Objektorientierte Modellierung

Udo Kelter

15.03.2003

Zusammenfassung dieses Lehrmoduls

Objektorientierte Analysemodelle modellieren sowohl Daten als auch Verhalten eines geplanten Systems. Die Konzepte der Datenmodellierung sind eine Erweiterung von Konzepten, die schon aus der ER-Modellierung bekannt sind. In diesem Lehrmodul verwenden wir Konzepte und Notationen der UML. Wir beschränken uns hier auf die wichtigsten Grundlagen, i.w. Klassendiagramme und die darin auftretenden Notationsformen.

Vorausgesetzte Lehrmodule:

obligatorisch: – Datenmodellierung mit ER-Modellen

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Übersicht und Einordnung **3**

2 Objekte, Objekttypen und Klassen **5**
2.1 Einzelobjekte 7

3 Attribute **8**

4 Operationen **11**

5 Beziehungen **12**

6 Typhierarchien **15**

7 Pakete **16**

Literatur 18
Glossar 19
Index 21

1 Übersicht und Einordnung

In diesem Lehrmodul stellen wir Grundlagen der objektorientierten Analyse (OOA), insb. Klassendiagramme vor. Klassendiagramme modellieren sowohl die Daten als auch die Funktionen eines Systems. Lernziel dieses Lehrmoduls ist primär, OOA-Modelle in UML-Notationen lesen und verstehen zu können.

Die wichtigsten grundlegenden Bücher über die OOA entstanden Ende der 80er und Anfang der 90er Jahre; die bekanntesten Autoren sind Booch [Bo91, Bo94], Coad/Yourdon [CoY91, CoY91a], Jacobson [Ja+92], Rumbaugh [Ru+91] und Shlaer/Mellor [ShM88, ShM92]. Die unterschiedlichen Methoden weisen viele gemeinsame grundlegende Konzepte auf; dies und wirtschaftliche Gründe[1] führten Mitte der 90er Jahre dazu, daß die verschiedenen Ansätze zu einer *unified modeling language* (UML) zusammengeführt wurden. Die UML wurde von der OMG als Standard übernommen, was ihre Praxisrelevanz noch erhöht hat. Wir werden uns daher weitgehend an der UML und ihren Notationen orientieren.

Die objektorientierten Analysemethoden sind deutlich jünger als die "strukturierten" Analysemethoden, wozu auch die ER-Modellierung gehört, und haben viele Konzepte der älteren Methoden, oft in verfeinerter Form, übernommen. Die objektorientierten Analysemethoden vereinigen Konzepte aus zwei Bereichen:

1. aus der Datenmodellierung mit ER-Diagrammen bzw. mit semantischen Datenmodellen; reduziert man die OOA-Methoden auf Konzepte im Bereich der Datenmodellierung, so sind diese weitgehend konsistent mit Konzepten aus der ER-Modellierungs-Welt.
2. aus den objektorientierten Programmiersprachen – die historisch gesehen ebenfalls erheblich älter sind, zu nennen ist hier Simula 67 oder Smalltalk. Von diesen wurden Konzepte wie Datenkapselung und Klassenhierarchien übernommen.

[1] Zu nennen ist hier die Firma Rational, die mehrere führende Köpfe "aufkaufte" und zu der heute die Herren Booch, Jacobson und Rumbaugh gehören.

Etwas vereinfachend kann man sagen, daß die (fortgeschrittenen) ER-Modelle ein Spezialfall der in OOA-Modellen dargestellten Datenmodelle sind. Dies ist der Grund, warum wir hier die ER-Modellierung voraussetzen (s. Lehrmodul [DMER]) und nur noch Erweiterungen gegenüber der ER-Modellierung vorstellen. Eine praktische Konsequenz hieraus ist, daß man methodisch bei der Modellierung analog zur ER-Modellierung vorgehen kann (s. Abschnitt 4 in [DMER], ferner Abschnitt 5 in [SASM]) und daß man aus einem OOA-Modell Datenbankschemata analog dazu ableiten kann, wie dies für ER-Diagramme in Abschnitt 2.2 in [TAE] vorgestellt worden ist.

Neu gegenüber der ER-Modellierung ist das grundlegende, schon in den modularen Programmiersprachen bekannte Prinzip der Datenkapselung. Dieses besagt, daß der Zustand eines Objekts nach außen hin verborgen bleibt und auf ein Objekt nur über dessen Operationen, die (hoffentlich) eine wohldefinierte Semantik haben, zugegriffen wird. Dahinter steht die softwaretechnische Erkenntnis, daß Datenstrukturen und Algorithmen zusammengehören. Den Datentypen einer Anwendung sind daher nicht nur, wie in den ER-Modellen, Attribute, sondern zusätzlich Operationen zugeordnet.

Wir hatten schon bei den ER-Modellen angemerkt, daß ein ER-Modell aus mehreren (heterogenen) Dokumenten besteht. Diese Heterogenität trifft auf OOA-Modelle noch in verstärktem Umfang zu. Die UML, an der wir uns hier orientieren, definiert insg. 8 (!) verschiedene Dokumenttypen [UML99], wovon 6 für die Analyse relevant sind:

- **Klassendiagramm** (*class diagram*)
- **Anwendungsfalldiagramm** (*use case diagram*), auch **Geschäftsprozeßdiagramm** genannt.
- Verhaltensdiagramme:
  - **Zustandsübergangsdiagramm** (*statechart diagram*)
  - **Aktivitätsdiagramm** (*activity diagram*)
  - **Interaktionsdiagramme** (*interaction diagrams*):
    - **Sequenzdiagramm** (*sequence diagram*)
    - **Kollaborationsdiagramm** (*collaboration diagram*)

Von den für die Analyse relevanten Diagrammtypen konzentrieren wir uns in diesem einführenden Lehrmodul auf das (Analyse-) Klassendiagramm[2] und werden fast nur solche Konzepte und Notationen vorstellen, die in Klassendiagrammen auftreten. Ein Klassendiagramm modelliert die relevanten Daten des Systems und die zugeordneten Operationen. Ferner strukturiert es das Gesamtsystem in Pakete.

Wegen der Vielzahl von Diagrammtypen und demzufolge auch der Zahl der Dokumente, die ein OOA-Modell konstituieren, stellt sich die Frage nach der Konsistenz dieser Dokumente verschärft. Obwohl Methoden im Prinzip unabhängig von Werkzeugen auch mit Papier und Bleistift praktiziert werden können, unterstellt die UML relativ leistungsfähige Werkzeuge und ist ohne solche, wenn man den vollen Sprachumfang ausnutzt, auch kaum praktikabel. So enthält [UML99] an vielen Stellen Hinweise, ein Werkzeug könnte diese oder jene Fähigkeiten offerieren. Dies erklärt auch teilweise, warum die UML für viele Konzepte unterschiedliche Darstellungen definiert: In Werkzeugen lassen sich aus einer Datenstruktur, die das logische Dokument (s. Abschnitt 5 in [DMER]) darstellt, relativ leicht unterschiedliche externe Darstellungen generieren.

2 Objekte, Objekttypen und Klassen

Analog zur ER-Modellierung werden interessierende Entitäten der realen Welt – Personen, Gegenstände, Ereignisse usw. – durch Objekte modelliert. **Objekte** haben stets einen Typ; dieser Objekttyp bzw. die Menge seiner Instanzen – zwischen beiden wird in Analyseklassendiagrammen nicht unterschieden – werden als Klasse bezeichnet.

Spezifiziert wird eine **Klasse** (bzw. ein **Objekttyp**) durch

- ihren Namen
- eine Liste von Attributen
- eine Liste von Operationen

[2] Eine detailreichere Variante der Klassendiagramme wird zur Darstellung von Architekturen benutzt. Derartige Entwurfsklassendiagramme werden in einem separaten Lehrmodul vorgestellt.

Der Klassenname ist normalerweise ein Substantiv im Singular und wird groß geschrieben (auch im Englischen). Er muß innerhalb eines Pakets des OOA-Modells eindeutig sein. Mit Hilfe von Paketen können große OOA-Modelle strukturiert werden; wir kommen später auf Pakete zurück. Anzustreben ist, daß Klassennamen sogar im ganzen OOA-Modell eindeutig sind.

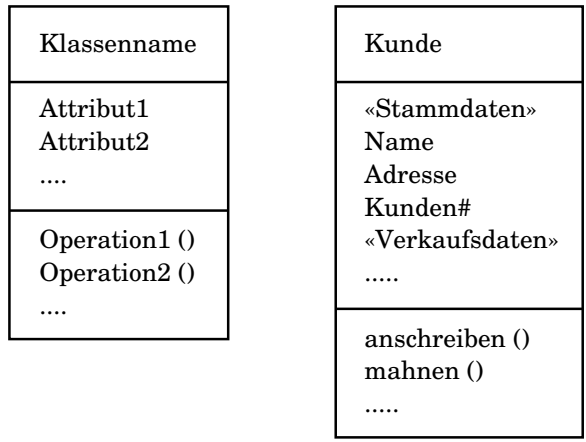

Abbildung 1: Beispiele für Klassen

Die graphische Darstellung einer Klasse in einem Klassendiagramm ist in Bild 1 angegeben. In Bild 1 findet sich auch ein Beispiel, die Klasse Kunde. Die Attribute dieser Klassen sind durch zwischengeschobene Überschriften gruppiert, um die Übersicht zu verbessern. Derartige Überschriften nennt man **Stereotypen**. Mit Stereotypen können Elemente eines OOA-Modells annotiert oder klassifiziert werden. Die Bezeichnung wird in $\ll \ldots \gg$ (französische Anführungszeichen) eingeschlossen.

Das Klassendiagramm enthält nur die Namen der Attribute und Operationen; insofern ist es analog zu einem ER-Diagramm unvollständig und muß durch weitere Angaben[3] ergänzt werden.

[3]Diese Angaben entsprechen dem Inhalt des data dictionary bei der ER-Model-

Gelegentlich werden Klassen auch noch unvollständiger dargestellt, indem die Attributliste oder die Operationenliste oder beide weggelassen werden.

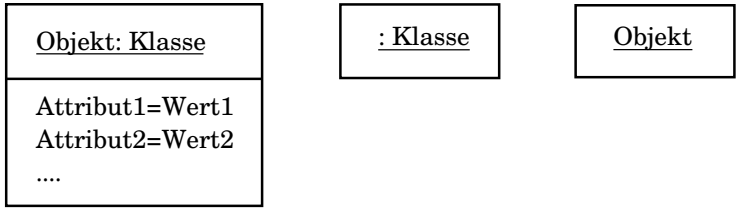

Abbildung 2: Darstellungen von Objekten

2.1 Einzelobjekte

In einigen Diagrammtypen der UML werden nicht nur Klassen, sondern auch einzelne Objekte aus einer Klasse graphisch repräsentiert. Bild 2 zeigt unterschiedliche graphische Repräsentationen für Objekte. Die Notation links in Bild 2 ist die detaillierteste. Angegeben wird

- der Name des Objekts
- der Name der Klasse bzw. des Typs des Objekts
- eine Liste von Attributwerten

Der Name des Objekts und seines Typs werden zusammen im Kopf in der Notation `Objektname: Klassenname` angegeben. Zur Unterscheidung von Klassendarstellungen wird diese Angabe unterstrichen.

Die Liste der Operationen wird nicht angegeben, diese ergibt sich aus den Angaben zur Klasse, zu der das Objekt gehört.

Die Attributwerte können in einer der folgenden Notationen angegeben werden:

```
Attributname
Attributname = Wert
Attributname : Typ = Wert
```

lierung; der Begriff data dictionary ist aber in der OOA nicht üblich.

Ein Objekt ist als Beispiel oder sozusagen als Variable aufzufassen, es stellt irgendein Objekt der Klasse dar. Der Name des Objekts ist im Prinzip belanglos, allerdings stellen Objekte mit ungleichem Namen verschiedene Objekte dar. Diese Unterscheidung wird allerdings nur selten benötigt, weswegen der Name weggelassen werden kann.

In der Mitte von Bild 2 ist die verkürzte Darstellung gezeigt; der Kopf enthält hier nur noch die Angabe `:Klassenname`. Derartige Objekte nennt man *anonym*. Rechts in Bild 2 ist eine weitere verkürzte Darstellung gezeigt, bei der die Angabe der Klasse fehlt. Dies kann sinnvoll sein, wenn die Klasse aus dem Kontext klar ist.

3 Attribute

Im einfachsten Fall werden Attribute wie in der ER-Modellierung angegeben, also mit

- Namen
- Typ
- Initialwert.

Attributnamen sind normalerweise Substantive im Singular.

Die Angabe eines Attributs widerspricht scheinbar dem Prinzip der Datenkapselung. Der Widerspruch läßt sich dadurch auflösen, daß man die Angabe eines Attributs X als kompakten Ersatz für die Angabe zweier Operationen `setzeX` und `liesX` ansieht, die einen Wert des entsprechenden Typs setzen bzw. liefern. Diese beiden Operationen werden in der Operationenliste *nicht* explizit aufgeführt.

Als Typ von Attributen sind komplexe Typen zulässig; man geht hierbei davon aus, daß auch in den Folgephasen objektorientierte Sprachen und Systeme (insb. DBMS) eingesetzt werden, die komplexe Attributwerte verarbeiten können. Diese Annahme ist oft falsch. In solchen Fällen müssen Attribute mit komplexen Typen durch Netzwerke von Tupeln realisiert werden (vgl. Abschnitt 4.3 in [DMER]).

Als Attributtyp kann auch eine benutzerdefinierte **elementare Klasse** (*support class*) verwendet werden. Elementare Klassen sind

üblicherweise nicht relevant für die Architektur eines Systems und werden deshalb nicht in das Klassendiagramm eingetragen.

Abgeleitete Attribute. In einem OOA-Modell sind redundante oder voneinander ableitbare Attribute nicht verboten, wenn sie aus Sicht der Anwender sinnvoll sind. Das Verbot redundanter Daten in Analysemodellen entstammt einer Denkweise, daß Datenmodelle praktisch unverändert in Datenbankschemata umgesetzt werden können. Für die Redundanzfreiheit in Datenbankschemata gibt es sehr gute Gründe; für Analysemodelle treffen diese Gründe nicht immer zu. Beispiele für redundante Daten sind

- die Postleitzahl und der Name des Orts (der sich schon eindeutig aus der PLZ ergibt)
- Bankleitzahl und Name der Bank
- Geburtsdatum und Alter einer Person

Ein abgeleitetes Attribut `X` kann nicht direkt gesetzt werden, da sonst die Konsistenz der Daten gefährdet wäre. Bei einer redundanzfreien Speicherung wird sein Wert bei jedem Lesen aus anderen Daten berechnet. Es gibt daher nur eine Operation `liesX`, keine Operation `setzeX`. In der Notation für Attribute wird dieser Sachverhalt ausgedrückt, indem ein `/` vor den Attributnamen gesetzt wird, beispielsweise `/Bankname`.

Selbst wenn redundante Daten erlaubt sind, sollte man vorsichtig mit ihnen sein. So ist z.B. das obige Beispiel `/Alter` u.U. falsch, weil man einen Stichtag für die Berechnung des Alters aus dem Geburtsdatum vorgeben muß und als Stichtag u.U. nicht einfach das aktuelle Datum genommen werden kann. Hier wäre eine Operation, die das Alter an einem gegebenen Stichtag berechnet, angemessener.

Klassenattribute. Attribute modellieren normalerweise Eigenschaften genau einer Entität. Oft will man auch "Attribute" wie den Durchschnittsstand aller Konten, die Zahl der Mitarbeiter, Häufigkeiten, Durchschnittswerte, Minimal- oder Maximalwerte oder andere

Aggregationen abfragen. Diese Attribute sind abgeleitet aus *allen* Objekten, die zu einer Klasse gehören. Da man in OOA-Modellen nicht zwischen einem Typ und seinen Instanzen unterscheidet, werden diese Attribute auch der jeweiligen Klasse zugeordnet.

Zur Unterscheidung von normalen Attributen werden Klassenattribute unterstrichen dargestellt. Klassenattribute sind selbst dann definiert, wenn die Klasse leer ist, also kein Objekt enthält.

Objektidentität. Zu den grundlegenden Konzepten der Objektorientierung gehört die Annahme, daß Objekte eine Identität besitzen; die Attribute eines Objekttyps brauchen daher keinen Identifikationsschlüssel zu enthalten. Es kann also sein, daß zwei Objekten existieren, bei denen alle Attribute gleich sind. Ein Beispiel könnten zwei eineiige Zwillinge (Typ: Person) sein, bei denen Alter, Haarfarbe usw. gleich sind.

Das Konzept der Objektidentität ist in der Denkwelt objektorientierter DBMS mit dem Begriff Surrogat verbunden. Ein Surrogat ist ein systemkontrolliertes Attribut aller Objekte, das beim Erzeugen eines Objekts vom DBMS einen eindeutigen Werte erhält. Die Werte werden nicht wiederverwendet, d.h. ein Surrogatwert identifiziert ein Objekt über die gesamte Existenzdauer der Datenbank hinweg. Diese Merkmale von Surrogaten entsprechen dem, was man auch intuitiv unter einer "Identität" verstehen wird.

Surrogate sind andererseits ein implementierungstechnisches Konzept; in der OOA haben die Objekte nicht automatisch ein benennbares Surrogat oder einen Identifizierer. Um auf unser Beispiel mit den Zwillingen zurückzukommen: wir hätten Probleme, eine Löschoperation zu beschreiben, die nur einen der beiden Zwillinge löscht, weil er auswandert; wir könnten anhand der bekannten Attribute nur beide Zwillinge gemeinsam löschen. Vermutlich sollten wir in unserem Beispiel die Personalausweisnummer – also letztlich einen Identifikationsschlüssel – als weiteres Attribut von Person einrichten. Wenn ein Objekttyp keinen Identifikationsschlüssel hat, sollte man stets sorgfältig prüfen, ob nicht doch ein Modellierungsfehler vorliegt.

4 Operationen

Operationen[4] modellieren "Dienste", die mit einem Objekttyp zusammenhängen. Von einer Operation wird nur der Name angegeben, diesem folgt eine leere Parameterliste: `()`. Angaben zur Sichtbarkeit (wie in objektorientierten Programmiersprachen, z.B. `public`, `protected` oder `private`) werden ebenfalls nicht gemacht. Operationen einer Klasse können stets Operationen der gleichen Klasse benutzen; hieraus folgt, daß sie insb. auf die Attribute einer Klasse zugreifen können.

Analog zu Klassenattributen gibt es **Klassenoperationen**. Klassenoperationen arbeiten auf der Menge der Objekte einer Klasse; beispielsweise sind Operationen, die Listen von Objekten drucken, Klassenoperationen. Klassenoperationen werden durch Unterstreichung kenntlich gemacht.

Normale Operationen arbeiten mit genau einem Objekt. In gewisser Hinsicht eine Zwitterstellung nehmen Operationen ein, die Objekte erzeugen oder löschen. Zum einen wird dadurch die Menge der Objekte einer Klasse natürlich verändert, d.h. diese Operationen arbeiten zumindest implizit auf der Menge der Instanzen einer Klasse. Weiterhin hat eine objekterzeugende Operation, die man auch **Konstruktoroperation** nennt, kein existierendes Objekt als Ausgangspunkt, d.h.

[4] Operationen (bzw. Funktionen oder Prozeduren) werden in der objektorientierten Begriffswelt oft als "Methoden" bezeichnet. Den Erfindern dieser Bezeichnung war wahrscheinlich unbekannt, daß Methode ein schon lange etablierter Begriff ist und daß man darunter eine systematische, strukturierte Vorgehensweise zur Erreichung eines Ziels versteht (s. Abschnitt 4 in [SASM]). Da das Überladen von Bezeichnungen in didaktischer Hinsicht eher kritisch zu sehen ist, vermeide ich sie, wenn möglich, in diesem Text.

Die OMG definiert eine Methode als Implementierung (also Rumpf) einer Operation (s. [UML99], 2.5, S. 2-34: "A method is the implementation of an operation."). Man mag bedauern, daß der vorliegende semantische Unfug nicht komplett abgeschafft worden ist, aber von Standardisierungskomitees sind normalerweise nur Kompromisse derart zu erwarten, daß zwischen völligem Unfug und einer sinnvollen Lösung interpoliert wird und dabei ein mittlerer Unfug entsteht.

Die OMG-Begriffsbildung hat immerhin den entscheidenden Vorteil, daß der Begriff Methode im Sinne von Operationsrumpf in den Bereich Entwurf bzw. Implementierung fällt und wir ihn im Kontext der Analyse ignorieren können.

als Eingabeparameter ist allenfalls der Objekttyp sinnvoll. Dennoch werden diese Operationen nicht als Klassenoperationen betrachtet, da das involvierte Objekt im Vordergrund steht.

5 Beziehungen

Beziehungen werden in der UML **Assoziationen** genannt. Sie existieren (wie bei der ER-Modellierung) nur zwischen Objekten: Beispielsweise sind zwei Personen miteinander verheiratet, oder ein Student schreibt Diplomarbeit bei einem Professor. Dargestellt werden aber analog zur ER-Modellierung nur Beziehungsmengen bzw. der Typ von Beziehungen.

Bei 2-stelligen Beziehungstypen besteht die graphische Notation aus einer Verbindungslinie zwischen den involvierten Klassen, s. Bild 3. An dieser Linie steht der Name der Assoziation. Die Leserichtung wird durch ein Dreieck, das einen Pfeil symbolisiert, angeben.

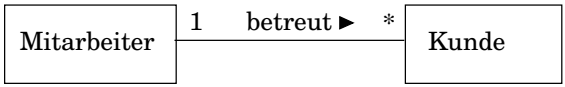

Abbildung 3: Beispiel für eine Assoziation

Rollennamen können angegeben werden, wenn dies die Verständlichkeit steigert, müssen aber nicht. Die Rollennamen werden an das jeweilige Ende der Verbindungslinie geschrieben. Bild 4 zeigt ein Beispiel.

Binäre Beziehungstypen auf einer einzigen Menge – ein Beispiel war ist_verheiratet_mit – werden **reflexive** Assoziationen genannt. Bei diesen *muß* wenigstens ein Rollenname angegeben werden.

Kardinalitäten. In den beiden letzten Bildern sind die Kardinalitäten (*multiplicities*[5]) der Rollen in der Assoziation angegeben. Im

[5]Die UML benutzt die Bezeichnung *cardinality* für die Zahl der Elemente einer Menge. Eine Menge derartiger Zahlen, also letztlich eine Teilmenge der positiven

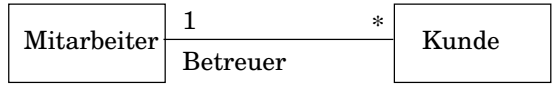

Abbildung 4: Beispiel für Rollennamen an einer Assoziation

Vergleich zu ER-Modellen müssen die Kardinalitäten immer angegeben werden, und es sind hier vielfältigere Ausdrucksformen möglich:

1	genau 1
3	genau 3
0..1	0 oder 1
2..*	2 oder mehr
*****	beliebig viele, incl. 0

Mehrere der vorstehenden Angaben können als kommagetrennte Liste angegeben werden.

Attributierte Beziehungen. Eine Assoziation kann Eigenschaften haben. In diesem Fall wird analog zu schwachen Entitätstypen eine sog. **assoziative Klasse** gebildet. In der graphischen Darstellung wird das Symbol für die assoziative Klasse durch eine gestrichelte Linie mit der Assoziationslinie verbunden. Bild 5 zeigt ein Beispiel.

Aggregationen. Aggregationen sind spezielle Assoziationen: sie modellieren Teil-von-Beziehungen. Dargestellt wird eine Aggregation (*aggregation*) durch eine Verbindungslinie mit einer Raute auf der Seite der Klasse, die das Ganze darstellt.

Ein Objekt kann Komponente mehrerer anderer Objekte sein; beispielsweise kann ein Glossar Teil mehrerer Bücher sein. In diesem Fall spricht man von einer gemeinsamen Komponente (*shared aggregation*).

Eine verschärfte Form der Aggregation ist die **Komposition** (*composite aggregation*). Die Komponente kann hier nur exklusiv einem anderen Objekt zugeordnet sein. Das Objekt, das das Ganze repräsen-

ganzen Zahlen, ist eine *multiplicity*.

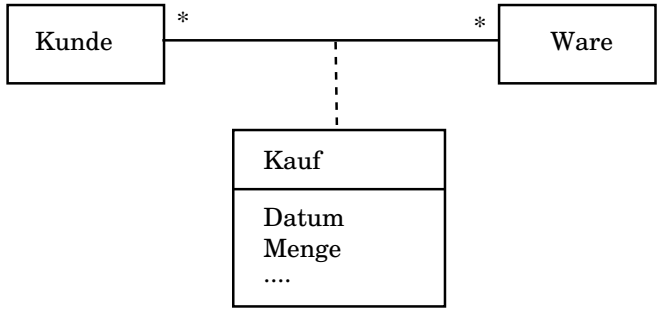

Abbildung 5: Assoziative Klasse

tiert, ist für die Erzeugung und Löschung seiner Komponenten verantwortlich; wenn es gelöscht wird, muß es alle seine Komponenten löschen. Bei einer Komposition wird in der graphischen Darstellung die Raute schwarz gefüllt, bei (einfachen) Aggregationen ist sie hingegen innen weiß. Bild 6 zeigt ein Beispiel für eine Komposition: in einem POSIX-Dateisystem enthält ein Verzeichnis mehrere Links; ein Link führt zu genau einer Datei[6].

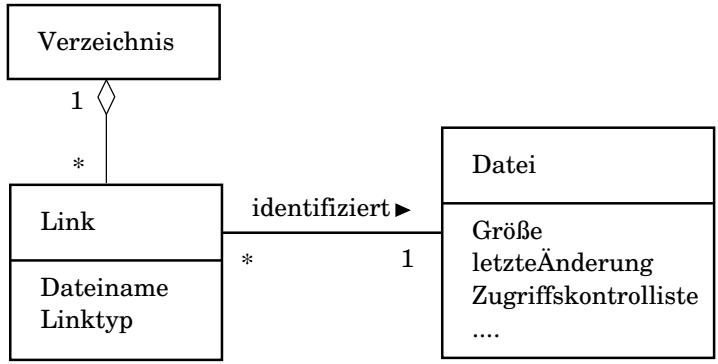

Abbildung 6: Komposition

[6]Zu derselben Datei können mehrere Links in verschiedenen Verzeichnissen führen; eine Datei ist somit nicht (exklusiver) Teil eines Verzeichnisses.

n-stellige Beziehungstypen. Die UML stellt n-stellige Beziehungstypen durch eine Raute dar (genauso wie Beziehungstypen in ER-Modellen), von der Raute aus führen Verbindungslinien zu den beteiligten Klassen. Aggregationen sind bei n-stelligen Beziehungstypen nicht erlaubt. Kardinalitäten können angegeben werden, allerdings ist ihre Bedeutung anders definiert als bei binären Beziehungstypen[7]. Wenn die Beziehungen Attribute haben, kann eine assoziative Klasse einbezogen werden.

6 Typhierarchien

Die grundlegenden Konzepte der Typhierarchien und die zugehörigen Modellierungsrichtlinien sind bei der OOA die gleichen wie bei der ER-Modellierung. Bild 7 zeigt eine Typhierarchie mit 4 Klassen. In der graphischen Darstellung geht ein Pfeil von Subtyp zum Supertyp. Die Pfeile mehrerer Subtypen können wie in Bild 8 gezeigt zusammengefaßt werden. Mehrfaches Erben ist zulässig.

Eine Subklasse erbt von ihren Superklassen

- die Attribute
- die Operationen
- die Assoziationen (incl. Aggregationen und Kompositionen)

Klassen, die keine Instanzen haben und die üblicherweise im Rahmen von Generalisierungen entstehen, werden **abstrakte Klassen** genannt. Bei abstrakten Klassen wird entweder der Name kursiv geschrieben (wie in Bild 7) oder das Merkmal `abstract` wie in Bild 8 gezeigt angegeben.

Merkmale (*properties*) sind bei diversen Modellelementen möglich und spezifizieren jeweils ein oder mehrere Merkmale des Modellelements. Geschrieben werden sie in geschweiften Klammern.

Ein weiteres Beispiel für ein Merkmal ist `{query}`; angewandt auf Operationen bedeutet es, daß diese Operation rein lesend ist, den Zustand des Objekts also nicht ändert.

[7] "The multiplicity on a role represents the potential number of instance tuples in the association when the other N-1 values are fixed." [UML99], 3.46.1.

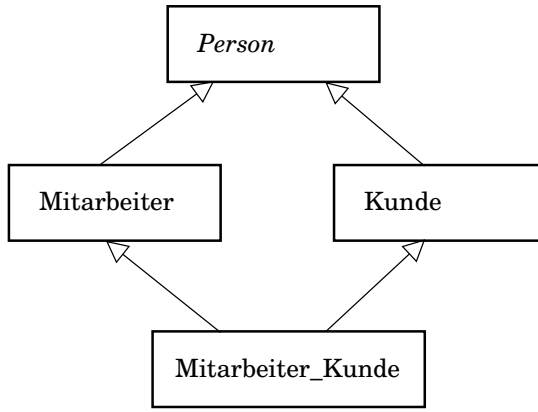

Abbildung 7: Typhierarchien

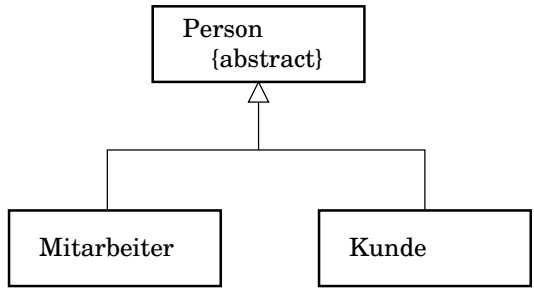

Abbildung 8: Alternative Notationen in Typhierarchien

Eine Klasse kann eine geerbte Operation **überschreiben** (*redefine, override*), indem eine eigentlich geerbte Operation bei der Unterklasse erneut angegeben wird.

7 Pakete

Pakete dienen dazu, OOA-Modelle zu strukturieren. Ein **Paket** (*package*) faßt mehrere Modellelemente zusammen; in unserem Kon-

text sind dies Klassen oder Pakete, Pakete können also geschachtelt werden. Das gesamte System ist ein alles umgebendes, "äußerstes" Paket. Enthält ein Klassendiagramm nur noch Pakete (und keine Klassen mehr), nennt man es auch ein **Paketdiagramm**.

Für Pakete gibt es zwei graphische Darstellungen. In beiden wird ein Rechteck mit einem kleinen Reiter links oben (wie bei Karteikarten) verwendet. Bei der kompakten Darstellung steht der Name des Pakets im Rechteck, sonst sind keine Angaben vorhanden (s. Bild 9).

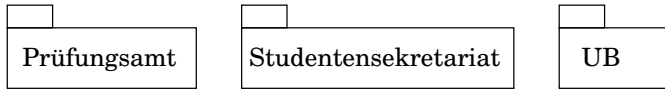

Abbildung 9: Pakete: kompakte Darstellung

Bei der detaillierten Darstellung steht der Name des Pakets im Reiter, das Rechteck ist sozusagen die Zeichenfläche, auf der die Symbole für die im Paket enthaltenen Modellelemente angeordnet werden können.

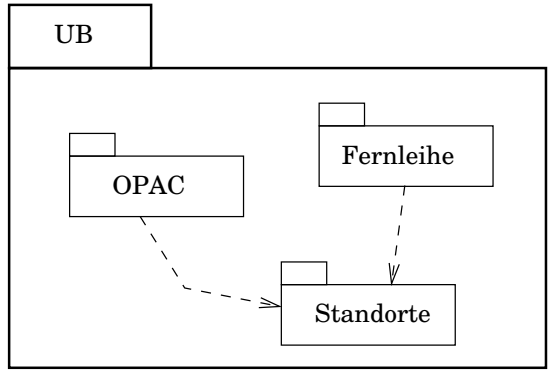

Abbildung 10: Pakete: detaillierte Darstellung

Pakete sind wichtig für die Bildung von Namensräumen. Pakete haben keine operationale Semantik, aber jedes Paket bildet einen Namensraum für die in ihm enthaltenen Modellelemente.

Zwischen Paketen kann eine "benutzt"-Beziehung (*import or access relationship*) bestehen. Graphisch notiert wird sie als gestrichelter Pfeil vom benutzenden zum benutzten Paket (s. Bild 10). Die "benutzt"-Beziehung zeigt nur die Möglichkeit einer Benutzung an; worin die Benutzung exakt besteht, wird nicht dargestellt, es ist sogar möglich, daß sie aktuell überhaupt nicht ausgenutzt wird.

Jedes Modellelement gehört zu genau einem Paket. Modellelemente können in anderen Paketen mit der Notation

`paketname::modellelement`

referenziert werden. Beispielsweise könnte

`UB::Standorte::Öffnungszeiten`

die Klasse `Öffnungszeiten` im Paket `Standorte`, das im Paket `UB` enthalten ist, bezeichnen.

Subsysteme. Pakete haben, wie schon erwähnt, keine inhaltliche Bedeutung; diese liegt allein bei den enthaltenen Modellelementen.

Ein **Subsystem** ähnelt einem Paket insofern, als es ebenfalls Modellelemente "in seinem Inneren" gruppiert, es hat aber wie eine Klasse eine Schnittstelle nach außen und definiert daher selbständig eine Semantik. Im Gegensatz zu Paketen können Subsysteme instantiierbar sein.

Modelle. Ein **Modell** in der Diktion der UML ist ein Paket, das Pakete, Modelle oder Subsysteme umfaßt, die ein Modell im allgemeinen Sinne bilden. Beispiele sind ein Analysemodell oder ein Entwurfsmodell. Modelle bilden somit die obersten "Etagen" der Pakethierarchie. Für das alles umfassende Modell, das alle anderen Modelle enthält, wird das Stereotyp «`systemModel`» vergeben.

Literatur

[Bo91] Booch, G.: Object-oriented design with applications; The Benjamin/Cummings Publ. Comp.; 1991

[Bo94] Booch, G.: Object-oriented analysis and design with applications, 2nd edition; The Benjamin/Cummings Publ. Comp.; 1994

[CoY91] Coad, Peter; Yourdon, Edward: Object-oriented analysis, 2nd edition; Yourdon Press, Prentice-Hall, Englewood, New-Jersey; 1991

[CoY91a] Coad, Peter; Yourdon, Edward: Object-oriented design; Yourdon Press, Prentice-Hall; 1991

[Ja+92] Jacobson, I.; Christerson, M.; Jinsson, P.; Övergaard, G.: Object-oriented software engineering - a use case driven approach; Addsion Wesley; 1992

[Ru+91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Object-oriented modelling and design; Prentice-Hall, Englewood Cliffs, New-Jersey; 1991

[ShM88] Shlaer, S.; Mellor, S.: Object-oriented system analysis; Yourdon Press, Prentice-Hall; 1988

[ShM92] Shlaer, S.; Mellor, S.: Object lifecycles - modeling the world in states; Yourdon Press, Prentice-Hall; 1992

[UML99] OMG Unified Modeling Language Specification (draft, Version 1.3 alpha R5, March 1999); OMG; 1999

[DMER] Kelter, U.: Lehrmodul "Datenmodellierung mit ER-Modellen"; 2003

[SASM] Kelter, U.: Lehrmodul "Systemanalyse und Systemmodellierung"; 2003

[TAE] Kelter, U.: Lehrmodul "Transformation von Analyse-Datenmodellen in Entwurfsdokumente"; 2003

Glossar

Aggregation (*aggregation*): Art von Assoziationen (Beziehungen) zwischen Klassen, die ausdrückt, daß Instanzen der einen Klasse als Teil von einer Instanz der anderen Klasse auftreten; Sonderfall: Komposition

Aktivitätsdiagramm (*activity diagram*): Diagramm, das den Kontrollfluß innerhalb eines Systems oder eines Anwendungsfalls darstellt

Anwendungsfalldiagramm, auch **Geschäftsprozeßdiagramm** (*use case diagram*): Diagramm, das stark vereinfacht dargestellte Anwendungsfälle, deren Beziehungen sowie die involvierten Akteure darstellt

Assoziation (*Assoziation*): Beziehung (in Analysediagrammen der UML stets ungerichtet)

Interaktionsdiagramm (*interaction diagram*): Sammelbegriff, unter dem Sequenzdiagramme und Kollaborationsdiagramme zusammengefaßt werden

Kardinalität (einer Rolle eines Beziehungstyps) (*cardinality, multiplicity*): beschränkt die Häufigkeit, mit der eine Entität bzw. ein Objekt eine bestimmte Rolle in Beziehungen eines bestimmten Beziehungstyps einnehmen darf

Klasse, abstrakte: Klasse, von der keine Instanzen existieren und die nur dazu dient, gemeinsame Attribute mehrerer konkreter(er) Subklassen zu tragen

Klasse, assoziative: Klasse, die die Attribute einer Assoziation trägt; wird in der UML mit einer gestrichelten Linie mit der Assoziationslinie verbunden

Klasse, elementare (*support class*): Klasse, die nicht relevant für die Systemarchitektur ist; kann als Typ von Attributen benutzt werden

Klassenattribut: Eigenschaft der Menge der Instanzen einer Klasse

Klassendiagramm (*class diagram*): Diagramm, das die Klassen eines Systems und deren Beziehungen (Subtypen, Komponenten, einfach Beziehungen u.a.) anzeigt; vereinfachte Darstellung zum Einsatz in der Systemanalyse, detailreicher zum Einsatz beim Architekturentwurf

Kollaborationsdiagramm (*collaboration diagram*): Diagramm, das Operationsaufrufe zwischen mehreren Objekten darstellt, wobei die Kommunikationsstruktur besonders hervorgehoben wird

Komposition (*composite aggregation*): Aggregation, bei der eine Komponente nur exklusiv einem anderen Objekt zugeordnet sein kann

Modell (im Kontext der UML) (*model*): Paket, das Pakete, Modelle oder Subsysteme umfaßt, die ein Modell im allgemeinen Sinne bilden

Objekt (*object*): repräsentiert eine Entität im betrachteten Weltausschnitt; hat einen durch die Werte seiner Attribute und die Beziehungen, in

denen es eine Rolle spielt, definierten Zustand; hat einen Type und bietet die durch den Objekttyp definierten Operationen an

Objektorientierte Analyse (*object-oriented analysis*): Systemanalyse unter Einsatz von Analyseklassendiagrammen und ergänzenden Dokumenten

Objekttyp (*object type*): definiert jeweils eine Menge von Attributen, von Typen von Beziehungen zu anderen Objekten und von Operationen

Operation (*operation*): Dienstleistung, die von einer Klasse angeboten wird

Paket (*package*): faßt andere Pakete und Klassen zusammen; bildet einen neuen Namensraum für diese Einheiten

Paketdiagramm (*package diagram*): stellt Pakete und ggf. deren Enthaltenseinsstruktur dar; verschiedene Darstellungen möglich

Sequenzdiagramm (*sequence diagram*): Diagramm, das die zeitliche Abfolge von Operationsaufrufen zwischen mehreren Objekten darstellt, wobei der zeitliche Ablauf besonders hervorgehoben wird

Stereotyp (*stereotype*): Annotation von Elementen in einem UML-Diagramm

Subsystem (*subsystem*): ähnlich wie Paket, definiert aber Schnittstelle nach außen

Zustandsübergangsdiagramm (*statechart diagram*): Diagramm, das die Zustände, die ein modelliertes System annehmen kann, die Zustandsübergänge, die diese auslösenden Ereignisse und die ausgelösten Aktionen darstellt

Index

abstrakte Klasse, 15
activity diagram, 4
Aggregation, 13, 19
Aktivitätsdiagramm, 4, 19
Anwendungsfalldiagramm, 4, 20
Assoziation, 12
    reflexive, 12
assoziative Klasse, 13
Attribut
    abgeleitetes, 9

Beziehung, 12

class diagram, 4
collaboration diagram, 4

Datenbankschema, 4

Geschäftsprozeßdiagramm, 4

Interaktionsdiagramm, 4, 20

Kardinalität, 12, 20
Klasse, 5
    abstrakte, 15, 20
    assoziative, 13, 20
    elementare, 20
Klassenattribut, 20
Klassendiagramm, 4, 20
Klassenoperationen, 11
Kollaborationsdiagramm, 4, 20
Komposition, 13, 20
Konstruktoroperation, 11

Merkmale, 15
Methode, 11
Modell, 18, 20
multiplicity, 13

Namensraum, 17

Objekt, 5, 7, 20
Objektorientierte Analyse, 21
Objekttyp, 5, 21
Operation, 21

package, 16
Paket, 16, 21
Paketdiagramm, 17, 21
properties, 15

Redundanz, 9

sequence diagram, 4
Sequenzdiagramm, 4, 21
statechart diagram, 4
Stereotyp, 6, 21
Subsystem, 18, 21
support class, 8

use case diagram, 4

Zustandsübergangsdiagramm, 4, 21