.uni-hamburg.de/iwi/tagungen-/wi98/kurz/doben.doc, 26. 2. 1998

[Dörn97] Dörner, D.: Die Logik des Mißlingens — Strategisches Denken in komplexen Situationen, Rohwolt, 1997

[Dude93] Duden-Redaktion: Duden Informatik, 2. Auflage, Duden, 1993

[Dude96] Duden-Redaktion: Duden — Rechtschreibung der deutschen Sprache, 21. Auflage, CD-ROM, Duden, 1996

[Dunn93] Dunn, R.: Software-Qualität — Konzepte und Pläne, Hanser/Prentice Hall, 1993

[Düsi99] Düsing, R.: Knowledge Discovery in Databases and Data Mining in: [ChGl99a], S. 345-353

[Edel97] Edelstein, H.: An Introduction to Data Warehousing in: [BaEd97], S. 31-50

[ElNa94] Elmasri, R., Navathe, S.: Fundamentals of Database Systems, 2. Auflage, Benjamin/Cummings, 1994

[Embl98] Embley, D.: Object Database Development — Concepts and Principles, Addison Wesley, 1998

[Expr93] IRI: Express-Introduction, Version 4.0, IRI, 1993

[FeSi90] Ferstl, O., Sinz, E.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM) in: Wirtschaftsinformatik, 32. Jahrgang, Heft 6, 12/1990

[FGJ+90] Fritchman, B., Guck, R., Jagannathan, D., Thompson, J., Tolbert, D.: SIM — Design and Implementation of a Semantic Database System in: [CáLe90a], S. 241-266

[FHS00] Frank, U., Hampe, F., Schwabe, G.: Wirtschaftsinformatik aus unserer Sicht, http://www.uni-koblenz.de/~iwi/allgemeines/-wi_aus_unserer_sicht.html, 12. 3. 2000

[FiHo98] Fitzgerald, B., Howcroft, D.: Competing Dichotomies In IS Research And Possible Strategies For Resolution in: Proceedings der ICIS '98 (International Conference on Information Systems) in Helsinki, 1998, S. 155-164

[Fink93] Finkenzeller, H.: Paradox für Windows — Professioneller Einsatz im Berufsalltag, Markt & Technik, 1993

[Flor00] Flores-Mendez, R.: Towards a Standardization of Multi-Agent System Frameworks, 1999, http://www.acm.org/crossroads/xrds5-4/-multiagent.html, 17. 4. 2000

[Fran99] Frank, U.: An Object-Oriented Architecture for Knowledge Management Systems, Arbeitsbericht des Instituts für Wirtschaftsinformatik, Nr. 16, Universität Koblenz-Landau, 1999

[Gaar00] Gaarder, J.: Sofies Welt, 3. Auflage, DTV, 2000

[Gabl97] Gabler-Redaktion: Gabler Wirtschaftslexikon, 14. Auflage, CD-ROM, Gabler, 1997

[Gems96a] GemStone Systems: GemStone Kernel Reference, Version 5.0, 1997

[Gems96b] GemStone Systems: GemStone Programming Guide, Version 5.0, 1997

[Gems96c] GemStone Systems: GemBuilder for VisualWorks, Version 5.0, 1997

[Gems96d] GemStone Systems: GemStone System Administration Guide — for Windows NT, Version 5.0, 1997

[Gens98] Gens, W. (Hrsg.): Tagungsband der STJA '98, 4. Fachkongreß Smalltalk und Java in Industrie und Ausbildung — Objektorientierte Sprachen, Konzepte und Methoden, 1998

[Gepp97] Geppert, A.: Objektorientierte Datenbanksysteme — Ein Praktikum, Dpunkt, 1997

[GeRo92] Gebert, D., v. Rosenstiel, L.: Organisationspsychologie, Kohlhammer, 1992

[GGC97] Gluchowsky, P., Gabriel, R., Chamoni, P. (Hrsg.): Management Support Systeme — Computergestützte Informationssysteme für Führungskräfte und Entscheidungsträger, Springer, 1997

[GHJV96] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Entwurfsmuster — Elemente wiederverwendbarer objektorientierter Software, Addison Wesley, 1996

[GKP92] Gray, P., Kulkarni, N., Paton, W.: Object-Oriented Databases — A Semantic Data Model Approach, Prentice Hall, 1992

[GlPr93] Gladis, R., Pretzsch, A.: CASE-Tools im Vergleich in: PC-Professionell, Nr. 3/1993, S. 202-242

[GoRo89] Goldberg, A., Robson, D.: Smalltalk-80 — The Language, Addison Wesley, 1989

[GoRu95] Goldberg, A., Rubin, K.: Succeeding with Objects — Decision Frameworks for Project Management, Addison Wesley, 1995

[GrWa98] Gray, P., Watson, H.: Decision Support in the Data Warehouse, Prentice Hall, 1998

[HaCa96] Hammer, H., Champy, J.: Business Reengineering — Die Radikalkur für das Unternehmen, 6. Auflage, Campus, 1996

[Hack97] Hackathorn, R.: Data Warehousing´s Credibility Crisis in: Byte, 8/1997, S. 43-44

[Hahn99] Hahne, M.: Logische Datenmodellierung für das Data Warehouse — Bestandteile und Varianten des Star Schemas in: [ChGl99a], S. 145-170

[Heue97] Heuer, A.: Objektorientierte Datenbanken, 2. Auflage, Addison Wesley, 1997

[HiJo96] Hinkle, B., Johnson, R.: The Active Life Is The Life For Me in: The Smalltalk Report, 5/1996, S. 14-20

[HLW99] Huang, K., Lee, Y., Wang, R.: Quality Information and Knowledge, Prentice Hall, 1999

[HoHo97] Hopkins, T., Horan, B.: Objektorientierte Programmierung mit Smalltalk — Einführung in die Applikationsentwicklung mit VisualWorks, Hanser/Prentice Hall, 1997

[Holt98] Holthuis, J.: Multidimensionale Datenstrukturen — Modellierung, Strukturkomponenten, Implementierungsaspekte in: [MuBe98], S. 143-193

[Holu99] Holub, A.: Buildung User Interfaces For Object-Oriented Systems, Part 1 — What Is An Object? The Theory Behind Building Object-Oriented User Interfaces, http://www.javaworld.com/javaworld/-jw-07-1999/jw-07-toolbox.html, 13. 7. 1999

[Howa95] Howard, T.: The Smalltalk Developer´s Guide to VisualWorks, SIGS, 1995

[HoWh96] Holsapple, C., Whinston, A.: Decision Support Systems — A Knowledge-based Approach, West, 1996

[Huff97] Hufford, D.: Metadata Repositories — The Key to Unlocking Information in Data Warehouses in: [BaEd97], S. 225-262

[Hugh92] Hughes, J.: Objektorientierte Datenbanken, Hanser/Prentice Hall, 1992

[HuKi90] Hull, R., King, R.: A Tutorial on Semantic Database Modeling in: [CáLe90a], S. 1-33

[Humm97] Hummeltenberg, W.: Statistik-Programmpaket in [MBB+], S. 381-382

[InHa94] Inmon, W., Hackathorn, R.: Using the Data Warehouse, John Wiley & Sons, 1994

[Inmo96] Inmon, W.: Building the Data Warehouse, 2. Auflage, John Wiley & Sons, 1996

[Inmo99] Inmon, W.: Building the Operational Data Store, John Wiley & Sons, 1999

[IWG97] Inmon, W., Welch, J., Glassey, K.: Managing the Data Warehouse, John Wiley & Sons, 1997

[Jani82] Janis, I.: Groupthink — Psychological Studies of Policy Decisions and Fiascoes, 2. Auflage, Hougthon Mifflin, 1982

[JCJÖ95] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Engineering — A Use Case Driven Approach, Neuauflage, ACM/Addison Wesley, 1995

[JJQ+00] Jarke, M., Jeusfeld, M., Quix, C., Sellis, T., Vassiliadis, P.: Metadata and Data Warehouse Quality in: [JLVV00], S. 122-158

[JLVV00] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P. (Hrsg.): Fundamentals of Data Warehouses, Springer, 2000

[John99] Johnson, R.: Lectures And Notes For CS497REJ: Object-Oriented Programming and Design, http://st-www.cs.uiuc.edu/users/johnson/-cs497/notes98/online-course.html, 24. 11. 1999

[Kage84] Kageyama, T.: Lessons in the Fundamentals of GO, 2. Auflage, Ishi, 1984

[KaSc96] Kappel, G., Schrefl, M.: Objektorientierte Informationssysteme — Konzepte, Darstellungsmittel, Methoden, Springer, 1996

[Kay93] Kay, A.: The Early History of Smalltalk in: ACM SIGPLAN Notices (History of Programming Languages-II), Volume 28, Nr. 3, 3/1993, S. 69-95

[Kell96] Kelly, S.: Data Warehousing — The Route to Mass Customization, John Wiley & Sons, 1996

[KeMo94] Kemper, A., Moerkotte, G.: Object-Oriented Database Management — Applications in Engineering and Computer Science, Prentice Hall, 1994

[KGZ94] Kilberth, K., Gryczan, G., Züllighoven, H.: Objektorientierte Anwendungsentwicklung — Konzepte, Strategien, Erfahrungen, 2. Auflage, Vieweg, 1994

[KhAb95] Khoshafian, S., Abnous, R.: Object Orientation — Concepts, Analysis and Design, Languages, Databases, Graphical User Interfaces, Standards, 2. Auflage, John Wiley & Sons, 1995

[Khos93] Khoshafian, S.: Object-Oriented Databases, John Wiley & Sons, 1993

[KIKT99] Künzel, R., Ipsen, J., Kambas, C., Trapp, H. (Hrsg.): Profile der Wissenschaft — 25 Jahre Universität Osnabrück, Rasch, 1999

[KiLo89] Kim, W., Lochovsky, F.: Object-Oriented Concepts, Databases, and Applications, ACM/Addison-Wesley, 1989

[Kimb96] Kimball, R.: The Data Warehouse Toolkit — Practical Techniques for Building Dimensional Data Warehouses, John Wiley & Sons, 1996

[Kimb97] Kimball, R.: Relocating the ODS — Moving the Operational Data Store Will Solve a Number of Problems, http://dbmsmag.com/-9712d05.html, 22. 9. 1998

[King89] King, R.: My Cat Is Object-Oriented in: [KiLo89], S. 23-30

[King96] King, D.: Intelligent Support Systems in: [SpWa96], S. 182-208

[Klot93] Klotz, M.: Integrierte Anwendungssoftware und Unternehmensorganisation — Ein neues aufbauorganisatorisches Konzept zum Abbau von Hierarchieebenen, Erich Schmidt, 1993

[KMS97] Krallmann, H., Mertens, P., Schiemann, I.: Entscheidungsunterstützendes System (EUS) in: [MBB+97], S. 149-150

[Knig96] Knight, A.: Principles of OO design — or, everything I needed to know in life, I learned from Dilbert in: The Smalltalk Report, 2/1996, S. 13-15

[KnJo98] Knapik, M., Johnson, J.: Developing Intelligent Agents for Distributed Systems — Exploring Architecture, Technologies, and Applications, McGraw-Hill, 1998

[KoLo95] Koch, G., Loney, K.: Oracle — The Complete Reference, 3. Auflage, McGraw-Hill, 1995

[Kral90] Krallmann, H. (Hrsg.): Innovative Anwendungen der Informations- und Kommunikationstechnologie der 90'er Jahre, Oldenbourg, 1990

[Kral94] Krallmann, H.: Systemanalyse im Unternehmen, Oldenbourg, 1994

[KrRi87] Krallmann, H., Rieger, B.: Vom Decision Support System (DSS) zum Executive Support System (ESS) in: HMD, Heft 138, 1987, S. 28-38

[Kurb97] Kurbel, K.: Wirtschaftsinformatik in: [Gabl97]

[LaLa95] Laudon, K., Laudon, J.: Information Systems — A Problem-Solving Approach, 3. Auflage, The Dryden Press, 1995

[LaLa96] Laudon, K., Laudon, J.: Management Information Systems — Organization and Technology, 4. Auflage, Prentice Hall, 1996

[LaLo95] Lang, S., Lockemann, P.: Datenbankeinsatz, Springer, 1995

[Lars95] Larson, J.: Database Directions — From Relational to Distributed, Multimedia, and Object-Oriented Database Systems, Prentice Hall, 1995

[LaVo96] Lausen, G., Vossen, G.: Objektorientierte Datenbanken — Modelle und Sprachen, Oldenbourg, 1996

[Lehn97] Lehner, F.: Wirtschaftsinformatik, Forschungsgegenstände und Erkenntnisverfahren (Information Systems Research, Research Questions and Scientific Methods) in: [MBB+97], S. 438-439

[Lehr00] Lehre-Datenbank des Fachgebiets BWL/Wirtschaftsinformatik II, Universität Osnabrück, http://www.oec.uos.de/lehredb, 22. 4. 2000

[Lewi95] Lewis, S.: The Art and Science of Smalltalk, Prentice Hall, 1995

[Liu96] Liu, C.: Smalltalk, Objects, and Design, Manning, 1996

[Loom95] Loomis, M.: Object Databases — The Essentials, Addison Wesley, 1995

[Lore95] Lorenz, M.: Rapid Software Development with Smalltalk, SIGS, 1995

[MaMa97] Martin, W., v. Maur, E.: Data Warehouse in: [MBB+97], S. 105-106

[Mano94] Manola, F.: An Evaluation of Object-Oriented DBMS Developments, Edition 1994, ftp://ftp.gte.com/pub/dom/reports/MANO94a.pt1.ps, ftp://ftp.gte.com/pub/dom/reports/MANO94a.pt2.ps, 31. 8. 1994

[Mant99] Manthey, A.: Knowledge Management — Perspektiven und Technologien, White Paper, Oracle, 3/1999

[Mart97] Martin, W.: Die Zukunft hat bereits begonnen, IS Report, 9/1997, S. 24-27

[Mart98a] Martin, W. (Hrsg.): Data Warehousing — Data Mining — OLAP, International Thomson, 1998

[Mart98b] Martin, W.: Data Warehouse, Data Mining, OLAP — Von der Datenquelle zum Informationsverbraucher in: [Mart98a], S. 17-37

[Mart98c] Martin, W.: Vom Data Warehouse zum Knowledge Warehouse in: [Mart98a], S. 415-424

[Maur99] v. Maur, E.: Object Warehouse zur MSS-adäquaten Informations-Modellierung am Beispiel der Standardsoftware Baan in: [BEGR99], S. 69-76

[MBB+00] Mertens, P. (Haupt-Hrsg.), Back, A., Becker, J., König, W., Krallmann, H., Rieger, B., Scheer, A., Seibt, D., Stahlknecht, P., Strunz, H., Thome, R., Wedekind, H. (Hrsg.): Lexikon der Wirtschaftsinformatik, 4. Auflage, Springer, 2000, in Vorbereitung, Vorveröffentlichung neuer Stichworte: http://www.wi1.uni-erlangen.de/buecher/-lexikon/

[MBB+97] Mertens, P. (Haupt-Hrsg.), Back, A., Becker, J., König, W., Krallmann, H., Rieger, B., Scheer, A., Seibt, D., Stahlknecht, P., Strunz, H., Thome, R., Wedekind, H. (Hrsg.): Lexikon der Wirtschaftsinformatik, 3. Auflage, Springer, 1997

[Mcgu98] McGuff, F.: Data Modeling for Data Warehouses — Designing the Perfect Data Warehouse, 2/1998, http://members.aol.com/fmcguff/-dwmodel/index.htm, 14. 4. 1999

[Mdc00] Metadata Coalition, http://www.mdcinfo.com, 22. 3. 2000

[MeGr93] Mertens, P., Griese, J.: Integrierte Informationsverarbeitung (Band 2) — Planungs- und Kontrollsysteme in der Industrie, 7. Auflage, Gabler, 1993

[Meie98] Meier, M.: Integration von Informationen aus dem Internet in das SAP-EIS — Konzeption und prototypische Implementierung eines Redaktions-Leitstandes in: [UhBr98], S. 105-122

[Mert93] Mertens, P.: Integrierte Informationsverarbeitung (Band 1) — Administrations- und Dispositionssysteme in der Industrie, 9. Auflage, Gabler, 1993

[Mert98a] Mertens, P.: Konzeption für ein Data Warehouse — Unternehmensdatenmodell als Basis in: Datenbank Fokus, 1/1998, S. 54-63

[Mert98b] Mertens, P.: Das Data Warehouse und seine Inhalte — Die Gefahr der Terabyte-Schrotthalden in: Datenbank Fokus, 5/1998, S. 56-74

[Mert98c] Mertens, P.: Integration interner, externer, qualitativer und quantitativer Daten auf dem Weg zum Aktiven MIS in: [UhBr98], S. 9-30

[Meye90] Meyer, B.: Objektorientierte Softwareentwicklung, Hanser/Prentice Hall, 1990

[Mik99] http://www.mik.de, 23. 9. 1999

[Mins88] Minsky, M.: The Society of Mind, Touchstone, 1988

[Mint91] Mintzberg, H.: Mintzberg über Management — Führung und Organisation, Mythos und Realität, Gabler, 1991

[MRB98] Martin, R., Riehle, D., Buschmann, F. (Hrsg.): Pattern Languages of Program Design 3, PLoP '96 und EuroPLoP '96, Addison Wesley, 1998

[MSOP90] Maier, D., Stein, J., Otis, A., Purdy, A.: Development of an Object-Oriented DBMS in: [CáLe90a], S. 200-220

[MuBe96] Mucksch, H., Behme, W. (Hrsg.): Das Data Warehouse-Konzept — Architektur, Datenmodelle, Anwendungen mit Erfahrungsberichten, 1. Auflage, Gabler, 1996

[MuBe98] Mucksch, H., Behme, W. (Hrsg.): Das Data Warehouse-Konzept — Architektur, Datenmodelle, Anwendungen mit Erfahrungsberichten, 3. Auflage, Gabler, 1998

[Muck99] Muksch, H.: Das Data Warehouse als Datenbasis analytischer Informationssysteme — Architektur und Komponenten in: [ChGl99a], S. 171-189

[Nast97] Nastansky, L.: Tabellenkalkulationssystem in: [MBB+], S. 397-399

[Nieb99] Niebuhr, B.: Implementierung eines objektorientierten Vertriebsinformationssystems für einen Landmaschinenhersteller auf Basis von Smalltalk und GemStone und der Standardsoftware Baan, Diplomarbeit an der Universität Osnabrück, 1999, unveröffentlicht

[Nier89] Nierstrasz, O.: A Survey of Object-Oriented Concepts in: [KiLo89], S. 3-21

[NoTa95] Nonaka, I., Takeuchi, H.: The Knowledge-Creating Company, Oxford University, 1995

[Obje97] Objectivity: Using Objectivity/Smalltalk, Version 4, Release 4, 1997

[OdGr98] O'Dell, C., Grayson, J.: If Only We Knew What We Know — The Transfer of Internal Knowledge and Best Practice, The Free Press, 1998

[OHE96] Orfali, R., Harkey, D., Edwards, J.: The Essential Client/Server Survival Guide, 2. Auflage, John Wiley & Sons, 1996

[Ohle98] Ohlendorf, T.: Objektorientierte Datenbanksysteme für den Einsatz im Data Warehouse-Konzept in: [MuBe98], S. 219-244

[PaDi95a] ParcPlace-Digitalk: VisualWorks — User's Guide, Revision 2.0, 1995

[PaDi95b] ParcPlace-Digitalk: VisualWorks — Cookbook, Revision 2.0, 1995

[Parc94] ParcPlace Systems: VisualWorks' Database Tools — Tutorial and Cookbook, Revision 1.1, 1995

[Parc95] ParcPlace Systems: VisualWorks — Object Reference, Revision 1.1, 1995

[Pete98] Peterson, S.: Stars: A Pattern Language for Query-Optimized Schemas in: [CoSc98], S. 163-177

[BlPr98] Blaha, M., Premerlani, W.: Object-Oriented Modeling and Design for Database Applications, Prentice Hall, 1998

[PoBl96] Pomberger, G., Blaschek, G.: Object Orientation and Prototyping in Software Engineering, Prentice Hall, 1996

[Poe96] Poe, V.: Building a Data Warehouse for Decision Support, Prentice Hall, 1996

[Popp94] Popper, K.: Ausgangspunkte — Meine intellektuelle Entwicklung, 2. Auflage, Hoffmann und Campe, 1994

[Popp95] Popper, K.: Alles Leben ist Problemelösen, 6. Auflage, Piper, 1995

[Post00] Postert, S.: Gestaltungspotenzial eines MSS-gestützten Hochschul-Controlling am Beispiel der Universität Osnabrück, Dissertation an der Universität Osnabrück, unveröffentlichtes Manuskript, 2000

[PrBü94] Probst, G., Büchel, B.: Organisationales Lernen — Wettbewerbsvorteil der Zukunft, Gabler, 1994

[PRR99] Probst, G., Raub, S., Romhardt, K.: Wissen managen — Wie Unternehmen ihre wertvolle Ressource nutzen, 3. Auflage, Gabler, 1999

[Quot99] http://www.quoteland.com/quotes/author/1.html, 26. 12. 1999

[Raut97] Rautenstrauch, C.: Modellierung und Implementierung von Data-Warehouse-Systemen, Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, 1997

[RBK+98] Rieger, B., Brodmann, K., Krüger, D., v. Maur, E., Postert, S.: Uni-Web — Ein integratives Konzept zur datenbank-gestützten Verwaltung, Navigation und Distribution multimedialer Informationsobjekte in: [UhBr98], S. 137-150

[RBP+93] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Objektorientiertes Modellieren und Entwerfen, Hanser/Prentice Hall, 1993

[RDM00] Rieger, B., Dalinghaus, H., v. Maur, E.: Objektorientiertes dynamisches Web-Publishing mit semistrukturierten Daten, NetObjectDays 2000, http://www.netobjectdays.org/pdf/00/papers/ooss/rieger.pdf

[Reen96] Reenskaug, T.: Working with Objects — The OOram Software-Engineering Method, Manning, 1996

[RiBr99] Rieger, B., Brodmann, K.: Mastering Time Variances of Dimension Tables in the Data Warehouse — Implementation Concept Based on a Practical Evaluation of Modeling Approaches in Theory, Universität Osnabrück, 1999, http://andromeda.oec.uni-osnabrueck.de/uwdstift/DMOS-Res.pdf, 22. 3. 2000

[Rieg85] Rieger, B.: Leistungsfähigkeit von Entscheidungsunterstützungssystemen im Unternehmen — dargestellt am Beispiel der Preisgestaltung von Mehrproduktunternehmen, Dissertation, TU-Berlin, 1985

[Rieg90] Rieger, B.: Executive Information Systems (EIS): Rechnergestützte Aufbereitung von Führungsinformationen in: [Kral90], S. 103-125

[Rieg93] Rieger, B.: Der rechnerunterstützte Arbeitsplatz für Führungskräfte, Habilitationsschrift, TU-Berlin, 1993

[Rieg96] Rieger, B.: OLAP — Stand der Forschung und Entwicklung in: Tagungsband Online-Hamburg '96, 1996

[Rieg97] Rieger, B.: Führungsinformationssysteme in Standard-Softwarepaketen — Welchen Nutzen ziehen Manager aus integrierten Systemen? in: Industrie-Management, Nr. 13, 3/1997, S. 8-12

[Rieg99] Rieger, B.: Potentiale und kritische Erfolgsfaktoren der Informations- und Kommunikationstechnologie zur betriebswirtschaftlichen Evolution von Unternehmungen am Beispiel der Universität Osnabrück in: [KIKT99], S. 283-297

[Riel96] Riel, A.: Object-Oriented Desgin Heuristics, Addison Wesley, 1996

[RiKl00] Rieger, B., Kleber, A.: Semantic Integration of Heterogeneous Information Sources: Experiences from a MSS Case in: Roantree, M., Hasselbring, W., Conrad, S. (Hrsg.): Engineering Federated Information Systems, Proceedings of the 3rd Workshop EFIS 2000, June 19-20, Dublin (Ireland), Infix-Verlag, S. 89-100

[RiMa00] Rieger, B., v. Maur, E.: Object Warehouse in: [MBB+00], erscheint demnächst

[RKM00] Rieger, B., Kleber, A., v. Maur, E.: Metadata-Based Integration of Qualitative and Quantitative Information Resources Approaching Knowledge Management, Proceedings der ECIS 2000 (European Conference on Information Systems) in Wien, S. 372-378

[RKM98] Rieger, B., Krüger, K., v. Maur, E.: Das Morph-Paradigma zur Überwindung von Akzeptanzbarrieren bei Smalltalkprojekten in Praxis und Lehre in: [Gens98], S. 238-246

[RMK98] Rieger, B., v. Maur, E., Krüger, K.: Kritische Erfolgsfaktoren für Projektkooperationen zur Entwicklung objektorientierter Management Support Systems — Ein Erfahrungsbericht in: [Gens98], S. 247-256

[RMK99] Rieger, B., v. Maur, E., Krüger, K.: Constraints persistenter Geschäftsobjekte im Smalltalk-basierten Object Warehouse in: Tagungsband der STJA '99, CD-ROM, 1999

[RMP99] Rieger, B., v. Maur, E., Postert, S.: Object Warehouse — Rekonzilierte Entscheidungsbasis des Decision Supports in: 2. Workshop Data Mining und Data Warehousing als Grundlage moderner entscheidungsunterstützender Systeme, LWA '99-Sammelband, 1999, S. 97-108

[RoDe88] Rockart, F., DeLong, D.: Executive Support Systems — The Emergence of Top Management Computer Use, Dow Jones-Irwin, 1988

[RuGo92] Rubin, K., Goldberg, A.: Object Behavior Analysis in: Communications of the ACM, Volume 35 (9), 9/1992, S. 48-62

[Rusk00] Ruske, N.: Wiederverwendung fördernde Dokumentation objektorientierter Anwendungsentwicklung am Beispiel eines Liegenschaftsverwaltungssystems, Lang, 2000

[Saxe96] Saxer, M.: Die „Zehn Gebote“ für Data-Warehouse-Projekte in: Computerwoche, Nr. 27, 1996, S. 49-50

[ScBa99] Schinzer, H., Bange, C.: Werkzeuge zum Aufbau analytischer Informationssysteme — Marktübersicht in: [ChGl99a], S. 45-74

[ScHa92] Scheer, A., Hars, A.: Extending Data Modeling to Cover the Whole Enterprise in: Communications of the ACM, Nr. 35, 9/1992, S. 166-172

[Sche88] Scheer, A.: Wirtschaftsinformatik — Informationssysteme im Industriebetrieb, Springer, 1988

[Sche90a] Scheer, A.: EDV-orientierte Betriebswirtschaftslehre — Grundlagen für ein effizientes Informationsmanagement, 4. Auflage, Springer, 1990

[Sche90b] Scheer, A.: CIM: Computer Integrated Manufacturing — Der computergesteuerte Industriebetrieb, 4. Auflage, Springer, 1990

[Sche97a] Schein, E.: Organizational Culture and Leadership, 2. Auflage, Jossey-Bass, 1992

[Sche97b] Scheer, A.: Unternehmensdatenmodell in: [MBB+97], S. 417-419

[Sche98a] Scheer, A.: ARIS — Modellierungsmethoden, Metamodelle, Anwendungen, 3. Auflage, Springer, 1998

[Sche98b] Scheer, A.: ARIS — Vom Geschäftsprozeß zum Anwendungssystem, 3. Auflage, Springer, 1998

[Schi97] Schinzer, H.: Entscheidungsorientierte Informationssysteme im Vergleich (Teil 1) — Labortest — Vorgehen und Vergleichskriterien in: IS Report, 9/1997, S. 4-11

[Schm91] Schmidt, D.: Persistente Objekte und objektorientierte Datenbanken — Konzepte, Architektur, Implementierung und Anwendung, Dissertation an der Universität Zürich, Hanser, 1991

[ScKa97] Schiffman, L., Kanuk, L.: Consumer Behavior, 6. Auflage, Prentice Hall, 1997

[Scot83] Scott Morton, M.: State of the Art of Research in Management Support Systems, Working Paper, Nr. 107, MIT, Center for Information Systems Research, 1983

[Seng90] Senge, P.: The Fifth Discipline — The Art and Practice of The Learning Organization, Doubleday, 1990

[Shar97] Sharp, A.: Smalltalk by Example — The Developer's Guide, McGraw-Hill, 1997

[ShMe96] Shlaer, S., Mellor, S.: Objektorientierte Systemananlyse — Ein Modell der Welt in Daten, Hanser/Prentice Hall, 1996

[ShMe98] Shlaer, S., Mellor, S.: Objekte und ihre Lebensläufe — Modellierung mit Zuständen, Hanser/Prentice Hall, 1998

[Shoh93] Shoham, Y.: Agent-Oriented Programming in: Artificial Intelligence, Volume 60, Nr. 1, 1993, S. 51-92

[SIG97] Silverston, L., Inmon, W., Graziano, K.: The Data Model Resource Book — A Library of Logical Data and Data Warehouse Design, John Wiley & Sons, 1997

[Sinz90] Sinz, E.: Das Entity-Relationship-Modell (ERM) und seine Erweiterungen in: HMD, Heft 152, 1990

[Soef99] Soeffky, M.: Knowledge Management — Ein weiteres Schlagwort der Informationstechnologie? in: IT Fokus, 2/1999, S. 22-28

[SpWa96] Sprague, R., Watson, H. (Hrsg.): Decision Support for Management, Prentice Hall, 1996

[SST97] Saake, G., Schmitt, I., Türker, C.: Objektdatenbanken — Konzepte, Sprachen, Architekturen, International Thomson, 1997

[Stac73] Stachowiak, H.: Allgemeine Modelltheorie, Springer, 1973

[Stae91] Staehle, W.: Management — Eine verhaltenswissenschaftliche Perspektive, 6. Auflage, Vahlen, 1991

[Stah73] Stahlknecht, P.: Erfahrungen über den Einsatz computergestützter Informationssysteme in: Industrial Engineering, Heft 2, 3/1973, S. 93-97

[Stah97] Stahlknecht, P.: Management-Informationssystem (MIS) in: [MBB+97], S. 247-248

[StHa97] Stahlknecht, P., Hasenkamp, U.: Einführung in die Wirtschaftsinformatik, 8. Auflage, 1997

[Stif99] Web-Site des Stifterverbandsprojekts zur Entwicklung eines MIS für die Universität Osnabrück, http://www.oec.uos.de/stift, 24. 12. 1999

[Stro88] Stroustrup, B.: What Is Object-Oriented Programming? in: IEEE Software, Volume 5 (3), S. 10-20

[Tayl92] Taylor, D.: Object-Oriented Information Systems — Planning and Implementation, John Wiley & Sons, 1992

[Thom97] Thomsen, E.: OLAP Solutions — Building Multidimensional Information Systems, John Wiley & Sons, 1997

[ToSc89] Tomlinson, C., Scheevel, M.: Concurrent Object-Oriented Programming Languages in: [KiLo89], S. 79-124

[Tulk99] Tulku, T.: Dynamik von Zeit und Raum — Wege zur Öffnung der Grenzen von Erkenntnis, Dharma, 1999

[Turb95] Turban, E.: Decision Support and Expert Systems — Management Support Systems, 4. Auflage, Prentice Hall, 1995

[UhBr98] Uhr, W., Breuer, S. (Hrsg.): Tagungsband der Wirtschaftsinformatik Fachtagung Integration externer Informationen in Management Support Systems, Technische Universität Dresden, 1998

[VCK98] Vlissides, J., Coplien, J., Kerth, N. (Hrsg.): Pattern Languages of Program Design 2, PLoP '95, 4. Auflage, Addison Wesley, 1998

[Vets95] Vetschera, R.: Informationssysteme der Unternehmensführung, Springer, 1995

[Vett95] Vetter, M.: Objektmodellierung — Eine Einführung in die objektorientierte Analyse und das objektorientierte Design, Teubner, 1995

[Voss94] Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme, 2. Auflage, Addison Wesley, 1994

[Walp93] Walpoth, G.: Computergestützte Informationsbedarfsanalyse — Strategische Planung und Durchführung von Informatikprojekten, Physica, 1993

[WaOe97] Wallrabe, A., Oestereich, B.: Smalltalk — für Ein- und Umsteiger, Oldenbourg, 1997

[Wegn87] Wegner, P.: Dimensions of Object-Based Language Design in: ACM SIGPLAN Notices, OOPSLA '87 Conference Proceedings, Volume 22, Nr. 12, 12/1987, S. 168-182

[Weus99] Weustink, M.: Entwicklung eines objektorientierten Vertriebsinformationssystems für einen Landmaschinenhersteller auf Basis von Smalltalk und GemStone und der Standardsoftware Baan, Diplomarbeit an der Universität Osnabrück, 1999, unveröffentlicht

[WhDa98] Whitener, A., Davis, J.: Minding Your Business with MapInfo, Onword, 1998

[Whit99] White, C.: The IBM Business Intelligence Software Solution, Version 3, 3/1999, http://www.software.ibm.com/data/pubs/papers/-bisolution/index.html, 25. 6. 1999

[WHR97] Watson, H., Houdeshel, G., Rainer, K.: Building Executive Information Systems — and other Decision Support Applications, John Wiley & Sons, 1997

[Wilk95] Wilkinson, N.: Using CRC Cards — An Informal Approach to Object-Oriented Development, SIGS, 1995

[Wiss99] Tagung Wirtschaftsinformatik und Wissenschaftstheorie 1999, http://caladan.wiwi.uni-frankfurt.de/IWI/, 1. 5. 2000

[WJK99] Wooldridge, M., Jennings, N., Kinny, D.: A Methodology for Agent-Oriented Analysis and Design in: Proceedings of the Third International Conference on Autonomous Agents (Agents-99), S. 69-76

[WKK+93] Wittmann, W., Kern, W., Köhler, R., Küpper, H., v. Wysocki, K. (Hrsg.): Handwörterbuch der Betriebswirtschaftslehre, 5. Auflage, 1993

[Wöhe91] Wöhe, G.: Einführung in die Allgemeine Betriebswirtschaftslehre, 17. Auflage, Vahlen, 1991

[Wool94] Woolf, B.: Understanding and Using the ValueModel Framework in VisualWorks Smalltalk, 1994 in: [CoSc98], S. 467-494

[WRK91] Watson, H., Rainer, K., Koh, C.: Executive Information Systems — A Framework for Development and a Survey of Current Practices in: MIS Quarterly, Volume 15, Nr. 1, 3/1991, S. 13-30

[WWW90] Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software, Prentice Hall, 1990

[Yip99] Yip, J.: Summary of eXtreme Programming, http://www.sern.enel.ucalgary.ca/yip/summaries/eXtreme.htm, 18. 09. 1999

[YoAr96] Yourdon, E., Argila, C.: Case Studies in Object-Oriented Analysis and Design, Yourdon, 1996

[Your93] Yourdon, E.: Die westliche Programmierkunst am Scheideweg — Die Schlüsseltechniken der Softwareentwicklung für das 21. Jahrhundert, Hanser/Prentice Hall, 1993

[Zach97] Zachman, J.: Vorwort in: [BiAl97], S. VII-X