Georgische Technische Universität
Humboldt Universität zu Berlin

Klaus Bothe, Gia Surguladze

OBJEKTORIENTIERTE MODELLIERUNG UND PROJEKTIERUNG

(Lehrmaterialien-1)

unterstützt durch DAAD (Deutschland)

Berlin - Tbilissi - 2006

bothe@informatik.hu-berlin.de **Prof.Dr. Klaus Bothe**

- 1979 Promotion (A) zur Thematik "Spezifikation und Verifikation abstrakter Datentypen";
- 1986 Promotion (B) zur Thematik "Ein algorithmisches Interface fuer Pascal-Compiler: Compiler-Portabilität durch Modularisierung";
- 09.1986-07.1987 Forschungsaufenthalt am ungarischen Forschungszentrum SZKI in Budapest;
- 09.1991 - 02.1992 Sonderforschungsstipendiat der Humboldt- Stiftung in Erlangen bei Prof. Stoyan;
- Seit 12.1993 Professur fuer Softwaretechnik und Theorie der Programmierung an der Humboldt-Universität zu Berlin.
- Bisherige Arbeitsgebiete: Theorie der Programmierung (Algebraische Spezifikation abstrakter Datentypen); Implementation prozeduraler Programmiersprachen (Compilertechnik); Logische Programmierung und Implementation von Prolog Wissensverarbeitung und Expertensysteme.

Prof.Dr. Gia Surguladze ***gsurg@gmx.net***

- 1980 Promotion (A), Elektrotechnisches Institut, St-Peterburg (Leningrad);
- 1991 Sonderforschungsstipendiat DAAD in Erlangen bei Prof. Stoyan (Deutschland);
- 1993 Habilitation (B) an der GTU, Tbilissi;
- 1995,96, 99, 2003 Sonderforschungsstipendiat DAAD in Erlangen bei den Professoren: H. Wedekind, F. Hofmann, G.. Bolch;
- 2000, 03 Sonderforschungsstipendiat DAAD an der Humboldt Universität zu Berlin, bei Prof. K.Bothe, Prof. W.Reisig;
- 1994 - bis h/Z Professor an dem Lehrstuhl „Management Information Systems“, GTU.
- 2001 – bis h/Z Direktor des Deutsch-Georgischen wissenschaftlichen Gemeinschaftslehrzentrum GTU.
- Forschungsinteressen: UML, OO-Programmierung; Relationale verteilte Datenbanken; Petrinetzen, PNML; .NET Technologie, C#, ADO, ASP.

ISBN 99940-56-01-8

Inhaltsverzeichnis

1. EINFÜHRUNG IN DIE OBJEKTORIENTIERTE MODELLIERUNG

Mit dem objectorientierten Ansatz sind nach Auffasung zahlreicher Fachleute signifikante Verbesserungen bei Entwurf, Realisierung, Test und Wartung von Anwendungen zu erzielen.

Bei der datenorientierten Vorgehensweise konzentriert sich das Interesse zunächst auf Objekte der Realität. Die Ermittlung von Funktionen (Tätigkeiten) wird erst dann in Angriff genommen, nachdem besagte Objekte bestimmt sind und im Sinne eines Leitbildes modellmässig zur Verfügung stehen /1,2/.

In der Vergangenheit war die Betrachtung eines Datenverarbeitungsystems sehr stark durch das Daten in Funktionen geprägt. Das rührt sicher auch daher, dass das Entwickeln von Software als Schreiben von Programmen gesehen wird und ein Programm realisiert eben eine oder auch mehrere Funktionen /3/.

In den 80-er Jahren haben die Daten zunehmend stärkere Beachtung erfahren. Indiz dafür ist die hohe Bedeutung, die man Datenmodellen, relationalen Datenbanken, Data Dictionaries, der Datenadministration u.ä.m. bemisst /5/ .

Beide Sichten – die funktions- wie die datenorientierte – haben natürlich ihren Sinn, aber sie vermitteln jeweils nur einen einseitigen und somit unvollständigen Blick auf ein System. Von daher ist eine Methode, deren Wesen in einer organischen Verbindung von Daten und Funktionen liegt, genau das Richtige. Und so ist es bei der objektorientierten Methodik.

Die Methodik der objektorientierten Modellierung umfasst die statistische Objektmodellierung (Zustandsmodellierung) und die dynamische Objektmodellierung (Verhaltensmodellierung).

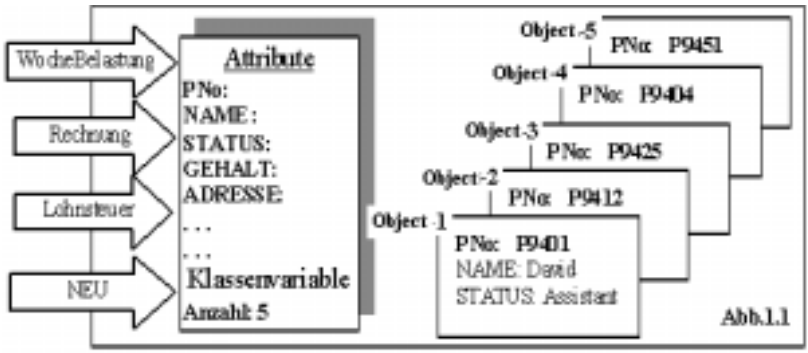
Objekte repräsentieren individuelle und identifizierbare Exemplare von Dingen, Personen oder Begriffen der realen oder der Vorstellungswelt. Ein Objekt entspricht also einem Einzelfall und ist als solcher wenig geeignet, die Realität modellhaft abzubilden.

Methoden (auch Services) bestimmen ja bekanntlich die Reaktion von Objekten auf einen äusseren Einfluss und sind damit für das Verhalten von Objekten zuständig. Zunächst soll aber von Nachrichten die Rede sein. Mit letzteren sind Methoden zu aktivieren und Objekte zum Agieren aufzufordern.

2. KLASSEN und OBJEKTE

Eine Klasse repräsentiert das Muster für die Kreierung von Objekten gleichen Typs. In einer Klasse sind somit Methoden und Variablen vorzufinden, welche an die zu kreierenden Objekte weiterzugeben sind.

Von Objekten gleichen Tzps spricht man, wenn ihr Status aufgrund identischer Variablen (Attributen) festzuhalten ist (jeder Lektor hat PNo, NAME, STATUS, GEHALT, ADRESSE u.s.w.) und ein identisches Verhalten vorliegt (identische Methoden wie WOCHEBELASTUNG, RECHNUNG, LOHNSTEUER u.s.w.). Abb. 1.1 illustriert das Prinzip einer Klasse.



Zu unterschieden ist zwischen konkreten (instanzierbaren) Klassen und abstrakten (nicht instanzierbaren) Klassen. Eine konkrete Klasse enthält im Normalfall Objekte, während in einer abstrakten Klasse niemals Objekte vorzufinden sind. Der Zweck einer abstrakten Klasse liegt darin, Attribute und Methoden an konkrete Klasse zu vererben.

Zu beachten ist, dass ein Objekt grundsätzlich irgend ein individuelles und identifizierbares Exemplar von Dingen, Personen oder Begriffen der realen oder der Vorstellungswelt zu repräsentieren vermag (Abb.1.2) /2/.

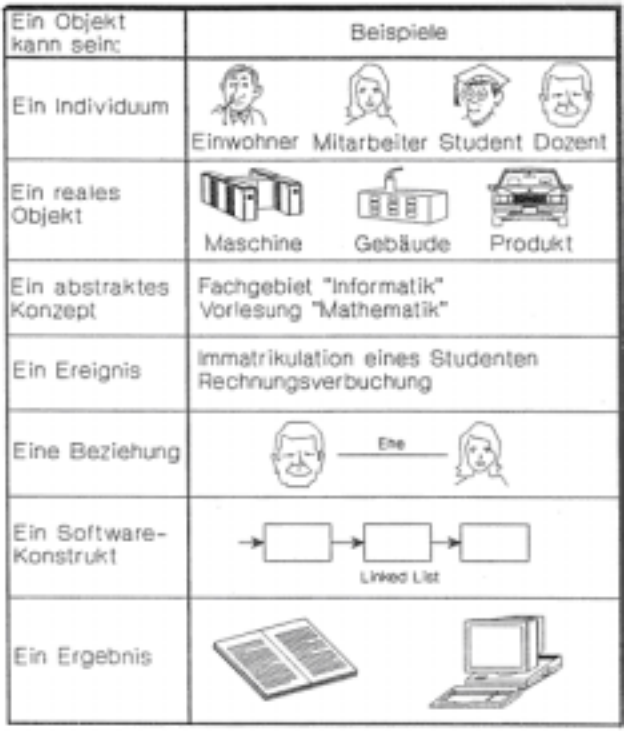


Abb.1.2. Beispiele des Objekts

Ein Objekt besitzt individuelle Eigenschaften, die sich in Daten (Attributen) sowie Verhalten (Methoden) gliedern lassen. Die Kommunikation zwischen Objekten erfolgt ausschlieslich durch das Senden von Nachrichten. Das Verhalten eines Objekts wird durch seine Methoden bestimmt. Erhält ein Objekt eine Nachricht, so wählt es eine geeignete Methode aus, welche ein Ergebnis ermittelt und dieses an Sender-Objekt zurückschickt.

Ein Objekt kann mehrere Zustände einnehmen und in jedem Zustand anders reagieren. Der Zustand eines Objekts wird

mittels Daten (Attributen) festgehalten. Ein Objekt verkapselt sowohl seine Daten (Attribute) wie auch seine Methoden und verkehrt mit der Aussenwelt über eine wohldefinierte Schnittstelle.

Von besonderer Bedeutung ist die auch unter dem Begriff Geheimnisprinzip (information hiding) bekannte Verkapselung von Daten und Methoden. Damit sind nämlich Implementationsdetails zu verstecken und damit Benutzer von Implementationsdetails zu entlasten.

In der Literatur gibt es zahlreiche Vorschläge für die Notation von Klassen und Objekte (Abb.1.3) /4/.

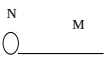
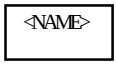
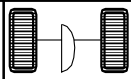
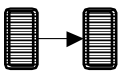
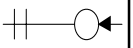
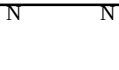
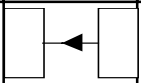
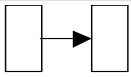
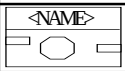
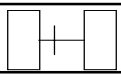
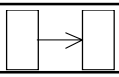
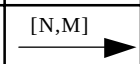
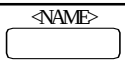
Element / Autor	Class, Object	Assotiation	Subsysiem	Inheritance	Message
G. Booch					
P. Coad E. Yordon					
D. Martin					
S. Shleer S. Melor					
I. Jacobson					

Abb.1.3 Notation verschiedener OO-Methoden

Abb.1.4 zeigt z.B., die Notation Gemäss Coad-Zordon (a) und gemäss Booch (b) für konkrete bzw. abstrakte Klassen.

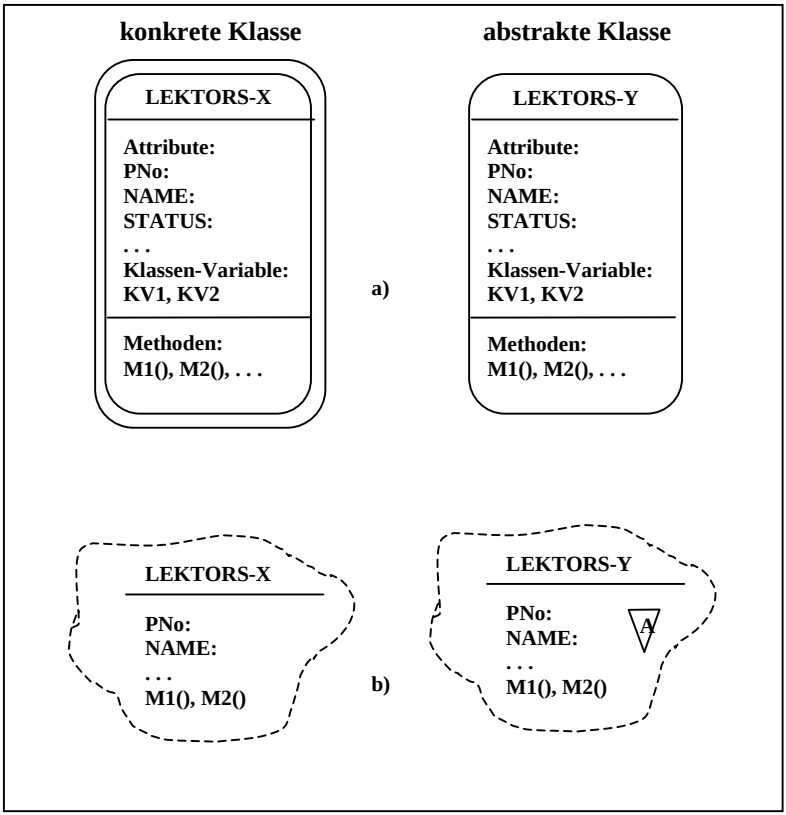


Abb.1.4 Notation der Klasse

3. ANALYSMETHODEN ZUR ERMITTLUNG VON KLASSEN

In der klassischen Datenmodellierung betrachtet man folgende Techniken zur Ermittlung von Klassen: die Textanalyse, Realitätsbeobachtungen, die Benützersichtanalyse, die Generalisierung und Spezialisierung, die Ereignisanalyse.

3.1. Die Textanalyse

Für die textuelle Beschreibung eines Problems verwendet man die grammatische Kategorien der Sprache, und zwar die Substantive als Kandidate für Klassen. In analoger Weise sind Variable anhand von Adjektiven und Methoden anhand von Verben ausfindig zu machen. Dieses Verfahren ist nicht perfekt, führt aber sehr schnell zu ersten Hinweisen. Selbstverständlich steht und fällt das Verfahren mit der Qualität des verfügbaren Textes.

Beispiel: Eine Unternehmung möchte die Administration ihrer internen Kurse automatisieren. Hinsichtlich der Kursorganisation liegen die im nachfolgenden Rahmen festgehaltenen Hinweise vor. Eine Textanalyse liefert folgende Kandidaten für Klassen (im Text-1 durch unterstrichene Substantive markiert):

MITARBEITER
DOZENT
STUDENT
KURSTYP
KURS
LOKAL
SCHLAFRAUM

Zudem sind folgende Methoden in Betracht zu ziehen (im Text kursiv markiert):

Kursanmeldungen pro Student durchführen
Kursabschlüsse pro Student durchführen
Teilnehmerliste pro Kurs erstellen

Atteste pro Kurs erstellen
Kursliste pro Dozent erstellen
Belegungsplan pro Lokal bzw. Schlafraum ermitteln

Offensichtlich sind mittels einer Textanalyse neben Klassen, Variablen sowie Methoden auch Beziehungen zwischen Klassen ausfindig zu machen. Beispielsweise deutet die Aussage „Jeder Kurs erfordert einen Dozenten" darauf hin, dass die Klassen KURS und DOZENT miteinander in Beziehung stehen.

Zu beachten ist, dass Variable nicht nur anhand von Adjektiven, sondern auch von Methoden her abzuleiten sind. So ist beispielsweise die Methode Belegungsplan pro Lokal ermitteln nur dann zu realisieren, wenn bekannt ist, welche Kurse von wann bis wann in welchen Lokalen stattfinden. Hierfür sind aber die Variablen AB.DATUM und BIS.DATUM für die Klasse KURS erforderlich.

Text-1: „KURSORGANIZATION"

1. *Es werden **Kurse** unterschiedlichen **Typs** (beispielsweise Informatik, Betriebswirtschaftslehre, Branchenkunde, etc.) angeboten. Für jeden **Kurstyp** gibt es mehrere **Kurse** (beispielsweise Informatik im Früihjahr, Sommer und Herbst).*
2. *Jeder **Kurs** erfordert einen **Dozenten.** Ein **Dozent** ist in der Regel für mehrere **Kurse** zuständig.*
3. *Ein **Kurs** kann von mehreren **Studenten** besucht werden. Umgekehrt kann ein **Student** mehrere **Kurse** besuchen.*
4. ***Dozenten** und **Studenten** sind **Mitarbeiter** ein und derselben **Firma**.*
5. *Jeder **Kurs** findet in einem **Lokal** statt. Ein **Lokal** kann im Verlaufe der Zeit von verschiedenen **Kursen** belegt sein.*
6. *Jeder **Dozent** und jeder **Student** braucht normalerweise einen **Schlafraum**. Ein **Schlafraum** kann im Verlaufe der Zeit von verschiedenen **Mitarbeitern** belegt werden.*
7. *Pro **Student** sind **Kursanmeldungen** sowie **Kursabschlusse durchzuführen**.*
8. *Pro **Kurs** ist eine **Teilnehmerliste** und sind **Kursatteste zu erstellen**.*
9. *Pro **Dozent** ist eine **Kursliste zu erstellen**.*
10. *Pro **Lokal** ist ein **Belegungsplan zu erstellen**.*
11. *Pro **Schlafraum** ist ein **Belegungsplan zu erstellen**.*

3.2. Realitätsbeobachtungen

Bei diesem Verfahren werden die für einen Problembereich relevanten Objekttypen und damit Klassen aufgrund von Realitätsbeobachtungen ermittelt. Auch dem vorstehend analysierten Text liegen Realitätsbeobachtungen zugrunde. Start diese Beobachtungen nun aber mühsam in textueller Form festzuhalten, dokumentiert man sie gleich in objektgerechter Form. Wie man sich dies für das Beispiel Kursorganisation vorzustellen hat, ist Abb. 1.5 (OO-Diagramm nach Coad/ Yourdon) zu entnehmen.

Empfehlenswert ist, bei der Definition einer Klasse gleich auch die zur Identifikation tatsächlicher Objekte erforderliche Variable - die identifizierende Variable also - festzulegen. Für die Klasse MITARBEITER kommt hierfür beispielsweise die Variable M# in Frage.

Realitätsbeobachtungen eignen sich hervorragend, um möglichst rasch ein, wenn auch grobes, OO-Modell zu ermitteln.

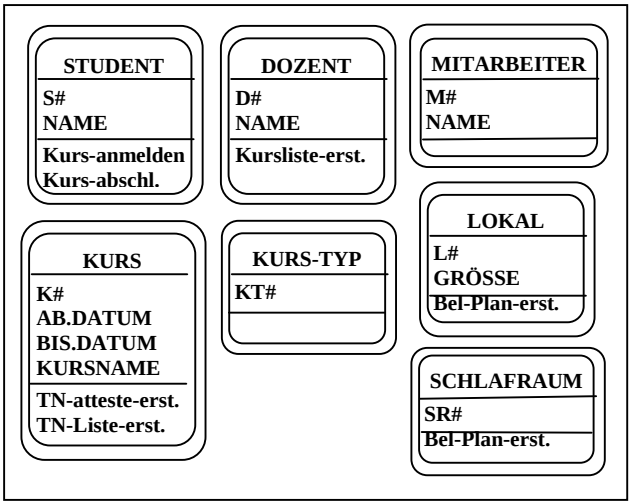


Abb. 1.5. OO-Diagramm nach Coad/Yourdon

Die Benützersichtanalyse

Bei der Bearbeitung von Wirtschafts- und Verwaltungsproblemen geht man zweckmassigerweise ergebnisorientiert vor. Zu diesem Zwecke ermittelt man kooperativ - also mit Beteiligung von Führungskräften, Sachbearbeitern und Informatikern - die für eine Anwendung zu produzierenden Ergebnisse in Form von Benützersichten für Bildschirmausgaben, Listen, Formulare, Belege etc.). Heute setzt man hierfür geeignete Tools ein, mit denen die angesprochenen Ergebnisse prototypmässig zu erzeugen sind (das heisst: ohne dass die gezeigten Daten in gespeicherter Form vorliegen oder die sie verarbeitenden Programme existieren müssen). Wir illustrieren die Benützersichtanalyse im folgenden an einem Beispiel und unterstellen zu diesem Zweck, dass für den Problembereich Kursorganisation die in Abb. 1.6 gezeigte Kursbeschreibung zu erstellen ist.

STUDENTEN					LEITER		BELEGTE KURSE		
St N	Stud. Name	Geb.-Datum	Sprache	Kentnis	Dozent-Numm.	Doz-Name	Kurs-Num.	Kurs-Name	Evaluation
15	Dvali	05.25.85	Engl. Deut Fran.	gut no schl	D05	Kio	k1 k2 k3 k4	Inf. Math. Deut. Alg.	gut gut gut gut
22	Guhua	12.03.80	Engl. Deut Fran.	schl gut no	D05	Kio	k1 k3 k5	Inf. Engl. Log.	mittel gut schlecht

Abb. 1.6. Beispiel für die Benützersichtanalyse

KURSBESCHREIBUNG

KURSNUMMER: K1 *DOZENTENNUMMER: D5*
KURSNAME: Informatik *DOZENTENNAME: Shonia*
LOKALNUMMER: L50
LOKALGROESSE: 55

EINGESCHRIEBENE STUDENTEN:

Pro Kurs ist also das Lokal, der Dozent, die Studenten, deren Studienleiter sowie die pro Student besuchten Kurse auszuweisen. Damit sind aber auch schon die wesentlichen Objekttypen und damit Klassen ganannt, die von der in Abb.1.6 gezeigten Benützersicht abzuleiten sind.

Mit der Benützersichtanalyse sind auch Beziehungen zwischen Klassen, sowie Variable (Attribute) zu ermitteln. So ist beispielsweise zu erkennen, daß für die Klasse KURS eine Variable KURSNAME, für LOKAL eine Variable GRÖSSE, für STUDENT die Variablen NAME sowie GEBURTSDATUM und für DOZENT die Variable NAME erforderlich sind.

Eine Benützersichtanalyse eignet sich hervorragend, wenn ein grobes, beispielsweise mittels Realitätsbeobachtungen ermitteltes OO-Modell zu verfeinern ist.

3.3. Die Generalisierung und Spezialisierung

Bei dieser Technik überlegt man sich:

1. Ist für eine (oder mehrere) Klasse(n) eine generellere, also allgemeinere Klasse denkbar ?

Beispiel: Hätte man für den Problembereich Kursorganisation entsprechend Abb. 1.7 zunächst nur die Klassen STUDENT und DOZENT festgelegt, dann hatte eine Generalisierung die Klasse MITARBEITER ergeben.

2. Sind für eine Klasse Spezialisierungen denkbar, oder mit andern Worten: Ist eine Klasse nach innen zu gliedern ?

Beispiel: Hätte man für den Problembereich Kursorganisation entsprechend Abb. 1.7 zunächst nur die Klasse MITARBEITER festgelegt. dann hätte eine Spezialisierung die Klassen STUDENT und DOZENT ergeben.

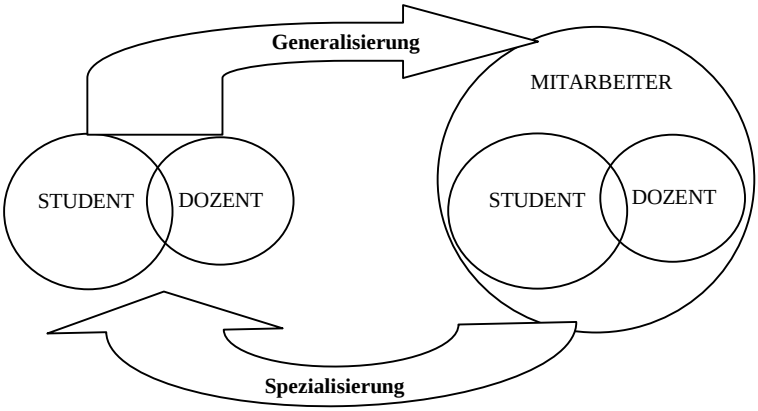


Abb.1.7. Generalisierung und Spezialisierung für Klassen

3.4. Die Ereignisanalyse

Sind über bereits erfolgte oder zu erwartende Ereignisse Informationen festzuhalten, so sind hierfür Ereignisklassen vorzusehen. Mit Ereignisklassen ist also das Geschehen auf der Zeitachse zu modellieren.

Beispiel: Im Problembereich Kursorganisation erfordert die Reservierung von Schlafräumen eine die Variablen AB.DATUM sowie BIS.DATUM aufweisende Ereignisklasse RESERVATION, denn nur so ist die zukünftige Belegung von Schlafräumen aufzuzeichnen. Analog erfordert die Belegung von Kursen durch Studenten eine Ereignisklasse BELEGUNG, denn nur so sind die von den Studenten im Verlaufe der Zeit erbrachten Kursleistungen festzuhalten. Letztere sind in Abb. 1.6 mit EVALUATION bezeichnet und müssten im OO-Modell als Variable der Klasse BELEGUNG berücksichtigt werden.

Eine Ereignisklasse ist auch erforderlich, wenn die zeitliche Abfolge von Kursen zu modellieren ist (einem Kurs für Anfanger folgen normalerweise mehrere Kurse für Fortgeschrittene, oder

umgekehrt: einem Kurs für Fortgeschrittene gehen normalerweise mehrere einfachere Kurse voraus). In unserem OO-Modell soll die zeitliche Abfolge von Kursen mit der Ereignisklasse ABFOLGE zum Ausdruck kommen.

Wie später darzulegen sein wird, steht die Ereignisklasse RESERVATION mit den Klassen SCHLAFRAUM und MITARBEITER in Beziehung. Analog steht die Ereignisklasse BELEGUNG mit den Klassen STUDENT und KURS in Beziehung, während die Ereignisklasse ABFOLGE zweckmässigerweise mit der Klasse KURS-TYP in Beziehung zu setzen ist. In der klassischen Datenmodellierung würde man bezüglich der genannten Ereignisklassen von Beziehungsmengen sprechen.

Abb. 1.8 von Coad/Yourdon zeigt das mit den Ereignisklassen RESERVATION, BELEGUNG und ABFOLGE ergänzte OO-Diagramm für das Beispiel Kursorganisation.

Neben den klassischen, der Datenmodellierung entlehnten Techniken zur Bestimmung von Klassen, gelangen im objektorientierten Umfeld zunehmend auch die im nachfolgenden Rahmen aufgeführten Techniken zum Einsatz, z.B. Verhaltensanalyse (Behavior Analyis), CRC-Karten (Class-Responsibility-Collaboration) u.s.w.

Anmerkungen:

Objektklassen (und -typen):

- *Gleichartige Objekte werden in einer Klasse zusammengefaßt.*
- *Jedes Objekt ist Instanz genau einer Klasse.*
- *Ein Objekt ändert seine Klasse während seiner Lebenszeit nicht.*
- *Die Klasse legt für ihre Instanzen fest:*
  - *alle Attribute mit zulässigen Wertebereichen.*
  - *alle Operationen mit ihren Implementierungen.*
  - *die Trennung von Schnittstelle und Rumpf.*
- *Objekttyp = Schnittstelle einer Klasse ganz ohne Implementierung.*
- *Eine Klasse kann mehrere Schnittstellen (Typen) implementieren.*
- *Abstrakte Klasse = Klasse ohne Instanzen mit unvollständiger Implementierung*

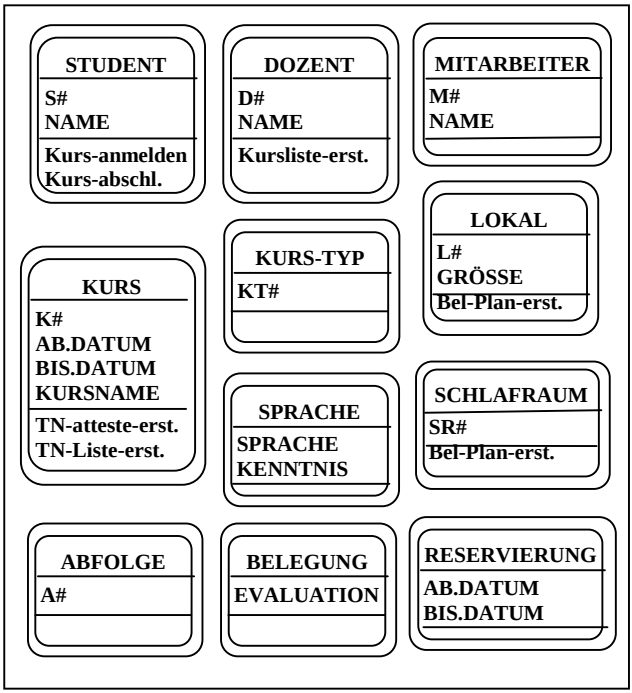


Abb. 1.8. Mit Ereignisklassen ergänztes OO-Diagramm nach Coad/Yourdon

Anmerkungen:

Objekte:

- *Konkretes Objekt der modellierten Welt, abstraktes Abbild im Programm.*
- *Kleinste (gekapselte) Einheit in einem objektorientierten System.*
- *Eindeutig identifizierbar.*
- *Hat einen internen Zustand:*
  - *repräsentiert durch **Attribute** (Instanzvariablen) = Datenelement.*
  - *Hat ein dynamischesVerhalten:*
  - *repräsentiert durch **Operationen** (Methoden) = Prozeduren.*
- *Objekte besitzen eine unveränderliche Identität.*
- *Objekte besitzen einen veränderlichen Zustand.*
- *Objekte sind Instanzen von Klassen, die ihr Verhalten festlegen.*
- *Objekte kommunizieren über Nachrichtenaustausch.*
- *Klassen bilden eine Vererbungshierarchie und*
- *auf Programmiersprachen ebene: Polymorphie (dynamisches Binden).*

4. VERERBUNGSHIERARCHIE

Vererbungsstrukturen resultieren, wenn man die Objekte einer Klasse nach bestimmten Kriterien gruppiert. Der Vorgang läuft darauf hinaus, daß man die Objekte einer sogenannten Superklasse als Menge auffast und diese unter Bildung von Untermengen - sogenannten Subklassen - nach innen gliedert.

Abb.1.7 illustriert ein Beispiel für Vererbungs- bzw. Generalisierungs-Spezialisierungsstruktur. Zu erkennen ist eine nach innen gegliederte Menge (Superklasse) MITARBEITER. Als Untermengen (Subklassen) treten zunächst LEKTOR und STUDENT in Erscheinung. Als Superklasse ist möglich eine Menge von Personen (PERSON) auszufassen, die alle Mitarbeiters der Institution beinhaltet Abb.1.9.

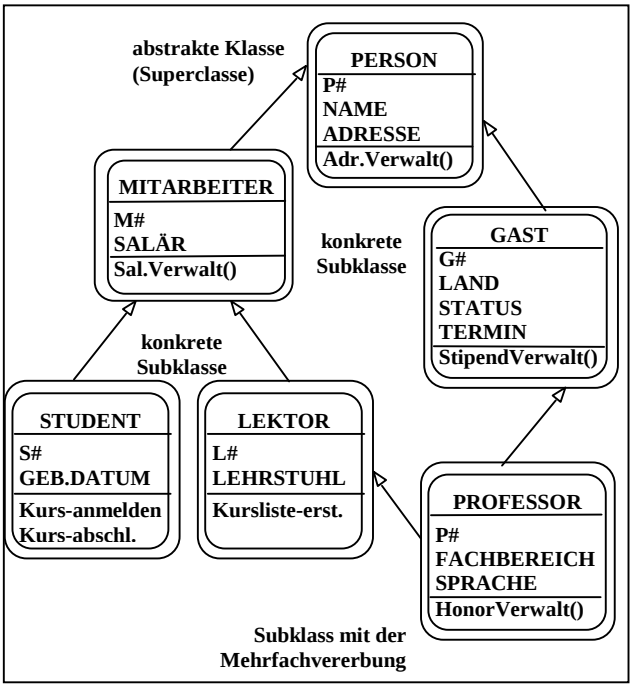


Abb. 1.9. Vererbungsstrukturen

Mit einer abstrakten Klasse sind Variable (Attribute) sowie Methoden an konkrete Klassen zu vererben. Also, mit Vererbung bezeichnet man die Funktionalität, mit welcher Variablen und Methoden einer Superklase an Subklasse weiterzugeben sind.

Falls eine vererbte Variable oder Methode in einer Spezialisierung nicht passt, so kann sie überschrieben werden.

Einfache Vererbung liegt vor, wenn jede Spezialisierung nur eine Generalisierung aufweist.

Mehrfache Vererbung liegt vor, wenn eine Spezialisierung mehrere Generalisierungen aufweist.

Vererbungsstrukturen nutzt und damit insofern einen hohen Grad an Wiederverwendbarkeit bietet, als in Superklassen definierte Variable (Attribute) und Methoden in Subklassen anzusprechen sind. Ein objektorientiertes Vorgehen nutzt das Prinzip der Daten- und Funktionskapselung sowie jenes von Klassen (Abstraktion), Abb.1.10.

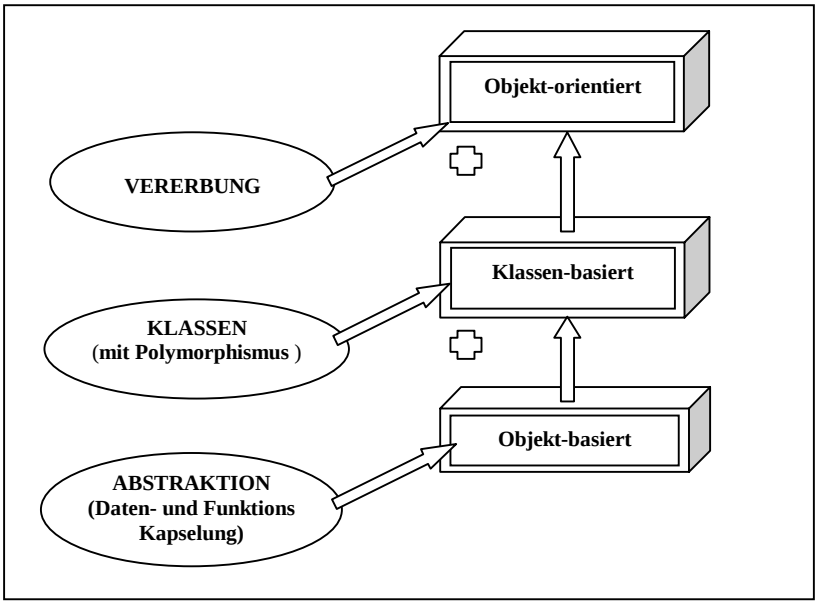


Abb. 1.10. Hauptprinzip der OO-Ansatzes

5. POLYMORPHISMUS

Polymorphismus gewährleistet, daß unterschiedlichen Klassen angehörende Objekte aufgrund ein und derselben Nachricht verschieden reagieren können.

Anzumerken ist, daß Polymorphismus natürlich auch bei Klassen funktioniert, die nicht an Vererbungsstrukturen beteiligt sind. Mit Polymorphismus ist nicht nur eine erstaunliche Flexibilität, sondern auch eine signifikante Vereinfachung der Logik von Programmen und damit eine Reduktion des Wartungsaufwandes zu erzielen (Abb.1.11). Kommt dazu, daß für jeden neuen Objekttyp Anpassungen in der Programmlogik

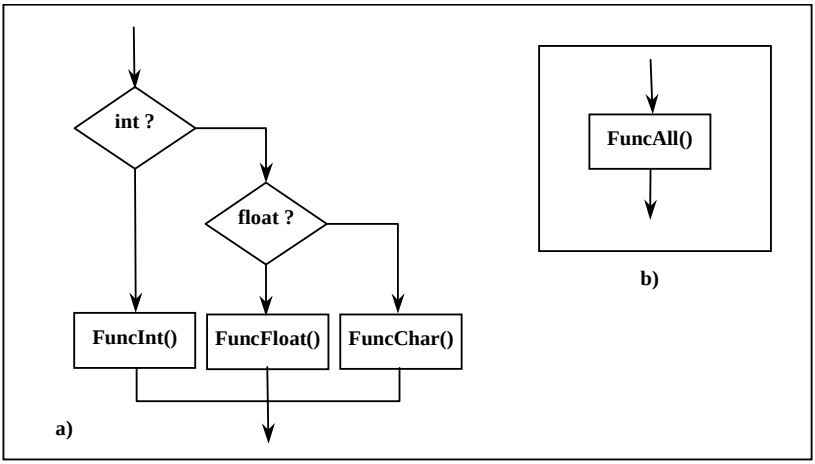


Abb.1.11. Das Programmfragment ohne Polymorphismus (a) und mit Polymorphismus (b)

erforderlich wären. All dies entfällt mit Polymorphismus, kennt doch jedes zu einer Aktion aufgeforderte Objekt die auszuführende Methode.

Polymorphie kommt aus dem Griechischen und heißt „Vielgestaltung“. Als polymorph werden alle Programmiersprachen bezeichnet, die die Definition von Prozeduren (Operationen) erlauben, die auf einer bestimmten Eingabeparameterposition Aktualparameter unterschiedlicher

Typen (Klassen) erlauben. Beispiel für Polymorphie wird in Abb.1.12 illustriert.

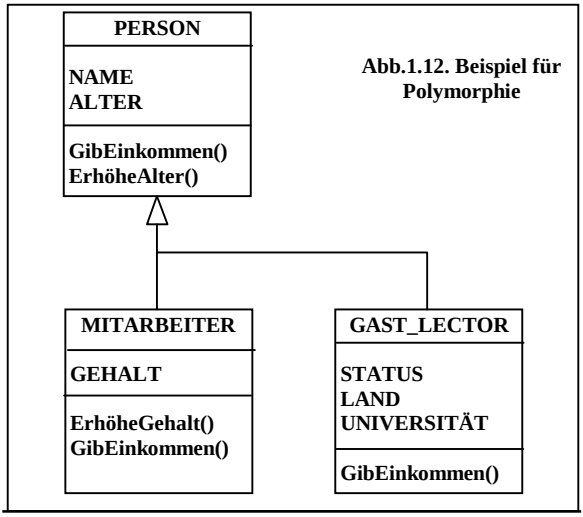


Abb.1.12. Beispiel für Polymorphie

Die Operation GibEinkommen ist polymorph, da sie auf Objekten der Klassen PERSON, MITARBEITER und GAST_LEKTOR aufgerufen werden kann.

In polymorphen funktionalen Sprachen wird derselbe Programmcode für Eingabe-parameter unterschiedlicher Typen abgearbeitet (kein dynamisches Binden). Bei objektorientierten Sprachen kann ein bestimmter Operationsaufruf in jeder erbenden Unterklasse durch eine völlig andere Methode implementiert sein. Die klassenabhängige Auswahl einer bestimmten Implementierungsmethode zu einem Operationsaufruf (zu einer empfangenen Nachricht) zur Programmlaufzeit nennt man dynamisches Binden.Im Gegensatz dazu wird die Auswahl einer bestimmten Prozedur aus einer Menge gleichbenannter Prozeduren zur Übersetzungszeit Overloading genannt (anhand der Aktualparametertypen). Beispiel: der Operator „+“ kann auf integer, float u.s.w. realisiert sein. Der Übersetzer entscheidet anhand der Aktualparameter welchen Maschinencode er für das „+“ erzeugen muß.

6. BEZIEHUNGSSTRUKTUREN

6.1. Beziehungstypen

Mit Beziehungsstrukturen sind die Objekte einer Klasse oder verschiedener Klassen miteinander in Beziehung zu setzen. Je nach Anzahl der Objekte, mit denen ein bestimmtes Objekt in Beziehung stehen kann, unterscheidet man in Analogie zur Datenmodellierung folgende vier Fälle:

- *Einfache (man sagt auch: Typ 1) Beziehungen.*
- *Konditionelle (Typ 0,1 oder auch Typ C genannte) Beziehungen.*
- *Komplexe (Typ 1,M oder auch Typ M genannte) Beziehungen.*
- *Komplex-konditionelle (Typ 0,M oder auch Typ MC genannte) Beziehungen.*

Anstelle von Beziehungen ist mitunter auch von Instance Connections (Coad/Yourdon) oder Associations (Booch) die Rede, während man bezüglich der Beziehungsarten (also einfach, konditionell, komplex, komplex-konditionell) auch von Zuordnungskardinalitäten spricht.

Einfache (Typ 1) Beziehung:

Wenn jedes Objekt einer Klasse A jederzeit mit einem Objekt einer Klasse B in Beziehung steht, so liegt von A nach B eine einfache (Typ 1) Beziehung vor (im Spezialfall steht jedes Objekt der Klasse A mit einem Objekt der gleichen Klasse in Beziehung). Es ist üblich, eine einfache Beziehung von einer Klasse A zu einer Klasse B wie folgt darzustellen (Abb. 1.13): ***A → B,*** *z.B.: STUDENT_ID → GROUP_ID.*

Konditionelle (Typ 0,1 oder C) Beziehung:

Wenn jedes Objekt einer Klasse A jederzeit mit einem oder keinem Objekt einer Klasse B in Beziehung steht, so liegt von A nach B eine konditionelle (Typ 0,1 oder C) Beziehung vor (im Spezialfall steht jedes Objekt der Klasse A mit einem oder keinem Objekt der gleichen Klasse in Beziehung). Es ist üblich, eine konditionelle Beziehung von einer Klasse A zu einer Klasse

B wie folgt darzustellen (Abb. 1.13): A —) B.

Z.B.: LEKTOR_NAME —) EHEPAAR_NAME.

Komplexe (Typ M oder 1,M) Beziehung:

Wenn jedes Objekt einer Klasse A jederzeit mit einem oder mehreren Objekten einer Klasse B in Beziehung steht, so liegt von A nach B eine komplexe (Typ M oder 1,M) Beziehung vor (im Spezialfall steht jedes Objekt der Klasse A mit einem oder mehreren Objekten der gleichen Klasse in Beziehung). Es ist üblich, eine komplexe Beziehung von einer Klasse A zu einer Klasse B wie folgt darzustellen (Abb. 1.13): A —>> B.

Z.B.: STUD_Id —>>LEKTOR_Id.

Komplex-konditionelle (Typ 0,M oder MC) Beziehung:

Wenn jedes Objekt einer Klasse A jederzeit mit beliebig vielen Objekten einer Klasse B in Beziehung steht, so liegt von A nach B eine komplex-konditionelle (Typ 0,M oder MC) Beziehung vor (im Spezialfall steht jedes Objekt der Klasse A mit beliebig vielen Objekten der gleichen Klasse in Beziehung). Es ist üblich, eine komplex-konditionelle Beziehung von einer Klasse A zu einer Klasse B wie folgt darzustellen (Abb. 1.13):

A –)> B, z.B.: STUD_Id —)> FACH_ID.

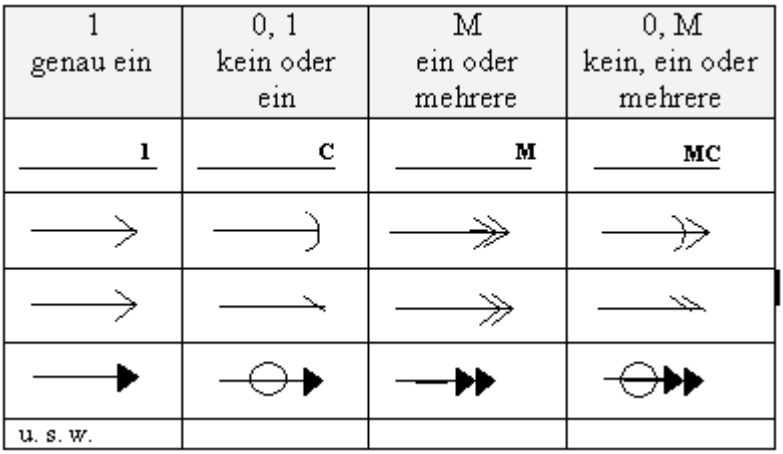


Abb. 1.13. Gebräuchlichste Notationen für Beziehungsarten

6.2. Abbildungen

Mit einer Abbildung sind Beziehungsarten einer Beziehung in beiden Richtungen auszuweisen.

Eine Abbildung macht eine Aussage über die Beziehungsarten, die einer Beziehung zwischen den Objekten einer Klasse A und jenen einer Klasse B zugrunde liegen (im Spezialfall betrifft die Abbildung Beziehungen von Objekten ein und derselben Klasse).

Ist die Beziehungsart von den Objekten einer Klasse A zu jenen einer Klasse B vom Typ T (also einfach, konditionell, komplex oder komplex-konditionell) und ist die dazu inverse Beziehungsart vom Typ T', so ist eine Abbildung formal wie folgt festzuhalten: (T' : T).

Dabei sind T und T' durch 1 (für einfache Beziehungen) bzw. C oder 0,1 (für konditionelle Beziehungen) bzw. M oder 1,M (für komplexe Beziehungen) bzw. MC oder 0,M (für komplex-konditionelle Beziehungen) zu ersetzen.

Gebräuchlich ist auch eine Notation, die neben den Beziehungsarten T und T' die Klassen A und B der an der Beziehung beteiligte Objekte wie folgt zum Ausdruck bringt:

A T'—T B.

Dabei sind T und T' mit geeigneten Symbolen aus Abb. 1.14 (verschiedene Abbildungstypen) zu ersetzen.

Nachdem die Beziehungsart der Objekte einer Klasse A zu jenen einer Klasse B einfach, konditionell, komplex sowie komplex-konditionell sein kann, und nachdem der gleiche Sachverhalt auch für die inverse Beziehungsart zutrifft, unterscheidet man 4 x 4 = 16 verschiedene Abbildungstypen.

Zu beachten ist, dass beispielsweise eine (l:M)-Abbildung durch eine Vertauschung der Klassen A und B in eine (M:l)-Abbildung umzufunktionieren ist. Entsprechend sagt man, dass die Abbildungen (1:M) und (M:l) zueinander symmetrisch seien. Von den 16 Abbildungstypen in Abb. 1.14 sind insgesamt 6 zueinander symmetrisch, namlich:

T / T'	1	0,1	M	0,M
1	(1 : 1)	(1 : 0,1)	(1 : M)	(1 : 0,M)
0,1	(0,1 : 1)	(0,1 : 0,1)	(0,1 : M)	(0,1 : 0,M)
M	(M : 1)	(M : 0,1)	(M : M)	(M : 0,M)
0,M	(0,M : 1)	(0,M : 0,1)	(0,M : M)	(0,M : 0,M)

Abb.1.14. Notation für Abbildungstypen

7. AGGREGATIONSSTRUKTUREN

Mit einer Aggregationsstruktur ist Zusammensetzung eines Ganzen aus mehreren Teilen zum Ausdruck zu bringen.

(1:M)	und	(M: 1)
(1:C)	und	(C : 1)
(1 : MC)	und	(MC : 1)
(C:M)	und	(M:C)
(C : MC)	und	(MC : C)
(M : MC)	und	(MC : M)

Zu unterscheiden sind also grundsatzlich 10 verschiedene Abbildungstypen.

G. Booch spricht bezüglich Aggregationsstrukturen von Containing Relationship und verwendet dafür die in Abb.1.15 gezeigte Notazion /4/.

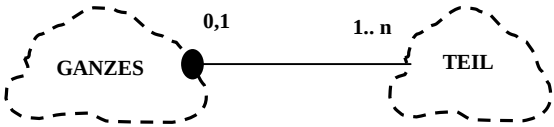


Abb.1.15. Agregationsstrukturen

8. VERHALTENSMODELLIERUNG

Unter der Verhaltensmodellierung verstehet man eine dynamische Objektmodellierung. Methoden (oder Services) bestimmen die Reaktion von Objekten auf einen äusseren Einfluss und sind damit für das Verhalten von Objekten zuständig. Die Methoden werden aufgrund von Nachrichten aktivieren.

Als Nachrichtenquellen kommen Objekte, Klassen sowie externe Ereignisse in frage. Der Austausch von Nachrichten wird in einem OO-Diagramm mittels Nachrichten-Verbindungen (Message Connections) dargestellt. Abb.1.16. illustriert anhand der Booch Notation daß Nachrichten grundsätzlich wie folgt zu versenden sind. Ein Senderobjekt (Client) schickt die Nachricht an das Empfängerobjekt (Server).

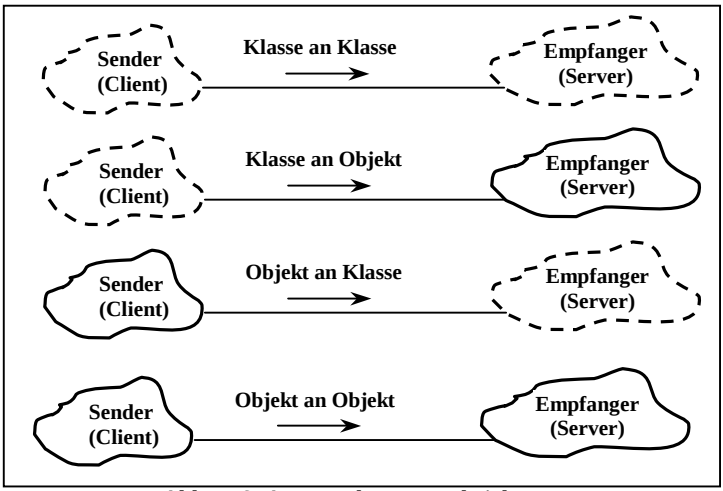


Abb.1.16. Austausch von Nachrichten

Das Empfängerobjekt führt die Methode aus und schickt das Ergebnis an das Senderobjekt zurück.

Mit Objekt-Diagrammen sind Ablaufszenarien - also exakte Abläufe von Systemfunktionen – festzuhalten. Die folgende Abbildungen 1.17 - 1.20 zeigen die Ablaufszenarien (OO-Diagramme) für eine Bibliothek.

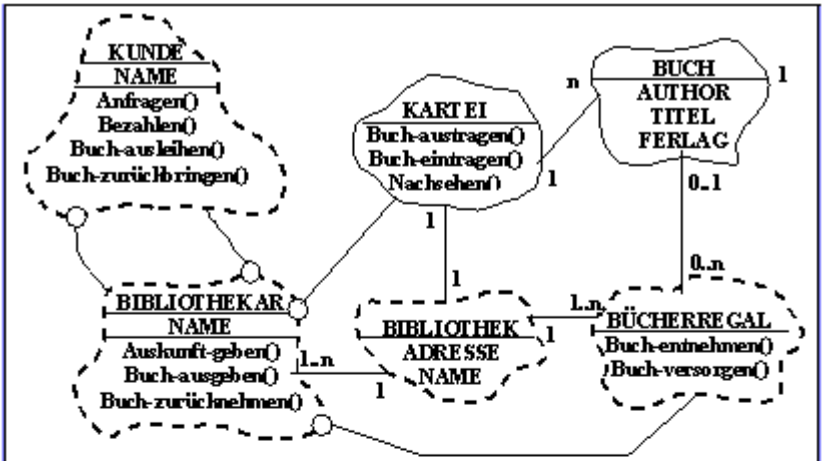


Abb. 1.17. OO-Diagramm für Bibliothek

Abb. 1.18. Ablaufszenario für Anfragen

Abb. 1.19. Ablaufszenario für Buch-ausleihen mit negativem Ergebnis

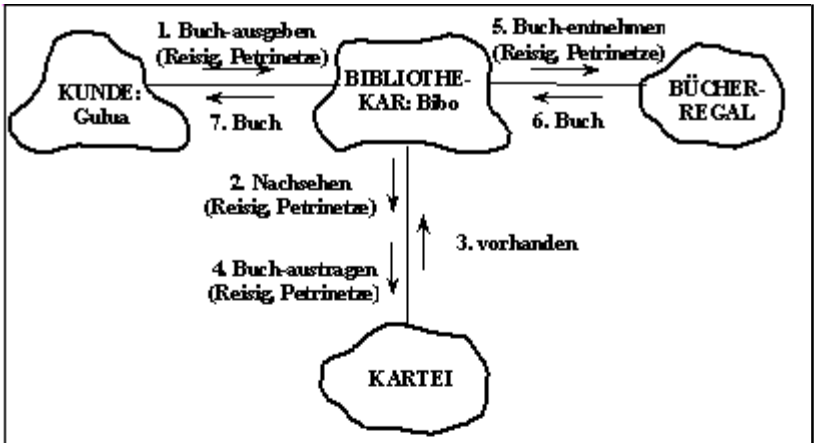


Abb. 1.20. Ablaufszenario für Buch-ausleihen mit positivem Ergebnis

Textuelle Beschreibung des Ablaufszenarios Buchausleihen:

1. ***Bibliothekar.Buch-ausgeben(Reisig, Petrinetze)***
2. ***Kartei.Nachsehen(Reisig, Petrinetze) → ···· nden***
3. ***Kartei.Buch-austragen(Reisig, Petrinetze)***
4. ***Bücherregal.Buch-entnehmen(Reisig, Petrinetze) → Buch***
5. ***Ende der Methode → Buch.***

Anmerkungen:

- Zustandsmodellierung wird als statishe Objektmodellierung betrachtet. Für die Realitätsmodellierung verwendet man bekannte Begriffen der Modellierungskonstrukte: Klasse, Objekt, Vererbung, Beziehung, Relation, Aggregation.
- Zustandsmodellierung hat viele Ähnlichkeit mit der Entitäten-Relations Modellierung (ER-M).
- Verhaltensmodellierung wird als dynamishe Objektmodellierung betrachtet. Hier verwendet man die Begriffe: Nachrichten, Methoden, Senderobjekt (Client), Empfängerobjekt (Server), Erreignisse, Ablaufszenario.
- Verhaltensmodellierung ist als Basis für die Use-Case Werkzeuge der Unified Modeling Language (UML).

9. GLOSSAR

Abstraction. *Abstraction ist eine Methode, bei der unter einem bestimmten Gesichtspunkt die wesentlichen Merkmale eines Gegenstandes oder Begriffes herausgesondert werden.*

Abstrakte Klasse. *Klassen, von denen keine konkreten Exemplare erzeugt werden koennen, d.h. von denen es grundsaetzlich keine Objekte geben wird.*

Aggregation. *Eine Aggregation ist eine Sonderform der Assotiation, bei der die beteiligten Klassen keine gleichwertige Beziehung fuhren, sondern eine Ganzes-Teile-Hierarchie darstellen. Eine Aggregation beschreibt, wie sich etwas Ganzesaus seinen Teilen zusammensetzt.*

Analyse*. Mit objektorientierter Analyse werden alle Proceduren im Rahmen des Softwareentwicklungsprocesses bezeichnet, die der Ermittlung, Klaerung und Beshreibung der Anforderungen an das System dienen (d.h. die Klaerung was das System leisten soll).*

Attribut*. Eine benannte Eigenschaft eines Typs. Ein Attribut ist ein Datenelement, das in jedem Objekt einer Klasse gleichermassen enthalten ist und von jedem Objekt mit einem individuellen Wert repraesentiert wird.*

Design. *Mit objektorientiertem Design werden alle Proceduren im Rahmen des Softwareentwicklungsprocesses bezeichnet, mit denen ein Modell logisch und physisch strukturiert wird und die dazu dienen zu beschreiben, wie das System die in der Analyse beschriebenen Anforderungen erfuellt.*

Ereignis (event) *ist ein Geschehen, das in einem gegebenen Kontext eine Bedeutung hat und sich raeumlich und zeitlich lokalisieren laest.laest.*

Exemplar (Objekt, Instanz).

Generalisierung *- (Ant. von* ***Spezialisierung*** *- konkretisierung).*

Geschaeftsprozess (Workflow) *ist eine Zusammenfassung von organisatorisch verteilten, fachlich jedoch zusammenhaengenden Prozeduren, die notwendig sind, um einen Geschaeftsvorfall (z.B. einen konkreten Antreg) ergebnisorientiert zu bearbeiten.*

Geschaeftsvorfall (Vorgang). *Ein Geschaeftsvorfall ist ein (Geschaefts-) Objekt (z.B ein konkreter Vertrag), das durch ein Ereignis ausgeloest (z.B. Antragseingang) durch innerhalb eines Geschaeftsprozesses beschriebenen Prozeduren bearbeitet wird..*

GUI (Graphical User Interface) *ist Grafische Benutzeroberflaeche.*

Interface - Schnittstelle.

Kardinalitaet - Multiplizitaet. *Anzahl der Elemente.*

Klasse. *Eine Klasse beschreibt die Struktur und das Verhalten einer Menge gleichartiger Obekte.*

Mehrfachvererbung. *Eine Klasse hat mehrere direkte Oberklassen.*

Merkmal - Eigenschaftswert.

Methode. *Das ist eine* ***Operation*** *(oder der Implementierung der Operation).*

Nachricht. *Nachrichten sind ein Mechanismus, mit dem Objekte untereinander kommunizieren koennen. Ein Nachricht fordert ein Objekt zur Ausfuerung einer Operation auf.*

Objekt. *Ein Objekt ist eine konkret vorhandene und agierende Einheit mit eigener Identitaet und definierten Grenzen das Zustand*

und Verhalten kapselt. Der Zustand wird repraesentiert durch die Attribute und Beziehungen, das Verhalten durch Operationen bzw. Methoden.

OO *ist die Abkuerzung fuer Objektorientierung.*

Polymorphismus. *Vielgestaltigkeit heisst, dass gleichlautende Nachrichten an kompatible Objekte unterschiedliches Verhalten bewirken koennen.*

Sichtbarkeitskennzeichen *schraenken die Zugriefbarkeit von Attributen und Methoden (private, protected, public).*

Vererbung *(inheritance). Unterklassen duerffen die Eigenschaften ihrer Oberklassen mitbenutzen.*

Virtuelle Klasse *(Operation) ist die Abstrakte Klasse (Operation).*

Workflow *ist die computergestuetzte Autometizierung des Geschaeftsprozesses.*

10. FRAGEN UND ÜBUNGEN

1. *Womit ist der Zustand eines Objekts festzuhalten ?*
2. *Was bestimmt das Verhalten eines Objekts ?*
3. *Was bedeutet der Begriff Datenkapsel ?*
4. *Was repräsentieren Klassen und Objekte ?*
5. *Wie erfolgt die Kommunikation zwischen Objekten ?*
6. *Was bedeutet Generalizierung und Spezialisierung ?*
7. *Man generalisiere die Klasse MITARBEITER.*
8. *Man spezialisiere die Klasse MITARBEITER.*
9. *Was bedeutet Klassifikation und Aggregation ?*

10. Was versteht man unter Verebung, Mehrfachvererbung ? 11. Was ist Polymorphismus ?

12. Wie sind objektbasiertes, klassenbasiertes und objektorientiertes Vorgehen zu charakterisieren ?

13. Worin unterscheiden sich Beziehungsstrukturen von Vererbungsstrukturen ?

14. Welche Beziehungsarten unterscheidet man ?

15. Was ist Abbildung und wie klassiert man sie ?

16. Was wird von Superklassen an Subklassen vererbt ?

17. Was sind Methoden und womit sind sie ativieren ?

18. Was unterscheidet ein Objektdiagram von einem Klassendiagramm ?

19. Was ist ein Ablaufszenario ?

20. Kostruiere ein Ablaufszenario für PRÜFUNG im Institut.

21. Kostruiere ein Objektdiagram für EINKAUF im Gescheft.

LITERATUR

1. Bothe K., Surguladze G. Modern Platforms and Languages of Programming. Tbilisi, 2003.

2.Vetter M. Objektmodellierung. Eine Einfuerung in die objektorientierte analyse und das objektorientierte Design. Zuerich, Stutgart, 1995.

3. Gogichaishvili G., Surguladze G., Shonia. Methods of Programming: C & C++, GTU, Tbilisi, 1997..

4. Oestereich B. Objektorientierte Softwareentwicklung. München, 1998.

5. Bothe K., Surguladze G.. Inheritance in the Programming of MIS: from the Databases to the UML Technology. GTU, 4(437),2001

Georgische Technische Universität
Humboldt Universität zu Berlin

Klaus Bothe, Gia Surguladze

SOFTWARE ENGINEERING MIT UML

(Lehrmaterialien-2)

unterstützt durch DAAD (Deutschland)

Berlin - Tbilissi - 2006

Inhaltsverzeichnis

2. EINFÜHRUNG IN UML

UML – die Unified Modeling Language ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.

Die UML ist keine Methode, aber sie kann die Basis für verschiedene Methoden sein, denn sie stellt eine definierte Menge von Modellierungskonstrukten mit einheitlicher Notation und Semantik bereit. Die UML enth]ält eine Vielzahl von Modellelementen und Details. Für den Software-Entwickler ist die UML eine Beschreibung mehrerer Diagrammtypen, die zur Spezifikation der statischen Struktur und des dynamischen Verhaltens der Objekte sowie zur Dokumentation von Implementierungsdetails benutzt werden können (Abb.2.1).

Folgende Diagrammtypen existieren:

- Anwendungsfalldiagramm *- zeigt Akteure, Anwendungsfälle und ihre Beziehungen.*

- Klassendiagramm *- zeigt Klassen und ihre Beziehungen untereinander.*

- Verhaltensdiagramme:

- Aktivitätsdiagramm *- zeigt Aktivitäten, Objektzustände, Zustände, Zustandsübergänge und Ereignisse.*

- Kollaborationsdiagramm *- zeigt Objekte und ihre Beziehungen inklusive ihres räumlich geordneten Nachrichtenaustausches.*

- Sequenzdiagramm *- zeigt Objekte und ihre Beziehungen inklusive ihres zeitlich geordneten Nachrichtenaustausches.*

- Zustandsdiagramm *- zeigt Zustände, Zustandsübergänge und Ereignisse.*

- Implementierungsdiagramme:

- Komponentendiagramm *- zeigt Komponenten und ihre Beziehungen.*

- Verteilungsdiagramm *- zeigt Komponenten, Knoten und ihre Beziehungen.*

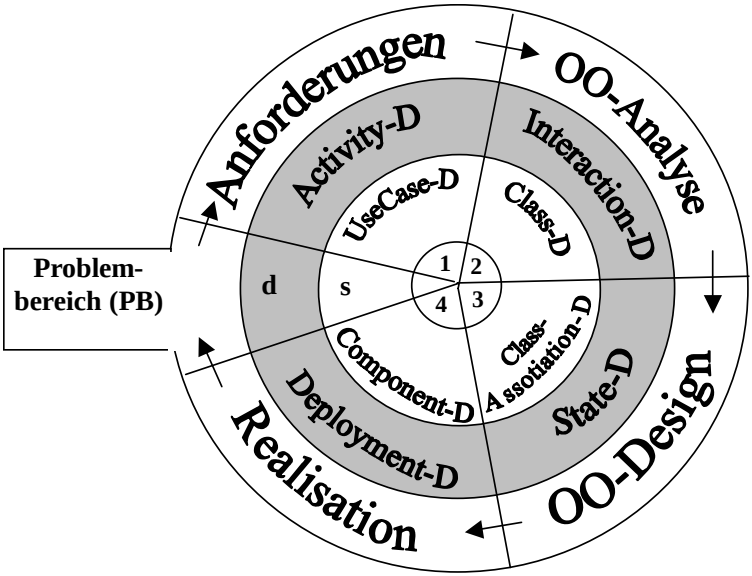


Abb.2.1. Problembereich und 4-Phasen fuer UML Programmierung
d - dynamische Modelle,
s - statische Modelle,
D - Diagramms

Anmerkungen:

- *Software Engineering* ist die Entwicklung, Pflege und Einsatz qualitativ hochwertiger Software unter Einsatz von wissenschaftlichen Methoden, wirtschaftlichen Prinzipien, geplante Vorgehensmodellen, Werkzeugen und quantifizierbaren Zielen.
- Die *Modellierungssprache UML* vielfдltige Mцglichkeiten zur objektorientierten Beschreibung von Software-Systemen zur Verfьgung stellt, die in den meisten Fдllen nur teilweise genutzt werden mьssen.
- *CASE-Werkzeuge* (Computer Aided Software Engineering) unterstьtzen grapische Darstellung der entworfenen Modelle, z.B. Rational Rose, Paradigm-Plus, ObjectivF, Together u.s.w.

Die Anforderungen des Systems können durch eine Anzahl verschiedener ***Anwendungsfälle*** *(Use Cases) ermittelt werden.*

Anwendungsfalldiagramme halten diese Anwendungs-Szenarien fest und beschreiben so den Funktionsumfang der Software aus der Sicht des späteren Benutzers. Anwendungsfälle bilden die Kommunikationsgrundlage mit dem Auftraggeber und sollten zum Entwurf von Testfällen herangezogen werden.

Das zentrale Element eines UML-Modells ist das ***Klassendiagramm*** *(Class Diagram). Darin werden die Klassen und ihre unterschiedlichen Beziehungen dargestellt, wobei die Klassen als Rechtecke und die Beziehungen als beschriftete Verbindungslinien gezeichnet werden. Im Verlauf des Entwicklungs-prozesses nimmt der Detaillierungsgrad dieser Darstellung zu, d.h. die Modelle werden immer präziser, bis sie schließ lich implementiert werden. Zu den Klassen können die Attribute und Operationen angegeben werden.*

Die UML betont die detaillierte Modellierung von ***Assoziationen zwischen Klassen*** *(Assotiation Class Diagram), zu denen Kardinalitäten und Rollennamen der beteiligten Objekte angegeben werden. Eine Aggregation ist eine spezielle Assoziation zwischen zwei Klassen, in der eine Klasse ein Bestandteil der anderen ist. Eine Komposition ist eine Aggregation, bei der eine Klasse ein Attribut einer anderen ist. Vererbungsbeziehungen werden durch Pfeile symbolisiert.*

Mit Hilfe von ***Aktivitätsdiagrammen*** *(Activity Diagram) läß t sich ein Vorgang (Workflow) veranschaulichen. Die verschiedenen Phasen oder Zustände eines z.B. Geschäftsprozesses werden dabei durch Transitionen miteinander verbunden. Dabei wird der Kontrollfluß in Form von Bedingungen und Verzweigungen, und der Datenflulß mit der Weitergabe von Objekten veranschaulicht. Die Verantwortung der einzelnen Objckte wird durch Aufteilen des Diagramms in mehrere Spalten,*

den sogenannten Schwimmbahnen, verdeutlicht.

Das dynamische Verhalten (Interaction Diagrams) von Komponenten kann übersichtlich in ***Sequenzdiagrammen*** *veranschaulicht werden, die den MSC-Diagrammen (Message Sequence Charts) sehr ähnlich sind. Der Nachrichtenaustausch der Objekte wird durch Pfeile angezeigt. Bedingungen, z.B. für Echtzeitsysteme, können angegeben werden.*

Die ***Kooperationsdiagramme*** *(Collaboration Diagrams) berücksichtigen zusätzlich noch die strukturellen Beziehungen. So kann z.B. der Einsatz von Entwurfsmustern dokumentiert werden.*

Mit ***Zustandsdiagrammen*** *(State Diagrams) wird das dynamische Verhalten von Objekten in der Form eines erweiterten Zustandsautomaten beschrieben. So wird das dynamische Verhalten einzelner Objekte präzise spezifiziert. Aus dieser Beschreibung lassen sich leicht Testfälle für die entsprechende Klasse ableiten. Auch Aktivitätsdiagramme können die Funktionsweise von Methoden veranschaulichen.*

Durch ***Komponenten- und Verteilungsdiagramme*** *(Component and Deployment Diagrams) wird die Architektur des Software-Systems und dessen Implementation in einer verteilten Rechnerumgebung (z.B. Client/ Server-Systeme) beschrieben.*

Die UML sieht die Modularisierung von Systemen in Paketen vor. Ein Paket faßt Modellelemente zusammen und regelt die Sichtbarkeit auf die Bestandteile. Die Abhängigkeit der einzelnen Pakete untereinander wird durch Paketdiagramme veranschaulicht. Also, Modellierungssprache UML stellt vielfältige Möglichkeiten zur objektorientierten Beschreibung von Software-Systemen zur Verfügung, die in den meisten Fallen nur teilweise genutzt werden müssen. Für die Hersteller von graphischen CASE-Werkzeugen hat die Standardisierung der UML einen einheitlichen Marktgeschaffen und nicht zuletzt

2.1. ANWENDUNGSFALL- UND AKTIVITÄTSDIAGRAMME

Ein Anwendungsfalldiagramm (Use Case) beschreibt die Zusammenhänge zwischen einer Menge von Anwendungsfällen und den daran beteiligten Akteuren. Es bildet somit den Kontext und eine Gliederung für die Beschreibung, wie mit einem Geschäftsvorfall umgegangen wird.

Ein Geschäftsvorfall ist beispielsweise die schriftliche Schadensmeldung eines Hausrat-Versicherten. Der Geschäftsprozeß (z.B. „Schadensmeldung Hausrat“) beschreibt den gesamten Ablauf, um ein solches Ereignis zu verarbeiten. Der Geschäftsprozeß enthalt dabei unter Umständen auch Aktivitäten die nicht direkt durch Software bzw. die zu entwickelnde Anwendung unterstützt werden (z.B. „Besichtigung des Schadenortes durch einen Sachverstän- digen“).

Anwendungsfälle beschreiben gewöhnlich nur die Aktivitäten, die durch die zu entwickelnde Software unterstiitzt werden sollen, und deren Berührungs punkte zum Umfeld dieser Software. Alle Anwendungsfälle zusammen bilden ein Modell, das die Anforderungen an das externe Verhalten des Gesamtsystems beschreibt. Was genau einen Anwendungsfall ausmacht, wird im nachsten Abschnitt detailliert beschrieben.

Zu beachten ist, daß Anwendungsfälle primer keinen Designansatz darstellen und nicht das interne Verhalten des zukunftigen Systems beschreiben, sondern ein Hilfsmittel zur Anforderungsermittlung sind. Anwendungsfälle sollten nicht zur funktionalen Dekomposition verwendet werden, sie sind kein

Ablaufdiagramme, Datenfluß diagramme oder Funktionenmodelle.

Anwendungsfälle sind vor allem dazu da, die Kommunikation mit den zukünftigen Anwendern, dem Auftraggeber, der Fachabteilung o.ä. zu unterstützen. Anwendungsfälle beschreiben das externe Systemverhalten, d.h. was das System leisten soll. Wie dieses entsteht, d.h. welches Systemdesign und welche Realisierung zu diesem äüß eren Systemverhalten beiträgt, darüber treffen die Anwendungsfälle keine Aussage.

Ein Anwendungsfalldiagramm enthält eine Menge von Anwendungsfällen, die durch einzelne Ellipsen dargestellt werden und eine Menge von Akteuren und Ereignissen, die daran beteiligt sind (Akteure). Die Anwendungsfälle sind durch Linien mit den beteiligten Klassen verbunden. Ein Rahmen um die Anwendungsfälle symbolisiert die Systemgrenzen (Abb 2.2).

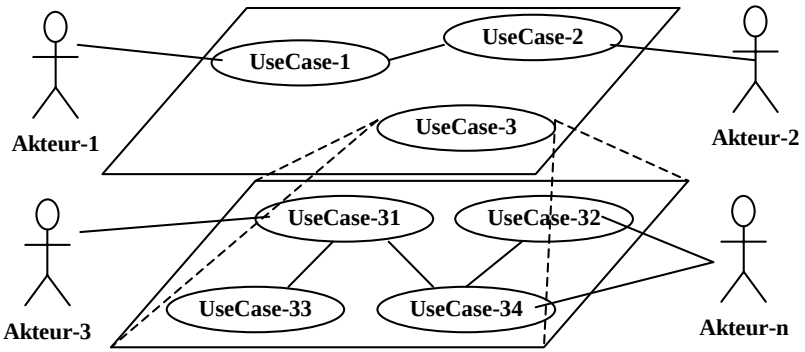


Abb.2.2. Hierarchisches AF-Diagramm

Anwendungsfalldiagramme können hierarchisch verschachtelt werden, d.h. ein Anwendungsfall in einem Anwendungsfalldiagramm kann durch ein weiteres Diagramm in detailliertere Anwendungsfälle untergliedert werden.

Innerhalb eines Diagramms können die Anwendungsfälle durch Beziehungen verbunden werden. Ein Vererbungspfeil von einem Anwendungsfall zum anderen soll zeigen, daß das Verhalten des Anwendungs-falles, auf den der Pfeil zeigt, Teil des Anwendungsfalles ist, von dem der Pfeil losgeht. Der Pfeil kann mit den Stereotypen «uses» oder «extends» beschriftet werden, je nachdem ob eine Benutzt- oder eine Erweiterungsbeziehung ausgedrückt werden sol l .

«uses» oder «benutzt» wird verwendet, wenn das gleiche Stück Anwendungsfallbeschreibung in verschiedenen Anwendungsfällen vorkommt. Um dies zu vermeiden, wird der entsprechende Teil separiert und mit einer «uses»-Beziehung in die andere Anwendungsfallbeschreibung wieder eingebunden. Der Pfeil zeigt in Richtung auf den mitbenutzten Anwendungsfall.

«extends» oder «erweitert» wird verwendet, um Variationen eines Anwendungsfalles zu zeigen, beispielsweise Fehler- und Ausnahmesituationen, spezielle Abweichungen oder Erweiterungen des Standardfalles. Anstelle von «extends»-Beziehungen können Varianten auch direkt als Text im Anwendungsfall beschrieben werden. Der Pfeil zeigt von der Variante zum Standard-Anwendungsfall.

Die Stereotypen «uses» und «extends» sind nützliche, aber entbehrliche Modellkonstrukte, manche Modelliererlnnen verzichten darauf Siehe hierzu.

Das folgende Beispiel zeigt die Verwendung von «uses» und «extends» und leutet gleichzeitig auf die damit verbundenen Schwierigkeiten hin. Das Beispiel zeigt den Anwendungsfall Vertrag schließ en, der unter anderem den Anwendungsfall Kunde identifizieren mitbenutzt (Abb.2.3).

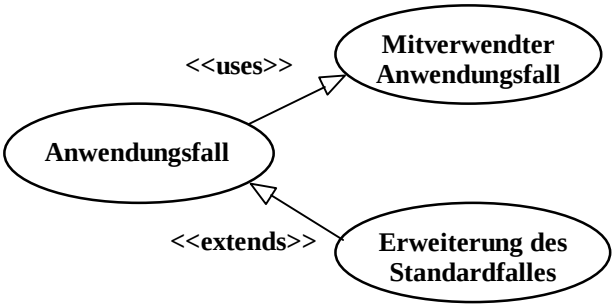


Abb.2.3

Kunde identifizieren vird auch in verschiedenen anderen Kontexten verwendet und existiert deslalb als eigener Anwendungsfall. Innerhalb des Anwendungsfalles Kunde identifizieren wird in bestimmten Fällen auß erdem der Anwendungsfall Kunde neuanlegen verwendet, nämlich immer dann, wenn der Kunde im Kundenbestand noch nicht enthalten ist und neuangelegt werden muß .

Das Anwendungsfalldiagramm zeigt zwar sehr allgemein die Verbindungen zwischen den drei Anwendungsfällen, beschreibt aber keinerlei Details. Man sieht beispielsweise, daß der Anwendungsfall Kunde identifizieren irgendwie durch Kunde neuanlegen erweitert wird, aber nicht wie.

*Das nebenstehende **Aktivitaetsdiagramm** beschreibt diesen Zusammenhang sehr viel konkreter (Abb.2.4).*

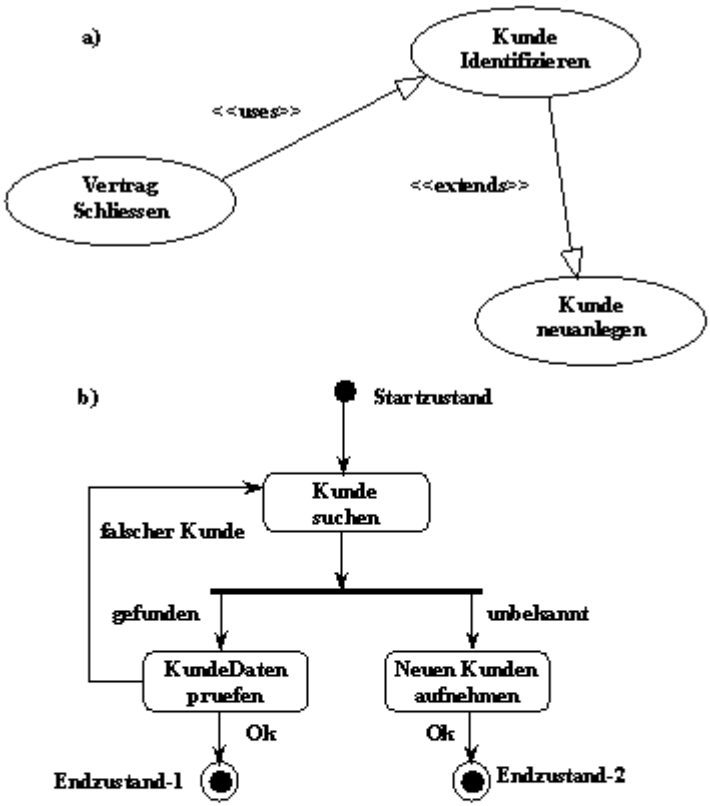


Abb.2.4. Use Case-Diagramm (a) und Aktivitaetsdiagramm (b)

Wie die einzelnen Anwendungsfälle zusammenhängen, d.h. die anwendungsfallübergreifende Beschreibung von Ablaufen, ließe sich zwar textuell innerhalb der Anwendungsfallbeschreibungen notieren, es wäre aber wenig anschaulich. Aktivitätsdiagramme vermitteln solche Zusammenhange visuell und damit einfacher.

2.2. KLASSENDIAGRAMM

Klassen werden durch Rechtecke dargestellt, die entweder nur den Namen der Klasse tragen oder zusätzlich auch Attribute und Operationen. Sie sind durch eine horiyontale Linie getrent (Abb.2.5). Dazu verwendbar sind alle Regeln OO-Modellierung von 1.Kapitel dieses Buch.

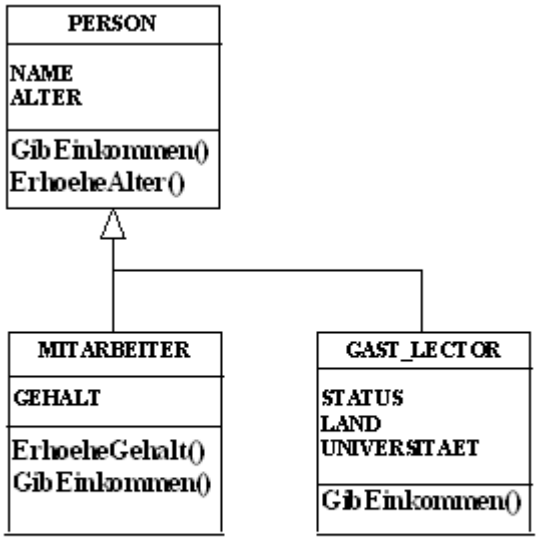


Abb.2.5. Beispiel fuer Klassendiagram

Ein Sequenzdiagramm zeigt eine Reihe von Nachrichten, die eine ausgewählte Menge von Objekten in einer zeitlich begrenzten Situation austauscht, wobei der zeitliche Ablauf betont wird (Abb.2.6).

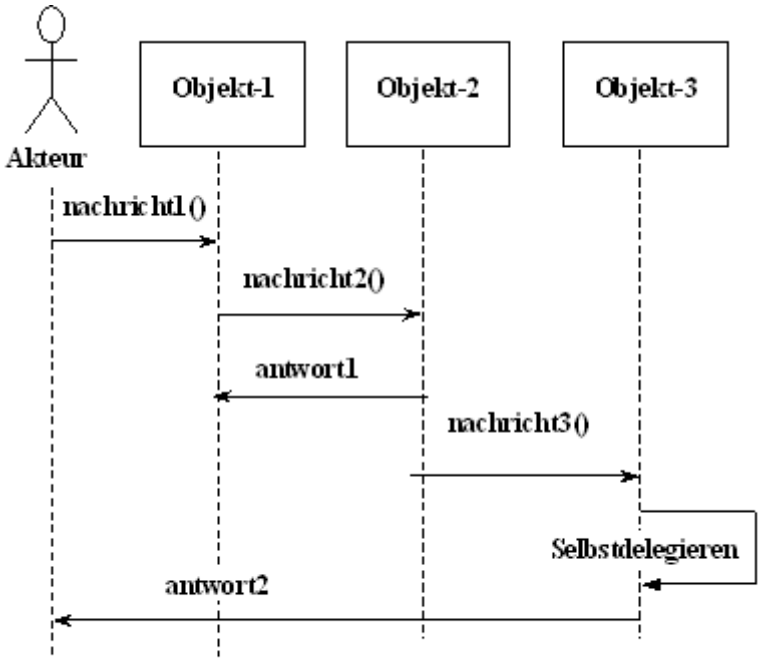


Abb.2.6. Sequenzdiagramm

Ein Kollaborationsdiagramm zeigt eine Menge von Interaktionen zwischen ausgewählten Objekten in einer bestimmten begrenzten Situation (Abb.2.7).

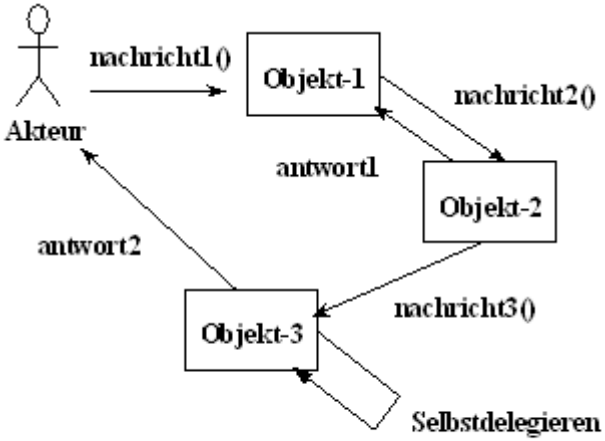


Abb.2.7. Kollaborationsdiagramm

Ein Kollaborationsdiagramm zeigt im Grunde die gleichen Sachverhalte wie ein Sequenzdiagramm, jedoch aus einer anderen Perspektive. Der zeitliche Verlauf der Kommunikation zwischen den Objekten, der beim Sequenzdiagramm im Vordergrund steht, wird beim Kollaborationsdiagramm durch Numerierung der Nachrichten verdeutlicht.

Die Nacrichten werden als Pfeile zwischen den Objekt-Linien gezeichnet. Die vertikale gestrichelte Lebenslinien symbolisieren den Steuerungsfokus (welches Objekt gerade aktiv ist).

2.4. KOMPONENTENDIAGRAMM

Eine Komponente stellt ein physisches Stück Programmkode dar, entweder als Quellkode, Binärkode, DLL oder ausführbares Programm. Komponentendiagrame zeigen die Beziehungen der Komponenten unter einander (Abb.2.8).

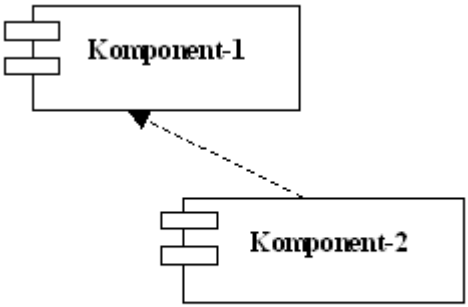


Abb.2.8. Komponentendiagramm

2.5. VERTEILUNGSDIAGRAMM

Verteilungsdiagramme zeigen, welche Komponenten und Objekte auf welchen Knoten (Prozessen, Computern) laufen, also wie diese konfiguriert sind und welche kommunikations-beziehungen dort bestehen (Abb.2.9).

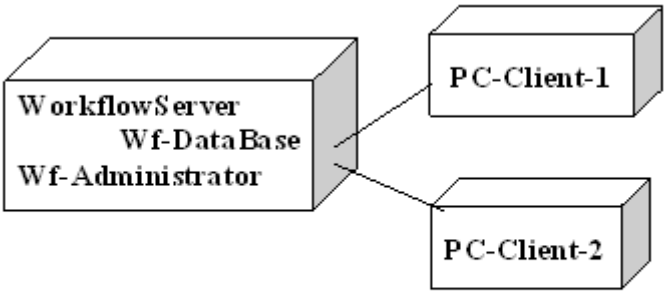


Abb.2.9. Verteilungsdiagramm

2.6. GLOSSAR

***Abhäengigkeit** (dependency) is eine Beziehung zwischen zwei Modellelementen, die zeigt, daß eine Änderung in dem einen (unabhängigen) Element eine Änderung in dem anderen (abhängigen) Element notwendig macht.*

***Akteur** (actor) ist eine außerhalb des Systems liegende Klasse, die an der in einem Anwendungsfall beschriebenen Interaktion mit dem System beteiligt ist. Akteure nehmen in der Interaktion gewöhnlich eine definierte Rolle ein.*

***Aktivität** (action state) ist ein Zustand mit einer internen Aktion und einer oder mehreren ausgehenden Transitionen, die automatisch dem Abschluß der internen Aktion folgen. Eine Aktivität ist ein einzelner Schritt in einem Ablauf. Sie kann mehrere ausgehende Transitionen haben, wenn diese durch Bedingungen unterschieden werden können.*

***Aktivitätsdiagramm** (activity diagram) ist eine spezielle Form des Zustandsdiagramms, das überwiegend oder ausschlieslich Aktivitäten entält.*

***Anwendungsfall** (use case) beschreibt eine Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure, die für die Akteure zu einem wahrnehmbaren Ergebniss führen. Ein Anwendungsfall wird stets durch einen Akteur initiert. Er ist ansonsten eine komplette, unteilbare Beschreibung.*

***Anwendungsfalldiagramm** (use case diagram) zeigtdie Beschreibung ywischen Akteuren und Anwendungsfällen.*

***Assoziation** (assotiation) beschreibt eine Relation zwischen Klassen.*

Assoziationsklasse *(assotiation class) ist ein Modellelement, das sowohl über die Eigenschaften einer Klasse, als auch über die einer Assoziation verfügt (Attributierte Assoziation).*

Assoziationsrolle *(assotiation role) is die Rolle, die ein Typ oder eine Klasse in einer Assoziation spielt. Eine Rolle repräsentiert eine Klasse in einer Assoziation.*

Beziehung *(relationship) ist eine Verbindung zwischen Modellelementen mit semantischem Gehalt. Sie ist Oberbegriff für Assoziation, Aggregation, Komposition, Generalisierung und Spezialisierung.*

Interaktionsdiagramm *(interaction diagram) ist Sammelbegriff für Sequenzdiagramm, Kollaborationsdiagramm und Aktivitätsdiagramm.*

Klassendiagramm *(class diagram) zeigt eine Menge statischer Modellelemente, allem Klassen und ihre Beziehungen.*

Knoten *(node) ist ein physisches Laufzeit-Objekt, das über Rechnerleistung (Prozessor, Speicher) verfügt. Laufzeitobjekte und Komponenten können auf Knoten residieren.*

Komponente *(component) ist ein ausführbares Softwaremodul mit eigener identität und wohldefinierten Schnittstellen (Sourcecode, Binärcode, DLL oder ausführbares Programm).*

Komponentendiagramm *(component diagram) zeigt die Organisation und Abhängigkeiten von Komponrnten.*

Komposition *(composite) ist eine strenge Form der Agregation, bei der die Teile vom Ganzen existenzabhängig sind.*

Objektdiagramm *(object diagram) zeigt Obiekte und ihre Beziehungen untereinander zu einem bestimmten Zeitpunkt.*

***Pakete** (package) sind Ansammlungen von Modellelementen beliebigen Typs. Pakete und Komponenten können zu sehr ähnlichen Zwecken verwendet werden. Während Pakete eine mehr logische Sicht darstellen, betonnen Komponneten die Physische Sicht.*

***Problembereich** (Domäne) bzw. Anwendungsgebiet, innerhalb dessen die fachliche Modellierung stattfindet.*

***Sequenzdiagramm** (sequence diagram) zeigt eine Menge von Interaktionen zwischen einer Menge ausgewählter Objekte in einer bestimmten begrenzten Situation unter Betonung der zeitlicher Abfolge.*

***Szenario** (scenario) ist eine spezifische Folge von Aktionen.*

***Transition** (transition) ist eine Zustandsübergang*

***Verantwortlichkeitsbereiche** (swimlane) sind durch Linien getrennte Bereiche in Aktivitätsdiagrammen, die Verantwortlichkeit der im Diagramm enthaltenen Elemente beschreiben.*

***Verteilungsdiagramme** (deployment diagram) zeigen, welche Komponenten und Objekte auf welchen Knoten (Prozessen, Computern) laufen.*

***Zustand** (state) ist eine Abstraktion der möglichen Attributwerte eines Objektes.*

***Zustandsdiagramm** (state diagram) zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann.*

2.7. FRAGEN UND ÜBUNGEN

1. Was versteht man unter UML und CASE ?

2. *Welche Diagrammtypen sind für Sie bekannt ?*
3. *Welche Phasen für Systementwicklung Sie kennen ?*
4. *Was für ein Unterschied gibt es zwischen statischen und dinsmischem Modellen ?*
5. *Was bedeutet der Begriff Use-Case ?*
6. *Was repräsentiert Activity Diagram ?*
7. *Was versteht man unter Interaction Diagram ?*
8. *Wozu braucht man Sequence Diagram ?*
9. *Wozu braucht man Collaboration Diagram ?*
10. *Was für ein Unterschied gibt es zwischen den Sequence und Collaboration Diagrams ?*
11. *Was repräsentiert State Diagram ?*
12. *Was repräsentiert Activity Diagram ?*
13. *Was bedeutet Klassen Diagramm und wie repräsentiert man Kommunikation zwischen Klassen ?*
14. *Was versteht man unter Component Diagram ?*
15. *Was versteht man unter Deployment Diagram ?*
16. *Man konstruiere ein Anwendungsfalldiagramm für den Lehrprozeß an der Universitätslehrstull.*
17. *Worin unterscheiden sich die Begriefe <<uses>> und <<extends>> ?*
18. *Man konstruiere ein Aktivitätsdiagramm für den Lehrprozeß an der Universitätslehrstull.*
19. *Man konstruiere ein Sequenzdiagramm für den Lehrprozeß an der Universitätslehrstull.*
20. *Man konstruiere ein Kollaborationsdiagramm für den*

Lehrprozeß an der Universitätslehrstull.

21. Man konstruiere ein Klassendiagramm für den Lehrprozeß an der Universitätslehrstull.

22. Was ist ein Komponentendiagram ?

23. Was ist ein Verteilungsdiagramm ?

LITERATUR

1. Bothe K., Surguladze G. Modern Platforms and Languages of Programming. Tbilisi, 2003.

2. Booch G., Jacobson I., Rumbaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corp., Santa Clara, 1996.

3. Bothe K., Surguladze G.. Inheritance in the Programming of MIS: from the Databases to the UML Technology. GTU, 4(437),2001

4. Seemann J., Von Gudenberg J.W. UML – Unified Modeling Language. „Informatik Spektrum“, N21,1998, Springer, S.89-90.

5. Kahlbrandt B. Software-Engineering. OO Software-Entwicklung mit der UML. Springer, Berlin, 1998.