

Andreas Pöschek

# Objektorientierte **Datenbanken**

Diplomarbeit

# Inhaltsverzeichnis

<b>1</b>	<b>EINLEITUNG .....</b>	<b>3</b>
<b>2</b>	<b>GRUNDPRINZIPIEN VON OBJEKTORIENTIERTEN SYSTEMEN UND DATENBANKEN .....</b>	<b>3</b>
2.1	WAS SIND OBJEKTE? .....	3
2.2	DAS PRINZIP DER KAPSELUNG .....	3
2.3	DIE VERERBUNGSEIGENSCHAFT UND SUBKLASSEN .....	3
2.3.1	<i>Overriding</i> .....	4
2.4	TYPES UND DATENTYPEN .....	4
2.5	IDENTITÄT .....	4
<b>3</b>	<b>EIGENSCHAFTEN UND KONZEPT OBJEKTORIENTIERTER DATENBANKEN .....</b>	<b>5</b>
<b>4</b>	<b>DAS MODELL DER OODB .....</b>	<b>6</b>
4.1	DER STRUKTURTEIL .....	6
4.1.1	<i>Typenkonstruktoren und komplexe Objekte</i> .....	6
4.1.2	<i>Objektidentität</i> .....	8
4.1.3	<i>Klassen und Typen</i> .....	10
4.1.4	<i>Beziehungen zwischen Klassen</i> .....	11
4.1.5	<i>Strukturvererbung</i> .....	12
4.1.6	<i>Integritätsbedingungen</i> .....	12
4.2	DER OPERATIONENTEIL .....	13
4.2.1	<i>Generische Operationen</i> .....	13
4.2.2	<i>Relationale Operationen</i> .....	13
4.2.3	<i>Objekterzeugende Operationen</i> .....	13
4.2.4	<i>Objekterhaltende Operationen</i> .....	13
4.3	METAKLASSEN .....	14
<b>5</b>	<b>DER ODMG-STANDARD .....</b>	<b>15</b>
5.1	DAS ODMG-OBJEKTMODELL .....	15
5.2	ODMG-ODL .....	15
5.3	ODMG-OQL .....	16
5.3.1	<i>Grundprinzip einer Abfrage</i> .....	16
5.3.2	<i>Semantik einer Anfrage</i> .....	17
5.4	TRANSAKTIONEN .....	17
5.5	KRITIK AM ODMG-STANDARD .....	18
5.6	DIE SQL-3-NORM .....	18
5.6.1	<i>Abstrakte Datentypen</i> .....	19
5.6.2	<i>Einkapselung und Funktionen</i> .....	19
5.6.3	<i>Typenkonstruktoren</i> .....	19
5.6.4	<i>Typ- und Tabellenhierarchie</i> .....	19
5.6.5	<i>Anfragen</i> .....	20
<b>6</b>	<b>OBJECT ORIENTATED DATABASE SYSTEM .....</b>	<b>21</b>
6.1	ANFORDERUNGEN AN OODBS .....	21
6.1.1	<i>Erweiterbarkeit</i> .....	21
6.1.2	<i>Persistenz</i> .....	21
6.2	INTERNE EBENE UND SPEICHERSTRUKTUR EINES OODBS .....	21
6.3	TRANSAKTIONEN, CONCURRENCY CONTROL .....	22
6.3.1	<i>Concurrency Control</i> .....	22
6.4	RECOVERY .....	22
6.5	VERSIONEN-MANAGEMENT .....	22

<b>7</b>	<b>ANHANG .....</b>	<b>23</b>
7.1	GLOSSAR.....	23
7.2	VERWENDETE NOTATION.....	23
7.3	LITERATURVERZEICHNIS.....	24
7.4	WEITERFÜHRENDE LITERATUR UND INFORMATIONSMATERIAL.....	24

Copyright© 2000 by Andreas Pöschek. Alle Rechte vorbehalten. Dieses Schriftstück dient als wissenschaftliche Arbeit.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Kein Teil dieses Dokumentes darf ohne schriftliche Genehmigung des Autors in irgendeiner Form durch Fotokopie, Mikrofilm, elektronische Datenverarbeitung, Drucker oder andere Verfahren reproduziert oder in eine für Maschinen, insbesondere Datenverarbeitungsanlagen, verwendbare Sprache übertragen werden. Auch die Rechte der Wiedergabe durch Vortrag, Funk und Fernsehen sind vorbehalten. Die in diesem Werk erwähnten Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Marken und unterliegen als solche den gesetzlichen Bestimmungen. Das Dokument darf nur nach schriftlicher Genehmigung seitens des Autors im Internet, (Web, Newsgroups, Fileservers, Bulletin-Boards, etc.) und anderen Netzen, sowohl öffentlich als auch im eingeschränkten Kreis, für dritte Personen bereitgestellt (zum Download etc.) bzw. veröffentlicht werden. Hierbei ist eine Abänderung des Dokumentes nicht gestattet. Ausgenommen von den Bestimmungen sind wissenschaftliche Arbeiten bzw. die Lehre und Bildung an Schulen wie AHS, HTL, etc. bei Nennung des Autors und der Herkunft des Textes.

Andreas Pöschek  
Informatik – Wirtschaftsingenieur  
Floßgasse 3 / 4, A-1020 Wien, Österreich  
andreas@poeschek.com

Informationen und weitere Werke im Internet unter <http://www.poeschek.com>

# 1 Einleitung

Alle angeführten Beispiele sind in einer Notation von Prof. Dr. Andreas Heuer<sup>1</sup> dargestellt. Eine kurze Erklärung zur Notation ist im Anhang zu finden. Beispiele zur Implementierung in Form von Sourcecode sind in der Sprache von OODBS O<sub>2</sub> angeführt.

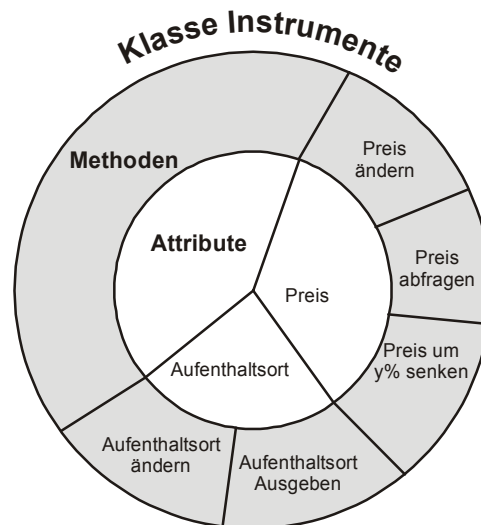
## 2 Grundprinzipien von objektorientierten Systemen und Datenbanken

### 2.1 Was sind Objekte?

Objekte sind Grundbausteine, aus welchen objektorientierte Anwendungssysteme aufgebaut werden. Jedes Objekt ist eine eigenständige Einheit, die über *Daten* (*Attributswerte* des Objektes) und *Verhalten* (*Methoden*<sup>2</sup> oder Operationen des Objekts) verfügt. Objekte mit identischer Struktur und gleichem Verhalten werden zu Klassen abstrahiert. Auf die Daten (Attribute) innerhalb eines Objektes kann nur mithilfe der definierten Methoden zugegriffen werden. Ein direkter Zugriff auf Daten ist somit von außen nicht möglich. Attribute und Methoden werden auch als *Komponente* eines Objekttypes bezeichnet.

### 2.2 Das Prinzip der Kapselung

Die eigentlichen Attributswerte sind dem Anwender unzugänglich, außer er arbeitet mit den vorgeschriebenen Methoden. Die Kapselung verhindert unzulässige Zustandsänderungen von Objekten und führt zu einer wünschenswerten Entkoppelung von Anwendern dieser Klasse und interner Klassendetails.



### 2.3 Die Vererbungseigenschaft und Subklassen

Ein weiteres Grundprinzip der Objektorientierung bildet die Vererbung bezüglich einer Klassenhierarchie. Sind die Objekte einer Klasse z.B. Blasinstrument spezieller als die einer anderen Klasse z.B. Instrument, so kann man die Blasinstrumente als Unterklasse von Instrumenten definieren. Dies bedeutet, dass die Attributswerte und das Verhalten von Instrument an Blasinstrument vererbt werden – jedes Blasinstrument besitzt die selben Eigenschaften der Instrumente, jedoch haben Blasinstrumente weitere zusätzliche spezifischere Eigenschaften, die nur für sie gelten.

Eine Klasse kann eine beliebige Anzahl an Subklassen (Unterklassen) haben, die wiederum Subklassen aufweisen. Vererbungseigenschaften gelten sowohl für Attribute als auch für Methoden der Klassen einer Klassenhierarchie. Die Vererbungseigenschaft ist eine der wichtigsten Grundprinzipien objektorientierter Entwicklung.

<sup>1</sup> Siehe [HEU97]

<sup>2</sup> Methoden werden oft als Routinen bezeichnet.

### 2.3.1 Overriding

Methoden können bei der Vererbung durch Methoden der Subklasse überschrieben werden. So kann z.B. die Klasse Blasinstrumente eine andere Methode zur Preisermittlung haben, als allgemein die Instrumentenklasse.

## 2.4 Types und Datentypen

In objektorientierten Datenbanksystemen sind nicht nur Standard-Datentypen für Eigenschaften von Objekten erlaubt, sondern auch die wiederholte Anwendung von Typenkonstruktoren.

Beispiel für ein Objekt in O<sub>2</sub>:

**CLASS** Personen

```

TYPE          TUPLE(Name:TUPLE(Vorname:      'Franz',
                                   Nachname: 'Eberharter',))
  Adresse:      TUPLE(PLZ:      '1050',
                    Ort:      'wien',
                    Strasse:  'Margarethenstraße',
                    Hausnr: 3)
  Hobbies: SET(Hobby: STRING)
  Geburtsdatum: DATE);

```

Wie hier im Beispiel ersichtlich, sind der Name und die Adresse wieder eigene Types von Personen. In O<sub>2</sub> werden Recordsets als Tuple bezeichnet.

## 2.5 Identität

In relationalen Datenbankmodellen musste die Identität eines Datensatzes durch Schlüssel definiert werden, die geändert werden können. Objektorientierte Datenbanksysteme wahren die Objektidentität. Objekte existieren unabhängig von den Werten ihrer Eigenschaften (Attribute). Während sich die Werte der Eigenschaften ändern können, bleibt die Identität des Objektes unverändert. So sind auch Objekte mit gleichen Eigenschaften möglich. Ein Beispiel hierfür sind Personen mit den selben Vor- und Nachnahmen. Im relationalen Modell müssten jetzt noch weitere Attribute als Schlüssel definiert werden – sind diese aber auch bei diesen Datensätzen identisch, so kann das System die Personen nicht mehr unterscheiden.

### 3 Eigenschaften und Konzept Objektorientierter Datenbanken

Objektorientierte Datenbanksysteme (OODBS) sind im Laufe der 80er Jahre entwickelt worden und seit einiger Zeit kommerziell verfügbar.

Unter einem objektorientierten Datenbanksystem versteht man ein Softwaresystem, das acht Regeln der Objektorientierung und fünf Grundsätze der Datenhaltung und -nutzung erfüllt.

Unter dem Vorschlag des Manifesto<sup>3</sup> wurden folgende Kriterien für objektorientierte Datenbanksysteme aufgestellt:

#### 1. Komplexe Objekte

Ein OODBS unterstützt komplexe Objekte, deren Attribute selbst wieder Objekte sein können. Attribute können einerseits atomare Elemente einfacher Datentypen wie `Integer`, `String`, `Char` sein – andererseits auch Listen, Mengen oder ganze Objekte.

#### 2. Objektidentität

Jedes Objekt trägt unabhängig von den Werten der Attribute eine eindeutige Identität. Die Identifikation eines Objekts muss systemweit eindeutig und unveränderbar sein.

#### 3. Einkapselung

Datenkapselung verlangt, Implementierungsdetails nach außen zu verbergen. Auf Attributswerte kann nur durch Methoden zugegriffen werden.

#### 4. Typen und Klassen

Objekte mit äquivalenter Struktur und gleichem Verhalten werden zu Klassen zusammengefasst.

#### 5. Typen und Klassenhierarchie

Typen und Klassen können hierarchisch oder im Netzwerk aufgebaut sein.

#### 6. Overriding

Überladen von Methoden

#### 7. Datenbank Programmiersprache

#### 8. Erweiterbarkeit

Das System kann um neue benutzerdefinierte Typen und Klassen ergänzt werden. Nach Außen hin ist keine Unterscheidung zwischen systemeigenen und benutzerdefinierten Klassen ersichtlich. Die Erweiterung kann auf allen Ebenen um Typen, Funktionen und Speicherstrukturen erweitert werden.

#### 9. Mehrfach-Vererbung

Eine Subklasse (Unterklasse) in einer Klassenhierarchie kann Eigenschaften (Methoden und Attribute) von übergeordneten Klassen erben. Daneben darf die Unterklasse eigene Eigenschaften hinzufügen.

#### 10. Polymorphismus

Dieselben Methodennamen können auf Objekte unterschiedlicher Klassen vielgestaltig (polymorph) angewendet werden, obwohl die entsprechenden Methoden je nach Klassenzugehörigkeit abweichende Implementierungen aufweisen. Subklassen können hiermit gleichnamige Methoden der Oberklasse überschreiben.

#### 11. Vollständigkeit

Jede Operation in der Datenbank kann durch die Sprache ausgedrückt werden.

#### 12. Persistenz

Ein Datenbankzustand muss über die Laufzeit eines Programms hinweg erhalten bleiben. Diese Übertragung auf den externen Speicher erfolgt ohne explizite Kommandos.

#### 13. Ad-hoc-Abfragemöglichkeit

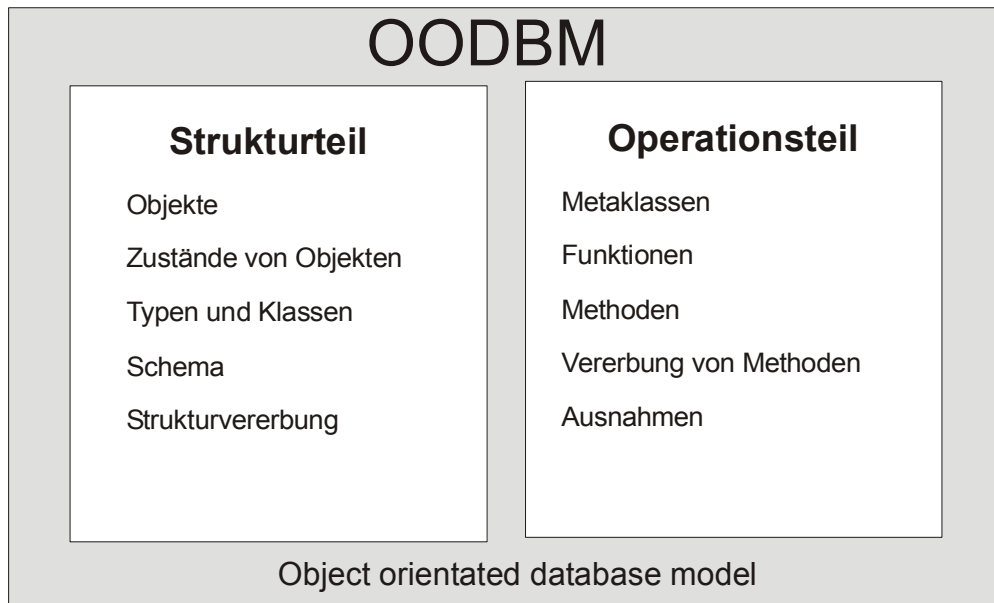
Das OODBS beinhaltet eine mächtige Abfragesprache zur Auswertung von Daten.

---

<sup>3</sup> [ABD89]

## 4 Das Modell der OODB<sup>4</sup>

Das Modell wird in zwei wesentliche Bereiche eingeteilt<sup>5</sup> – dem Struktur- und Operationsteil.



### 4.1 Der Strukturteil

Hier werden die statischen Aspekte der Objekte aus der zu modellierenden Datenbank und ihre Beziehungen untereinander beschrieben.

#### 4.1.1 Typenkonstruktoren und komplexe Objekte

Objektorientierte Datenbankmodelle integrieren neben Standard-Datentypen mindestens die drei Typenkonstruktoren `TUPLE OF`, `SET OF` und `LIST OF`. Jeder Typkonstruktor besitzt eine Reihe von Standard-Operationen wie Zugriffsoperationen auf Elemente oder Komponenten und Vergleichsoperationen.

##### Tupelkonstruktor

Ein Tupelkonstruktor `TUPLE OF` fasst mehrere Komponenten unterschiedlicher Typen zu einem neuen Typ zusammen. Die zugrundeliegenden Typen heißen *Komponententypen*.

##### Mengenkonstruktor

Ein Mengenkonstruktor `SET OF` erzeugt aus mehreren Elementen eines zugrundeliegenden Typs einen neuen Typ. Der zugrundeliegende Typ heißt *Elementtyp*. Die Menge enthält keine Elemente mehrfach und sie haben auch keine Ordnung.

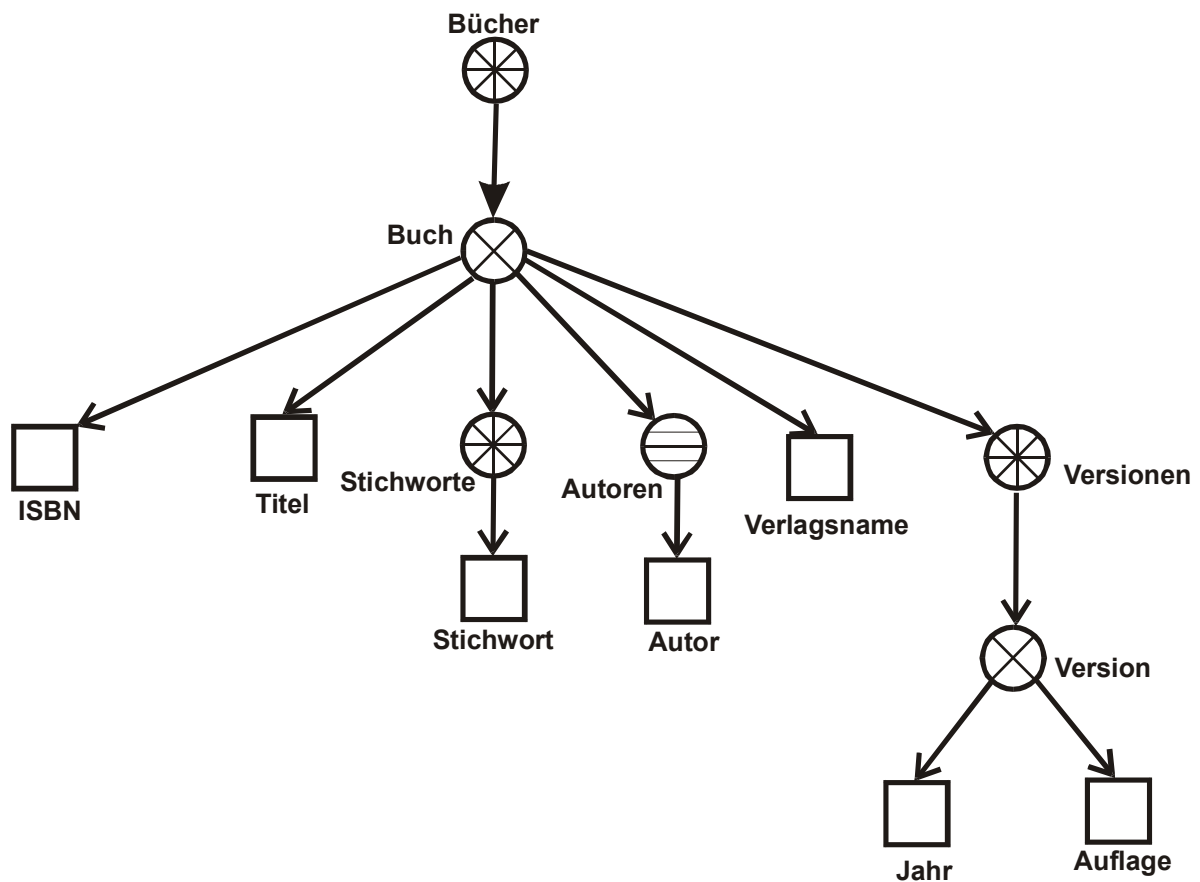
##### Listenkonstruktor

Ein Listenkonstruktor `LIST OF` erzeugt ebenfalls aus mehreren Elementen eines zugrundeliegenden Typs einen neuen Typs.

<sup>4</sup> Object oriented data base

<sup>5</sup> Einteilung nach [Bee89] S.370-395

Beispiel: Darstellung des Objekttypes Bücher mit Typkonstruktoren



Implementierung:

```

SET OF(TUPLE OF(ISBN: STRING,
  Titel: STRING,
  verlagsname: STRING,
  Autoren: LIST OF(Autor: String),
  Stichworte: SET OF(Stichworte: STRING),
  Versionen: SET OF(TUPLE OF(Auflage: INTEGER,
    Jahr: INTEGER))))
  
```

### 4.1.2 Objektidentität

Im Relationenmodell können Datenbankobjekte nur durch Schlüssel identifiziert werden. Leider haben solche „Objektidentifikatoren“ Nachteile in Bezug auf die Datenmodellierung (Updates, Abfragen etc.)

Eine Hauptforderung an objektorientierte Datenbankmodelle ist deshalb die Trennung des in der Datenbank dargestellten Objektes von seinen Werten: jedes Objekt hat seine unveränderbare Identität unabhängig von allen Attributswerten, die es beschreiben.

Es gibt mehrere Arten, die Objektidentität in OODBMs zu verwirklichen.

- physische Adressen (direkte Referenzen, Pointer)
- Namen (aus einem Namensraum, vom Benutzer definiert)
- Surrogate oder Identifier-Attribute
- abstrakte Objekte

Beim Erzeugen eines Objektes wird vom System eine global eindeutige und bis zur Löschung des Objektes unveränderbare Objektidentität vergeben. Zu jedem Objekt gehört ein Zustand, der es näher beschreibt. In dem Zustand können sich neben Objekten auch noch Werte befinden, die eine anwendungsunabhängige Semantik haben und nicht extra erzeugt werden müssen. Zum Beispiel die Buchstaben und Zahlen. Zu den Operationen auf Objekten zählen verschiedene Tests auf Identität und Gleichheit sowie verschiedene Kopieroperationen.

Surrogate und Identifier Attribute haben den Nachteil, dass sie nicht wie normale Attribute behandelt werden können. Physische Adressen sind wiederum an den physischen Speicherplatz gebunden.

Ein Beispiel für einen Identifier:

```

SET OF(TUPLE OF(Bücher_ID:IDENTIFIER,
                ISBN: STRING,
                Titel:STRING,
                Verlags_ID:IDENTIFIER,
                Autoren: LIST OF(Autor:STRING),
                Stichworte: SET OF(Stichwort: STRING),
                Versionen: SET OF(TUPLE OF(Auflage: INTEGER,
                                           Jahr: INTEGER))))

```

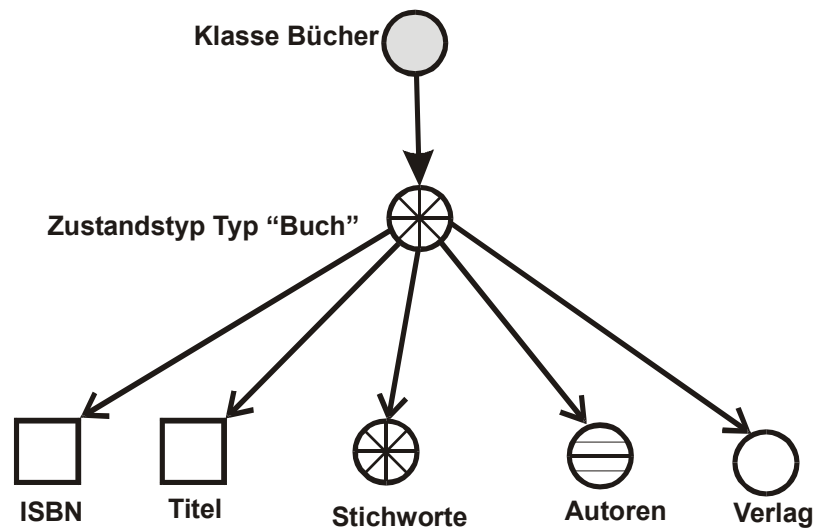
Objekt	Wert
nicht druckbar	druckbar
müssen erzeugt oder definiert werden	müssen nicht erzeugt oder definiert werden
trägt selbst keine Information	trägt Information
werden beschrieben	beschreiben etwas
anwendungsabhängige Abstraktion	anwendungsunabhängige Abstraktion

- **Nicht druckbar/druckbar:**  
Objekte wie Personen und Bücher sind als solches nicht druckbar – nur ihre Werte, wie z.B. die ISBN-Nummer sind auf Papier druckbar.
- **Zu erzeugen/bereits erzeugt:**  
Objekte sind zur Installation des OODBS noch nicht vorhanden. Die Objekte wie Personen und Bücher müssen erst auf Benutzeranforderung erzeugt werden und werden dann persistent in der Datenbank gespeichert. Andererseits gibt es bereits erzeugte Objekte wie z.B. die Zahlen und Buchstaben. Der Benutzer muss nicht zuerst die Zahl 5 und das A erzeugen, wenn er es als Attributswert einsetzen möchte. Solche Objekte zu löschen oder zu ändern ist nicht möglich, da sonst die Rechnung  $4+1$  nicht mehr existieren würde und der *Apfel* könnte auch nicht mehr geschrieben werden.
- **Zu definieren/bereits definiert:**  
Objekte sind nach der Erzeugung noch nicht „definiert“. Man muss den Objekten noch Informationen, etwa Attributswerte, zuweisen, um ihnen eine Bedeutung zu geben. Integer-Zahlen sind dagegen von vornherein definiert. Eine 5 ist für sich verständlich.
- **Trägt Information/trägt keine Information:**  
Objekte tragen keine Information sondern erhalten erst Informationen durch Zuweisung von Attributswerten zu einem Objekt.
- **Werden beschrieben/ beschreiben etwas:**  
Objekte werden durch Werte (Attribute) und Komponentenobjekte beschrieben. Werte hingegen müssen nicht beschrieben werden, da sie dem Menschen bekannt sind.
- **Anwendungsabhängig/anwendungsunabhängige Abstraktion:**  
Bei Objekten wie Personen und Büchern hängt die Semantik stark von der Anwendung ab. Die Semantik von Werten dagegen ist unabhängig –  $5 \text{ plus } 1$  ist immer 6.

### 4.1.3 Klassen und Typen

Klassen und Typen sind zwei Werkzeuge zur Definition eines Schemas in OODB. Eine Klasse beschreibt eine Objektfabrik, die zur Laufzeit Objekte aus einem Objektvorrat (der Domäne) erzeugt und in einer Objektmenge (der Instanz der Klasse) sammelt. Der zugeordnete Zustandstyp strukturiert die Objekte der Klasse. Auf Klassen muss es Operationen zum Erzeugen und Löschen von Objekten geben.

Beispiel des Objekttypes *Bücher*<sup>6</sup>



<sup>6</sup> Hinweise und Erklärung der Notation im Anhang

#### 4.1.4 Beziehungen zwischen Klassen

Beziehungen treten als Referenzen (1:n Beziehungen) oder m:n Beziehungen auf. Die Beziehungen zwischen Klassen und Komponentenklassen kann man aufgrund einiger Kriterien unterscheiden:

##### **Gemeinsame (shared objects) Komponentenobjekte**

Ein Komponentenobjekt kann Komponente mehrere anderer Objekte sein, sogar Komponente von Objekten unterschiedlicher Klassen.

Ein Beispiel hierfür wäre ein Verlag, der Herausgeber mehrere Bücher ist.

##### **Private Komponentenobjekte**

Diese Komponentenobjekte dürfen nur in einem Objekt als Komponente auftreten.

Ein Beispiel hierfür wäre ein Automotor eines bestimmten Autofabrikates. Der Motor kann nur in diesem Fabrikat genutzt werden. Es müssen auch für alle Autos Motoren vorhanden sein, denn es ist eben unmöglich einen gemeinsamen (shared) Motor bei 1000 Fahrzeugen zu verwenden<sup>7</sup>.

##### **Abhängige und unabhängige Komponentenobjekte**

Ein Objekt kann von der Existenz des umgebenden Objektes abhängen: es wird mit dem anderen Objekt erzeugt und auch wieder mit ihm gelöscht. Objekte, die von der Existenz anderer Objekte nicht abhängig sind bezeichnet man als unabhängige Objekte.

Ein Beispiel für abhängige Komponentenobjekte sind die Eltern eines Studenten oder der Verlag eines Buches in einer Universitätsdatenbank. Diese Komponenten können beim Austritt des Studenten mit den Studentendaten mitgelöscht werden, da sie alleine nur wenig Sinn machen. Genauso kann der Verlag beim Löschen des letzten Buches mitgelöscht werden. Ohne das eigentliche Objekt hat das abhängige Objekt keine Existenzberechtigung.

##### **Eingekapselte Komponentenobjekte**

Diese Objekte sind nur von ihrem umgebenen Objekt aus sichtbar. Man kann nur von dem zusammengesetzten Objekt aus auf das Komponentenobjekt zugreifen.

Ein Beispiel dafür ist, wenn der Zugriff auf die Verlage der Bücher nur durch den Zugriff auf ein konkretes Buch-Objekt möglich ist. Eine Ausgabe aller Verlage aller Bücher wäre dann aber unmöglich.

---

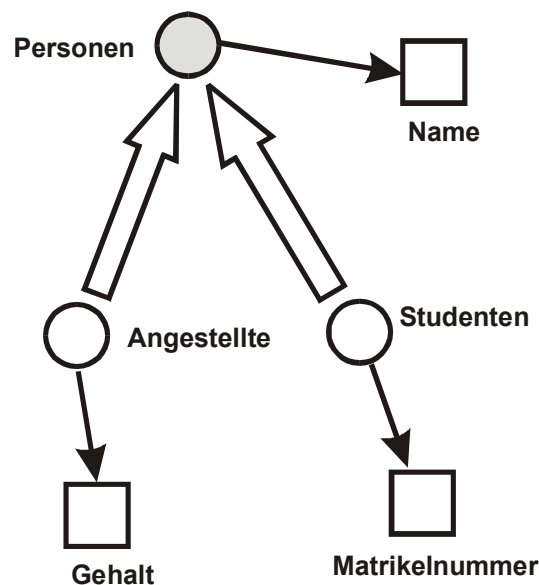
<sup>7</sup> wir gehen davon aus, das die Fahrzeuge auch gleichzeitig real funktionieren sollen!

### 4.1.5 Strukturvererbung

Die Strukturhierarchie (**Inclusion**) steuert die Vererbung von Attributen und Attributwerten (oder Zustandstypen und Zuständen) zwischen Klassen. Die Unterklasse hat weniger Objekte als die Oberklasse.

Unter der Typhierarchie (**Substitution**) versteht man eine Vererbung des Verhaltens von Objekten: Die Unterklasse hat mehr Attribute als die Oberklasse zur Verfügung (erbt alle Attribute der Oberklasse). Stimmen Klassen und Typhierarchie überein, so nennen wir sie Is-a-Hierarchie (**Specialization**).

Meistens findet die **Is-a-Hierarchie** Anwendung in OODBMs.



### 4.1.6 Integritätsbedingungen

Integritätsbedingungen sind ein optionales Konzept des Strukturteils. OODB besitzt keine eigenen Integritätsbedingungen, da sie vielmehr im OODB selbst implementiert und eingebaut sind. Einige der Integritätsbedingungen sind aus dem RDBM übernommen.

#### Schlüssel

Schlüssel werden im OODB nur mehr als Zugriffshilfe benötigt, da die Objekte intern durch abstrakte Werte identifiziert und unterschieden werden. Der Schlüssel kann benutzerdefinierte Werte enthalten – dies wird aber bei einer großen Objektanzahl sehr umständlich – benennen Sie einmal 10000 Objekte (z. B. Schrauben) ! Schlüssel dienen als lokaler Wert – er wird nicht als Referenz verwendet und keinesfalls zur Objektidentifizierung – die Objektidentität bleibt unveränderlich. Wie im RDBMM sind auch im OODB mehrere Schlüssel pro Klasse möglich. Schlüssel können auch vererbt werden und aus komplexen Komponenten (Komplexe Schlüssel beinhalten die Schlüssel der Komponenten - Tupels) bestehen.

#### Kardinalitäten

Es gibt totale Zustandsfunktionen, die Nullwerte verbieten. Einige Objekte aus einer Menge können jedoch Nullwerte bzw. undefinierte Werte als Attributswerte enthalten. Schlüsselattribute fordern jedoch totale Zustandsfunktionen.

Kardinalitäten von Beziehungen können im OODB Konzept implizit festgelegt werden. Beziehungen zwischen Klassen können als 1:n oder 1:1 ausgeprägt sein. m:n Beziehungen benötigen keine weiteren Einschränkungen.

#### Dynamische und Statische Integritätsbedingungen

Es gibt dynamische Integritätsbedingungen, die auch Daten aus der Vergangenheit mit den aktuellen vergleicht (Bsp. Lohn darf nur steigen). Statische Bedingungen überprüfen nur aktuelle Werte.

## 4.2 Der Operationenteil

### 4.2.1 Generische Operationen

Unter diesen Operationen zählen Anfrageoperationen und Updateoperationen, die nicht objektspezifisch sind. Sie können generisch zu allen Objektmengen angewendet werden. Diese Operationen müssen nicht definiert oder erzeugt werden, da sie im System schon vorhanden sind.

- **Erzeugen**  
Erstellung eines neuen Objekts in einer Objektmenge einer Klasse.
- **Löschen**  
Löschen eines Objektes aus einer abstrakten Klasse. Das Objekt muss zugleich aus den anderen Objektmengen, in denen es vorkommt, herausgenommen werden.
- **Einfügen**  
Ein bereits erzeugtes Objekt wird einer Objektmenge einer anderen Klasse hinzugefügt, wenn sie z. B. Unterklasse der Klasse ist.
- **Herausnehmen**  
Ein Objekt wird aus der Objektmenge einer Klasse herausgenommen, ohne es in der Klasse zu löschen, in der es erzeugt wurde.
- **Updates auf Zustände**  
Durch ein Update auf Zustände können von einem bestehenden Objekt die Attributswerte von Tupel oder Standard Datentypen geändert werden. Zusätzlich können bei Mengen und Listen Elemente eingefügt, geändert oder gelöscht werden.

### 4.2.2 Relationale Operationen

Diese Operationen verzichten auf die Berücksichtigung der Objektidentität des Klassenkonzeptes und der Strukturhierarchie und sehen die zugrundeliegenden Relation oder Menge „komplexer Werte“ an. Vorteile relationaler Operationen sind neben der Existenz vieler Algebren und Sprachen auch die unproblematische Anwendbarkeit und eine leicht definierbare Semantik. Der Nachteil ist, dass diese Operationen alleine nicht adäquat zum Datenbankmodell sind.

### 4.2.3 Objekterzeugende Operationen

Objekterzeugende Operationen stellen aus den Zuständen von Objektmengen Informationen zusammen. Es werden Ergebnisklassen erzeugt. Sie sind alleine nicht adäquat zum objektorientierten Datenmodell, aber relativ unproblematisch anwendbar.

### 4.2.4 Objekterhaltende Operationen

Werden zur Erstellung von dynamischen Klassen und Sichten verwendet. Sie bewahren die Objektidentität der in die Operation einfließenden Objekte. Die Informationen über Objekte und die zu ihnen gehörenden Methoden gehen nicht verloren. Objekterhaltende Operationen können nicht immer angewendet werden und es treten Inkonsistenzen und Konflikte zwischen sich widersprechenden Zuständen auf.

### **4.3 Metaklassen**

Metaklassen halten die Struktur und das Verhalten von Klassen fest. Dies ermöglicht das Anpassen von Klassen oder das Generieren neuer Klassen. Attribute einer Metaklasse werden auch Klassenattribute genannt.

Ein Beispiel:

In einem Maschinenbau-Konstruktionsbüro gibt es diverse Zylinder mit unterschiedlichen Durchmessern, Längen und Materialien. Einige Eigenschaften wie die Formel zur Berechnung des Volumens und der Außenfläche etc. sind aber bei allen Zylindern gleich. Diese Attribute (Formeln) sind nun Werte auf der Metaebene in den Metaobjekten.

## 5 Der ODMG-Standard

Es gibt eine große Anzahl an Entwicklungsrichtungen, Architekturen und Sprachschnittstellen für objektorientierte Datenbanksysteme. Diese Uneinheitlichkeit liegt natürlich unter anderem darin begründet, dass keine „Definition“ eines OODB, einer Anfragesprache sowie einer Systemarchitektur vor Beginn der Entwicklung vorlag.

Deshalb wurde im Jahre 1991 die *Object Management Group* (OMG) ins Leben gerufen. (Unter anderem mit der Beteiligung von Sun).

Der Standard besteht aus folgenden Teilen

- Objektmodell
- ODL
- OQL
- C++ Sprachanbindung
- Smalltalk Sprachanbindung

### 5.1 Das ODMG-Objektmodell

### 5.2 ODMG-ODL

Die ODMG-ODL (Object Definition Language) beschreibt die Typen des Datenbankschemas. Für diese Typen werden Schnittstellen und Klassen definiert. Es werden die Attribute, Beziehungen, Operationen, Klassendefinitionen und Schlüssel festgelegt.

Beispiel:

Wir definieren den Typ Person als Klasse und als Wurzel der folgenden Klassenhierarchie.

```
class Person(
  extent Personen
  keys PANr, (Name, Geburtsdatum, Adresse)
):{
  attribute long PANr;
  attribute Struct { string Vorname, string Nachname } Name;
  attribute Struct { integer PLZ, string Ort, ... }Adresse;
  attribute date Geburtsdatum;
  attribute Set <string> Hobbies
  ...
}
```

Als nächstes definieren wir die Unterklasse Student

```
class Student extends Person (
  extent Studenten
  keys Matrnr
):{
  attribute long Matrnr;
  attribute string Studienfach;
  attribute Person Mutter;
  attribute Person Vater;
  relationship Angestellter Betreuer inverse Angestellter::Betreut;
  attribute Set < Struct { float Note, string Fach } >Zeugnis;
  float Durchschnittsnote () raises (Keine_note);
  void Exmatrikulation (in Art: string) raises (Bücher_ausgeliehen);
}
```

In dieser Klasse wurden mit Vater und Mutter zwei (unidirektionale) objektwertige Attribute und zusätzlich eine (bidirektionale) Beziehung zur Klasse Angestellter deklariert. Neben dem relationenwertigen Attribut Zeugnis wurden auch noch zwei Operationen aufgenommen:

- Anfrage Operation Durchschnittsnote, die bei Nichtexistenz von Noten eine Ausnahmebehandlung aufruft
- Die Update-Operation Exmatrikulation, die einen Eingabeparameter Art verlangt und eine Ausnahmebehandlung aufruft, falls der Student noch Bücher ausgeliehen hat.

## 5.3 ODMG-OQL

Die ODMG-OQL (Object Query Language) ist eine Anfragesprache, die auf dem **select-from-where** Block von SQL-92 aufbaut. Zusätzlich zu SQL kann man mit OQL

- in Anfragen komplexe Werte, Objektidentitäten, Pfadausdrücke über Komponentenobjekte hinweg, Methoden und sogar das Overriding von Methoden ausnutzen,
- nicht nur auf Mengen, sondern auf jeder Collection in der gleichen Weise arbeitet, und
- neben dem SFW-Block auch beliebige andere Anfrageblöcke benutzen.

### 5.3.1 Grundprinzip einer Abfrage

Anfragen in OQL können von jedem Namen eines atomaren, strukturierten oder Collectionwertigen Objektes oder Wertes ausgehen.

So ist bereits

`Personen`

eine gültige Anfrage, da der Name die Extension des Typs Person bezeichnet.

Ist Hugo der Name für ein Objekt des Typs Student so ist

`Hugo.Zeugnis`

eine Anfrage, die eine Menge von Tupeln zurückliefert.

Zum Filtern verwendet man den SFW-Block:

```
select distinct struct(f: s.Studienfach, b: s.Betreuer)
from Studenten s
where s.Adresse.Ort = "wien"
```

Mit der Abfrage wird die Extension Studenten nach dem Wohnort Wien gefiltert.

Sowohl die Klasse-Unterklasse-Beziehungen als auch die Klasse-Komponentenklasse-Beziehungen werden von OQL über Pfadausdrücke direkt unterstützt.

### 5.3.2 Semantik einer Anfrage

OQL unterstützt relationale, objekterzeugende und objekterhaltende Anfragen, letztere jedoch nur sehr eingeschränkt.

Eine objekterzeugende Operation verwendet statt des Typenkonstruktors **struct** einen Objektkonstruktor in der Anfrage. Objektconstructoren sind die Namen der durch ODL definierten Typen. So wird mit

```
Person(PANr: 773495,
      Name: struct(Vorname: "Otto", Nachname: "Ohly"), ...)
```

ein neues Objekt vom Typ Person mit den angegebenen Werten erzeugt. Hier können jedoch nur für bestehende Typen Objekte erzeugt werden. Eine dynamische Typisierung und Klassifizierung wird nicht vorgenommen.

Eine objekterhaltende Anfrage ist

```
select s
from Studenten s
where s.Adresse.Ort = "Wien"
```

Hier werden die Objektidentitäten der Wiener Studenten in einer Multimenge aufgesammelt. Um ein Element selbst als Anfrageergebnis zu bekommen gibt es die Funktion **element**. In der folgenden Anfrage wird eine Person mit der PANr 5580 ermittelt.

```
element(select p
        from Personen p
        where p.PANr = 5580
)
```

## 5.4 Transaktionen

Das Sperren von Objekten wird im Standard vorgesehen. Die Sperren werden nach dem Read-Write Modell ausgeführt. Es gibt getrennte Sperren für Anfragen und Updates

Folgende Operationen sind in der Schnittstelle **Transaction** vorgesehen:

```
begin() Starten einer Transaktion
commit() Erfolgreiches Ende einer Transaktion
abort() Abbruch einer Transaktion
checkpoint() Synchronisation zwischen mehreren Transaktionen, um einen konsistenten
                Zustand im Logfile zu erreichen.
active() Überprüfen, ob eine Transaktion aktiv ist und läuft
```

Weiters gibt es für die Administration die Schnittstelle **Database** mit den Funktionen **open** und **close** für Datenbanken und **move**, **copy**, **reorganize**, **backup**, **restore** für die Datensicherung.

## 5.5 Kritik am ODMG-Standard

Der ODMG-Standard ist bis jetzt noch nicht ausgereift. Hier sind die wichtigsten Kritikpunkte aufgeführt. (Das ist nur eine kleine Auswahl aus einer Liste)<sup>8</sup>

- **Inkonsistenz und Unvollständigkeit**  
Der ODMG ist stolz auf die Kürze der Standards. Jedoch werden viele Themen sehr ungenau behandelt, so dass viele Fragen offen bleiben. Viele Konzepte werden nur syntaktisch und nicht semantisch eingeführt – nirgends wird die genaue Semantik erwähnt.
- **Schlüssel**  
Es fehlen genaue Angaben bezüglich dem Schlüsselkonzept. Nirgends wird auf die Semantik, und deren Bestandteile eingegangen.
- **Ständige Abschwächung des Standards**  
Es kommt zu keinen Verbesserungen am ODMG-Standard, die aufwärtskompatibel sind. Gibt es einmal Änderungen, so wird das Konzept nicht nur vollständig geändert, sondern womöglich entfernt.
- **Schwere Änderungen in der Semantik**

## 5.6 Die SQL-3-Norm

SQL-3 ist das aktuelle Normungsprojekt der ANSI und ISO als Weiterentwicklung von SQL-89 und SQL-92. Da SQL-3 erst voraussichtlich nach 2000 eingeführt wird, kann noch nichts konkretes gesagt werden. Ich stütze mich auf aktuelle Unterlagen.

SQL-3 fügt etliche neue Konstrukte dem bisherigen Standard hinzu. Sehr wichtig sind insbesondere die objektorientierten Erweiterungen wie

- abstrakte Datentypen (ADTs), die mit `CREATE TYPE` definiert werden,
- Objekt-Identifikatoren für einige ADTs neben Tupel-Identifikatoren für Tupel in Tabellen
- ADT-Hierarchien (ähnlich den Typhierarchien), die Substituierbarkeit von Objekten zusichern,
- Tabellen Hierarchien (ähnlich der Inklusion von Extensionen zwischen Unter und Oberklassen, hier bezogen auf Tabellen), wobei alle Attributnamen und Schlüssel geerbt, aber auch verändert werden dürfen,
- Möglichkeit zur Definition von Funktionen für ADTs,
- Überladen des Funktionsnamens mit Möglichkeiten zur dynamischen Auswahl der Funktionsimplementierung (ähnlich Overriding),
- komplexe Datentypen wie Mengen (`SET`), Multimengen (`MULTISET`), Listen (`LIST`) und Tupel (`ROW`).

---

<sup>8</sup> die vollständige Liste ist aus [HEU97] zu entnehmen.

### 5.6.1 Abstrakte Datentypen

Ein abstrakter Datentyp kann für Objekte (mit Objektidentität) und Werte (ohne Objektidentität) stehen.

Ein Beispiel:

```
CREATE TYPE Student
(
    PUBLIC Matrnr INTEGER,
      Studienfach CHAR(30),
    ...
    PUBLIC FUNCTION
      Durchschnittsnote (s Student) RETURNS REAL
      BEGIN
    ...
      END
)
```

### 5.6.2 Einkapselung und Funktionen

Die Einkapselung von Attributen wird so ähnlich wie in C++ vorgenommen. Es gibt die Sichtbarkeitsstufen PUBLIC, PROTECTED und PRIVATE.

Im ADT werden Funktionen mit Rückgabewert (Anfrage) und Prozeduren ohne Rückgabewert (Update) unterscheiden.

### 5.6.3 Typenkonstruktoren

In SQL-3 können mit den Typenkonstruktoren SET, MULTISSET, LIST und ROW verallgemeinerte geschachtelte Relationen gebildet werden.

Mit DISTINCT TYPES kann eine Kopie eines vorhandenen Types unter anderem Namen erstellt werden. Obwohl die Implementierung der Typen übereinstimmt, gelten sie für Anfragen als unvergleichbar.

### 5.6.4 Typ- und Tabellenhierarchie

SQL-3 unterstützt die Typ- und Klassenhierarchie des OODB. Letztere wird in SQL-3 Tabellenhierarchie genannt.

Ein Beispiel:

```
CREATE TYPE Student UNDER Person
(
    PUBLIC Matrnr INTEGER,
      Studienfach CHAR(30),
    ...
    PUBLIC FUNCTION
      Durchschnittsnote (s Student) RETURNS REAL
      BEGIN
    ...
      END,
    ...
)
```

Bei solchen Definitionen ist ein Overriding möglich. So kann eine Methode von Student die von Person überschreiben.

Neben der Typhierarchie gibt es in SQL-3 auf Tabellen noch die Tabellenhierarchie, die folgendermaßen definiert wird:

```
CREATE TABLE Studenten OF Student  
UNDER Personen OF Person
```

Dabei bezeichnen Studenten und Personen die Extensionen, in diesem Fall also Relationen der beiden Typen Student und Person. Jede **INSERT**-Anweisung auf der Untertabelle wird an die Obertabelle propagiert. Jede **DELETE**-Anweisung auf der Obertabelle an die Untertabelle. Die Untertabelle enthält alle Attribute der Obertabelle.

### **5.6.5 Anfragen**

Anfragen werden in SQL-3 immer an Tabellen gerichtet und haben immer eine relationale Semantik.

```
SELECT Adresse  
FROM Personen
```

## 6 Object Orientated Database System

### 6.1 Anforderungen an OODBS

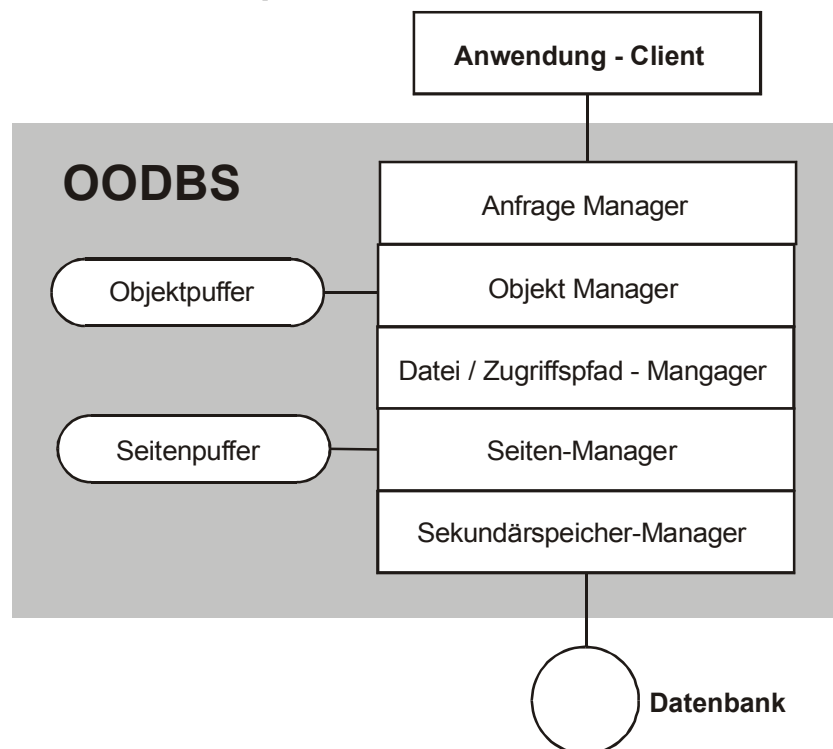
#### 6.1.1 Erweiterbarkeit

Das Datenbanksystem soll in allen Ebenen erweitert werden können. Sowohl in der konzeptuellen Ebene um neue Klassen, Methoden, Typen und ihre Funktionen, Sichten etc. In der internen Ebene wird Erweiterbarkeit bei Transaktionstechniken, eingesetzten Verfahren zur Optimierung, Datenspeicherstrukturen usw. gefordert. Oft wird ein Baukastensystem verwendet, um Datenbanken zu erweitern. Bei der externen Ebene wird Erweiterbarkeit bei generischen Anfragen und Updateoperationen verlangt.

#### 6.1.2 Persistenz

Fähigkeit der Daten (Werte oder Objekte) beliebige Lebensdauern lang zu erhalten (so kurz wie möglich – so lange wie nötig. Persistenz bedeutet, dass der Inhalt der Datenbank auch nach Ausfall des flüchtigen Speichers erhalten bleibt – sozusagen die Lebensdauer der Objekte und Daten. Bei der Datenbank ist das trivial erfüllt – hingegen im Programmiersprachenbereich benötigt es eigene Methoden um eine lange Lebensdauer von Objekten zu erlangen.

### 6.2 Interne Ebene und Speicherstruktur eines OODBS



Der Aufbau von einem OODBS besteht aus einer Fünf-Schichten-Architektur.

- **Sekundärspeicher-Manager:** Verwaltung des Sekundärspeichers, Transport der Daten (Seiten) in den Hauptspeicher
- **Seiten-Manager:** Verwaltet den Seitenpuffer und das Sperren (von Seiten) und den Recovery-Prozess.
- **Datei/Zugriffspfad-Manager:** Rechnet die Objektidentitäten der Objekte in Seitenadressen um.

- **Objekt-Manager:** Hat im OODBS eine zentrale Aufgabe.
  - Umwandlung von Seitenstrukturen in Objektstrukturen
  - Stellt Komponenten- und Vererbungsbeziehungen her
  - Verwaltet den Objektpuffer
  - Generiert und verwaltet Objektidentitäten
  - Sperrt Objekte auf Objektebene
  - Stellt Informationen für allfälliges Recovery zur Verfügung
- **Anfragen-Manager:** Ist meist im Anwendungsclient integriert und nicht im eigentlichem OODBS vorhanden. Optimiert Anfragen und verwaltet Sichten.

### 6.3 Transaktionen, Concurrency Control

Transaktionen, wie sie beim relationalen Modell vorkommen sind für OODBS unzureichend. Die Strukturen und die Transaktionen sind viel komplexer. Sie sind nicht mit einfachen Update- oder Anfrageoperationen aus dem RDBS vergleichbar.

In ODBS sind Transaktionen

- sehr lange
- setzen sich meist aus mehreren Teiltransaktionen zusammen.

Es gibt auch kooperierende Transaktionen, die ihr Ergebnis anderen parallel laufenden Transaktionen mitteilen, bevor der Abschluss der Transaktion erreicht wird. Unter ihnen liegen atomare Transaktionen, die wiederum abgeschlossen sein müssen.

Transaktionen können geschachtelt und strukturiert sein. So kann eine Transaktion mehrere Subtransaktionen beinhalten.

#### 6.3.1 Concurrency Control

Es gibt zwei Strategien zum Concurrency Control:

- **Pessimistische Verfahren** sperren Seiten, Sätze, Tupel, Objekte oder Objektmengen, um Konflikte mit anderen Transaktionen beim versuchten Zugriff auf die Daten zu erkennen und zu vermeiden.
- **Optimistische Verfahren** sperren dagegen nicht, sondern versuchen durch andere Techniken beim Schreiben von Daten Konflikte zu erkennen.

### 6.4 Recovery

Bei einem unbeabsichtigten Abbruch einer Transaktion führt das Recovery-System die Datenbank mithilfe von Log-Protokollen in einen konsistenten Zustand zurück.

### 6.5 Versionen-Management

Optional kann das OODBS ein Versionen-Management beinhalten. Es erlaubt mehrere Versionen von Objekten zu haben. Es kann nur die letzte Version bearbeitet werden. Neue Versionen können generiert werden und alte Versionen können gelöscht werden. Versionen können auch in mehrere parallele Versionen aufgeteilt werden bzw. es können verschiedene Versionen zusammengeführt werden.

Einen praktischen Anwendungsbereich findet ein Versionsmanagement im CAD-Bereich, wo mehrere Zeichnungen und Modelle eines Komponentenobjektes bestehen.

## 7 Anhang

### 7.1 Glossar

#### RDBM

Relational database model

#### RDBS

Relational database system

#### OODBM

Object orientated database model

#### OODBS

Object orientated database system

#### OOPL


Object orientated programming language


### 7.2 Verwendete Notation<sup>9</sup>


Abstrakte Klasse 

Komponente 

Freie Klasse 

Spezialisierung  


Generalisierung  


Zuordnung  
Komponente ->  
Komponentenklasse  


Mengenkonstruktor 

Tupelkonstruktor 

Listenkonstruktor 

Standard-Datentyp 

Zustandsfunktion  


Typkonstruktion  


<sup>9</sup> siehe Definitionen von Andreas Heuer aus [HEU97]

### **7.3 Literaturverzeichnis**

[HEU97] Andreas Heuer: Objektorientierte Datenbanken, Konzepte, Modelle, Standards und Systeme - Bonn: Addison Wesley Longman Verlag GmbH 1997, 2. Auflage

[HEU91] Andreas Heuer: Konzepte objektorientierter Datenmodelle. – Bonn: R. Oldenbourg Verlag, 1991

[GEP97] Andreas Geppert: Objektorientierte Datenbanksysteme: ein Praktikum. – Heidelberg: dpunkt, Verlag für digitale Technologie 1997, 1. Auflage

[MEW97] Andreas Meier, Thomas Wüst: Objektorientierte Datenbanken, ein Kompass für die Praxis. – Heidelberg: dpunkt, Verlag für digitale Technologie 1997, 1. Auflage

[BOO94] Grady Booch: Objektorientierte Analyse und Design, Mit praktischen Anwendungsbeispielen – Bonn: Addison Wesley Longman Verlag GmbH 1994

[BEE90] C. Beeri: A formal approach to object-oriented databases, Data and Knowledge Engineering, Band 5, Nr. 4, 1990

[ABD89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik: The object-oriented Database System Manifesto, 1989

### **7.4 Weiterführende Literatur und Informationsmaterial**

Object Data Management Group: <http://www.odmg.org/>

Object Management Group: <http://www.omg.org>

DBMS Website der Fachschaft Informatik der Universität Rostock, Deutschland:  
<http://www.db.informatik.uni-rostock.de/News/dbms.html>