Transformation von Analyse-Datenmodellen in Entwurfsdokumente

Udo Kelter

04.10.2003

Zusammenfassung dieses Lehrmoduls

Aus einem Analyse-Datenmodell, z.B. einem ER-Diagramm oder einem Analyse-Klassendiagramm, können wesentliche Architekturelemente abgeleitet werden, speziell die Datenbankschemata und bestimmte Fachkonzeptklassen in der Programmarchitektur. Die Ableitung ist nicht mechanisch, sondern es sind bei diversen Details bewußte Entwurfsentscheidungen erforderlich.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Datenmodellierung mit ER-Modellen

empfohlen:
- Objektorientierte Modellierung
- Software-Architekturen

Stoffumfang in Vorlesungsdoppelstunden: 0.9

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der Systemanalyse bestimmt man mit Hilfe von Entity-Relationship- (ER-) Modellen, Analyse-Klassendiagrammen oder ähnlichen Analyse-Datenmodellen, welche Daten verwaltet werden sollen. Die Realisierung bleibt bei der Analyse weitgehend offen und ist beim Entwerfen der Systemarchitektur zu entscheiden. Betroffen sind insb. folgende Entwurfsdokumente:

- die Module bzw. Klassen, die die Datentypen der Laufzeitobjekte definieren, die Analyse-Entitäten bzw. -Objekte repräsentieren
- die Datenbankschemata oder allgemeiner gesagt die Strukturen im Datenverwaltungssystem

Die Analyse-Datenmodelle können i.a. nicht unverändert als Entwurfsdokumente verwendet werden; Gründe hierfür sind:

- Differenzen in den jeweiligen Typwelten: z.B. Typhierarchien in der Analyse (ggf. sogar mehrfaches Erben), keine Typhierarchien hingegen im Datenverwaltungssystem (z.B. in relationalen DBMS)
- die vereinfachte Darstellung in den Analyse-Datenmodellen, z.B. fehlende Verwaltung von Objektmengen

Die Transformation von Analyse-Datenmodellen in Entwurfsdokumente hängt im Detail davon ab, ob man von einem ER-Modell oder von Klassendiagrammen ausgeht und welche Typwelten in der Architekturbeschreibung bzw. Programmiersprache und im Datenbankmodell vorliegen. Dennoch gibt es gemeinsame Grundzüge bei allen Varianten. Wir konzentrieren uns hier auf diese Gemeinsamkeiten.

Die Transformation ist nicht rein mechanisch. Viele Umformungsschritte kann man mechanisch durchführen (von Hand oder durch ein Werkzeug), bei manchen Details muß jedoch zwischen verschiedenen Alternativen entschieden werden. Bei diesen Entscheidungen spielen oft Effizienz- bzw. Performance-Optimierungen eine Rolle.

2 Umsetzung von Analysemodellen in Tabellen

Wir betrachten zunächst die Strukturierung der persistenten Speicher, also der Datenverwaltungssysteme.

Sofern ein DBMS benutzt wird, besteht diese Strukturierung in der Definition von Schemata. Ein besonders wichtiges Beispiel sind relationale DBMS, bei denen Daten aus Benutzersicht in Tabellen gespeichert werden. Im Sinne unserer Betrachtungen ist wesentlich, daß relationale DBMS im Gegensatz den den Analyse-Datenmodellen keine Typhierarchien unterstützen und daß hierdurch eine wesentliche Diskrepanz zwischen beiden Typwelten entsteht.

Sofern man die Daten in Dateien speichert, die Datei einzelne Sätze enthält und die Sätze wiederum (atomare) Felder enthalten, entspricht dies strukturell ebenfalls einer Tabelle.

Sofern die Implementierungssprache nicht objektorientiert ist und zur Speicherung von Objektmengen Kollektionen von Records benutzt werden, läuft dies ebenfalls auf eine Tabellenstruktur hinaus. Die nachstehenden Umformungsregeln sind dann auch für die Entwicklung der Programmarchitektur anwendbar.

Zur Vereinfachung unterstellen wir i.f. stets Tabellen in relationalen DBMS. Wir betrachten i.f. für Entitätstypen (bzw. Klassen), Beziehungen und Attribute aus Analyse-Datenmodellen, welche "Umbaumaßnahmen" beim Übergang in Tabellen erforderlich sein können.

2.1 Umsetzung von Entitätstypen bzw. Klassen

Im Normalfall wird für jeden Entitätstyp bzw. jede Analyseklasse eine eigene Tabelle angelegt. Für jedes Attribut des Entitätstyps ist eine eigene Spalte vorhanden.

Eine Ausnahme hiervon sind schwache Entitätstypen bzw. assoziative Klassen, die dazu dienen, Attribute eines Beziehungstyps zu realisieren. Diese Typen werden durch die Tabelle mit abgedeckt, die für den Beziehungstyp angelegt wird (s.u.).

Eine weitere Ausnahme sind abstrakte Entitätstypen bzw. Klassen;

für diese wird in bestimmten Fällen keine Tabelle angelegt.

Umsetzung von Typhierarchien. Ein Entitätstyp bzw. eine Klasse repräsentiert eine Menge realer Objekte. Bei einem Entitätstyp, der keine Subtypen hat, ist dies offensichtlich die Menge der Instanzen dieses Typs. Bei einem Entitätstyp, der Subtypen hat, ist dies nicht so eindeutig, man kann ihm zwei verschiedene Entitätsmengen zuordnen:

1. die Entitäten, die exakt diesen Typ haben
2. die Entitäten, die diesen Typ oder einen Subtyp haben

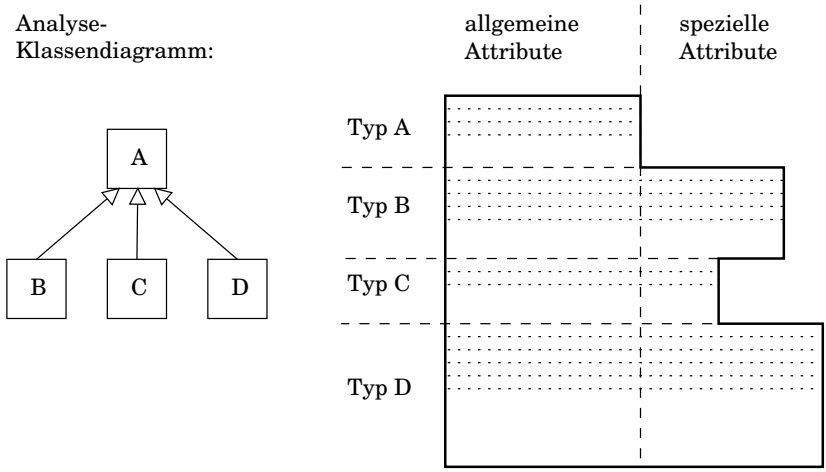

Abbildung 1: Inhomogene Entitätsmenge

Im zweiten Fall ist die Entitätsmenge inhomogen. Bild 1 zeigt rechts symbolisch die Entitätsmenge zu einem Typ A, der die Subtypen B, C und D hat. Die Inhomogenität wird veranschaulicht, indem die Tupel verschiedener Typen verschieden lang gezeichnet sind.

Bildlich gesprochen können wir nur Rechtecke auf Tabellen abbilden; das unregelmäßige Gebilde in Bild 1 muß also irgendwie mit Rechtecken überdeckt werden. Hierzu stellen wir drei Ansätze vor:

1. **horizontale Partitionierung**: bildlich gesprochen wird unser unregelmäßiges Gebilde hier horizontal durchgeschnitten, so daß Rechtecke entstehen. Für jeden Entitätstyp wird eine Tabelle gebildet, die die Tupel für diejenigen Entitäten enthält, die exakt diesen Typ haben (s. linken Teil in Bild 2).

   Sofern der Supertyp abstrakt ist, braucht für ihn natürlich keine Tabelle gebildet zu werden.

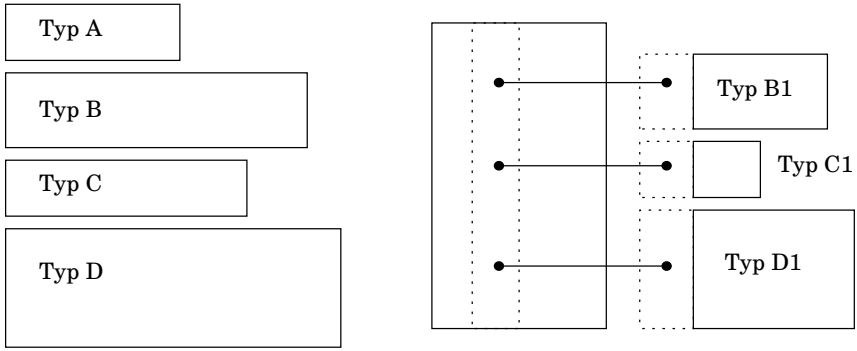

Abbildung 2: Horizontale und vertikale Partitionierung

2. **vertikale Partitionierung**: bildlich gesprochen wird unser unregelmäßiges Gebilde hier vertikal durchgeschnitten (s. rechten Teil in Bild 2). Folgende Tabellen werden gebildet:
   - für den Supertyp: eine Tabelle, die Tupel für alle Entitäten dieses Typs und seiner Subtypen enthält; die Tupel umfassen aber nur die allgemeinen Attribute, also die des Supertyps.
   - für jeden Subtyp: eine Tabelle, die Tupel für alle Entitäten dieses Typs enthält, allerdings keine kompletten Tupel, sondern nur
     (a) die speziellen Attribute dieses Subtyps sowie
     (b) einen Identifizierungsschlüssel des Supertyps.
     Der Typ dieser "Resttupel" ist in Bild 2 als B1, C1 bzw. D1 bezeichnet. Der Identifizierungsschlüssel des Supertyps wird benötigt, um die "durchgeschnittenen" Tupel der Subtypen wie-

der zusammensetzen zu können. Die Attributwerte des Identifizierungsschlüssels werden also doppelt gespeichert.

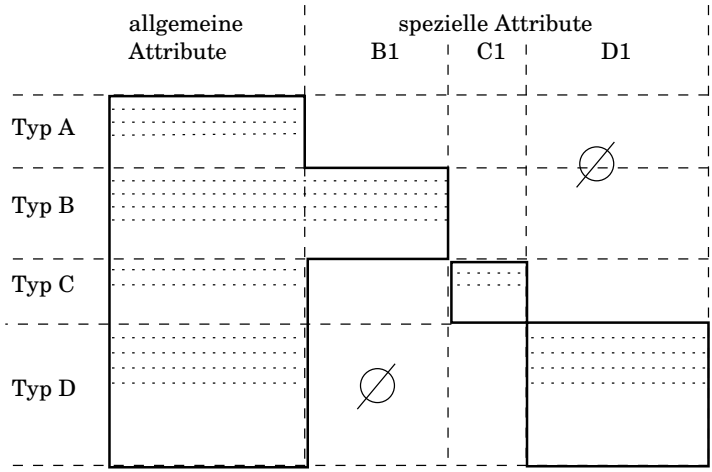

Abbildung 3: Aufrundung auf einen gemeinsamen Subtyp

3. **"Aufrundung" auf einen gemeinsamen Subtyp**: hier wird eine einzige Tabelle mit *allen* Attributen gebildet. Diese entspricht einen gemeinsamen Subtyp aller auftretenden Typen. Die Tabelle enthält jetzt für die einzelnen Tupel zu viele Spalten. In die "unzutreffenden" Felder werden Nullwerte eingetragen.

Alle Ansätze haben Vor- und Nachteile. Nachteil der horizontalen Partitionierung ist, daß die Verarbeitung der Gesamtmenge aller Tupel umständlich und ineffizienter ist, denn es müssen ja alle Tabellen durchlaufen werden. Ferner hat man nicht automatisch einen Schlüssel für die Gesamtmenge, denn selbst dann, wenn man das gleiche Attribut als Identifizierungsschlüssel in allen Tabellen festlegt, kann ein bestimmter Schlüsselwert in mehreren Tabellen auftreten.

Ein Nachteil der vertikalen Partitionierung ist offensichtlich die doppelte Speicherung der Identifizierungsschlüssel. Gravierender dürf-

te bei performancekritischen Systemen der Aufwand sein, den das Zusammensetzen der beiden Hälften des Subtyp-Tupels verursacht (zwei Zugriffe statt einem). Bei mehrstufigen Vererbungsstrukturen ist dieser Aufwand noch höher.

Die Aufrundung auf einen gemeinsamen Subtyp kann leicht zu erheblicher Platzverschwendung führen, daher ist sie nur dann anwendbar, wenn die Zahl der speziellen Attribute sehr klein ist. Außerdem werfen die Nullwerte diverse logische Probleme auf.

Die genannten Vor- und Nachteile müssen fallbezogen bewertet werden, um zu einer fundierten Entscheidung zu kommen.

2.2 Umsetzung von Beziehungstypen

Auch hier ist es der Normalfall, daß für einen Beziehungstyp eine eigene Tabelle angelegt wird. Weil sie Tupel der Entitäten verbindet, nennt man solche Tabellen auch **Verbindungstabellen**. Bei einem n-stelligen Beziehungstyp enthält die Tabelle folgende Spalten:

1. für jede Rolle einen Identifizierungsschlüssel des Entitätstyps in dieser Rolle
2. ggf. Attribute des Beziehungstyps

Der Identifizierungsschlüssel dieser Tabelle besteht aus den Identifizierungsschlüsseln der Rollen und ggf. den Diskriminator-Attributen.

Id.Schl. Rolle 1		Id.Schl. Rolle 2	Id.Schl. Rolle 3	Attribute	
AA	AB	B	C	K	L
..	..	..	..	..	..

Die vorstehende Tabelle zeigt ein Beispiel. Gegeben ist ein 3-stelliger Beziehungstyp, bei dem der Entitätstyp in der 1. Rolle einen Identifizierungsschlüssel hat, der aus dem Attributen `AA` und `AB` besteht. `B` bzw. `C` seien die Identifizierungsschlüssel für die 2. bzw. 3. Rolle. Ferner habe der Beziehungstyp die Attribute `K` und `L`, was durch einen entsprechenden schwachen Entitätstyp modelliert wird.

Das vorstehende Umsetzungsverfahren funktioniert in allen Fällen, ist aber bei 1:n-Beziehungstypen unnötig ineffizient. Ein Beispiel hierfür ist der Beziehungstyp schreibt_Diplomarbeit_bei zwischen den Analyseklassen Student und Professor (s. Bild 4).

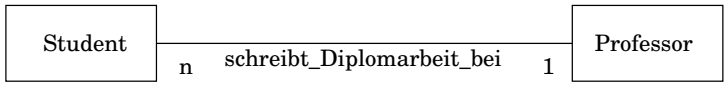

Abbildung 4: 1:n-Beziehungstyp

Die Umsetzung der beiden Analyseklassen und des Beziehungstyps zeigt Bild 5. Die Linien zwischen den Tabellen verbinden gleiche Werte.

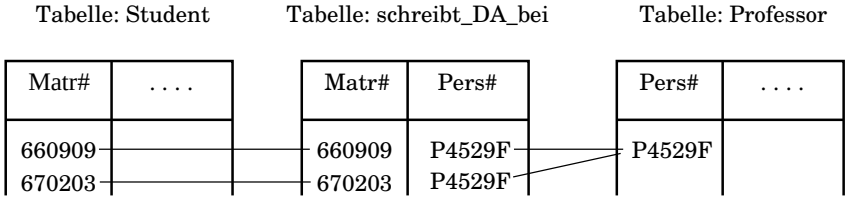

Abbildung 5: Tabellen bei einem 1:n-Beziehungstyp

Weil die Rolle "Diplomand" bei diesem Beziehungstyp die Kardinalität 1 hat, kann von einem Tupel in der Tabelle Student immer höchstens ein Tupel in der Tabelle schreibt_Diplomarbeit_bei erreicht werden. Es wäre hier einfacher, auf die Verbindungstabelle komplett zu verzichten und den Verweis auf den betreuenden Professor in die Tabelle Student als weiteres Attribut aufzunehmen (s. Bild 6).

Die kompaktere Speicherung in Bild 6 macht ein Traversieren einer Beziehung effizienter. Ferner spart sie Speicherplatz, sofern die Kardinalität der "1"-Rolle exakt [1,1] ist. Die Rolle Diplomand hat oben hingegen die Kardinalität [0,1] (nicht jeder Student schreibt Diplomarbeit). Bei den Studenten, die keine Diplomarbeit schreiben, muß als Wert in der Spalte Pers# ein Nullwert eingetragen werden. Zu

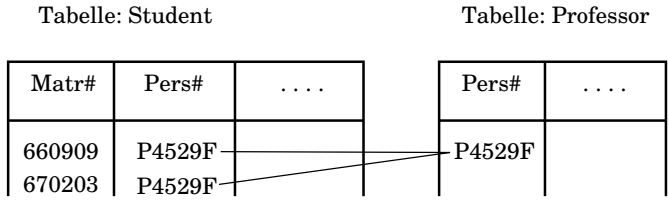

Abbildung 6: kompakte Tabellen bei einem 1:n-Beziehungstyp

einem Nullwert gibt es kein korrespondierendes Tupel in der anderen Tabelle. Sofern der Anteil der Nullwerte in der Spalte hoch ist, ist der Platzbedarf für diese Nullwerte ggf. höher als der Platzbedarf für die eingesparte Verbindungstabelle.

Umsetzung von schwachen Entitätstypen. Für einen schwachen Entitätstyp wird keine eigene Tabelle angelegt. Stattdessen wird die Tabelle des dominierenden Beziehungstyps um die Attribute des schwachen Entitätstyps erweitert.

2.3 Umsetzung von Attributen

Die Grundregel lautet hier, daß aus einem Analyseattribut eine Spalte der jeweiligen Tabelle mit einem geeigneten Typ wird. Einige Ausnahmen von dieser "direkten" Umsetzung werden unten diskutiert.

Im Analyse-Datenmodell sind die Typen der Attribute oft nur vage beschrieben. Beispielsweise kann bei Text-Attributen die Maximallänge noch offen sein oder bei Fließkommazahlen die Präzision. Derartige Entscheidungen müssen nun nachgeholt werden. Auf der anderen Seite kann es seitens des DBMS funktionale Einschränkungen oder Performance-Schwächen geben, die bei der Wahl des Typs einer Spalte berücksichtigt werden müssen.

Strukturierte Attribute. In Tabellen (bzw. Relationen) müssen die Attribute elementare Wertebereiche haben[1]. Sofern also in einem Analyse-Datenmodell ein strukturiertes Attribut auftritt, muß eine Ersatzkonstruktion gewählt werden. Das Vorgehen hängt von der Struktur des Attribute (seinem Typkonstruktor) ab:

- Sofern der Attributtyp ein Record (Verbund) ist, kann man die einzelnen Komponenten separat auf Spalten der Tabelle abbilden.
- Sofern der Attributtyp ein Array mit "wenigen" Elementen ist, kann man analog zum vorigen Fall verfahren.
- Sofern der Attributtyp eine Menge, Liste oder ähnliche Kollektion ist, wobei die Zahl der Elemente unterschiedlich und veränderbar ist, spricht man auch von einem **mehrwertigen Attribut**. Hier muß man die einzelne Elemente der Menge in einer separaten Tabelle verwalten und mit Referenzen zur Haupttabelle verbinden. Als Beispiel betrachten wir ein Bücherverzeichnis; ein Buch kann mehrere Autoren haben:

Tabelle: `bücherverzeichnis`			
`ISBN`	`Titel`	`Autoren`	`Jahr`
12345	Software Engineering	Naur, P.; Randell, B.	1968
..	..	..	..

Das Attribut `Autoren` ist mehrwertig. Es wird entfernt und ersetzt durch eine Tabelle `autoren`, in die für jeden Autor ein Tupel eingetragen wird, ferner der Schlüsselwert (hier die ISBN) des Tupels, auf den sich die Autoren-Tupel beziehen.

[1] In der Datenbank-Sprachwelt bezeichnet man dies als die erste Normalform. Die Forderung nach elementaren Wertebereichen hat mehrere Gründe. Elementare Datentypen erlauben eine feste Länge der internen Speichersätze, frühe Implementierungen von DBMS unterstützten keine variable Satzlänge. Wenn Attribute strukturiert sind, müssen die Datenmodellierungskonzepte und die Abfrage- und Änderungsoperationen wesentlich komplizierter sein. Wenn man sich z.B. in Selektionen auf einzelne Komponenten der Struktur beziehen können will, muß die Abfragesprache entsprechende Konstrukte enthalten.

Objektorientierte DBMS unterstützen strukturierte Attribute, hier erübrigt sich oft eine Umsetzung.

Tabelle: bücherverzeichnis		
ISBN	Titel	Jahr
12345	Software Engineering	1968
..	..	..

Tabelle: autoren	
ISBN	Autor
12345	Naur, P.
12345	Randell, B.
..	..

Bei einer sortierten Menge (also einer Liste) ist aus der vorliegenden Darstellung die Reihenfolge der Elemente nicht rekonstruierbar. Daher muß zusätzlich ein Positionsattribut oder Nachfolger-Referenz-Attribut vorhanden sein.

Klassenattribute. Klassenattribute sollte man nicht etwa bei jedem Tupel speichern, sondern nur einmal; hierzu wird eine eigene Tabelle benutzt, deren Spalten die Klassenattribute sind und die nur ein einziges Tupel enthält, das die Werte aller Klassenattribute enthält.

Abgeleitete Attribute. Abgeleitete Attribute werden i.d.R. nicht gespeichert, sondern bei Bedarf berechnet.

2.4 Ausnahmen und (De-) Normalisierung

Die aufgrund der bisherigen Umsetzungsregeln erhaltenen Tabellen werden oft aus technischen Gründen verändert.

Künstliche Schlüsselattribute. Eine Tabelle benötigt praktisch immer einen (Identifizierungs-) **Schlüssel**. Ein Identifizierungsschlüssel ist eine Menge von Spalten bzw. Attributen, in denen keine Wertekombination doppelt auftritt. Ein Tupel kann also durch die Werte der Schlüsselattribute, die wir als den **Schlüsselwert** des Tupel bezeichnen, eindeutig identifiziert werden. Schlüssel werden insb. benötigt, um einzelne Tupel von anderen Tabellen aus referenzieren

zu können. Ferner muß in relationalen DBMS ein Identifizierungsschlüssel als Primärschlüssel gewählt werden[2].

Manchmal müssen mehrere Attribute herangezogen werden, um die Eindeutigkeit zu erreichen, z.B. Name, Vorname, Geburtsdatum und Geburtsort bei Personen. Der Schlüsselwert eines Tupels besteht dann aus der Kombination der Werte dieser vier Attribute. Derart umfangreiche Schlüsselwerte sind für Referenzierungen und als Primärschlüssel nicht geeignet. In solchen Fällen fügt man zu den "natürlichen" Attributen des Entitätstyps ein künstliches hinzu, meist eine laufende Nummer.

Normalisierung. Daten sollten möglichst redundanzfrei gespeichert werden. Manchmal führt indessen ein Datenmodell zu redundanter Speicherung. Als Beispiel betrachten wir den Bestandskatalog einer Lehrbuchsammlung: Jedes Buch (-exemplar) hat eine Bibliotheksnummer, Autor, Titel, Verlag, ISBN usw. Wenn von einem Buch 20 Exemplare vorhanden sind, haben alle die gleichen Autoren, Titel usw. Die Lösung des Problems besteht darin, zwei Tabellen zu verwenden: eine ordnet einer ISBN den Autorn, Titel, Verlag usw. zu, die zweite gibt zu einer Bibliotheksnummer nur noch die ISBN und den Standort des Buchs an.

In Endeffekt haben wir eine "zu breite" Tabelle in zwei "schmalere" zerlegt; dieser Vorgang wird in der Datenbankliteratur Normalisierung genannt.

Die Normalisierung der Tabellen wäre nicht notwendig gewesen, wenn bereits im Analyse-Datenmodell der Entitätstyp Buchexemplar analog zerlegt worden wäre. Die Normalisierung führt allerdings zu vielen "kleinen" Typen mit wenigen Attributen, sie zerreißt manchmal Daten, die aus Anwendersicht zusammengehören. Analyse-Datenmodelle sollen die Kommunikation mit dem Anwender unterstützen, daher ist ein Analyse-Datenmodell in normalisierter Form zwar anzustreben, wichtiger aber ist die Verständlichkeit für den Anwender.

[2]Ein Primärschlüssel ist ein Identifizierungsschlüssel, über den besonders effizient auf einzelne Tupel zugegriffen werden kann, weil er durch interne Indexstrukturen (z.B. B-Bäume) unterstützt wird.

Denormalisierung. Die Denormalisierung ist in Ausnahmefällen zur Performance-Optimierung nötig. Sie führt dazu, daß mehrere Entitätstypen durch eine Tabelle realisiert werden.

Wir unterstellen hier, daß das Analyse-Datenmodell in normalisierter Form vorliegt. Wie schon erwähnt können dadurch Daten, die z.B. zu einem elementaren Geschäftsvorgang gehören (Beispiel: Anzeige der vorhandenen Exemplare eines Buchtitels) auf mehrere Tabellen verteilt sein. Die entsprechenden Tupel müssen für die Anwendung wieder verbunden und hierfür einzeln gelesen werden. Dies steigert den Rechenaufwand erheblich. Bei einem stark belasteten System mit vielen parallelen Nutzern kann dieser Mehraufwand zuviel sein, und man speichert dann lieber die Daten redundant und in nichtnormalisierter Form. Die Redundanz macht besondere Maßnahmen bei Änderungen erforderlich, auf die wir hier nicht eingehen.

3 Ableitung der Programmarchitektur

Wir unterstellen i.f.,

- daß die Implementierungssprache eine objektorientierte Programmiersprache ist,
- daß es sich um eine Programmarchitektur handelt (d.h. eine Architekturklasse repräsentiert jeweils eine Programmklasse) und
- daß UML-Klassen- und Paketdiagramme zur Notation der Architektur verwendet werden.

Unter diesen Annahmen liegen nur wenige strukturelle Differenzen zwischen den Typen in Analyse und Architektur vor. Die Entitätstypen bzw. Klassen im Analyse-Datenmodell beeinflussen folgende Teile der Architektur bzw. des kompletten Programms:

- die Klassen, die die fachlichen Datentypen definieren, i.f. als Fachkonzeptklassen bezeichnet
- die Klassen, die die Ein-/Ausgabeschnittstellen realisieren, über die einzelne Objekte dieser Klasse erzeugt, gelöscht, verändert oder gesucht werden können.

Die Bezeichnung "beeinflussen" im vorigen Abschnitt wurde bewußt vage gewählt. Ideal wäre es natürlich, wenn man die genannten Teile der Architektur komplett aus dem Analyse-Datenmodell ableiten könnte. Dies ist aber i.a. unrealistisch, tatsächlich kann man meist nur erste Versionen dieser Teile der Architektur ableiten; diese Erstversionen können im Laufe der Entwicklung wesentlich verändert werden.

Fachkonzeptklassen. Die Architekturklassen, die die fachlichen Datentypen definieren, machen relativ wenig Probleme. Die einzige wesentliche Strukturdifferenz liegt darin, daß im Analyse-Datenmodell eine implizite Objektmengenverwaltung unterstellt ist. In der Architektur muß i.d.R. für jede Analyseklasse eine **Containerklasse** vorgesehen werden. Eine Instanz der Container-Klasse wird **Container** genannt. Ein Container verwaltet mehrere Instanzen der Klasse, die die realen Entitäten repräsentieren.

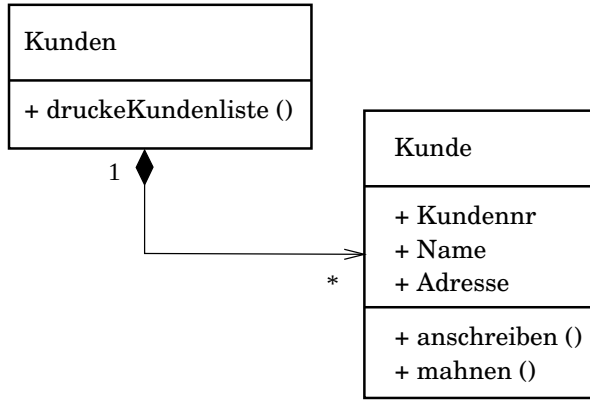

Abbildung 7: Container-Klasse

Bild 7 zeigt als Beispiel, was aus einem Entitätstyp `Kunde` entstehen kann: die Klasse `Kunde` und die Container-Klasse `Kunden`, die mehrere Objekte des Typs `Kunde` verwaltet. Diese `Kunde`-Objekte sind Komponenten des Containers, dementsprechend ist eine Kompositionsbeziehung zwischen den beiden Klassen eingetragen.

Die Container-Klasse muß eine Zugriffsstruktur realisieren, mit deren Hilfe einzelne Objekte im Container lokalisiert werden können. Beispiele für solche Zugriffsstrukturen sind u.a. die bekannten Typkonstruktoren wie Array, Liste oder Menge. Welche Zugriffsstruktur gewählt wird, hängt von den Einsatzbedingungen ab.

Sofern das Analyse-Datenmodell objektorientiert ist, sind Operationen, die mit Mengen von Objekten arbeiten, und Klassenattribute der Container-Klasse zuzuordnen.

Man beachte, daß im Analysemodell nur *eine* Klasse `Kunde` vorhanden war, die Analyseklasse wird hier also durch zwei Architekturklassen realisiert. Der im Beispiel gezeigte Übergang zur Architektur ist häufig anzutreffen, es gibt aber auch andere (kompliziertere) Realisationsformen. Entscheidend sind hier die Beobachtungen, daß

1. strukturelle Differenzen zwischen Analysemodell und Programmarchitektur auftreten
2. die Programmarchitektur deutlich detail- und umfangreicher als das Analysemodell ist.

Ein-/Ausgabeschnittstellen. Die Programmteile, die die Ein-/Ausgabeschnittstellen realisieren, sind i.a. nicht automatisch aus einem Analyse-Datenmodell ableitbar. Es liegt nahe, für jeden Entitätstyp im Analyse-Datenmodell ein Formular oder einen Tabelleneditor anzubieten, über den einzelne oder mehrere Entitäten angezeigt, eingegeben, gelöscht oder gesucht werden können. Selbst diese simple Schnittstelle ist nicht automatisch generierbar, weil z.B. Feldbeschriftungen, Feldlängen, Schriftgröße, Anordnung auf der Bildschirmseite, Hilfetexte und weitere Darstellungsmerkmale zusätzlich angegeben werden müssen. Hier können ggf. generische Module eingesetzt werden, die ein einheitliches Layout realisieren und die i.f. durch entsprechende Parametrisierung an konkrete Darstellungen angepaßt werden. In der Architektur ist dann aber i.w. nur das generische Modul sichtbar, seine Anwendung ist vergleichsweise simpel.

Ein weiteres Problem liegt darin, daß wegen der Normalisierung ein konkreter Geschäftsvorfall mehrere Entitäten betreffen kann, die

durch Beziehungen verbunden sind. Beziehungen können in generierten Ein-/Ausgabeschnittstellen nur auf Navigationsschritte abgebildet werden. Bei der Bearbeitung des Geschäftsvorfalls muß man daher mehrere Formulare bzw. Fenster öffnen anstatt alle benötigten Daten in einem einzigen Formular zu sehen. Das Hantieren mit mehreren Fenstern ist meist unergonomisch und nicht akzeptabel.

Glossar

Attribut, strukturiertes: Attribut eines Entitätstyps oder einer Klasse, das einen nichtelementaren Wertebereich hat, z.B. einen Record bzw. eine Klasse, einen Array oder eine Liste

Attribut, mehrwertiges: Attribut eines Entitätstyps oder einer Klasse, dessen Wert eine Liste oder Menge von Datenelementen ist

Container: Objekt, das dazu dient, eine Menge von Objekten einer zug. Basisklasse zu verwalten

Containerklasse: Klasse, deren Instanzen Container sind

Denormalisierung: Bildung von Datenbanktabellen, die nicht die Normalformbedingungen erfüllen; i.d.R. Verbindung zweier Tabellen zu einer einzigen, wodurch der Zugriff auf Daten performanter wird, andererseits aber Redundanz entsteht

Normalisierung: Umbildung eines Datenbankschemas dahingehend, zu große Tabellen in kleinere zu zerlegen, aus denen die ursprüngliche Tabelle wiederhergestellt werden kann

Partitionierung: Umsetzung einer Vererbungshierarchie (aus einem Analyse-Datenmodell) in tabellenförmige Strukturen

Partitionierung, horizontale: Partitionierung, bei der für jeden Typ des Analyse-Datenmodells eine Tabelle vorgesehen wird, die für jede Instanz exakt dieses Typs (ohne Subtypen) einen Eintrag enthält

Partitionierung, vertikale: Partitionierung, bei der für einen Supertyp des Analyse-Datenmodells eine Tabelle vorgesehen wird, die für jede Instanz dieses Typs oder seiner Subtypen einen Eintrag enthält; die Tabelle für einen Subtyp enthält nur noch die speziellen Attribute und einen Identifizierungsschlüssel

Index