Datenmodellierung mit ER-Modellen

Udo Kelter

08.11.2005

Zusammenfassung dieses Lehrmoduls

Die Datenmodellierung mit Entity-Relationship-Diagrammen gehört zu den wichtigsten Modellierungstechniken. Wir stellen zunächst die Konzepte und Notationsformen vor mit dem Ziel, das Lesen und Verstehen von vorgegebenen ER-Diagrammen zu lernen. In der zweiten Hälfte des Lehrmoduls gehen wir auf die Erstellung von ER-Modellen und die dabei zu beachtenden Modellierungsregeln ein.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Vorgehensmodelle
- Systemanalyse und Systemmodellierung

Stoffumfang in Vorlesungsdoppelstunden: 1.6

Inhaltsverzeichnis

1 Einordnung **3**

2 Grundlegende Konzepte der ER-Modellierung **5**
2.1 Entitätstypen 5
2.2 Attribute 6
2.3 Beziehungstypen 9
2.4 Schwache Entitätstypen 12
2.5 Kardinalitäten 15
2.6 Vererbungshierarchien 18

3 Das Data Dictionary **20**
3.1 Angaben über Attribute 21
3.2 Konsistenzkriterien 22

4 Modellierungsregeln und methodisches Vorgehen bei der ER-Modellierung **23**
4.1 Ein Vorgehensmodell für die ER-Modellierung 23
4.2 Modellierungsregeln für Entitätstypen 25
4.3 Modellierungsregeln für Attribute 28
4.4 Modellierungsregeln für Beziehungstypen 29

5 Exkurs: logische und physische Dokumente **30**

Literatur 32
Glossar 32
Index 33

1 Einordnung

Entity-Relationship- (ER-) Modelle sind Datenmodelle, sie modellieren also die relevanten Daten eines zu entwickelnden Systems. Anders gesehen stellen sie die Anforderung dar, daß das System die im ER-Modell beschriebenen Daten verwalten können muß.

Wie diese Datenverwaltungsfunktionen tatsächlich realisiert werden, sollte im Prinzip in der Analysephase noch nicht bestimmt werden, sondern erst in der Entwurfsphase. ER-Modelle sind so abstrakt, daß man aus ihnen für unterschiedliche Datenbankmanagementsysteme Schemata bzw. für dateibasierte Lösungen Satztypen ableiten kann.

In diesem Lehrmodul ist der Stoff daher so angeordnet, daß zunächst die Konzepte der ER-Modelle erklärt werden und der Leser zunächst (hoffentlich) lernt, vorhandene ER-Modelle zu lesen und als Vorlage für die Realisierung eines Systems zu benutzen. Erst anschließend gehen wir auf Modellierungsregeln und die Erstellung von ER-Modellen ein.

Historische Entwicklung. ER-Diagramme sind alt, um nicht zu sagen uralt; sie wurden erstmals in den 70er Jahren vorgestellt - die Original-Referenz ist [Ch76]. Daß sie schnell große Bedeutung gewannen und bis heute innehaben, kann als Beweis dafür gelten, daß sie ein zentrales Konzept der Informatik sind.

Publiziert wurde die ER-Modellierung in einer Datenbank-Zeitschrift, Titel und Autor waren "P.P. Chen: The entity relationship model - toward a unified view of data". Die "vereinheitlichte Sicht auf die Daten" betraf die seinerzeit virulente Konkurrenz zwischen den drei dominanten Datenbankmodellen, dem hierarchischen, dem Netzwerk- und dem relationalen Datenbankmodell. Absicht der ER-Modellierung war es, bei der Entwicklung von datenbankgestützten Informationssystemen die Entscheidung für eine bestimmte Datenbanktechnologie und damit auch für ein bestimmtes Datenbankmodell zunächst offenzulassen. Diese Absicht erfordert es, die Daten eines Systems auf einer Ebene zu beschreiben, die von technischen Details abstrahiert. Da man die Wahl eines bestimmten Datenverwaltungssystems zum Themen-

kreis der Entwurfsphase zählen wird[1], ist diese Denkweise konsistent mit dem Phasenmodell. Die ER-Modellierung hat sich daher auch weit über den Datenbankbereich hinaus verbreitet.

Es wurden sehr viele Varianten und Erweiterungen vorgestellt, vielfach als "**semantische Datenmodelle**" bzw. Modellierung bezeichnet, um auszudrücken, daß diese Modelle die Bedeutung der Daten erfassen können. Diese Vielfalt hat leider auch zu einer Begriffsvielfalt und -Uneinheitlichkeit geführt.

Die ER-Modelle können auch als vereinfachte Variante der inzwischen aktuelleren objektorientierten Analysemodelle aufgefaßt werden, wir benutzen sie hier sozusagen als Sprungbrett, um nicht alle Konzepte der objektorientierten Modelle auf einmal vorstellen zu müssen.

Im folgenden werden die wichtigsten Konzepte der ER-Modellierung vorgestellt. In anderen Quellen wird man für die gleichen Konzepte andere Bezeichnungen oder Varianten finden. Generell ist die Denk- und Begriffswelt hier etwas unsauber und unpräzise; als begrifflich gefestigter Informatiker mag man das bedauerlich finden, allerdings muß man immer wieder daran erinnern, daß Analysemodelle eine Kommunikationsbasis zwischen Anwendern und technischen Fachleuten bilden sollen und dabei formale Exzesse schlichtweg aussichtslos sind[2].

Eine erste Doppeldeutigkeit liegt übrigens schon im Begriff ER-Modell selbst. Unter einem ER-Modell versteht man erstens ein Modell des zu entwickelnden Systems, der Begriff Modell wird also im klassischen Sinn eingesetzt. Zweitens versteht man unter ER-Modell häufig die gesamte ER-Modellierung als *Methode*, also die Summe der Konzepte, Notationen und Vorgehensweisen. Die beiden Bedeutungen sind ungefähr so verschieden wie die Begriffe Programmiersprache und Programm in einer Programmiersprache. Wenn die Methode gemeint ist, spricht man oft auch vom **ER-Ansatz**.

[1] In der Praxis wird oft schon vorab entschieden, welches DBMS verwendet werden soll, z.B. weil das System im Unternehmen schon vorhanden ist.

[2] Ernstzunehmende Stimmen aus der Praxis behaupten übrigens gelegentlich, daß ER-Modelle immer noch viel zu formal sind, um mit ihrer Hilfe mit *wirklichen* Kunden über deren Anforderungen zu reden. Es gibt aber offenbar auch andere Kunden, die mit ER-Modellen ganz gut klarkommen.

2 Grundlegende Konzepte der ER-Modellierung

2.1 Entitätstypen

Zentral ist der Begriff der **Entität** bzw. synonym dazu der eines **Objekts**. Eine (Realwelt-) Entität ist irgendetwas in der realen Welt, was im gegebenen Gegenstandsbereich von Interesse und Gegenstand der Dienste des vorliegenden Softwaresystems ist. Eine Person, ein Gegenstand, ein Ereignis usw. kann eine Entität sein, sofern er, sie oder es von Interesse ist.

Jede reale Entität wird durch bestimmte Eigenschaften beschrieben; diese Eigenschaften repräsentieren in Form von Daten diese Entität innerhalb unseres Softwaresystems.

Entitäten werden bei der ER-Modellierung nur dann als relevant angesehen, wenn sie gleich scharenweise auftreten. Man betrachtet daher nur **Entitätsmengen**. Beispiele sind die Angestellten oder Kunden einer Firma, die Konten einer Bank, die Rechnungen einer Firma usw.

Man faßt nur solche Entitäten zu einer Entitätsmenge zusammen, die die gleichen Eigenschaften haben, also vom gleichen Typ sind. Jeder Entitätsmenge ist daher ein **Entitätstyp** zugeordnet; die Entitäten in der Menge sind **Instanzen** (oder Exemplare) dieses Typs.

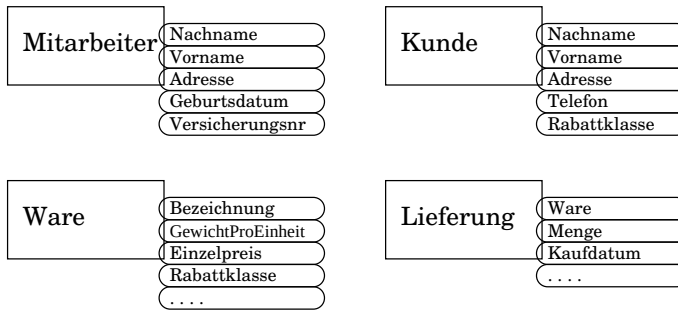

Abbildung 1: Beispiele für Entitätstypen

Man kann ER-Modelle sowohl textuell als auch graphisch darstellen. Üblich sind die graphischen Darstellungen, und diese nennt man **ER-Diagramme**. In einem ER-Diagramm wird nun jeder auftretende Entitätstyp durch ein Rechteck dargestellt, in dessen Mitte der Name des Entitätstyps steht. Bild 1 zeigt einige Beispiele.

Ob das Rechtecksymbol übrigens den Entitätstyp oder die Menge seiner Instanzen, also die Entitätsmenge darstellt, ist Glaubenssache. Ein Anwender versteht vermutlich besser die Mengen, mit Typen kann man Vererbungshierarchien, auf die wir unten eingehen, besser definieren.

Das ER-Modell erlaubt es uns nicht, zu einem Entitätstyp mehrere Entitätsmengen zu unterscheiden. Wenn wir beispielsweise eine Autovermietung mit drei Standorten haben, dann stellt der Typ "Fahrzeug" im ER-Modell implizit *alle* Fahrzeuge des Unternehmen dar. Die drei Fahrzeugmengen, die zu den Standorten gehören, können wir im ER-Modell nicht durch Symbole darstellen[3].

2.2 Attribute

Wir interessieren uns weiterhin nur dann für Realwelt-Entitäten, wenn wir irgendwelche Informationen über sie verwalten, d.h. wenn sie irgendwelche interessierenden Eigenschaften haben.

Die interessierenden Eigenschaften der Entitäten werden als **Attribute** modelliert.

Graphisch dargestellt werden Attribute meist durch Ovale. Den Sachverhalt, daß ein Attribut zu einem Entitätstyp gehört, stellt man durch eine Verbindungslinie zwischen dem Attribut und dem Entitätstyp dar oder dadurch, daß man das Attributsymbol leicht überlappend mit dem Symbol für den Entitätstyp anordnet.

Offensichtlich sollte man in der Lage sein, unterschiedliche Entitäten der realen Welt im geplanten System zu unterscheiden. Hieraus ergibt sich, daß jeder Entitätstyp ein oder mehrere Attribute haben

[3]Natürlich können wir dem Typ "Fahrzeug" das Attribut "Standort" geben; wir haben dann aber nach wie vor nur ein Symbol für alle Fahrzeuge im ER-Diagramm.

muß, die gemeinsam die Instanzen dieses Typs identifizieren. Eine solche Attributmenge nennt man einen **Identifizierungsschlüssel** oder kurz **Schlüssel**. Sofern die vorhandenen Attribute nicht ausreichen, um die Entitäten zu identifizieren (z.B. reichen in einer Adreßverwaltung Vorname, Nachname, Straße und Hausnummer i.a. *nicht* aus), muß zusätzlich ein künstliches Schlüsselattribut, in dem z.B. eine laufende Nummer vergeben wird, hinzugefügt werden. Ein Entitätstyp muß also mindestens ein Attribut haben.

Ein Entitätstyp, der nur genau ein Attribut (allgemeiner: außer dem Identifizierungsschlüssel keine weiteren Attribute) hätte, wäre allerdings seltsam: wir wüßten zwar, daß bestimmte Entitäten existieren, ansonsten aber nichts *über* diese Entitäten. Wie wir später sehen werden, kann in solchen Fällen der Entitätstyp meist aufgelöst und durch ein Attribut ersetzt werden.

Etwas formeller betrachtet wird ein Attribut definiert durch

- einen *Namen*
- einen *Wertebereich W*; Beispiele sind Strings, Integers, Felder usw.
- einen *Initial- bzw. Vorgabe-Wert*, der benutzt wird, wenn ein Attribut gelesen wird, das noch nie explizit bestimmt worden ist.

Der Begriff Attribut wird in zwei verschiedenen Kontexten benutzt und hat je nach dem Kontext eine andere Bedeutung:

- Ein **Attribut einer Entität** kann man sich vorstellen als ein Paar *(Attributname, Variable)*, worin die Variable den aktuellen Wert des Attributs enthält. Mögliche Werte eines Attributs ergeben sich aus seinem Wertebereich.

  Beispielsweise sagen wir "Beim Konto 1234567 hat das Attribut Stand den Wert 1000.00" und meinen dabei mit dem Wert des Attributs den aktuellen Inhalt dieser Variablen.
- Ein **Attribut eines Entitätstyps** besagt, daß jede Instanz dieses Entitätstyps dieses Attribut hat. Die Menge der aktuell auftretenden Werte aller Attribute können wir als eine Abbildung auffassen, die jeder Entität den aktuellen Wert für dieses Attribut zuordnet.

Wenn S ein Schlüssel ist und wir SW als Menge der aktuell vorhandenen Schlüsselwerte ansehen, dann ist ein Attribut A also eine Abbildung

$$A : SW \rightarrow W$$

Wenn man mit einem Kunden Daten modelliert, sollte man ihn mit der vorstehenden feinsinnigen Unterscheidung nicht belästigen. Als Informatiker sollte Ihnen aber klar sein, was Sie wirklich sagen bzw. meinen, wenn Sie den Begriff in einem der beiden Kontexte benutzen, und daß die beiden Benutzungen im oben definierten Sinne konsistent zueinander sind.

Darstellung von Attributen. Ein erneuter aufmerksamer Blick auf unsere graphische Darstellung läßt uns nun erkennen, daß diese nicht alle relevanten Angaben zu einem Attribut enthält: dort steht nur der Name, es fehlen der Wertebereich und der Vorgabewert. Diese Angaben können aus Platzgründen nicht mehr im Diagramm untergebracht werden und müssen (zusammen mit weiteren Angaben, z.B. einer Kurzbeschreibung des Sinns des Attributs, Ursache seiner Einführung usw.) an anderer Stelle notiert werden, und zwar im sog. **Datenlexikon** (**Data Dictionary**). Dieses ist ein weiteres Dokument, das wir neben dem ER-Diagramm entwickeln müssen; wir kommen später hierauf zurück.

Wie man in Bild 1 weiter unschwer erkennt, ist die graphische Darstellung der Entitätstypen ziemlich platzraubend; im Bild sind daher auch nicht alle sinnvollen Attribute aufgeführt. Wenn ein Entitätstyp 50 oder mehr Attribute hat - das kommt in der Praxis regelmäßig vor - wird die graphische Darstellung vollends unbrauchbar, man muß dann zu einer textuellen Darstellung übergehen und läßt die Attribute in der graphischen Darstellung komplett weg; auf die Darstellungsformen kommen wir später noch zurück. Der Wert der graphischen Darstellung liegt eher darin, Beziehungen zwischen den Entitätstypen zu visualisieren.

2.3 Beziehungstypen

Die Entitäten der realen Welt hängen fast immer durch bestimmte Beziehungen zusammen. Wenn beispielsweise die Entitätstypen aus Bild 1 die Verkaufsabteilung eines Geschäfts modellieren sollen, dann kann es z.B. sein, daß jedem Kunden ein bestimmter Mitarbeiter zugeordnet ist, der diesen Kunden betreut, oder jeder Ware (bzw. Warengruppe) ein oder mehrere Mitarbeiter, die sich mit diesen Waren auskennen und sie verkaufen dürfen. Den Sachverhalt

"Mitarbeiter A betreut den Kunden B"

fassen wir nun so auf, daß zwischen den Entitäten "Mitarbeiter A" und "Kunde B" ein **Beziehung** besteht, und zwar die Beziehung "betreut". Eine Beziehung verbindet hier also 2 konkrete Entitäten. Solche Beziehungen nennt man auch **binär**. Allgemein können in einer Beziehung auch mehr als zwei Entitäten involviert sein, z.B. im Sachverhalt "Mitarbeiter A verkauft dem Kunden B die Ware X aus dem Lager Y": "verkauft" ist hier eine vierstellige Beziehung.

Analog zu Entitätsmengen betrachten wir auch hier nur solche Beziehungen, die in Mengen auftreten. Eine **Beziehungsmenge** (*relationship set*) ist eine Menge von Beziehungen gleichen Typs. Wenn E_1, ..., E_n ($n \geq 2$) Entitätsmengen sind, kann man eine Beziehungsmenge auch als Relation im mathematischen Sinne auffassen, also als eine Teilmenge B des Kreuzprodukts der Entitätsmengen:

$$B \subseteq E_1 \times ... \times E_n$$

In unserem obigen Beispiel wären also "betreut" und "verkauft" Relationen

$$betreut \subseteq Mitarbeiter \times Kunde$$
$$verkauft \subseteq Mitarbeiter \times Kunde \times Ware \times Lager$$

Ein **Beziehungstyp** modelliert nun eine Art von Beziehung zwischen den Entitäten bestimmter Entitätsmengen. Graphisch dargestellt werden Beziehungstypen in ER-Modellen durch eine Raute, in deren Mitte der Name des Beziehungstyps steht und von der aus Verbindungslinien zu den involvierten Entitätstypen führen. Bild 2 zeigt zwei Beziehungstypen zwischen den schon früher definierten Entitätstypen.

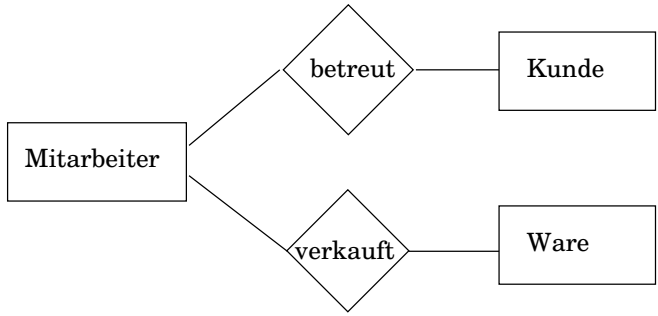

Abbildung 2: Beispiele für Beziehungstypen

Der Name eines Beziehungstyps ist typischerweise ein Verb in der Form 3. Person, Singular, Präsens.

In den beiden Beispielen in Bild 2 ist intuitiv klar, welche Rollen die Entitäten spielen, die in eine Beziehung dieses Typs involviert sind - die Ware verkauft sicherlich nicht den Mitarbeiter. Man kann allenfalls bemängeln, daß die Bezeichnung des Beziehungstyps eine bestimmte Leserichtung vorgibt; durch Übergang auf die Passiv-Form kann dieses Problem aber normalerweise leicht behoben werden (die Ware "wird verkauft vom" Mitarbeiter).

Schwieriger sind Fälle, in denen ein Beziehungstyp mehrfach den gleichen Entitätstyp verbindet. Betrachten wir hierzu als Beispiel ein Standesamt, das Einwohner verwaltet und die Beziehungstypen "ist_Mutter_von", "ist_Vater_von" und "ist_verheiratet_mit". In einer erweiterten Relationenschreibweise wären dies die Relationen

$$ist\_Mutter\_von \subseteq Einwohner[Mutter] \times Einwohner[Kind]$$
$$ist\_Vater\_von \subseteq Einwohner[Vater] \times Einwohner[Kind]$$
$$ist\_verheiratet\_mit \subseteq Einwohner[Ehefrau] \times Einwohner[Ehemann]$$

Hinter dem Namen der Entitätsmenge haben wir jeweils in [....] die Rolle notiert, die diese Entität spielt. In der graphischen Notation notieren wir die Rollennamen, sofern sie erforderlich sind, an der Verbindungslinie zwischen der Raute und dem entsprechenden Symbol für den Entitätstyp. Bild 3 zeigt einige Beispiele hierfür.

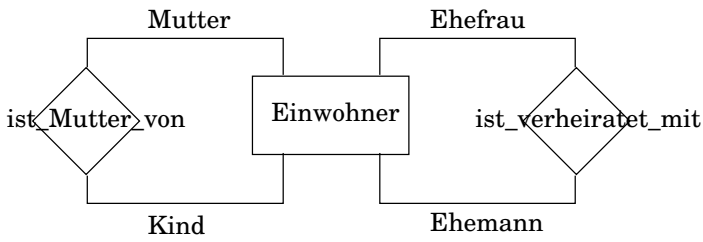

Abbildung 3: Beispiele für Rollen in Beziehungstypen

Formal können wir einen **Beziehungstyp** definieren durch[4]:

- einen Namen
- eine Menge von Rollen; jede Rolle hat einen zugeordneten Entitätstyp und ggf. einen eindeutigen Rollennnamen; sofern der Entitätstyp eindeutig innerhalb des Beziehungstyps ist, kann auf den Rollennamen verzichtet werden.

Auch bei Beziehungen wird man analog zu Entitäten verlangen, daß jede einzelne Beziehung innerhalb der Menge aller Beziehungen identifizierbar sein muß. Im Unterschied zu Entitäten können Beziehungen allerdings nicht alleine existieren, sondern nur dann, wenn auch die involvierten Entitäten vorhanden sind.

Wenn man nun Beziehungsmengen wie oben dargestellt als mathematische Relationen definiert, kann zwischen bestimmten Entitäten nur eine einzige Beziehung bestehen (eine mathematische Relation ist eine *Menge* von Tupeln, keine Multimenge). Eine einzelne Beziehung kann daher identifiziert werden, indem man die involvierten Entitäten benennt; man braucht hier also keine Schlüsselattribute. Anders gesagt können die Schlüsselattribute der involvierten Entitäten zur Identifizierung der Beziehungen genutzt werden.

Wenn allerdings zwischen den gleichen Entitäten mehrere Beziehungen des gleichen Typs existieren können, trifft das Modell der mathematischen Relationen nicht mehr zu, und wir müssen die Beziehun-

[4]Beim bisherigen Diskussionsstand; hinzu kommen Kardinalitäten und Attribute, auf die wir anschließend eingehen.

gen attributieren; dies wird im folgenden Abschnitt behandelt.

2.4 Schwache Entitätstypen

An dieser Stelle kommen wir noch einmal auf Bild 1 zurück: dort ist ein Fehler eingebaut; haben Sie ihn bemerkt?

Der Entitätstyp Lieferung - gemeint ist hier eine Position auf einem Lieferschein oder einer Rechnung - hat das Attribut Ware, das die gelieferte Ware beschreibt. Da wir Waren aber schon als Entitätstypen modelliert haben, können wir sie nicht zugleich als Attribut modellieren (auf die Modellierungsregeln gehen wir in Abschnitt 4 ausführlich ein). Statt des Attributs Ware muß jede Entität des Typs Lieferung eine Beziehung zu einer Entität des Typs Ware haben.

Wenn wir weiter annehmen, daß auch der belieferte Kunde mit in der Lieferung verzeichnet ist, hätten wir eine dreistellige Beziehung "kauft" zwischen Ware, Kunde und Lieferung (s. Bild 4); die Lieferung hätte keine Attribute namens Ware und Kunde mehr, diese Daten würden über die dreistellige Beziehung identifiziert. Ferner müßte hier sichergestellt sein, daß eine Entität des Typs Lieferung *immer genau eine* solche Beziehung zu je einer Ware und einem Kunden hat. Konkret kann eine Lieferung nur dann existieren, wenn

- eine "kauft"-Beziehung existiert, in der die Lieferung eine Rolle spielt
- die anderen Entitäten existieren, die in der "kauft"-Beziehung eine Rolle spielen

Derartige Entitäten nennen wir **schwache Entitäten**. Die Beziehung und die anderen Entitäten, von denen die Existenz einer schwachen Entität abhängt, nennt man die **dominierende Beziehung** bzw. die **dominierenden Entitäten**. Diese Begriffe gelten analog auch für die Typen dieser Entitäten und Beziehungen.

In der graphischen Darstellung werden schwache Entitätstypen unterschiedlich gekennzeichnet. Wir haben in Bild 4 eine Pfeilspitze an der Verbindungslinie zwischen dem schwachen Entitätstyp und seinem dominierenden Beziehungstyp eingetragen.

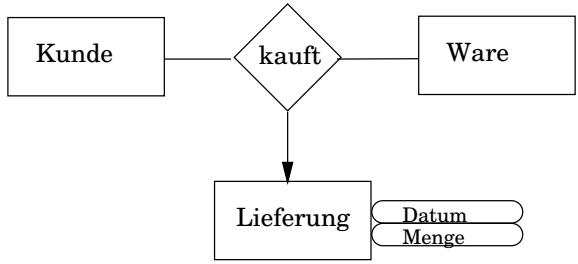

Abbildung 4: Schwache Entitätstypen

Was bedeutet die oben angegebene Integritätsbedingung nun in operationaler Hinsicht? Wenn versucht wird, eine Ware (also eine dominierende Entität) zu löschen, die gekauft worden ist, die also in eine oder mehrere "kauft"-Beziehungen involviert ist (also eine oder mehrere abhängige schwache Entitäten hat), muß die Löschung entweder abgelehnt werden oder die "kauft"-Beziehungen und die zugehörigen Lieferungsentitäten (also die abhängigen schwachen Entitäten und deren dominierende Beziehungen) müssen auch gelöscht werden.

Attributierte Beziehungen. In unserem Beispiel hätten wir übrigens auch gleich auf die Idee kommen können, daß, wenn ein Kunde eine Ware kauft, eine Beziehung zwischen dem Kunden und der Ware vorliegt[5], wir wären also als erstes auf die Beziehung "kauft" gestoßen. Wir hätten dann aber das Problem gehabt, daß diese Beziehung attributiert ist und daß wir in unserer bisherigen Notation keine Attribute an Beziehungstypen vorgesehen haben. Insofern hätten wir dann anschließend den Entitätstypen "Lieferung" als Träger dieser Attribute ebenfalls eingeführt.

Ein anderer Ausweg wäre natürlich, Attribute an Beziehungstypen zu erlauben und ähnlich wie bei Entitätstypen in die Diagramme einzuzeichnen. Dies würde allerdings die Diagramme unübersichtlich machen.

[5]Die Werbeabteilung unserer Firma gibt sich redliche Mühe, die Beziehung der Kunden zu den Waren unserer Firma möglichst fest werden zu lassen...

Mit anderen Worten entsprechen schwache Entitätstypen ziemlich genau attributierten Beziehungstypen. Ein subtiler Unterschied zwischen beiden Konzepten ist, daß ein schwacher Entitätstyp ggf. nur *einen* dominierenden Entitätstyp hat; wenn man dies auf attributierte Beziehungstypen übertragen würde, wäre dieser Beziehungstyp einstellig, was nicht sinnvoll ist. Ein Beispiel für diesen Fall ist ein Konto bei einer Bank (als dominierender Entitätstyp) und die Buchungen auf diesem Konto (als schwacher Entitätstyp). Weitere Beispiele sind Teil-von-Strukturen, bei denen die Teile nicht unabhängig vom Ganzen existieren können.

Identifikation von schwachen Entitäten bzw. attributierten Beziehungen. Wie schon im Abschnitt über nicht attributierte Beziehungen erwähnt können wir für die Identifikation von Beziehungen die Schlüssel der Entitäten heranziehen, die in die Beziehungen involviert sind. Diese Schlüsselwerte sind bereits ausreichend, sofern höchstens eine Beziehung zwischen einer Kombination von Objekten existieren kann.

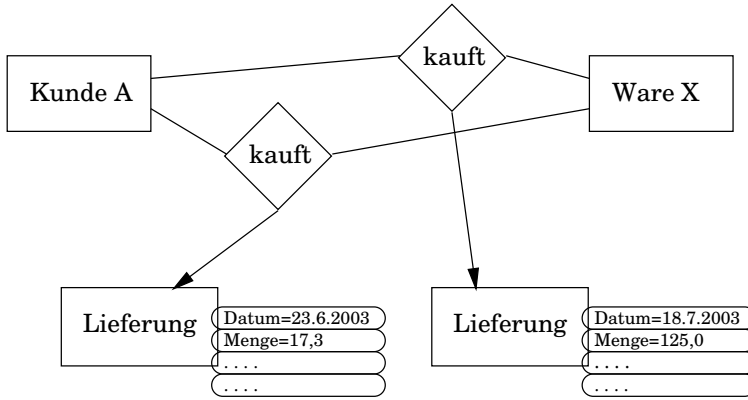

Abbildung 5: Entitäten und Beziehungen

Beim oben definierten Beziehungstyp "kauft" trifft diese Annahme nicht zu: Im Interesse unseres Kaufmanns wäre es zu wünschen, daß

ein Kunde A die gleiche Ware X mehrfach kauft. Jeder Kauf entspricht einer schwachen Entität, die durch ihre "private" dominierende Beziehung mit den gleichen dominierenden Entitäten verbunden wird. Bild 5 zeigt ein Geflecht entsprechender Entitäten und Beziehungen (nicht Typen!) als Beispiel.

In solchen Fällen können offensichtlich die schwachen Entitäten bzw. attributierten Beziehungen nicht mehr durch die Schlüssel der involvierten Entitäten identifiziert werden, man benötigt zusätzlich einen **Diskriminator**. Ein Diskriminator besteht ähnlich wie ein Schlüssel aus einem oder mehreren Attributen, deren Werte eine Beziehung identifizieren, aber nur innerhalb der Menge der Beziehungen mit den gleichen involvierten Entitäten.

Wenn unser Kaufmann z.B. seinen Kunden streng verbieten würde, eine Ware mehr als einmal am Tag bei ihm zu kaufen, dann wäre im obigen Beispiel das Attribut Kaufdatum ein brauchbarer Diskriminator. Ein solches Verbot widerspricht indessen den kaufmännischen Gepflogenheiten, außerdem wird ein normal veranlagter Kaufmann auch gerne dem gleichen Kunden mehrfach an einem Tag die gleiche Menge der gleichen Ware verkaufen. Dann reichen die beiden bisher vorhandenen Attribute nicht als Diskriminator aus, man muß ein weiteres Attribut wie z.B. die Uhrzeit hinzunehmen oder ein künstliches Zählerattribut.

Im Falle des künstlichen Zählerattributs würden die Käufe einer bestimmten Ware durch einen bestimmten Kunden an einem bestimmten Tag mit 1 beginnend durchnumeriert. Da einen der konkrete Wert des Zählers aber i.a. nicht interessiert, könnte man auch einfach die Käufe eines Kunden an allen Tagen durchnumerieren, oder noch einfacher ganz global alle Käufe; dann bräuchte man nur noch das Zählerattribut, um eine Beziehung bzw. eine schwache Entität zu identifizieren.

2.5 Kardinalitäten

In vielen Fällen darf eine Entität nicht in beliebig vielen Beziehungen eine Rolle spielen. Im zwischenmenschlichen Bereich kann man das wörtlich nehmen, im europäischen Recht kann z.B. eine Person

mit maximal einer anderen Person verheiratet sein. Bild 6 zeigt einen Ausschnitt aus dem Datenbestand eines Standesamts, der "ist_verheiratet_mit"-Beziehungen anzeigt.

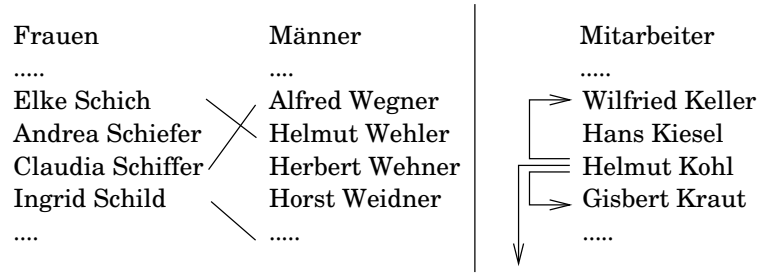

Abbildung 6: Ehepaare und Vorgesetzte

Im Islam könnte ein Mann legal zwischen 0 und 4 Beziehungen des Typs "ist_verheiratet_mit" zu Frauen haben. Eine Frau kann ungerechterweise nur mit maximal einem Mann verheiratet sein. Dieses Beispiel zeigt, daß die Häufigkeitseinschränkungen nicht einem Beziehungstyp zuzuordnen sind, sondern separat jeder Rolle in einem Beziehungstyp.

Im rechten Teil von Bild 6 sind in der Menge der Mitarbeiter eines Unternehmens die "ist_Vorgesetzter_von"-Beziehungen durch Pfeile, die von der Rolle "Vorgesetzter" ausgehen, dargestellt. Hier kann ein Mitarbeiter Vorgesetzter von beliebig vielen anderen Mitarbeitern sein, umgekehrt soll jeder Mitarbeiter höchstens einen Vorgesetzten haben. Man könnte versucht sein, zu verlangen, daß jeder Mitarbeiter auch mindestens einen Vorgesetzten haben soll; dies führt allerdings zu einem Problem mit einem speziellen Mitarbeiter, dem obersten Boß. Dieses Beispiel zeigt, daß man mit unteren Schranken für die Häufigkeit von Beziehungen vorsichtig sein sollte, auch wenn sie scheinbar naheliegen.

Einschränkungen dahingehend, wie oft eine Entität eine Rolle in einer Beziehung spielen darf, nennen wir **Kardinalitäten** . Eine Rolle R in einem Beziehungstyp B hat die Kardinalität [x,y], wenn es für jede Entität, die die Rolle R spielen kann, zu jedem Zeitpunkt mindestens

x und höchstens y Beziehungen des Typs B gibt. Statt ∞ benutzt man bei der Obergrenze die Buchstaben m oder n.

Die Rolle "Ehemann" hat beim Beziehungstyp "ist_verheiratet_mit" also beim europäischen Recht die Kardinalität [0,1] und im Islam die Kardinalität [0,4]. Die Rolle "Boß" hat beim Beziehungstyp "ist_Vorgesetzter_von" die Kardinalität [0,n], die Rolle "Untergebener" die Kardinalität [0,1].

Ein weiteres Beispiel sind schwache Entitäten: diese müssen in genau einer dominierenden Beziehung eine Rolle spielen. Diese Rolle hat bei dem dominierenden Beziehungstyp daher die Kardinalität [1,1].

Obwohl wir den Begriff Kardinalität oben für beliebige Beziehungstypen definiert haben, werden Kardinalitäten in der Praxis nur für binäre Beziehungstypen eingesetzt, wovon wir auch i.f. ausgehen.

In der graphischen Darstellung werden die Kardinalitäten einer Rolle eines Beziehungstyp *gegenüber* der Seite notiert, an der die Rolle steht. Da die Untergrenze meist 0 ist, schreibt man statt des Intervalls meist nur die Obergrenze. Bild 7 zeigt ein Beispiel.

Die gegenüberliegende Anordnung der Beschriftung ist eigentlich nicht naheliegend, hat aber den Vorteil, daß man die Kardinalitäten dann leichter lesen kann. Bild 7 kann so vorgelesen werden: "Ein Student schreibt Diplomarbeit bei maximal 1 Professor." "Ein Professor betreut die Diplomarbeiten von n Studenten."

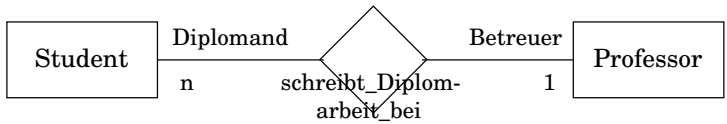

Abbildung 7: Kardinalitäten

Kardinalitäten gehören wie gesagt zu Rollen. Bei binären Beziehungstypen spricht man oft von **1:1-**, **1:n-** und **m:n-Beziehungen**[6],

[6]Genaugenommen müßten wir hier von *Beziehungstypen* reden; dies ist aber unüblich und klingt ziemlich holprig, deshalb leisten wir uns hier eine kleine sprachliche Unsauberkeit.

um die Kardinalitäten der beiden Rollen anzugeben. Bild 7 zeigt eine 1:n-Beziehung. "ist_verheiratet_mit" ist in Europa eine 1:1-Beziehung, während z.B. "ist_befreundet_mit" eine n:m-Beziehung ist.

Wie findet man nun Kardinalitäten heraus? Jedenfalls nicht dadurch, daß man vorhandene Datenbestände oder Strukturen durchzählt. Sonst käme man z.B. darauf, daß ein Vorgesetzter maximal 23 Mitarbeiter befehligen darf, weil in unserer Firma zufällig gerade die größte Abteilung 23 Mitarbeiter hat. Kardinalitäten sind aus irgendwelchen (z.B. gesetzlichen) Gründen *gewollte* Einschränkungen, die im Rahmen der Problemanalyse festgestellt werden.

2.6 Vererbungshierarchien

Oft treten ähnliche Entitätstypen auf, die viele gemeinsame und wenige spezielle Attribute haben. Beispiele sind:

- unterschiedliche Fahrzeugarten wie PKW, LKW, Omnibus, Fahrrad usw.
- unterschiedliche Konten wie Girokonto, Sparkonto oder Firmenkonto

In klassischen imperativen Programmiersprachen gibt es für derartige Fälle des Konzept der varianten Records, in objektorientierten Programmiersprachen Vererbungshierarchien. Alle ER-Modellierungsansätze beinhalten das Konzept von Vererbungshierarchien bei Entitätstypen; Vererbungshierarchien bei Beziehungstypen sind hingegen nicht üblich. Die Darstellungen differieren, wir verwenden hier einen dicken Pfeil vom Supertyp zum Subtyp. Bild 8 zeigt ein Beispiel.

Vom Supertyp vererben sich auf den Subtyp:

- die Attribute
- die Beziehungstypen

Die geerbten Attribute und Beziehungstype werden natürlich nicht mehr an den Subtypen notiert.

Ein Entitätstyp kann auch von mehreren Supertypen erben; in Bild 8 erbt der Entitätstyp Kunden-Mitarbeiter von den Entitätstypen

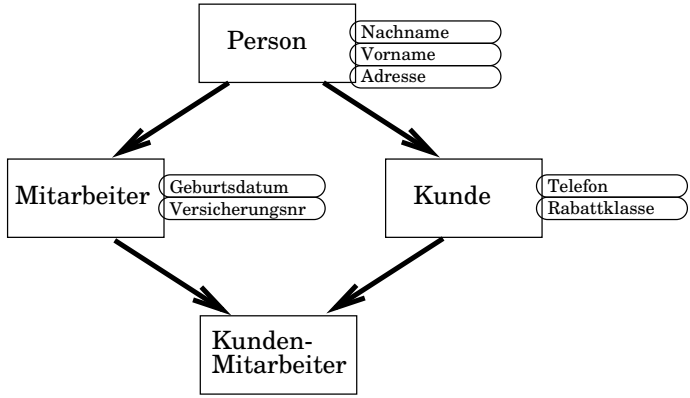

Abbildung 8: Eine Vererbungshierarchie

Kunde und Mitarbeiter. Man nennt dies **mehrfaches Erben**[7] bzw. *multiple inheritance*.

Vererbungshierarchien entstehen typischerweise auf eine der folgenden Arten:

Generalisierung: Ausgangsbasis sind 2 oder mehr ähnliche Entitätstypen (z.B. unterschiedliche Fahrzeugarten); zu diesen wird eine Verallgemeinerung in Form eines neuen Entitätstyps ("Fahrzeug") gebildet, der als **Generalisierungstyp** bezeichnet wird. Der Generalisierungstyp wird zum Supertypen der vorhandenen Entitätstypen und erhält deren gemeinsame Attribute.

Nach Konstruktion gibt es *keine* Realwelt-Entitäten, die durch Instanzen des Generalisierungstyps repräsentiert werden; Realwelt-Entitäten werden nur als Instanzen einer der Subtypen repräsentiert. Solche Entitätstypen bezeichnet man als **abstrakt**.

[7] Oft wird dies als Mehrfachvererbung bezeichnet; dies ist aber schlicht ein Übersetzungsfehler: *to inherit* bedeutet erben und nicht vererben. Mehrfachvererbung liegt in Bild 8 beim Typ Person vor, denn er vererbt seine Eigenschaften an mehrere Subtypen.

Spezialisierung: Ausgangsbasis ist hier ein Entitätstyp bzw. die zugehörige Entitätsmenge, bei der eine Teilmenge der Entitäten zusätzliche, besondere Eigenschaften hat. Man könnte beispielsweise für alle Studenten einer Universität die üblichen Angaben verwalten, für ausländische Studenten generell zusätzliche Angaben zum Visum und zur Aufenthaltserlaubnis sowie ferner vielleicht sogar spezifische Angaben abhängig vom Heimatland.

Im Gegensatz zur Generalisierung gibt es hier durchaus Instanzen des Supertyps, ferner sind Strukturen sinnvoll, wo nur *ein* Subtyp vorhanden ist; solche Strukturen wären bei der Generalisierung unsinnig.

Bei Vererbungshierarchien ist die Gleichsetzung von Entitätstyp und Entitätsmenge nicht mehr so einfach möglich. Man kann einem Entitätstyp eine der beiden folgenden Entitätsmengen zuordnen:

- die Menge der Entitäten, die *exakt* diesen Typ haben; bei Generalisierungstypen ist diese Menge stets leer.
- die Menge der Entitäten, die diesen Typ oder einen Subtyp davon haben. Die Menge dieser Entitäten ist zunächst inhomogen. Wenn man allerdings den Entitätstyp als Definition einer abstrahierenden Sicht auffaßt, ist die Menge wieder homogen, und man kann durchaus sinnvolle Funktionen darauf definieren ("Finde das Fahrzeug mit Kennzeichen XYZ", "Berechne den Gesamtwert aller Fahrzeuge").

Bei der zweiten Definition ist die Entitätsmenge des Supertyps eine Obermenge der Entitätsmenge eines Subtyps, bei der ersten nicht. Bei der ersten Definition ist eine Entität immer in genau einer Entitätsmenge enthalten, bei der zweiten kann sie (wenn auch unter verschiedenen Sichten) in mehreren Entitätsmengen enthalten sein.

3 Das Data Dictionary

Wir kommen an dieser Stelle wieder auf das Datenlexikon zurück. Es war zunächst dadurch motiviert worden, daß die Wertebereiche und

Vorgabewerte von Attributen irgendwo notiert werden müssen. Das Datenlexikon ist somit ein Dokument, das bestimmten Typnamen bestimmte Informationen zuordnet.

Mit **Data Dictionary System** bezeichnet man ein Softwaresystem, das Datenlexika verwaltet.

3.1 Angaben über Attribute

Interessierende Informationen über Attribute sind:

- die Angabe des Wertebereichs, ggf. incl. Angaben von Schreibweisen oder Restriktionen (Bsp: Postleitzahlen 5-stellig mit führenden Nullen; Autorennamen mit Initialen der Vornamen usw.)
- eine informelle Beschreibung des Sinns der Daten
- Konsistenzbedingungen mit anderen Daten
- administrative Daten wie Datum und Grund der Eintragung, Erfasser u.ä.

Eine unvermutet knifflige Frage ist, wie präzise die Angabe des Wertebereichs sein soll. Eine naheliegende Möglichkeit ist, das Typsystem irgendeiner Programmiersprache zu verwenden, also die dort definierten elementaren Datentypen und Typkonstruktoren. Nun gibt es nicht von ungefähr verschiedene Programmiersprachen in verschiedenen Anwendungsgebieten. C ist weitaus spartanischer als COBOL, bei dem man bei den Typen sogar angeben kann, in welchem Format sie angezeigt werden sollen. Eine Frage ist daher, einen wie mächtigen Vorrat an Typen man bei der Systemanalyse braucht und ob Darstellungsanweisungen mit vorgegeben werden sollen.

Eine noch weitergehende Frage ist, ob man überhaupt *exakte* Angaben machen kann oder will. Daß Postleitzahlen 5-stellig mit führenden Nullen geschrieben werden, weiß man lange im voraus (unter der optimistischen Annahme, daß Postleitzahlenreformen eher selten sind). Ob man dagegen die Personennamen in einer Anwendung auf 30 oder 40 Zeichen begrenzt oder ob die Personalnummer eine ganze Zahl oder ein Text ist, kann oder will der Kunde vielleicht so früh noch gar nicht festlegen.

Man findet daher recht unterschiedliche Meinungen hierzu in der Literatur über die ER-Modellierung. Bei einer Entscheidung sind zwei Aspekte zu berücksichtigen:

1. Wie schon erwähnt, sind ER-Modelle ja vielfach dazu gedacht, Informationssysteme, die auf einem Datenbankmanagementsystem (DBMS) basieren, zu entwickeln; ein zentraler Punkt hierbei ist die Ableitung der Datenbankschemata aus dem ER-Modell. Diese wird sehr vereinfacht und kann sogar teilweise automatisiert werden, wenn die Attributbegriffe in der Datenbank und im ER-Modell übereinstimmen[8]. Hierin liegt zugleich eine Gefahr, denn im Detail unterscheiden sich die DBMS in den Möglichkeiten zur Definition von Typen, d.h. wenn man diese Details in das Data Dictionary einschleppt, werden die Datenmodelle abhängig vom DBMS, was durch die ER-Modellierung gerade vermieden werden sollte.
2. Nur bei formalen Typangaben mit exakter Bedeutung kann man aus einem ER-Modell einen GUI-Prototypen generieren. Derartige Prototypen realisieren nur die Bedienschnittstelle (ohne die dahinterliegende Verarbeitungslogik) und geben dem Kunden eine erste Vorstellung, wie das System aussehen könnte.

Diese Aspekte sprechen eher für eine formal-exakte, aber nicht zu detaillierte Angabe der Wertebereiche von Attributen.

3.2 Konsistenzkriterien

Wir untersuchen jetzt, welche Konsistenzkriterien zwischen ER-Diagramm und Datenlexikon zu beachten sind.

Klarerweise muß für jedes Attribut, das im ER-Diagramm auftaucht, ein entsprechender Eintrag im Datenlexikon vorhanden sein.

Eine interessante Frage ist, ob auch die Entitäts- und Beziehungstypen im Datenlexikon dargestellt werden sollen. Der bei Attributen vor-

[8]DBMS benötigen übrigens intern immer ein Subsystem, das exakte Angaben über die Typen in den Schemata enthält. Dieses Subsystem wird teilweise ebenfalls als Datenlexikon bezeichnet, deckt aber einen sehr viel engeren und implementierungsnäheren Problemkomplex ab.

handene Grund, daß diese Typen im ER-Diagramm nicht vollständig definiert werden, trifft hier nicht zu: alle hinsichtlich der Typisierung relevanten Angaben sind in den Diagrammen enthalten. In den Diagrammen fehlen aber alle sonstigen oben aufgelisteten Angaben (informelle Beschreibung usw.). Diese Angaben müssen letztlich auch irgendwo erfaßt werden, naheliegenderweise ebenfalls im Datenlexikon.

Die Angabe des Wertebereichs bzw. Datentyps eines Entitäts- und Beziehungstyps muß dabei entweder offenbleiben - d.h. hier würde auf das ER-Diagramm verwiesen -, oder aber völlig konsistent mit dem ER-Diagramm sein. Im zweiten Fall könnte man einen Entitätstyp als Record mit den Attributen als Komponenten darstellen, einen Beziehungstyp als Record, dessen Komponenten Pointer (oder Fremdschlüssel) für die einzelnen Rollen sind, ferner ggf. Attribute. Um Typhierarchien zwischen Entitätstypen nachbilden zu können, müßte die Typdefinitionssprache des Datenlexikons mehrfaches Erben unterstützen.

Im zweiten Fall läge Redundanz vor, d.h. die gleichen Sachverhalte werden einmal graphisch und einmal textuell notiert. Eine doppelte Eingabe der Daten sollte durch Werkzeuge vermeidbar sein.

4 Modellierungsregeln und methodisches Vorgehen bei der ER-Modellierung

4.1 Ein Vorgehensmodell für die ER-Modellierung

In diesem Abschnitt stellen wir eines von diversen denkbaren Vorgehensmodellen vor, das den Prozeß der Entwicklung eines ER-Modells in mehrere Schritte gliedert[9]. Dieses Modell ist gut geeignet für kleine bis mittlere Systeme, also auch solche, die typischerweise im Programmierpraktikum oder bei Diplomarbeiten in der universitären Ausbildung anfallen.

Wir stellen zunächst nur den Gesamtablauf vor; detailliertere Mo-

[9]Bzgl. der Entwicklung mehrerer Modelle und genereller Modellierungsregeln sei auf [SASM] verwiesen.

dellierungsregeln folgen in den anschließenden Abschnitten.

Unsere generelle Regel, stabile Teile zuerst zu entwickeln, gilt auch innerhalb der ER-Modellierung. So sollte man beispielsweise nicht viel Arbeit in die detaillierte Beschreibung von Attributen stecken, solange unklar ist, ob die Entitätstypen, zu denen sie gehören, überhaupt von Interesse sind. Bei den Beziehungstypen sind solche, die Teil-von-Strukturen modellieren (z.B. zwischen einem Fahrzeug und seinem Motor) meist sehr stabil, andere Beziehungstypen sind weniger stabil. Folgende Vorgehensweise ist daher sinnvoll:

Schritt 1: erstellen einer initialen Menge von Kandidaten für Entitätstypen und provisorische Zuordnung von Attributen (nur Attributname)

Schritt 2: entfernen überflüssiger Entitätstypen

Schritt 3: Ähnlichkeiten zwischen den Entitätstypen identifizieren und Vererbungshierarchien aufbauen

Schritt 4: Teil-von-Beziehungen zwischen den Entitäten untersuchen und entsprechende Beziehungstypen anlegen

Schritt 5: wie 4. für sonstige Beziehungen

Schritt 6: Attribute genau bestimmen (incl. Eintragungen im Data Dictionary) und richtig plazieren, also in Vererbungshierarchien möglichst weit oben anordnen

Zu Schritt 1: Hierzu durchsucht man das Lastenheft und andere geeignete Unterlagen nach *Substantiven* bzw. führt mit dem Kunden Interviews, in denen danach gefragt wird, welche Informationen relevant sind. Besonders interessant sind:

- "Speicher" in der Problemwelt, also Gegenstände, die Ereignisse aufzeichnen
- Rollen von Personen (Beispiel: Sachbearbeiter, Administrator, Kassierer, ...)
- Organisationseinheiten (Abteilungen, Niederlassungen, Staaten)

– Rollen in Beziehungen; diese müssen von Entitäten irgendeines Typs eingenommen werden

Sehr hilfreich sind oft vorhandene Pläne und graphische Darstellungen, weil in diesen bereits eine Abstraktions- und Strukturierungsleistung vorliegt.

Zu Schritt 2: In Schritt 1 werden zunächst "auf Verdacht" alle möglichen und daher eher zu viele (Kandidaten für) Entitätstypen in die initiale Menge aufgenommen, um zunächst einen Gesamtüberblick zu bekommen und nichts zu übersehen. In Schritt 2 wird "entrümpelt" (s. Abschnitt 4.2). Nach Schritt 2 sollten bei kleinen bis mittleren Systemen ca. 30 bis 100 Entitätstypen übrig sein, nicht mehr.

Zu Schritt 3: Eine generelle Regel bei der Einführung neuer Entitätstypen in Generalisierungen bzw. Spezialisierungen (s. Abschnitt 2.6) ist, daß der neue Typ mit der Denk- und Begriffswelt der Nutzer korrespondieren soll. Auf keinen Fall sollten künstliche Entitätstypen ohne reale Entsprechung eingeführt werden. Ansonsten ist wie folgt vorzugehen:

– Bei Entitätstypen mit mehreren gemeinsamen Attributen: ggf. die Entitätstypen zu einem generelleren Entitätstyp verallgemeinern und diesem die gemeinsamen Attribute zuordnen.
– Falls eine Entitätsmenge, die zu einem Entitätstyp gehört, inhomogen ist, also nicht alle Entitäten exakt die gleichen Attribute haben: für die Entitäten mit zusätzlichen speziellen Attributen einen Subtyp bilden und dort Attribute zuordnen.

Zu Schritt 4: s. Abschnitt 4.4.

4.2 Modellierungsregeln für Entitätstypen

Ein Entitätstyp ist wahrscheinlich überflüssig, falls:

– keine Entitäten dieses Typs entstehen und wieder verschwinden können

- nur ein Nichtschlüssel-Attribut vorhanden ist: dies deutet auf einen zu hohem Detaillierungsgrad hin.
- nur eine Instanz vorhanden ist
- der Entitätstyp nur Realisierungsaspekte betrifft, also nicht direkt auf Anforderungen aus der Problemwelt basiert, sondern z.B. eine Datenstruktur zur Performance-Steigerung oder temporäre Daten (Zwischenergebnisse) betrifft. Alle effizienzsteigernden Maßnahmen, Pufferungen usw. sind Gegenstand der Entwurfs- oder Kodierungsphase und nicht Gegenstand der Analysephase.
- er redundant ist, also ein anders benannter, aber äquivalenter Entitätstyp vorhanden ist. Analog gilt dies für Attribute.

Anonyme vs. identifizierbare Realweltentitäten. Ein häufiges Problem beim Modellieren und Bilden von Entitätstypen ist die Unterscheidung zwischen anonymen und identifizierbaren Realweltentitäten. Betrachten wir hierzu einen Computerladen, der u.a. Festplatten, CD-Laufwerke und CD-Rohlinge verkauft.

- Die Festplatten und CD-Laufwerke haben jeweils eine Seriennummer, die auch beim Verkauf erfaßt werden muß, um die Garantiefrist kontrollieren zu können. Die Festplatten und CD-Laufwerke sind einzeln identifizierbare Realweltentitäten.
- Die CD-Rohlinge liegen hingegen als anonyme graue Masse auf einem Stapel im Regal. Von Interesse ist nur die Anzahl der vorhandenen Stücke, ob ein Exemplar von oben oder von unten aus dem Stapel verkauft wird, spielt keine Rolle, die Anzahl sinkt so oder so um 1. Die CD-Rohlinge sind anonym. Modelliert werden muß hier der Gesamtzahl der vorhandenen Exemplare.

Eine erste Version des Datenmodells könnte wie in Bild 9 gezeigt aussehen.

Wegen der Ähnlichkeit der beiden Attributlisten drängt sich die Idee auf, einen gemeinsamen Supertyp zu bilden. Das wäre hier allerdings falsch, was offensichtlich wird, wenn man nach dem Identifizierungsschlüssel des Supertyps fragt. Dies kann bei den CD-Rohlingen nur die Typbezeichnung (oder eine entsprechende Nummer) sein. Für

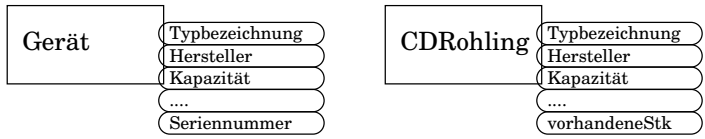

Abbildung 9: Modellierung anonymer und identifizierbarer Entitäten

Geräte ist die Typbezeichnung aber nicht identifizierend, als Identifizierungsschlüssel eignet sich hier nur die Seriennummer.

Das Datenmodell in Bild 9 enthält noch einen weiteren gravierenden Fehler: angenommen, wir haben 100 Festplatten des gleichen Typs im Lager, dann müssen die Angaben zum Hersteller, zur Kapazität usw. 100 Mal angegeben werden. Außerdem sind die Gerätedaten unabhängig davon von Interesse, ob gerade wenigstens ein Exemplar auf Lager vorhanden ist. Die richtige Vorgehensweise ist hier, die Angaben zum Gerätetyp nur einmal und separat von den Exemplaren zu speichern. Bild 10 zeigt das resultierende Datenmodell.

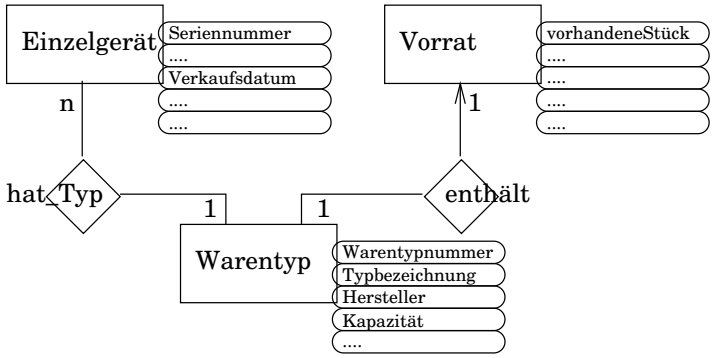

Abbildung 10: Modellierung von Typen und Instanzen

Identifizierungsschlüssel beim Entitätstyp Einzelgerät ist das Attribut Seriennummer, beim Entitätstyp Vorrat das Attribut Warentypnummer.

4.3 Modellierungsregeln für Attribute

Wir haben oben ganz bewußt die Frage, welche Typen Attribute haben können, nicht angeschnitten. Prinzipiell in Frage kommen z.B. alle Typen und Typkonstruktoren (Arrays, Listen, Mengen usw.), die man üblicherweise in Programmiersprachen benutzt. In den meisten Datenhaltungssystemen können indessen die Felder, in denen Attributwerte gespeichert werden, keine weitere Struktur haben und müssen meist sogar eine feste Länge haben. Bei relationalen DBMS bezeichnet man dies als die 1. Normalform.

Offensichtlich ist dies "nur" eine Implementierungsrestriktion, und Implementierungsaspekte sollten in der Analyse möglichst noch nicht betrachtet werden. Wenn man andererseits ER-Modelle als relativ exakte Spezifikation von Datenbankschemata auffaßt[10], stört dieser Mangel an Durchgängigkeit. In vielen ER-Modellierungsmethoden wird diese Implementierungsrestriktion daher schon bei der Analyse berücksichtigt; dementsprechend werden nur "atomare" Wertebereiche für Attribute als zulässig angesehen. Nicht zulässig sind dann

- mehrwertige Attribute (Arrays, Listen, Mengen usw.)
- Attribute, die ihrerseits Attribute haben (Records).

Solche Attribute müssen durch einen Entitätstyp ersetzt werden. Als Beispiel hierfür betrachten wir die Mitarbeiter und deren Telefone in einem Unternehmen. Wenn jeder Mitarbeiter nur ein Telefon hat, kann man die Telefonnummer als Attribut des Entitätstyps Mitarbeiter speichern.

Wenn ein Mitarbeiter mehrere Telefone haben kann, wäre das Attribut mehrwertig; in diesem Fall müssen wir statt des Attributs einen eigenen (schwachen) Entitätstyp einrichten und diesen durch einen Beziehungstyp hat_Telefon an den Entitätstyp Mitarbeiter anbinden (s. Bild 11).

Wenn über Telefone noch zusätzlich die Information verwaltet werden soll, ob dort ein Fax-Gerät angeschlossen ist, muß ebenfalls ein separater (starker) Entitätstyp verwandt werden.

[10] vgl. hierzu die Anmerkungen zur Historie des ER-Ansatzes und zur Durchgängigkeit der Modelle in Lehrmodul [SASM].

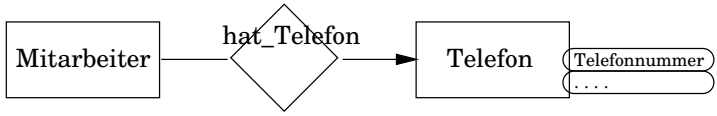

Abbildung 11: Behandlung von Attributen

Bei objektorientierten DBMS und Programmiersprachen mit Persistenzkonzept kann man beliebig komplexe Attributwerte speichern (z.B. einen Array von Records mit einer Komponente, die wieder ein Array ist). Bei solchen Datenverwaltungssystemen kann der Typ von Attributen auch nichtatomar sein.

Allerdings selbst jetzt können Komponenten innerhalb solcher Werte nicht von außen referenziert werden und insb. keine Rolle in Beziehungen spielen. Wenn also ein Attributwert oder eine Komponente daraus in Beziehung zu etwas anderem stehen kann, muß der komplexe Wert analog zu den vorstehenden Beispielen in eine Objektstruktur zerlegt werden.

4.4 Modellierungsregeln für Beziehungstypen

Beziehungen sind bei den meisten ER-Modellierungsmethoden nicht weiter in Arten untergliedert, d.h. es werden nur bei erweiterten Formen spezielle Klassen von Beziehungen unterschieden; der Nutzen einer solchen Unterscheidung liegt darin, daß man den einzelnen Klassen bestimmte semantische Merkmale zuordnen kann. Selbst wenn solche Klassen nicht durch eigene Modellierungskonstrukte direkt unterstützt werden, ist es immer noch sinnvoll, bei der Modellierung nach entsprechenden Strukturen zu suchen und durch eine geeignete Namenswahl zumindest ansatzweise etwas von der Bedeutung des Beziehungstyps zu vermitteln.

Eine häufig auftretende Klasse von Beziehungen sind Teil-von-Beziehungen. Hier suche man nach Komponenten von Objekten schon vorhandener Entitätstypen. Entsprechende Entitätstypen und Komponentenbeziehungen sollten eingeführt werden, wenn

- sie aus Sicht der Problemwelt sinnvoll sind,
- die Komponente für sich wichtig, nicht nur Attribut des Ganzen ist
- beim Löschen des Ganzen die Teile implizit mitgelöscht werden

5 Exkurs: logische und physische Dokumente

Ein ER-Modell besteht, wie wir gesehen haben, aus einem ER-Diagramm und dem Datenlexikon. Wenn diese auf Papier vorliegen, hat man also zwei physische Dokumente in der Hand. Zwischen diesen Dokumenten gibt es Bezüge, die anhand gleicher Typ- bzw. Attributnamen erkennbar sind. Obwohl ein Attribut sowohl im ER-Diagramm als auch in Datenlexikon definiert wird, wird logisch gesehen nur einmal das gleiche Attribut definiert.

Statt eines ER-Diagramms muß man übrigens in der Praxis oft mehrere verwenden: schon bei mittelgroßen Systemen hat man ca. 50 Entitätstypen und entsprechend viele Attribute und Beziehungstypen; das erforderliche DIN A0-Papier ist doch recht unhandlich und paßt in keinen normalen Kopierer. Daher zerlegt man üblicherweise das Gesamtdiagramm in Teile, die auf ein DIN-A4-Blatt passen. Ein bestimmter Entitätstyp kann dann auf mehreren Teildiagrammen vorkommen; gemeint ist immer der gleiche Entitätstyp und definiert wird auch hier nur ein Typ.

In beiden Beispielen werden "logische" Modellelemente mehrfach in Dokumenten repräsentiert, im ersten Beispiel in heterogenen Dokumenten, im zweiten in homogenen. Bei der objektorientierten Analyse werden noch viel mehr Dokumenttypen verwendet, so daß sich dort das Problem noch viel intensiver stellt.

Unter einem **logischen Dokument** verstehen wir nun eine Abstraktion des Inhalts der physischen Dokumente, in der

- jedes Modellelement nur einmal enthalten ist, auch wenn es mehrfach repräsentiert wird, die Redundanzen also vermieden sind,
- nur der "eigentliche" Sinn des Dokuments enthalten ist, nicht hingegen alle Layout-Angaben. Layout-Angaben in den physischen

Dokumenten sind z.B. die Positionen der Symbole in den Diagrammen. Obwohl diese sehr hilfreich für die Verstehbarkeit sein können, sind sie inhaltlich irrelevant: wenn man die Positionen beliebig verändert, hat man immer noch "das gleiche" Modell.

Ein **physisches Dokument**[11] repräsentiert somit i.a. nur eine Teilmenge der Information des logischen Modells[12] und fügt dieser noch Layout-Angaben hinzu.

Die Trennung zwischen logischen und physischen Dokumenten ist auch für Werkzeuge sehr relevant. Äußerlich merkt man es daran, daß es bei vielen Werkzeugen zwei Löschbefehle gibt; in den entsprechenden Menüs lauten die Einträge typischerweise (nachdem man z.B. ein Attribut selektiert hat):

delete from diagram
delete from model

diagram steht hier für physisches Dokument, *model* für logisches Dokument[13]. Wenn ein Modellelement nur im Diagramm gelöscht wird, bleibt es weiterhin im logischen Dokument erhalten. Wird es hingegen im "Modell", also im logischen Dokument gelöscht, verschwinden *alle* Vorkommen dieses Elements in den unterschiedlichen Diagrammen.

In diesen Kontext gehört auch die Frage, was die (Typ- oder Attribut-) Namen, die in den physischen Dokumenten auftreten, eigentlich bedeuten. Hier gibt es zwei Interpretationen, was eine Name ist:

[11] Unlogischerweise verstehen wir hier auch nicht ausgedruckte elektronische Dokumente als physische Dokumente.

[12] Man könnte von "Projektionen" des oder "Sichten" auf das logische Dokument reden, diese Bezeichnungen haben aber in der Datenbank-Begriffswelt eine andere Bedeutung.

[13] Diese Ausdrucksweise ist in der Tat griffig, allerdings benötigt diese Begriffsbildung ein explizit vorhandenes Gesamtmodell, die Elemente in Diagrammen sind sozusagen Referenzen auf Elemente des Gesamtmodells. Für CASE-Werkzeuge, die intern so ähnlich arbeiten, paßt diese Begriffswelt gut. Beim Arbeiten mit Papier und Bleistift paßt sie hingegen weniger gut.

1. eine Referenz auf das Modellelement (für POSIX-Fans: sozusagen ein symbolischer Link)
2. das Modellelement selbst (für POSIX-Fans: sozusagen ein harter Link)

Der Unterschied wird am klarsten, wenn man die Wirkung einer Änderung des Namens beschreibt. Im ersten Fall wird nach der Änderung ein anderes Modellelement referenziert (oder versehentlicherweise gar keines), das logische Dokument bleibt aber völlig unverändert. Im zweiten Fall wird das logische Dokument verändert; die Änderung muß in alle betroffenen physischen Dokumente propagiert werden.

Die vorstehende Unterscheidung ist übrigens kein rein technisches Problem der Werkzeuge; auch dann, wenn Sie mit Papier und Bleistift arbeiten und in einem Diagramm einen Entitätstyp, der noch einmal ein einem anderen Diagramm vorkommt, durchstreichen, stellt sich die Frage, was sie damit eigentlich gemeint haben.

Literatur

[Ch76] Chen, P.P.: The entity relationship model - toward a unified view of data; ACM TODS 1:1, p.9-36; 1976/03

[SASM] Kelter, U.: Lehrmodul "Systemanalyse und Systemmodellierung"; 1999/10

Glossar

Attribut: modelliert eine Eigenschaft einer Entität bzw. eines Entitätstyps; bei Entitätstypen formal definiert durch Namen, Wertebereich und Vorgabe-Wert; bei einer Entität definiert durch Namen und Variable, die den aktuellen Wert enthält

Beziehung (*relationship*): Sachverhalt, in den mehrere Entitäten involviert sind

Beziehungstyp (*relationship type*): gemeinsame Struktur einer homogenen Menge von Beziehungen; definiert durch einen Namen und eine

Menge von Rollen und deren zugehörige Bezeichnung und Entitätstyp, ferner ggf. beschreibende Attribute

Data Dictionary: Dokument, in dem Datendefinitionen verwaltet werden; auch entsprechende Komponente eines DBMS

Diskriminator (eines schwachen Entitätstyps): wird benötigt, wenn mehrere Beziehungen des zug. Entitätstyps zwischen den gleichen Entitäten existieren können; besteht aus einem oder mehrere Attributen, die innerhalb dieser Beziehungsmenge eindeutige Werte haben

Entität (*entity*): Person, Gegenstand, Ereignis usw. in der realen Welt, die, der oder das im gegebenen Gegenstandsbereich von Interesse ist

Entität, schwache (*weak entity*): modelliert die Eigenschaften einer Beziehung

Entity-Relationship-Modell: graphisch notiertes Datenmodell

Generalisierung: Bildung eines gemeinsamen Supertyps zu mehreren vorhandenen Entitätstypen

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

Spezialisierung: Bildung eines Subtyps zu einem vorhandenen Entitätstyp

Index

Attribut, 6, 28, 32
    mehrwertiges, 28

Beziehung, 9, 32
    attributierte, 13
    binäre, 9
    dominierende, 12
    Teil-von-∼, 29
Beziehungsmenge, 9, 11
Beziehungstyp, 9, 11, 29, 32
    1:n, 17
    dominierender, 12

Data Dictionary, 8, 20, 33
Data Dictionary System, 21
Datenbankmodell, 3
Datenlexikon, 8, 20, 30
Datenmodell, 3
    semantisches, 4
Datenmodellierung, 23
Diskriminator, 15, 33
Dokument, 30
    logisches, 30
    physisches, 31

Entität, 5, 33
    dominierende, 12
Entitätsmenge, 5
Entitätstyp, 5
    abstrakter, 19
    dominierender, 12
    schwacher, 12, 13
Entity-Relationship-.., *siehe ER-..*
ER-Ansatz, 4
ER-Diagramm, 3, 6, 30
ER-Modell, 4, 33

Generalisierung, 19, 25, 33
Generalisierungstyp, 19

Identifizierungsschlüssel, 7
Instanz, 5

Kardinalität, 15, 16, 33

Lastenheft, 24

mehrfaches Erben, 19
Modellelement, 30, 31
multiple inheritance, 19

Objekt, 5

Performance, 26

Redundanz, 26
Relation, 10
relationship set, 9
Rolle, 10, 15

Schlüsselattribut, 25
Spezialisierung, 19, 25, 33
Subtyp, *siehe Typhierarchie*

Typhierarchie, 18

Vererbung, *siehe Typhierarchie*
Vorgehensmodell, 23

Werkzeug, 31, 32