Gunter Saake
Kai-Uwe Sattler
Andreas Heuer

Datenbanken

Konzepte und Sprachen

Sechste Auflage

Dieses pdf-Kapitel ist eine kostenlose Ergänzung zum oben genannten Buch, das 2018 bei MITP erschienen ist.

Bitte beachten: Verweise auf Umgebungen B-X beziehen sich auf Teile innerhalb dieses Download-Kapitels. Verweise, die mit Kapitelnummern beginnen, wie 11-31, beziehen sich auf Umgebungen innerhalb des obigen Lehrbuches. Verweise auf Seiten vor 738 beziehen sich ebenfalls auf das zugrundeliegende Lehrbuch.

Literaturhinweise in diesen pdf-Kapiteln beziehen sich auf ein erweitertes Literaturverzeichnis zum Lehrbuch, das auch als kostenlose Ergänzung an derselben Stelle zum Download bereitsteht.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Bei der Herstellung des Werkes haben wir uns zukunftsbewusst für umweltverträgliche und wiederverwertbare Materialen entschieden. Der Inhalt ist auf elementar chlorfreiem Papier gedruckt.

ISBN 978-3-95845-776-8
6. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953/7189-079
Telefax: +49 7953/7189-082

Lektorat: Sabine Janatschek, Ernst-Heinrich Pröfener
Sprachkorrektorat: Jürgen Dubau, Astrid Langen
Covergestaltung: Christian Kalkert, www.kalkert.de
Bildnachweis Cover: iStock.com/Shawn Hempel | fotolia.com/karpenko_ilia
Satz: Gunter Saake, Magdeburg; Kai-Uwe Sattler, Ilmenau; Andreas Heuer, Rostock
Druck: Westermann Druck Zwickau GmbH

B.2 Erweiterte Entwurfsmodelle

Im ersten Teil des Buches haben wir bereits das Entity-Relationship-Modell (kurz: ER-Modell) als Entwurfsmodell für Datenbestände kennengelernt, das sich durch die Beschränkung auf wenige Konzepte auszeichnet. Derart universelle einfache Ansätze sind zwar in der Lage, beliebige Datenbestände tatsächlich zu modellieren, bieten aber für spezielle Situationen oft keine eleganten und adäquaten Beschreibungen.

Das ER-Modell kann um Konzepte zur Beschreibung spezieller Modellierungsmuster erweitert werden. Diese Muster korrespondieren mit semantischen Beziehungen, wie sie in der Wissensrepräsentation und in der Computerlinguistik eingesetzt sind. Hierzu gehören insbesondere die Spezialisierung mit Vererbung und die Aggregation von Objekten. Anhand eines konkreten erweiterten ER-Modells, dem EER-Modell, werden derartige Konzepte detailliert vorgestellt.

Im allgemeinen Softwareentwurf haben sich objektorientierte Modelle weitgehend durchgesetzt. Wir werden den Bezug zu erweiterten ER-Modellen anhand der verbreiteten UML-Notation vorstellen.

B.2.1 Erweiterungen des ER-Modells

Bisher haben wir das klassische ER-Modell nur um spezielle Konzepte angereichert, die in Modellierungen oft benötigt werden, etwa spezielle Einschränkungen für Beziehungstypen wie Kardinalitätsangaben. All diese Erweiterungen lassen aber die vier Kernkonzepte des ER-Modells im Wesentlichen unverändert. Die Spezialisierung mittels einer IST-Beziehung zeigt aber, dass es Modellierungskonzepte gibt, die eine spezielle Bedeutung haben und nur unvollkommen auf die Basiskonzepte des klassischen ER-Modells abgebildet werden können.

Diese Erkenntnis führte zu den sogenannten *semantischen Datenmodellen*, in denen anstelle eines allgemeinen Beziehungskonzepts mehrere „semantiktragende“ Beziehungsarten eingeführt wurden, etwa Spezialisierung/Generalisierung, Aggregation zu komplexen Objekten etc. Diese neuen Konzepte wurden bald auch zusammen mit anderen Erweiterungen in das ER-Modell integriert, und die erhaltenen Datenbankmodelle werden *erweiterte Entity-Relationship-Modelle* (EER-Modelle) genannt. Wir werden zuerst einige verbreitete Erweiterungen skizzieren, danach mit dem sogenannten EER-Modell einen typischen Vertreter ausführlich vorstellen und mit einer Diskussion weiterer Vorschläge, die von Interesse sind, abschließen.

B.2.1.1 Erweiterungen bei Attributen

Im klassischen ER-Modell sind nur Attribute möglich, die Werte von Standarddatentypen wie **`string`** oder **`integer`** annehmen. Eine naheliegende Erweiterung ist die Unterstützung *strukturierter* bzw. zusammengesetzter *Attributwerte*. Verbreitet sind Konstruktoren für die Tupelbildung (entsprechen dem **`record`**-Konstruktor aus Programmiersprachen) für die Mengenbildung sowie für geordnete Listen.

Die von Elmasri und Navathe [EN09] vorgeschlagenen graphischen Notationen werden in Abbildung B.13 vorgestellt. Hier werden Tupelbildung (dort *composite attribute*) und Mengenbildung (dort *multivalued attribute*) unterstützt. Elmasri und Navathe schlagen für die Tupelbildung eine hierarchische Aufspaltung des Attributs in Teilattribute vor. Ein Beispiel ist das Attribut `Adresse` in Abbildung B.13, das in die drei Teilattribute `Straße`, `Nummer` und `Ort` aufgeteilt ist.

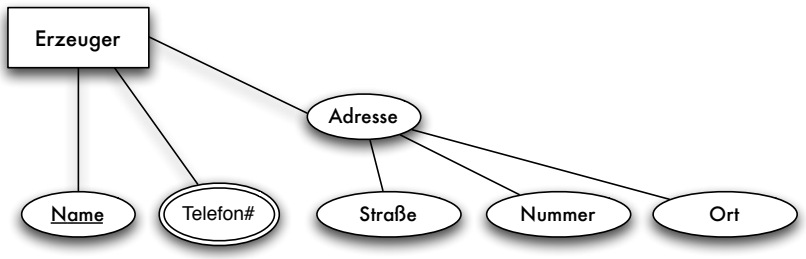

Abbildung B.13: Strukturierte Attributwerte im ER-Modell

Mengenwertige Attribute (auch *mehrwertige* Attribute genannt) werden durch eine doppelte Begrenzungslinie gekennzeichnet. Im Beispiel können Weingüter mehrere Telefonnummern haben (Attribut `Telefon#`). Tupel- und Mengenbildung können geschachtelt eingesetzt werden.

Neben den Standarddatentypen und den üblichen Konstruktoren wird in einigen erweiterten ER-Modellen vorgeschlagen, beliebige abstrakte Datentypen als Wertebereiche für Attribute zuzulassen [EGH+92]. Etwa könnte ein Attribut `Geometrie` eines Entity-Typs `Weinberg` den Wertebereich **`polygon`** haben, dessen Werte Polygonzüge in der Ebene sind.

Eine weitere Art von Attributen, die graphisch speziell notiert wird, sind die *abgeleiteten Attribute* (engl. *derived attributes*). Abgeleitete oder auch *berechnete* Attribute sind Attribute, deren Werte nicht abgespeichert werden müssen, sondern durch eine Anfrage an die Datenbank bestimmt werden können. Bei der Deklaration eines abgeleiteten Attributs muss somit jeweils eine Anfrage mit angegeben werden, die den aktuellen Wert bestimmt. Abgeleitete At-

tribute werden durch gepunktete Begrenzungslinien dargestellt. Ein Beispiel zeigt Abbildung B.14 mit dem Attribut `Jahresgehalt` für Angestellte von Weingütern.

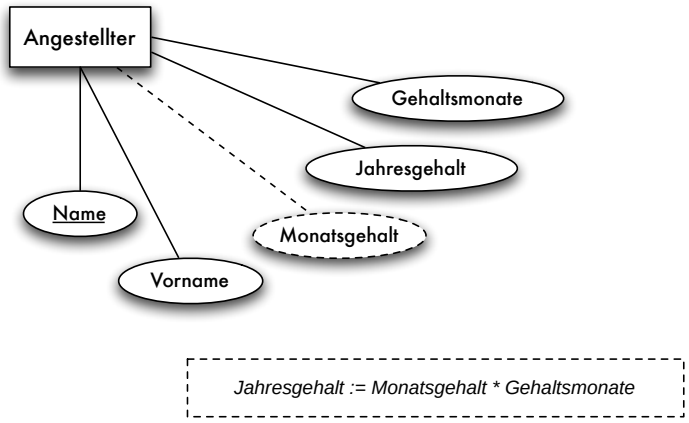

Abbildung B.14: Abgeleitete Attributwerte im ER-Modell

B.2.1.2 Spezialisierung und Generalisierung

Die bisher vorgestellte IST-Beziehung ist eine einfache Form, Spezialisierungs- oder Generalisierungsbeziehungen zwischen Entity-Typen zu modellieren. Aus der Wissensrepräsentation und Vorschlägen semantischer Datenmodelle können verschiedene, feiner differenzierte Beziehungen in ein erweitertes ER-Modell übernommen werden:

- Hinter dem Begriff der *Spezialisierung* verbirgt sich die klassische IST-Beziehung, beispielsweise der Entity-Typ `Schaumwein` als Spezialisierung von `Wein`. Eigenschaften der Spezialisierung sind die Teilmengenbeziehung auf den Ausprägungen und die Vererbung von Eigenschaften.

- Während die Spezialisierung einige Entitys in einem spezielleren Kontext betrachtet, möchte man bei der *Generalisierung* Entitys in einen *allgemeineren* Kontext setzen. Etwa kann ein `Erzeuger` ein `Abonnent` eines Weinführers sein – aber auch ein `Kritiker` kann `Abonnent` sein!

  Bei der Generalisierung verallgemeinern wir in dem Sinne, dass der Generalisierungs-Entity-Typ durch die Vereinigungsmenge der Instanzen mehrerer Entity-Typen gebildet wird. Dieser Entity-Typ erhält Attributdeklarationen, die dem allgemeinen Kontext zuzuordnen sind, etwa die

Lieferadresse. Auf die konzeptionellen Unterschiede zwischen Spezialisierung und Generalisierung werden wir später bei der Diskussion des EER-Modells tiefer eingehen.

Warnung: Die Begriffe Spezialisierung und Generalisierung werden in einigen Datenmodellvorschlägen in der umgekehrten Bedeutung oder auch synonym gebraucht – hier muss man sich jeweils die genauen Definitionen und Beispiele anschauen.

- Die *Partitionierung* bezeichnet den oft auftretenden Sonderfall der Spezialisierung, dass ein Entity-Typ in mehrere *disjunkte* Entity-Typen spezialisiert wird. Beispiel ist eine Partitionierung von Weinen in Schaumweine und Tafelweine, wobei ein konkreter Wein nur eines von beiden sein kann. Wir sprechen von *totaler* oder *vollständiger* Partitionierung, wenn alle Instanzen eindeutig einer Partition zugeteilt werden. Eine vollständige Partitionierung wird auch als *disjunkte Überdeckung* bezeichnet.

Wir verzichten an dieser Stelle auf die Diskussion geeigneter graphischer Notationen und der formalen Semantik, da wir dieses bei der Vorstellung des EER-Modells im folgenden Abschnitt ausführlich tun werden.

B.2.1.3 Komplexe Objekte

Speziell bei der Diskussion der Datenbankunterstützung für Ingenieuranwendungen und anderer Nicht-Standardanwendungsgebiete ist der Begriff des sogenannten *komplexen Objekts* populär geworden [Mit88]. Unter einem komplexen Objekt verstehen wir im Kontext des ER-Modells eine Entity-Klasse, deren Instanzen aus anderen Entitys *zusammengesetzt* sind. Wir unterscheiden mehrere Fälle:

- Unter dem Begriff *Aggregierung* verstehen wir die Situation, dass ein Entity aus einzelnen Instanzen anderer Entity-Typen zusammengesetzt ist. Typisches Beispiel ist ein Fahrzeug, das aus Motor, Karosserie und anderen Teilen besteht. Aggregierung entspricht der Tupelbildung auf Werten.

- Das entsprechende Gegenstück zur Mengenbildung auf Werten wird als *Sammlung* oder auch als *Assoziation* (engl. *association*) bezeichnet. Typisches Beispiel ist die Arbeitsgruppe als Sammlung einzelner Mitarbeiter.

- Sammlung und Aggregierung basieren auf der Zusammenfassung von Entitys zu komplexen Entitys. Einige Vorschläge für komplexe Objekte gehen insofern weiter, als dass auch Beziehungen in komplexe Objekte integriert werden. Ein komplexes Objekt entspricht dann einer Teildatenbank mit einem lokalen Schema bestehend aus lokalen Entity-Typen und lokalen Beziehungstypen.

Zwischen komplexen Entitys und ihren Teilen existieren in der Regel Existenzabhängigkeiten – je nach Anwendungsbereich ist das Ganze von der Existenz der Teile abhängig oder die Teile können nur in Abhängigkeit vom komplexen Objekt existieren. Es besteht somit eine starke konzeptionelle Nähe zu abhängigen Objekttypen, wie sie bereits im klassischen ER-Modell eingeführt wurden.

B.2.1.4 Beziehungen höheren Typs

Auch das Konzept der Beziehung lässt sich erweitern. Im Gegensatz zu den diskutierten neuen Konzepten bei den Entity-Typen sind Erweiterungen bei den Beziehungstypen in erweiterten ER-Modellen nicht verbreitet. Der Grund liegt vermutlich darin, dass sich Beziehungen auch als Entitys darstellen lassen und somit auf höhere Konzepte für Beziehungen zugunsten entsprechender Konzepte für Entity-Typen verzichtet werden kann.

- Die Konzepte der *Spezialisierung* und *Generalisierung* machen auch für Beziehungstypen einen Sinn. Etwa könnte die Beziehung `Abonnement` eines Weinführers zu `Probeabonnement` spezialisiert werden.

  Eine derartige Erweiterung ist sehr naheliegend und semantisch leicht zu fundieren – trotzdem ist sie in vielen populären erweiterten ER-Modellen nicht verwirklicht worden.

- Eine Spezialisierung zwischen Beziehungstypen ist genau genommen eine *Beziehung zwischen Beziehungsinstanzen*, also eine Beziehung zweiter Ordnung. Eine Erweiterung des klassischen ER-Modells um Beziehungen höherer Ordnung wurde etwa von Thalheim unter dem Namen *hierarchische Entity-Relationship-Modell*, kurz HERM, vorgeschlagen [Tha91, Tha00]. Beziehungen höherer Ordnung können Aggregierung und Spezialisierung als Spezialfälle modellieren.

Naturgemäß verwirklichen existierende erweiterte ER-Modelle in der Regel nicht alle hier skizzierten Erweiterungen, sondern beschränken sich auf eine sinnvoll handhabbare Teilmenge hinreichender Ausdrucksfähigkeit. Des Weiteren gibt es verschiedene erweiterte ER-Modelle, die auf bestimmte Anwendungsgebiete, etwa geographische Informationssysteme oder Ingenieuranwendungen, zugeschnitten sind und spezifische Erweiterungen für die Anwendungsgebiete anbieten. Ein Beispiel ist die explizite Modellierung von Versionen und Varianten für Entwurfsdatenbanken im Ingenieurbereich.

Nachdem wir einen Überblick über mögliche Erweiterungen des klassischen ER-Modells gegeben haben, werden wir nun ein spezielles erweitertes ER-Modell, das wir kurz EER-Modell nennen, genauer betrachten.

B.2.2 Das EER-Modell – ein erweitertes ER-Modell

Das vorgestellte erweiterte ER-Modell ist nur einer der Vorschläge für erweiterte ER-Modelle. Es wurde als konkrete Erweiterung ausgewählt, da mit den Büchern von Hohenstein [Hoh93] und Gogolla [Gog94] vollständige Beschreibungen sowohl in deutscher als auch englischer Sprache vorliegen. Das EER-Modell wurde im Rahmen eines Projekts zur Entwicklung einer Datenbankentwurfsumgebung festgelegt [EGH+92]. Weitere erweiterte ER-Modelle werden im Anschluss an die Vorstellung des EER-Modells kurz diskutiert.

B.2.2.1 Übernommene Grundkonzepte aus dem klassischen ER-Modell

Das EER-Modell erweitert das klassische ER-Modell um weitere Konstrukte, behält aber bis auf wenige Ausnahmen die Grundkonzepte des ER-Modells unverändert bei. Im Einzelnen werden folgende Konzepte übernommen:

- *Werte*: Die Standarddatentypen des ER-Modells werden auch im EER-Modell unterstützt.
- *Entitys* bzw. *Entity-Typen*: Werden unverändert übernommen.
- *Beziehungen* bzw. *Beziehungstypen*: Werden unverändert übernommen.
- *Attribute*: Werden unverändert übernommen.
- *Funktionale Beziehungen*: Werden unverändert übernommen.
- *Schlüssel*: Auch im EER-Modell werden nur die Primärschlüssel graphisch angegeben. Schlüssel können über Attribute und funktionale Beziehungen definiert werden. Im EER-Modell werden Schlüssel durch einen schwarz ausgefüllten Kreis an der Verbindungslinie des Teilschlüssels graphisch notiert. Der Grund dafür ist, dass das Schlüsselkonzept im EER-Modell um zusätzliche Möglichkeiten erweitert wurde, die mit der Notation durch Unterstreichung nicht adäquat notiert werden können.

  Ein Beispiel für die Schlüsselnotation zur Modellierung eines *abhängigen Entitys* zeigt Abbildung B.15. Ein `Weinjahrgang` wird hier durch das `Jahr` und den zugehörigen `Wein` identifiziert.

Nicht übernommen werden die folgende Konzepte:

- Die IST-Beziehung wird durch ein allgemeineres Konzept, den *Typkonstruktor*, ersetzt.
- Abhängige Entity-Typen werden im EER-Modell ausschließlich durch das erweiterte Schlüsselkonzept sowie objektwertige Attribute modelliert.

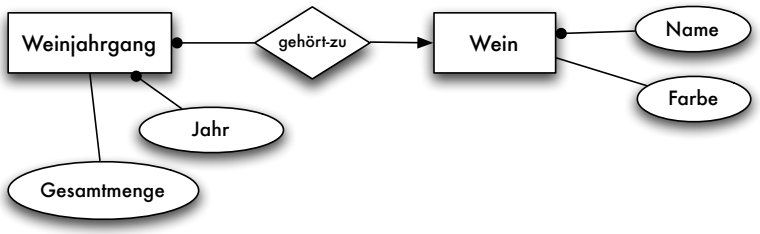

Abbildung B.15: Schlüsselnotation im EER-Modell

B.2.2.2 Erweiterung bei Wertebereichen

Das EER-Modell unterstützt *benutzerdefinierte* Datentypen, auch Nicht-Standarddatentypen genannt. Neue benutzerdefinierte Datentypen sind frei definierbar mittels Datentypkonstruktoren für Mengen-, Tupel- oder Listenkonstruktion. Das EER-Modell unterstützt die folgenden Konstruktoren:

prod: Tupelbildung, z.B. können Punkte in der Ebene durch den folgenden Datentyp beschrieben werden:

$$\texttt{point} = \texttt{prod}(\texttt{real}, \texttt{real})$$

list: Listen oder Folgen von Werten, z.B. können Polygonzüge als Folgen von Punkten beschrieben werden:

$$\texttt{polygon} = \texttt{list}(\texttt{point})$$

set: Mengen von Werten ohne Duplikate, etwa die Menge von Urlaubstagen:

$$\texttt{holidays} = \texttt{set}(\texttt{date})$$

Der Datentyp der Datumswerte kann, falls nicht bereits als Standarddatentyp unterstützt, durch Tupelbildung definiert werden.

bag: Multimengen von Werten, also Mengen, in denen Werte mehrfach vorkommen können. Beispiel ist die Multimenge der Geburtstage einer Gruppe, bei der einzelne Datumswerte mehrfach auftreten können:

$$\texttt{birthdays\_of\_a\_group} = \texttt{bag}(\texttt{date})$$

Jeder Nicht-Standarddatentyp D stellt einen Wertebereich $\mu(D)$ mit Operationen dar, z.B.:

$$\mu(\texttt{point}) = \mu(\texttt{real}) \times \mu(\texttt{real})$$

Für jeden Nicht-Standarddatentyp können spezifische Operationen wie $+$, $-$, **distance** etc. für den Datentyp **point** durch Gleichungen spezifiziert oder in einer imperativen, an Programmiersprachen angelehnten Notation definiert werden. In [EGH+92] werden mehrere Beispiele für die Spezifikation benutzerdefinierter Datentypen im EER-Modell gegeben.

B.2.2.3 Mengenwertige und strukturierte Attribute

Mengen- und tupelwertige (oder strukturierte) Attribute werden im EER-Modell nicht mittels der im vorigen Abschnitt vorgestellten graphischen Notation dargestellt. Stattdessen werden die Konstruktoren **prod**, **set**, **list** und **bag** direkt in die Attributdeklaration geschrieben. Das Beispiel aus Abbildung B.13 wird im EER-Modell somit wie in Abbildung B.16 gezeigt notiert.

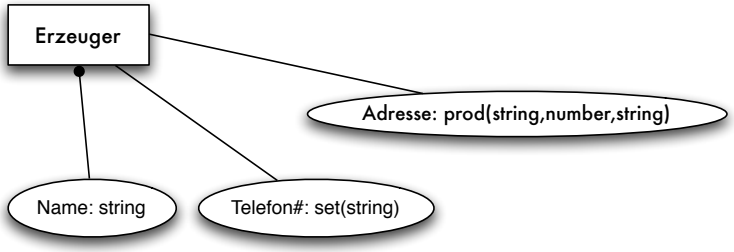

Abbildung B.16: Mengen- und tupelwertige Attribute im EER-Modell

B.2.2.4 Der Typkonstruktor: Spezialisierung, Generalisierung, Partitionierung

Die Konzepte der Spezialisierung, Generalisierung und Partitionierung lassen sich als Spezialfall eines allgemeinen Konstrukts auffassen: eines sogenannten *Typkonstruktors*, der als Dreieck mit Eingabe- und Ausgabe-Entity-Typen notiert wird. Eingabetypen werden mit einer Seite des Dreiecks verbunden, die Ausgabetypen sind mit der dieser Seite gegenüberstehenden Spitze verbunden. In [Hoh93, Gog94] wird der Typkonstruktor mit einem Namen beschriftet (analog zum Namen eines Beziehungstyps). Wir verzichten darauf, da als Namen in der Regel nur `ist` und `sind` auftreten.

Die bereits diskutierte Spezialisierung bzw. IST-Beziehung wird als Spezialfall mit dem Typkonstruktor ausgedrückt. Abbildung B.17 zeigt die Spezia-

lisierung von `Weinen` zu `Schaumweinen` aus Abbildung 4.32 in der Notation im EER-Modell:

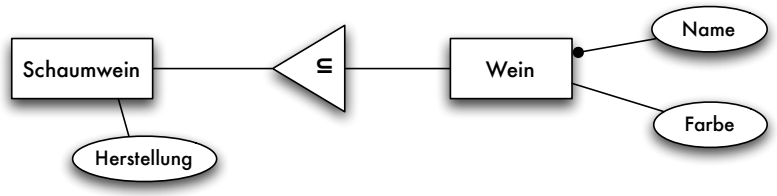

*Abbildung B.17: Spezialisierung (*IST*-Beziehung notiert mit dem Typkonstruktor des EER-Modells*

In das Dreieck wird ein $\supseteq$- bzw. $\subseteq$-Symbol geschrieben, um die Teilmengensemantik der IST-Beziehung zu verdeutlichen: $\sigma(\texttt{Schaumwein}) \subseteq \sigma(\texttt{Wein})$. In diesem Spezialfall ist die Semantik des Typkonstruktors identisch mit der vorgestellten Semantik der IST-Beziehung. Die offene Seite des Symbols zeigt im Diagramm in Richtung der Obermenge, sofern das in der graphischen Darstellung Sinn macht (die Untermenge ist jeweils mit der Spitze des Dreiecks verbunden!). Handelt es sich um eine totale Spezialisierung, wird das $\subseteq$-Symbol durch das Gleichheitszeichen $=$ ersetzt.

Generalisierung

Wird der Typkonstruktor mit mehreren Eingabetypen notiert, so modelliert er eine *Generalisierungsbeziehung*. Die allgemeine Notation für eine Generalisierung wird in Abbildung B.18 gezeigt.

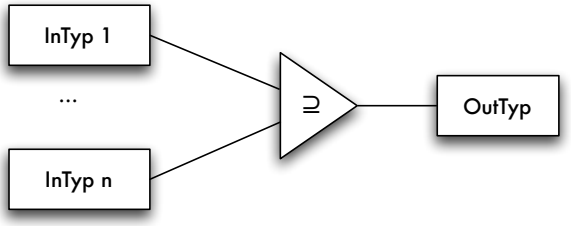

Abbildung B.18: Notation des Typkonstruktors für die Generalisierung

Die Semantik der Spezialisierungsbeziehung – die Ausprägung des spezialisierten Typs ist eine Untermenge der Ausprägung des Eingabetyps – kann direkt auf mehrere Eingabetypen erweitert werden. Der Typkonstruktor für die Generalisierung stellt somit im Zustand σ eine Inklusion dar:

$$\bigcup_i \sigma(\texttt{InTyp}_i) \supseteq \sigma(\texttt{OutTyp})$$

Als Beispiel betrachten wir den Fall, dass `Abonnenten` von Weinführern sowohl `Weingüter` als auch `Kritiker` sein können. Das Beispiel ist in Abbildung B.19 modelliert. Die Generalisierungsklasse hat keine eigene Identifizierung durch Schlüssel. Attribute des Generalisierungstyps modellieren gemeinsame Attribute, die nur im Kontext der Generalisierung auftreten. Im Beispiel ist dies die Angabe des Attributs `Lieferadresse`.

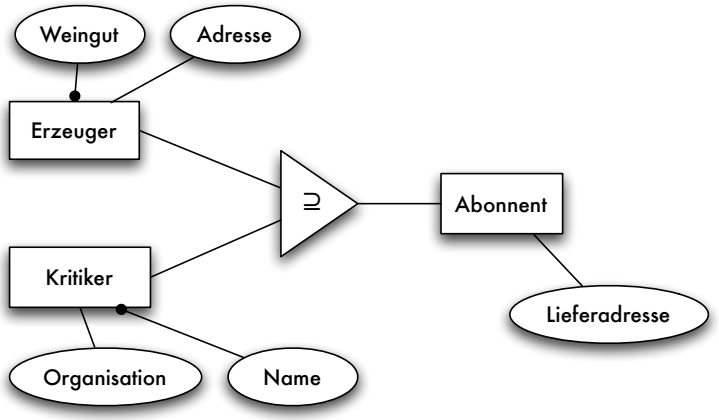

Abbildung B.19: Beispiel für Generalisierung im EER-Modell

In dem Beispiel aus Abbildung B.19 ist die Semantik der Generalisierung durch die folgende Inklusion gegeben:

$$\sigma(\texttt{Kritiker}) \cup \sigma(\texttt{Weingut}) \supseteq \sigma(\texttt{Abonnent})$$

Wie erwähnt kann beim Typkonstruktor statt des $\subseteq$-Symbols auch ein Gleichheitszeichen in das Dreieck gezeichnet werden. In diesem Fall liegt eine totale Generalisierung vor – in unserem Beispiel wäre dann *jeder* Kritiker und *jedes* Weingut ein Abonnent; wir hätten also:

$$\sigma(\texttt{Kritiker}) \cup \sigma(\texttt{Weingut}) = \sigma(\texttt{Abonnent})$$

Als letzte Bemerkung zum Beispiel in Abbildung B.19 weisen wir darauf hin, dass Ausgabetypen keine eigenen Schlüssel haben können – die Identifikation von Entitys der Ausgabetypen wird durch den Typkonstruktor definiert.

Abbildung der Generalisierung auf Beziehungen

Anhand eines Beispiels wollen wir zeigen, dass die Generalisierung ein relevantes Konstrukt ist, dessen Fehlen in einer Modellierungssprache zu Problemen führen kann. Hierfür nutzen wir erneut unser Abonnentenbeispiel. Abbildung B.20 zeigt die Generalisierung mit einer funktionalen Beziehung `abonniert`.

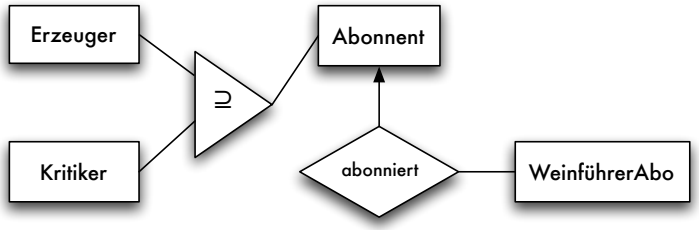

Abbildung B.20: Beziehung an einer Generalisierung

Um die `abonniert`-Beziehung im normalen ER-Modell darzustellen, haben wir mindestens drei Möglichkeiten, die allerdings jeweils mit Nachteilen verbunden sind (siehe Abbildung B.21). Die Schwierigkeit besteht darin, dass sowohl Kritiker als auch Erzeuger einen Weinführer abonnieren können.

- In Version a) werden zwei `abonniert`-Beziehungstypen definiert. Beide sind 1:n-Beziehungen, da ein Abonnement zu einem bestimmten Zeitpunkt einem Abonnenten zugeordent sein kann. Allerdings könnten in dieser Modellierung Weinführerabonnements gleichzeitig von einem Kritiker und einem Erzeuger abonniert sein. Grund dafür ist dass eine funktionale Beziehung nur jeweils lokal geprüft werden kann. Diese Definition erfüllt also nicht die Anforderungen der Anwendung.

- In Version b) wird ein zusätzlicher Entity-Typ `Abonnent` zwischen Personen[1] und Kritikern bzw. Erzeugern geschaltet. Die funktionale Beziehung ist somit gewährleistet. Aufgrund der Inklusionssemantik der IST-Beziehung muss nun aber wirklich jeder Erzeuger und jeder Kritiker als Abonnent auftreten. Eine andere Möglichkeit lässt diese Modellierung nicht zu.

[1] Laut unserer bisherigen Modellierung handelt es sich genauer um *juristische Personen*, da auch Firmen oder Vereine Erzeuger sein können.

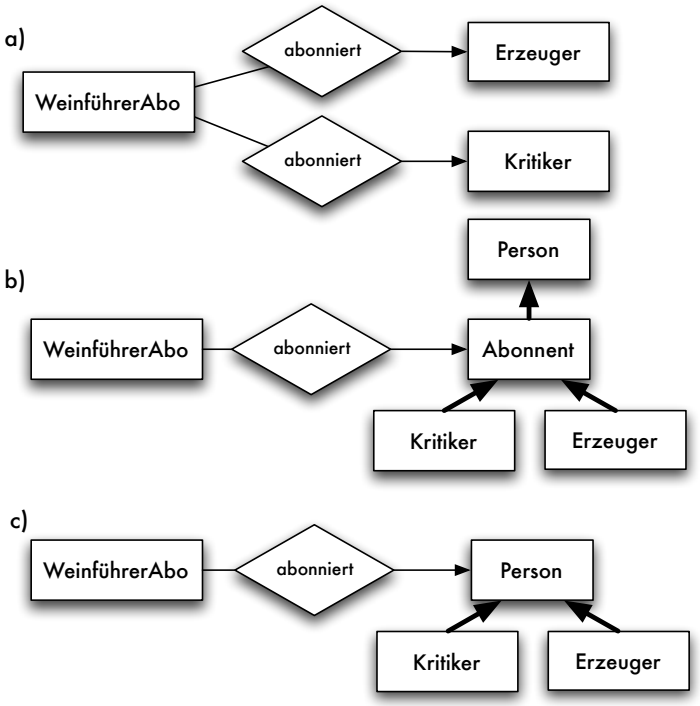

Abbildung B.21: Auswirkungen der fehlenden Generalisierung im klassischen ER-Modell

- In Version c) wird das Abonnieren direkt für `Person` definiert. Leider können hier nun auch Personen abonnieren, die es laut Anwendungsbeschreibung nicht dürfen, etwa Urlauber oder Verlage, die weder Kritiker noch Erzeuger sind.

Partitionierung

Werden beim Typkonstruktor mehrere Ausgabetypen angegeben, erhält man die *Partitionierung*. Die allgemeine Notation der Partitionierung ist in Abbildung B.22 gezeigt.

Die Ausgabetypen einer Partitionierung werden auch als *Partitionen* bezeichnet. Die Semantik einer Partitionierung wird durch zwei Bedingungen festgelegt:

- Die Partitionen (die Ausgabetypen des Typkonstruktors) bilden Spezialisierungen des Eingabetyps:

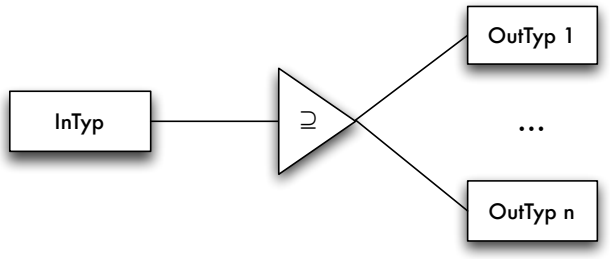

Abbildung B.22: Notation des Typkonstruktors für die Partitionierung

$$\sigma(\texttt{InTyp}) \supseteq \bigcup_i (\sigma(\texttt{OutTyp}_i))$$

- Die einzelnen Partitionen sind disjunkt:

$$\forall i, j: \quad i \neq j \implies \sigma(\texttt{OutTyp}_i) \cap \sigma(\texttt{OutTyp}_j) = \emptyset$$

Die Spezialisierung bzw. IST-Beziehung kann als Spezialfall für $n = 1$ aufgefasst werden, also als eine Partitionierung mit genau einer Partition.

Als Beispiel für eine Partitionierung betrachten wir die Aufteilung von Erzeugern in Privaterzeuger und Unternehmen in Abbildung B.23. Beide Ausgabetypen erben Attribute und Identifikation durch Schlüssel vom Eingabetyp `Erzeuger`. Das $\supseteq$-Symbol bedeutet, dass es auch Erzeuger geben kann, die weder Privaterzeuger noch Unternehmen sind (etwa staatliche Weingüter).

Eine Partitionierung ist nicht identisch mit einer *mehrfachen Spezialisierung*, da zusätzlich die Disjunktheitsbedingung gilt. Als Beispiel betrachten wir die mehrfache Spezialisierung in Abbildung B.24, in der wir `Erzeuger` in die Entity-Typen `Internethändler` und `Unternehmen` spezialisieren – es gibt Erzeuger, die in beiden Spezialisierungen enthalten sind.

Totale Partitionierungen werden auch als disjunkte Überdeckung bezeichnet und mit dem Gleichheitszeichen im Dreieck notiert. Für die Semantikfestlegung gilt die analoge Festlegung wie für totale Generalisierungen. Ein typisches Beispiel ist die totale Partitionierung von `Personen` in die Entity-Typen `Mann` und `Frau`.

Partitionierung versus Generalisierung

In vielen Modellierungssituationen ist es fraglich, ob man eine Partitionierung oder eine Generalisierung zur Beschreibung des Zusammenhangs von Entity-

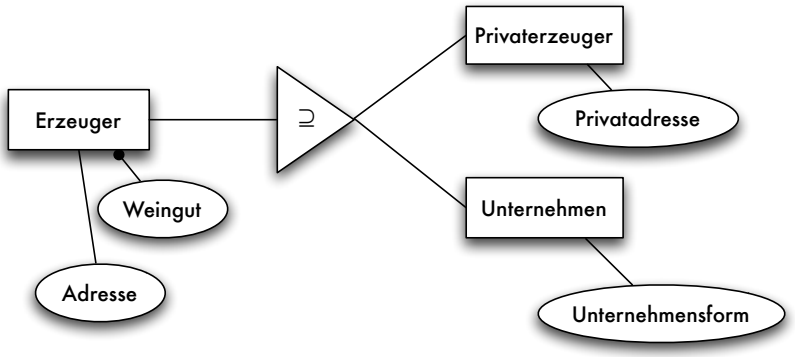

Abbildung B.23: Beispiel für Partitionierung im EER-Modell

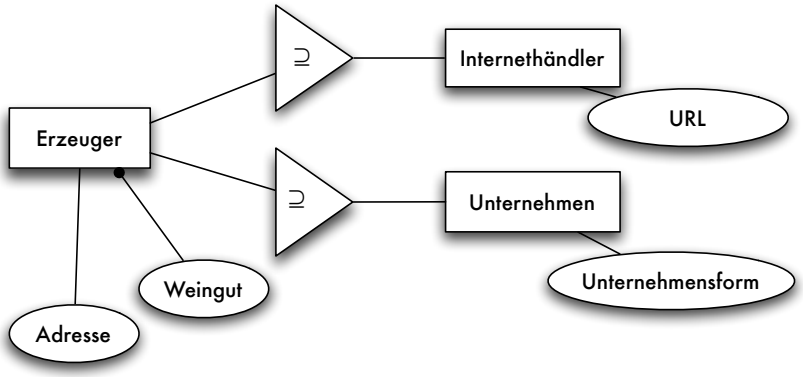

Abbildung B.24: Mehrfache Spezialisierung im EER-Modell

Typen einsetzen sollte. Die konzeptionellen Unterschiede (und damit Kriterien zur Auswahl) werden wir anhand des Beispiels in Abbildung B.25 erläutern.

Dafür greifen wir unser Beispiel der Partitionierung erneut auf. Der wichtigste Unterschied ist die Teilmengenbeziehung zwischen den Instanzmengen:

- Im Fall der *Partitionierung* haben wir die Beziehung:

$$\sigma(\texttt{Erzeuger}) \supseteq \sigma(\texttt{Privaterzeuger}) \cup \sigma(\texttt{Unternehmen})$$

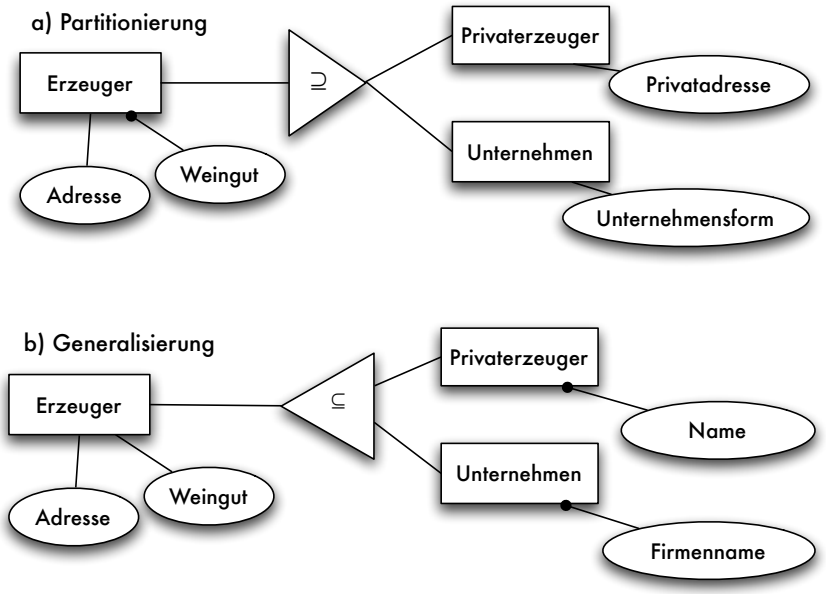

Abbildung B.25: Partitionierung und Generalisierung im Vergleich

Es kann somit Erzeuger geben, die weder Privaterzeuger noch Unternehmen sind – etwa Vereine oder staatliche Organisationen.

- Im Fall der *Generalisierung* haben wir hingegen die Beziehung:

$$\sigma(\texttt{Erzeuger}) \subseteq \sigma(\texttt{Privaterzeuger}) \cup \sigma(\texttt{Unternehmen})$$

Erzeuger sind nun ausschließlich Privaterzeuger und Unternehmen. Aber im Gegensatz zur Partitionierung muss nicht jedes Unternehmen auch ein Erzeuger sein. Hier können in der Datenbank also auch Daten über Unternehmen gespeichert werden, die mit Weinerzeugung nichts zu tun haben.

Die Unterschiede in der Modellierung treten auch bei der Vergabe von Schlüsseln auf: Schlüssel können jeweils nur bei Eingabetypen vergeben werden; für Ausgabetypen dient die Typkonstruktion zur Identifizierung.

Semantik des allgemeinen Typkonstruktors

Die Semantikfestlegungen für Spezialisierung, Generalisierung und Partitionierung weisen große Ähnlichkeiten auf. Es liegt darum nahe, alle diese Kon-

strukte durch einen einheitlichen Typkonstruktor auszudrücken, wie er in Abbildung B.26 dargestellt ist. Das X im Dreieck steht hierbei für eines der beiden Symbole $\supseteq$ (bzw. $\subseteq$ je nach Orientierung des Dreiecks) oder $=$.

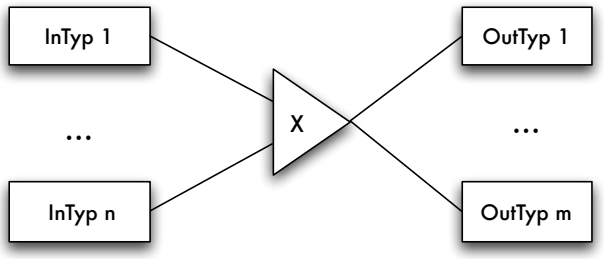

Abbildung B.26: Allgemeiner Typkonstruktor

Die Semantik des allgemeinen Typkonstruktors kann nun durch die folgenden beiden Bedingungen festgelegt werden:

- Die Ausgabetypen des Typkonstruktors bilden Spezialisierungen der Eingabetypen:
$$\bigcup_j(\sigma(\texttt{InTyp}_j)) \supseteq \bigcup_i(\sigma(\texttt{OutTyp}_i))$$
Steht im Dreieck an der Stelle X ein Gleichheitszeichen, so verschärft sich die Bedingung zu:
$$\bigcup_j(\sigma(\texttt{InTyp}_j)) = \bigcup_i(\sigma(\texttt{OutTyp}_i))$$

- Die einzelnen Ausgabetypen sind disjunkt:
$$\forall i,j: \quad i \neq j \implies \sigma(\texttt{OutTyp}_i) \cap \sigma(\texttt{OutTyp}_j) = \emptyset$$

Der allgemeine Typkonstruktor hat allerdings kein Gegenstück in den üblichen Abstraktionskonzepten beim Modellieren von Weltausschnitten im konzeptionellen Datenbankentwurf. Aus diesem Grund werden im EER-Modell nur die Spezialfälle unterstützt, obwohl der allgemeine Typkonstruktor als einheitliche semantische Grundlage benutzt werden kann.

Einschränkungen im Gebrauch des Typkonstruktors

Der Typkonstruktor kann nicht nur auf Basistypen angewendet werden, sondern auch auf Ergebnistypen von Typkonstruktionen. Etwa kann die Spezialisierung `Schaumwein` von `Wein` weiter zu `Flaschengärung` spezialisiert oder Teil

einer Generalisierung zu `Katalogeinträge` eines Internetversandes werden. Es gilt aber die folgende Einschränkung:

> *Der aus den Typkonstruktionen gebildete gerichtete Graph darf keine Zyklen enthalten.*

Die Einschränkung stellt die eindeutige Identifikation von Instanzen in den konstruierten Entity-Typen sicher. In den Originalquellen über das EER-Modell [EGH+92, Hoh93, Gog94] wird eine weitere Einschränkung getroffen:

> *Jeder konstruierte Entity-Typ ist Resultat genau einer Anwendung des Typkonstruktors.*

Diese Einschränkung verbietet sogenannte *Mehrfachspezialisierungen* mit Mehrfachvererbung von Eigenschaften. Andere erweiterte ER-Modelle (etwa das in [EN09] beschriebene) erlauben derartige Mehrfachspezialisierungen. Ein typisches Beispiel in unserem leicht modifizierten Weinszenario wäre die Spezialisierung `ErzeugenderKritiker`, die als Spezialisierung sowohl von `Erzeuger` als auch von `Kritiker` auftritt (Abbildung B.27).

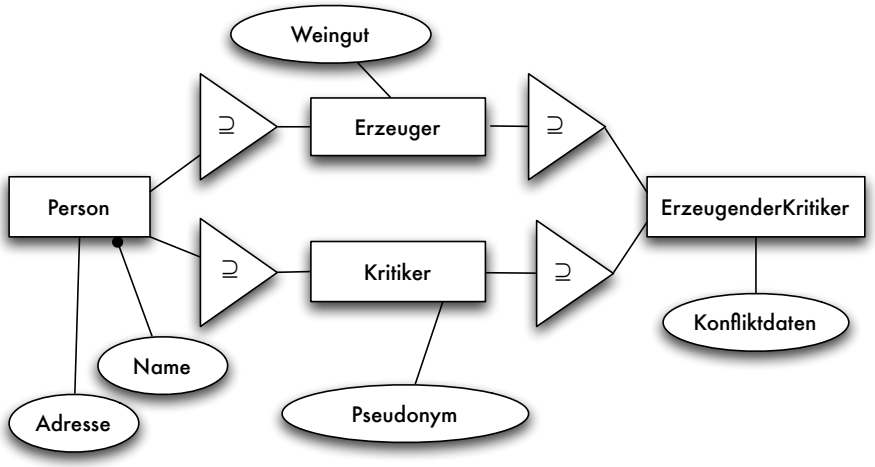

Abbildung B.27: Mehrfachspezialisierung zu `ErzeugenderKritiker`

Die Bedeutung dieser Mehrfachspezialisierung ist durch die folgende Beziehung beschrieben:

$$\sigma(\texttt{ErzeugenderKritiker}) \subseteq \sigma(\texttt{Erzeuger}) \cap \sigma(\texttt{Kritiker})$$

Eine Mehrfachspezialisierung impliziert aufgrund der Bedeutung der Spezialisierung als Teilmengenbildung eine Schnittmengenbildung der beteiligten Eingabetypen. Wir erlauben derartige Mehrfachspezialisierungen, da sie insbesondere beim Entwurf von Datenbanken für objektorientierte Datenbanksysteme ein Modellierungsmittel anbieten, das sich direkt in Beschreibungskonzepte dieser Systeme umsetzen lässt. Wir führen allerdings folgende Einschränkung ein:

> *Mehrfachspezialisierungen sind nur erlaubt, wenn die Eingabetypen direkt oder indirekt aus einer gemeinsamen Ausgangsklasse konstruiert wurden.*

Diese Einschränkung basiert auf der Tatsache, dass die Instanzmengen unabhängiger Basisklassen disjunkt sind. Eine Mehrfachspezialisierung macht aufgrund der Bildung der Schnittmenge nur dann einen Sinn, wenn die Eingabetypen gemeinsame Elemente enthalten.

B.2.2.5 Aggregierung und Sammlung mittels objektwertiger Attribute

Die Bildung komplexer Objekte, also von Objekten, die aus anderen Objekten „zusammengesetzt“ sind, wird im EER-Modell durch *objektwertige Attribute* ermöglicht. Abbildung B.28 zeigt den Einsatz objektwertiger Attribute bei der Modellierung eines Weinkatalogs. Die schwarz ausgefüllten Kreise werden für die Schlüsselfestlegung genutzt, näheres hierzu im folgenden Abschnitt.

Notiert werden objektwertige Attribute analog zu datenwertigen Attributen. Allerdings wird anstelle des Datentyps ein □-Symbol gezeichnet, das durch einen Pfeil mit dem Entity-Typ verbunden ist, dessen Instanzen mögliche Werte dieses Attributs sein können. In der Textnotation wird der Name des Entity-Typs anstelle der Datentypangabe geschrieben. Wie bei datenwertigen Attributen können die Angaben **`set`**, **`list`** und **`bag`** zur Deklaration mehrwertiger Attribute eingesetzt werden.

Ein objektwertiges Attribut A eines Entity-Typs $E(\ldots, A : E_A, \ldots)$, das Werte eines anderen Entity-Typs E_A annehmen kann, entspricht der Funktion:

$$\sigma(A) : \sigma(E) \rightarrow \sigma(E_A)$$

Bei mehrwertigen Attributen wird der Bildbereich jeweils durch Mengen-, Multimengen- bzw. Listenbildung über den Instanzen konstruiert. Objektwertige Attribute können nur aktuelle Instanzen als Werte annehmen. Das Attribut `Autoren` in Abbildung B.28 entspricht somit der folgenden Funktion:

$$\sigma(\texttt{Autoren}) : \sigma(\texttt{Weinkritik}) \rightarrow (\sigma(\texttt{Kritiker}))^*$$

Die Notation $(\sigma(E))^*$ bezeichnet die Menge aller endlichen Folgen aktueller Instanzen von E.

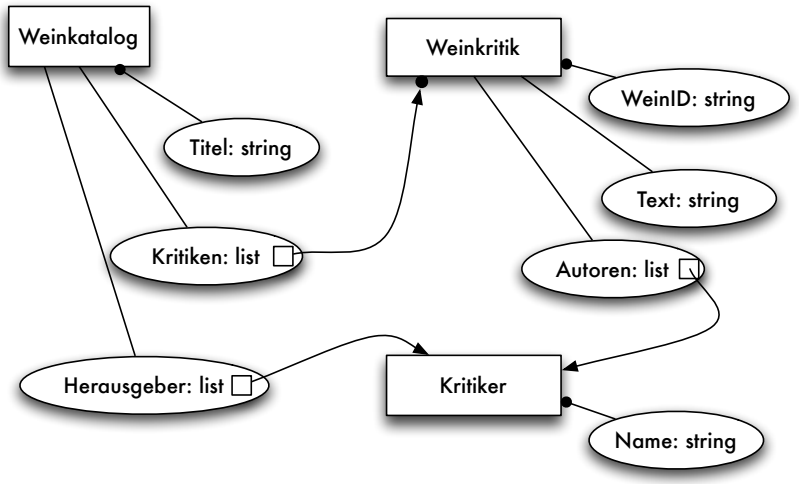

Abbildung B.28: Objektwertige Attribute im EER-Modell

Einfache objektwertige Attribute können äquivalent durch eine Umsetzung in funktionale Beziehungen ausgedrückt werden. Die Angabe **`set`** führt zu einer nicht-funktionalen zweistelligen Beziehung, während die Umsetzung der Angaben **`list`** und **`bag`** zusätzliche Attribute bzw. Entity-Typen erfordern würde.

B.2.2.6 Erweitertes Schlüsselkonzept

Abbildung B.28 zeigt eine weitere Besonderheit objektwertiger Attribute. Da einfache objektwertige Attribute funktionalen Beziehungen entsprechen, können sie auch als Schlüsselattribute auftreten. Bei dem Attribut `Kritiken` zeigt sich eine andere Art der Schlüsselbedingung: Eine `Weinkritik` in einem `Weinkatalog` wird durch den `Weinkatalog` und die `WeinID` identifiziert! Ein objektwertiges Attribut kann Schlüssel sowohl für den Entity-Typ, für den es deklariert ist, als auch für den Entity-Typ, der Bildbereich des Attributs ist, sein. Diese zweifache Möglichkeit erfordert die Abwandlung der graphischen Notation für Schlüssel.

Zusammen mit dem erweiterten Schlüsselkonzept können objektwertige Attribute somit abhängige Entity-Typen adäquat modellieren. Das Beispiel aus Abbildung 4.28 von Seite 98 kann im EER-Modell nun wie in Abbildung B.29 modelliert werden.

Die Modellierung in Abbildung B.30 beschreibt dieselbe Abhängigkeit mittels eines objektwertigen Attributs `Jahre`, das für jeden `Wein` auf die Menge der

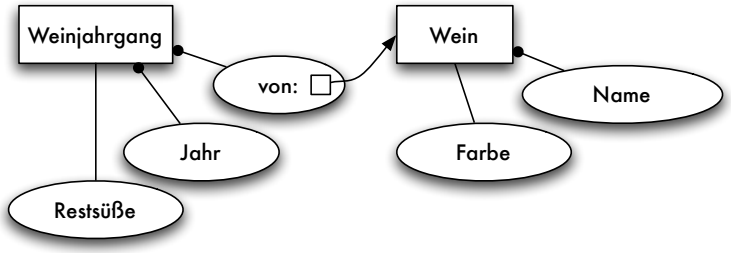

Abbildung B.29: Einsatz objektwertiger Attribute zur Modellierung abhängiger Entity-Typen im EER-Modell

vorhandenen `Weinjahrgänge` verweist. Auch hier dient die Beziehung zwischen Weinen und Weinjahrgängen zusammen mit dem Attribut `Name` zur Identifizierung von Weinjahrgängen.

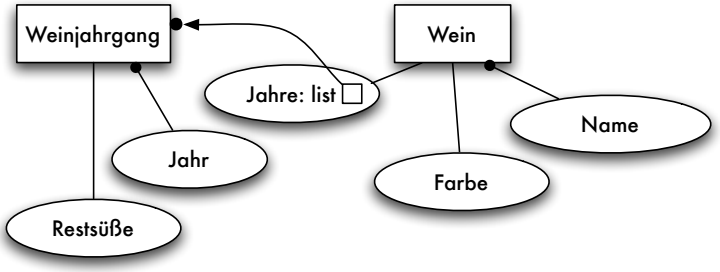

Abbildung B.30: Alternativer Einsatz objektwertiger Attribute zur Modellierung abhängiger Entity-Typen im EER-Modell

In [EGH+92, Hoh93] wird eine abgewandelte graphische Notation für die Modellierung abhängiger Entity-Typen vorgeschlagen, bei der ein fetter Pfeil anstelle eines Schlüssel-Symbols verwendet wird. Abbildung B.31 zeigt diese abgewandelte Notation.

B.2.2.7 Abgeleitete Konzepte

Neben Attributen können im EER-Modell beliebige andere Konzepte abgeleitet werden, etwa Typkonstruktionen, Beziehungen, objektwertige Attribute oder

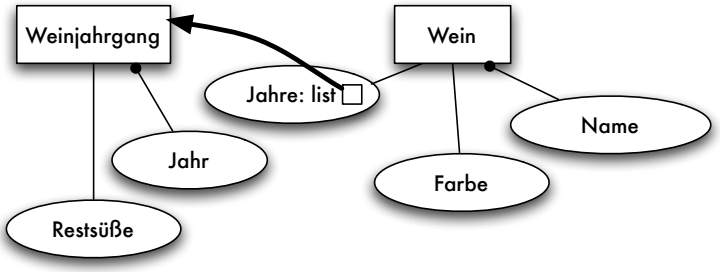

Abbildung B.31: Modellierung abhängiger Entity-Typen im EER-Modell (alternative Notation)

Entity-Typen. Die Ableitung muss jeweils mit einer *Anfrage* spezifiziert werden – wir gehen hier allerdings nicht auf die konkrete Notation von Anfragen im EER-Modell ein und verweisen stattdessen auf [EGH+92, Hoh93, Gog94]. Abgeleitete Konzepte werden wie abgeleitete Attribute mit gepunkteten Umrandungen notiert.

Ein interessanter Spezialfall abgeleiteter Konzepte sind abgeleitete Spezialisierungen und Partitionierungen, etwa die Partitionierung von `Person` in `Mann` und `Frau` anhand der Werte eines Attributs `Geschlecht`.

B.2.2.8 Vergleich zu anderen erweiterten ER-Modellen

Im Lehrbuch von Elmasri und Navathe [EN09] wird ein erweitertes ER-Modell vorgestellt, das dort ebenfalls EER-Modell (nach *Enhanced ER-Model*) genannt wird. Es basiert auf dem von Elmasri et al. in [EWH85] vorgestellten *Entity-Category-Relationship Model*, kurz *ECR-Modell*. Die Modellierungskonzepte sind ähnlich zu dem in diesem Buch vorgestellten EER-Modell. Zur besseren sprachlichen Abgrenzung bezeichnen wir auch das in [EN09] vorgestellte EER-Modell als ECR-Modell.

Das ECR-Modell unterscheidet sich vom EER-Modell hauptsächlich in den Notationen für Spezialisierungen und Generalisierungen. Einen Vergleich der Notationen für EER- und ECR-Modell gibt Abbildung B.32 (die $\subseteq$-Symbole im EER-Modell sind der Vereinfachung halber weggelassen).

Spezialisierungen werden im ECR-Modell durch ein $\subset$-Symbol an einer verbindenden Kante dargestellt, das den Aspekt der Mengeninklusion hervorheben soll. Die offene Seite des $\subset$-Symbols zeigt dabei auf die Superklasse der Spezialisierungshierarchie.

Mehrfache Spezialisierungen und Partitionierungen werden durch Kreise in Zusammenhang mit dem Spezialisierungssymbol graphisch notiert, wobei

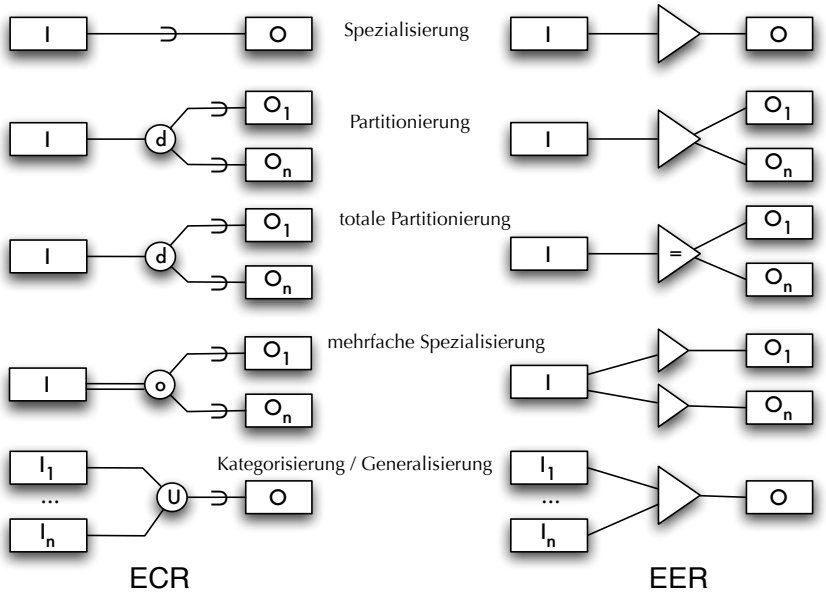

Abbildung B.32: Gegenüberstellung der Notation für Spezialisierung, Partitionierung und Generalisierung im ECR- und EER-Modell

ein **d** für Partitionierungen und ein **o** für Mehrfachspezialisierungen steht (**d** für *disjoint* und **o** für *overlapping*). Der Begriff Generalisierung wird im ECR-Modell synonym für (Mehrfach-)Spezialisierungen benutzt, je nachdem, in welcher Richtung die Entstehung der Beziehung während des Entwurfsprozesses gesehen wird. Totalität bei Partitionierungen wird durch eine doppelte Linie graphisch notiert.

Dem in diesem Buch vorgestellten Konzept der Generalisierung entspricht im ECR-Modell die *Kategorisierung*, bei der die Ergebnisklasse als *Kategorie* bezeichnet wird (engl. *category*). Kategorisierung wird als Kreis mit einem **U** notiert (**U** für *Union*). Totale Kategorisierung wird ebenfalls mit einer doppelten Linie gekennzeichnet.

Eine von den bisher vorgestellten Ansätzen verschiedene Vorgehensweise bietet HERM von Thalheim [Tha91, Tha00]. Die grundlegende Erweiterung von HERM sind Relationships *über Relationships*, also eine hierarchische Schachtelung von Beziehungen. Hiermit kann elegant eine mehrstufige Aggregation abgebildet werden: Aggregierte Objekte entsprechen Beziehungen höherer Ordnung.

Im Sinne einer Minimierung von Modellierungskonzepten schlägt HERM vor, auch Spezialisierung durch Beziehungen (und nicht durch Entity-Typen!) zu modellieren. Dieser Ansatz basiert auf der Erkenntnis, dass eine einstellige Beziehung formal einer Untermengenbildung auf den Instanzenmengen entspricht.

Thalheim diskutiert in [Tha00] weitere Modellierungskonzepte, auf die wir hier nicht näher eingehen wollen, und gibt eine durchgängige formale Semantik für das hierarchische ER-Modell an.

B.2.3 Ein Kalkül für das EER-Modell

Anfragekalküle hatten wir im Buch bisher nur für das Relationenmodell definiert und eingeführt. In einigen Lehrbüchern werden Anfragekalküle auch als fest mit dem Relationenmodell verbunden vorgestellt. Diese Verbindung ist naheliegend, entsprechen doch Tupel einer Relation direkt Fakten zu einem Prädikat in einem Logikansatz. Jedoch können Kalkülanfragen auch auf andere Datenmodelle übertragen werden, deren Datenobjekte nicht direkt Fakten und Prädikaten einer Logik entsprechen.

In diesem Abschnitt betrachten wir als konkreten Ansatz einen Kalkül für das in diesem Kapitel vorgestellte erweiterte ER-Modell.

Der *EER-Kalkül* nach Hohenstein und Gogolla [HG88, GH91, Hoh93, Gog94] basiert auf den vorgestellten Konzepten für relationale Anfragekalküle, bietet aber zusätzlich weitere Konzepte an:

- Anfragen im EER-Kalkül liefern *Multimengen* als Ergebnis, d.h. Anfrageergebnisse können Duplikate enthalten. Multimengenanfragen werden mit speziellen Klammersymbolen notiert:

  $$\{\{\cdots \mid \ldots\}\}$$

  Die Multimengensemantik entspricht eher den realisierten Datenbanksprachen wie SQL und ermöglicht eine einfache Integration von Aggregatfunktionen in den Kalkül.

- Ein Problem mit relationalen Kalkülen ist die Frage der Sicherheit. Im EER-Kalkül wird das Problem der Sicherheit dadurch gelöst, dass Variablen ausschließlich positiv an *endliche Bereiche* gebunden werden. Die Bindung an Bereiche erfolgt in *Deklarationen* δ_i zu Beginn der qualifizierenden Formel, so dass eine EER-Anfrage die folgende Form erhält:

  $$\{\{t_1, \ldots, t_n \mid \delta_1 \wedge \cdots \wedge \delta_k \wedge \phi\}\}$$

  wobei die Terme t_i wie im Bereichskalkül die Felder der Ergebnistupel bestimmen und ϕ eine qualifizierende Formel ist.

Ein Beispiel für eine typische Deklaration in einer Datenbankanfrage wäre die folgende Bindung der Variable w an den (endlichen) Bereich aller in der Datenbank gespeicherten Weine:

$$(w\colon \texttt{WEINE})$$

- Bereiche des EER-Kalküls sind endliche (Multi-)Mengen von Datenobjekten, an die eine Variablenbindung erfolgen kann. Neben Datenbankbereichen können auch Anfragen als Bereiche in Deklarationen verwendet werden – der EER-Kalkül hat einen *rekursiven Anfrageaufbau* wie in Abbildung B.33 gezeigt wird.

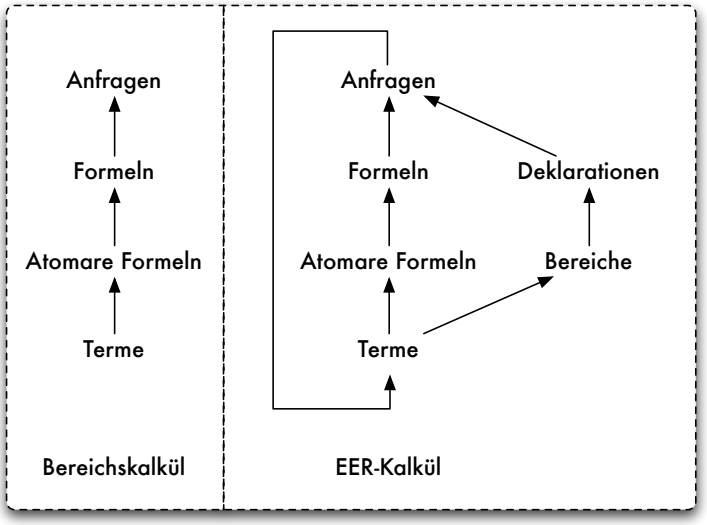

Abbildung B.33: Vergleich des Formelaufbaus zwischen Bereichskalkül und EER-Kalkül

- Da Anfragen als spezielle Terme vom Typ **`multiset`** aufgefasst werden, können *Aggregatfunktionen* wie Summenbildung, Durchschnittswerte und Zählen einfach als Funktionen notiert werden, etwa in der folgenden Anfrage:

$$\texttt{AVG}(\{\{Preis(w) \mid (w\colon \texttt{WEINE} \wedge \texttt{Weingut}(w) = \text{'Creek'}\}\})$$

`AVG` (für engl. *average*) ist eine Funktion, die den Durchschnittswert einer Multimenge von Zahlen bestimmt (vergleiche hierzu Aggregierung in SQL, Abschnitt 11.2).

- Da Anfrageergebnisse nur spezielle Werte sind, die mit dem Datentypkonstruktor **`multiset`** konstruiert werden, können alle Operationen auf Multimengen wie Vereinigung, Schnittmenge etc. auch auf Anfrageterme angewendet werden. Hier verbindet der EER-Kalkül den algebraischen Ansatz zur Anfrageformulierung mit den Kalkülansätzen.
- Eine spezielle Funktion **`bts`** (für *bag-to-set*) entfernt Duplikate aus Multimengen und schlägt damit die Brücke zu den klassischen mengenbasierten Kalkülen.
- So wie in relationalen Kalkülen Relationen als Prädikate behandelt werden, müssen die Konzepte des EER-Modells auf logische Konstrukte des EER-Kalküls abgebildet werden:
  - Die Entity-Typen definieren die Bereiche, an die Variablen gebunden werden können.
  - Beziehungstypen definieren Prädikatsymbole. Alternativ können sie auch Bereiche definieren.
  - Attribute von Entity-Typen werden als Funktionen modelliert.

Der EER-Kalkül kann auch auf Datenbanken angewendet werden, die im ER-Modell oder im relationalen Datenbankmodell beschrieben wurden, da beide als Spezialfälle des EER-Modells aufgefasst werden können. Auch für das Netzwerk- und hierarchische Datenbankmodell könnte er benutzt werden, obwohl diesen Modellen dann eine neue multimengenbasierte Semantik gegeben würde. Die Nähe der Modellierungskonzepte des EER-Modells zu objektorientierten Konzepten lässt auch hier einen Einsatz zu.

◂Beispiel B-3▸ Um einen Eindruck des EER-Kalküls zu vermitteln, geben wir einige Anfragen zum Beispiel-ER-Schema an.

$$\{\{\texttt{Weingut}(e) \mid (e : \texttt{ERZEUGER}) \wedge \exists (a : \texttt{ANBAUGEBIET}) \\ (\texttt{Name}(a) = \text{'Nahe'} \wedge \texttt{SitztIn}(e,a))\}\}$$

Diese erste Anfrage bestimmt die Weingüter im Anbaugebiet Nahe. Da im EER-Kalkül Variablen auch an Beziehungstypen gebunden werden können, kann die Anfrage auch wie folgt formuliert werden:

$$\{\{\texttt{Weingut}(si.\texttt{ERZEUGER}) \mid (si : \texttt{SitztIn}) \wedge \texttt{Name}(si.\texttt{ANBAUGEBIET}) = \text{'Nahe'}\}\}$$

Die Anzahl der Anbaugebiete wird mit einer Funktionsanwendung bestimmt:

$$\texttt{CNT}(\{\{a \mid (a : \texttt{ANBAUGEBIET})\}\})$$

Zuletzt bestimmen wir für jeden Erzeuger den durchschnittlichen Restsüße-Wert:

$$\{\{\texttt{Weingut}(e), \texttt{AVG}(\{\{\texttt{Restsüße}(w) \mid (w : \texttt{WEIN}) \wedge \texttt{produziert}(e, w)\}\} \mid (e : \texttt{ERZEUGER})\}\}$$

□

Die Struktur von Anfragen im EER-Kalkül ist eng an Anfragen in der Sprache SQL angelehnt, so dass er zur Semantikfestlegung von SQL-ähnlichen Sprachen dienen kann. Gogolla gibt in [Gog94] eine formale Semantik für eine Teilmenge des SQL-Standards an, während Hohenstein und Engels [HE92, Hoh93] eine SQL-basierte Sprache SQL/EER für das EER-Modell vorstellen, die auf dem EER-Kalkül basiert.

Nach dem EER-Modell und dem passenden Anfragekalkül werden wir nun die Entity-Relationship-Modelle verlassen und als alternatives semantisches Datenbankmodell das UML-Modell vorstellen, das nicht nur für den objektorientierten Softwareentwurf eingesetzt werden kann, sondern auch als objektorientierte Technik des Datenbankentwurfs.

B.2.4 UML für den Datenbankentwurf

Seit Anfang der 90er Jahre sind Ansätze zum objektorientierten Entwurf und zur objektorientierten Analyse populär. Bekannte Ansätze der ersten Generation können in den Büchern von Booch [Boo91] und Rumbaugh et al. [RBP+91, RBP+94] gefunden werden. Die von Rumbaugh eingeführte Methode *Object Modelling Technique* (OMT) ist gezielt für den Entwurf datenintensiver Anwendungen entwickelt worden, und basiert daher zum Teil auf erweiterten ER-Modellen. Aus diesen Ansätzen entstand (insbesondere durch Hinzufügung der Use Cases von Jacobson [JCJÖ92]) unter der Federführung von Booch, Jacobson und Rumbaugh die *Unified Modeling Language* (UML) [BJR97, FS97, Bur95, Oes97, Rum98, BRJ05, RJB04].

Oft wird der objektorientierte Entwurf als prinzipieller Fortschritt gegenüber dem ER-Modell-gestützten Entwurf angesehen, da der ER-Entwurf ausschließlich auf die (vermeintlich nicht moderne) relationale Technologie abzielen würde. Derartige Meinungen übersehen dabei allerdings, dass die Strukturbeschreibung in Form eines *Klassendiagramms* in UML von den Basiskonzepten her eine Modellierung in einem erweiterten ER-Modell darstellt.

In UML gibt es inzwischen eine Vielzahl von Diagrammen, die verschiedene Aspekte des Softwareentwurfs betreffen. Wir verzichten hier auf eine umfassende Übersicht und beschränken uns auf die Klassendiagramme, da diese für den Datenbankentwurf relevant sind. Klassendiagramme sind im Kern erweiterte ER-Diagramme, in denen die Entity-Typen nun Objektklassen eines

objektorientierten Entwurfs entsprechen. Es gibt eine ganze Reihe von Erweiterungen im Vergleich zum EER-Modell, die aus Programmierungssicht motiviert sind (etwa spezielle Notationen für abstrakte Klassen und Interfaces). Wir beschränken uns auf den Teil der Notation, der direkten Bezug zu gespeicherten Datenobjekten hat.

B.2.4.1 Das Objektmodell von UML

Das Objektmodell von UML kann als Erweiterung eines erweiterten ER-Modells um objektorientierte Konzepte aufgefasst werden. Im Vergleich zu ER-Modellen bestehen folgende Besonderheiten:

- Unterschieden wird zwischen *Klassendiagrammen* und *Objektdiagrammen*. Klassendiagramme entsprechen den bekannten Datenbankschemanotationen, beschreiben also Typen von Instanzkollektionen. Objektdiagramme hingegen beschreiben Einzelobjekte.
- Neben den Strukturaspekten (Attribute, Beziehungen) werden auch Operationen im Klassendiagramm notiert.

  Diesen Aspekt werden wir nur oberflächlich betrachten; hier sei auf die Originalliteratur verwiesen.
- Die textuelle Sprache zur Formulierung von Integritätsbedingungen und Ableitungsregeln ist nicht vorgegeben, hier kann also natürlichsprachlicher Text oder ein geeigneter Logikformalismus genutzt werden.

  Die *Object Constraint Language* (OCL) [WK99] erlaubt formalisierte Angaben von Integritätsbedingungen.
- Üblich sind Darstellungen auf unterschiedlicher Detaillierungsebene. Eine Klasse kann z.B. nur mit ihrem Namen, mit Namen und Attributbezeichnern oder vollständig mit allen Detailangaben wie Datentypen etc. angegeben werden.

Für unsere Zwecke können wir die semantische Modellbildung für ER-Modelle übernehmen, obwohl der funktionale und zeitliche Aspekt der Operationen dort nicht berücksichtigt ist.

B.2.4.2 Darstellung von Klassen in UML

Abbildung B.34 zeigt den prinzipiellen Aufbau einer Klassendarstellung im UML-Klassendiagramm. Attribute und Operationen werden im Gegensatz zu ER-Modellen innerhalb des Klassen-„Kastens“ notiert. Diese beiden Abschnitte können jeweils leer sein. Da aber die Reihenfolge beider Abschnitte relevant ist, muss die Trennungslinie auf jeden Fall angegeben werden, auch wenn der Attributabschnitt leer bleibt.

Abbildung B.34: Darstellung einer Klasse in UML

Klassennamen beginnen als Konvention typischerweise mit einem Großbuchstaben, Attribute hingegen mit einem kleinen. Wir werden dieser Konvention in unsern Beispielen nicht folgen sondern verwenden die Bezeichner aus unseren bisherigen Beispielen unverändert. Für Attributangaben und Operationen gelten die folgenden Konventionen:

- Attribute werden nach folgendem Muster angegeben:

  ```
  name: typ = initialer_Wert { Zusicherung }
  ```

  Die Angabe eines initialen Wertes und einer Zusicherung sind optional. Ein Beispiel wäre die folgende Angabe:

  ```
  Alter: integer = 0 { Alter > 0 and Alter < 125 }
  ```

- Das prinzipielle Muster für Operationen lautet:

  ```
  name( Parameterliste )
  ```

  Auch hier können Datentypen für Parameter und deren initiale Werte angegeben werden, Zusicherungen sind ebenfalls möglich.

Abbildung B.35 zeigt den Entity-Typ `Wein` aus unserer Beispielmodellierung in UML-Notation. Auf eine Angabe von Operationen wurde verzichtet. Für die Restsüße wurde (unrealistischerweise) ein Typ **`number`** angenommen, um den Einsatz von Datentypen zu zeigen. Für den Jahrgang ist 2007 der Default-Wert.

Im UML-Klassendiagramm können noch weitere Angaben stehen, die für den Datenbankentwurf zum Teil relevant sind:

- Verschiedene Arten von Klassen werden unterschieden. Für den Datenbankentwurf sind insbesondere die *abstrakten Klassen* relevant, die mit der Angabe {**`abstract`**} unter dem Klassennamen gekennzeichnet werden. Derartige Klassen haben keine eigenen Instanzen, nur ihre Unterklassen können Instanzen erzeugen.

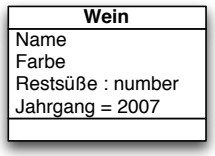

Abbildung B.35: Klasse `Wein` in UML

Weitere Klassenarten sind Metaklassen und parametrisierte Klassen, die für den Datenbankentwurf weniger relevant sind.

- Attribute können weitere Angaben haben. Für den Datenbankentwurf sind hier insbesondere die Definition von **`readonly`**-Attributen sowie die Angaben der Werte **`public`**, **`protected`** bzw. **`private`** für die Sichtbarkeit relevant.
- Abgeleitete Attribute können wie im EER-Modell definiert werden. Sie werden mit einem vorangestellten Schrägstrich (`/`) gekennzeichnet.
- Eine Besonderheit sind *Klassenattribute*, die durch Unterstreichen gekennzeichnet werden. Diese Attribute haben für alle Instanzen der Klasse denselben Wert. Ein Beispiel wäre das Attribut `MaximaleRestsüße` für Weine.

B.2.4.3 Beziehungen in UML

Der Standardfall in UML sind binäre Beziehungen. Beziehungen werden dabei der objektorientierten Terminologie folgend auch als *Assoziationen* bezeichnet.

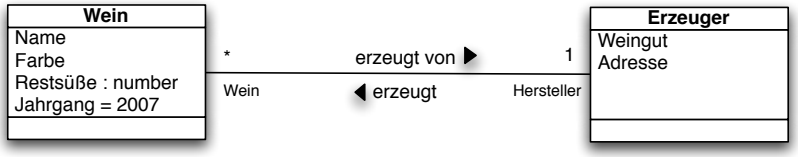

Abbildung B.36: Binäre Beziehung in UML

Abbildung B.36 zeigt ein einfaches Beispiel angelehnt an unsere Beispielmodellierung, anhand dessen wir die Notation erläutern können.

- Beim Beziehungsbezeichner können unterschiedliche Leserichtungen (durch ein ausgefülltes schwarzes Dreieck notiert) angegeben werden. Ein

Grund dafür ist, dass in der objektorientierten Sicht eine binäre Beziehung eigentlich ein Paar zueinander inverser Referenzattribute darstellt.

Warnung: Eine binäre Beziehung mit an der Verbindungslinie angebrachter Pfeilspitze ist in UML keine funktionale Beziehung wie im ER-Modell, sondern eine nur in eine Richtung verfolgbare *Referenz*, wie sie in der objektorientierten Programmierung üblich ist.

- Weiterhin können *Rollennamen* angegeben werden. Diese können in einer späteren Implementierung Namen von Referenzattributen werden.

  Benötigt werden Rollennamen insbesondere bei rekursiven Beziehungen (siehe auch Abbildung 4.4 auf Seite 78).

- Analog zum ER-Modell können Kardinalitäten angegeben werden. Das Beispiel ist wie folgt zu lesen: *„1 Erzeuger erzeugt * Weine."*

  Mögliche Kardinalitätsangaben sind z.B. `1` für genau 1, die Angabe `0,1` für 0 oder 1, `0..3` für 0 bis 3, `0..*` als Standardannahme, `1..*` für nicht optionale Beziehungen oder auch zusammengesetzte Angaben wie `0..2,6,10..*`.

An Beziehungen können auch Zusicherungen geschrieben werden, um etwa referentielle Integrität zu erzwingen.

Beziehungen mit Attributen

Die aus dem ER-Modell bekannten Beziehungen mit eigenen Attributen sind in UML auch möglich. Abbildung B.37 zeigt eine derartige Beziehung. Attributierte Beziehungen werden als degenerierte Klassen ohne eigenen Bezeichner aufgefasst und auch entsprechend notiert.

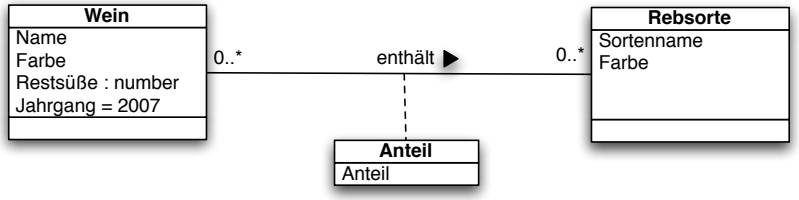

Abbildung B.37: Beziehung mit Attributen in UML

Qualifizierende Beziehungen

Statt der Mengensemantik von beteiligten Objekten kann man in UML eine sogenannte *qualifizierende Assoziation* spezifizieren, um einen Zugriffsschlüs-

sel anzugeben, der wiederum eine spätere Implementierung als Abbildung von Zugriffswerten auf Objektreferenzen ermöglicht.

Abgeleitete Beziehungen

Wie andere UML-Konzepte auch, können Beziehungen abgeleitet werden. Hierzu wird ihrem Bezeichner ein Schrägstrich vorangestellt. Typisches Anwendungsbeispiel ist eine berechnete Referenz als Abkürzung eines mehrstufigen Referenzpfades.

n-stellige Beziehungen

UML erlaubt auch drei- oder mehrstellige Assoziationen. Diese werden wie im ER-Modell als Raute notiert, wobei die Raute aber üblicherweise eine feste Größe hat und der Beziehungsname außerhalb der Raute steht.

Methodisch wird oft vorgegeben, dass derartige Beziehungen durch zusätzliche Klassen ersetzt werden sollen, weil sie dem „reinen“ objektorientierten Ansatz widersprechen.

B.2.4.4 Aggregation in UML

Für die Aggregation wird in UML eine besondere Notation genutzt. Aggregation erfolgt über binäre Assoziationen, die mit einer Raute als Aggregationsbeziehung gekennzeichnet werden. Abbildung B.38 zeigt diese Notation.

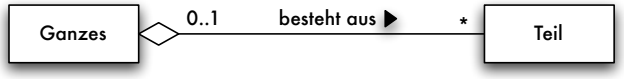

Abbildung B.38: Aggregation in UML

Mehrere aggregierte Teile werden üblicherweise als Baum dargestellt. Dies zeigt Abbildung B.39.

Eine Aggregationsbeziehung kann mittels der Angabe {**`geordnet`**}[2] zu einer listenwertigen Aggregation werden. Analog zu den abhängigen Entity-Typen des ER-Modells können in UML *abhängige Objekte* als Spezialfall der Aggregation modelliert werden. Diese Art der Aggregation wird als *Komposition* bezeichnet. Graphisch wird hier die Aggregationsraute schwarz ausgefüllt (siehe Abbildung B.40).

[2] Hier wurde eine deutsche Annotation nach [Oes97] verwendet.

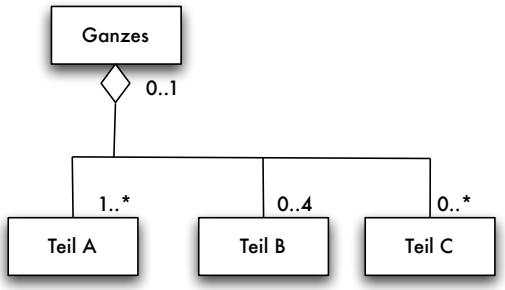

Abbildung B.39: Aggregation in UML in Baumdarstellung

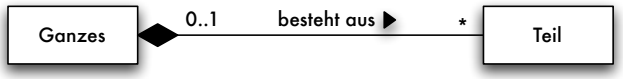

Abbildung B.40: Komposition in UML

B.2.4.5 Spezialisierung in UML

UML erlaubt die bekannte Spezialisierung mit Vererbung. Abbildung B.41 zeigt die verwendete graphische Notation. Die die Spezialisierung ausdrückenden Linien zeichnen sich durch große, nicht ausgefüllte Pfeilspitzen in Richtung der Oberklasse aus.

Eine besondere Rolle spielen Spezialisierungen mit einem *Diskriminator*. Hier teilt ein Aufzählungsattribut, der Diskriminator, die Instanzen auf die Unterklassen auf. Das Aufzählungsattribut wird nicht explizit notiert; sein Wertebereich wird durch die beteiligten Klassennamen festgelegt. Spezialisierungen mit Diskriminator werden graphisch speziell notiert.

Spezialisierungen in mehrere Unterklassen können mit den Zusicherungen **`overlapping`**, **`disjoint`**, **`complete`** und **`incomplete`** versehen werden, welche die im Zusammenhang mit dem EER-Modell ausführlich diskutierten Varianten (etwa Spezialfälle der Partitionierung) ausdrücken können.

B.2.5 Zusammenfassung

In diesem Kapitel haben wir Entwurfsmodelle vorgestellt, die über die Basiskonzepte des ER-Modells hinausgehende Modellierungsmöglichkeiten anbieten. Neben komplexen Attributen und Objekten betrifft dies insbesondere spe-

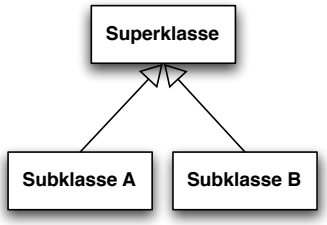

Abbildung B.41: Spezialisierung in UML

zielle Beziehungen wie Generalisierung, Spezialisierung und Partitionierung. Weiterhin haben wir die für den Datenbankentwurf relevanten Teile von UML behandelt, die ebenfalls als eine Erweiterung des ER-Modells angesehen werden können.

Eine Übersicht über die in diesem Kapitel eingeführten Begriffe und deren Bedeutung gibt Tabelle B.4.

Begriff	**Informale Bedeutung**
EER-Modell	erweitertes ER-Modell
strukturierte Attribute	explizite Modellierung von nicht atomaren Attributwertebereichen
komplexe Objekte	Modellierung zusammengesetzter Informationseinheiten
Generalisierung	logische Zusammenfassung von Objekten unterschiedlicher Charakteristik
Partitionierung	Spezialisierung in mehrere disjunkte Klassen
Typkonstruktor	vereinheitlichtes Konzept zur Beschreibung von Spezialisierung, Generalisierung und Partitionierung
Aggregation	Modellierung komplexer Objekte
UML-Notation	Sammlung von Notationen für den Softwareentwurf
Klassendiagramm	UML-Pendant zum EER-Schema

Tabelle B.4: Wichtige Begriffe bei erweiterten Entwurfsmodellen

B.2.6 Vertiefende Literatur

Das Entity-Relationship-Modell wurde in einem grundlegenden Artikel von P. P. Chen im Jahre 1976 [Che76] beschrieben. Seitdem wurde es sehr oft in Lehrbüchern und Übersichtsartikeln behandelt. Zu empfehlen ist insbesondere die Einführung im Buch von Elmasri und Navathe [EN09]. Dort wird auch auf eine Reihe von Erweiterungen und Varianten des ER-Modells verwiesen und weiterführende Literatur angegeben.

Das beschriebene EER-Modell geht auf die Arbeiten [HNS86, HNSE87, EGH+92] zurück. Die Arbeit [EGH+92] beschreibt die hier vorgestellte Notation. Ausführliche Beschreibungen enthalten die Bücher von Hohenstein [Hoh93] und Gogolla [Gog94].

Das ebenfalls kurz behandelte hierarchische Entity-Relationship-Modell (HERM) wird von Thalheim in [Tha91, Tha00] beschrieben. Eine um andere Strukturierungskonzepte erweiterte ER-Version ist das Structured Entity Relationship Model, kurz SERM von Sinz [Sin90]. Kommerzielle Werkzeuge zur ER-Modellierung sind etwa ERwin, Power-Builder und die Oracle-CASE-Produkte (Oracle Designer) [HM99].

Bekannte Ansätze der objektorientierten Modellierung können in den Büchern von Booch [Boo91] und Rumbaugh et al. [RBP+91, RBP+94] gefunden werden. Aus diesen Ansätzen entstand (unter Hinzunahme der Use Cases von Jacobson [JCJÖ92]) unter der Federführung von Booch, Jacobson und Rumbaugh die *Unified Modeling Language* (UML) [BJR97, FS97, Bur95, Oes97, Rum98, BRJ05, RJB04]. [HKKR05] ist eine gute Einführung in die verschiedenen Diagrammarten von UML. [WK99] präsentiert die Object Constraint Language (OCL) des UML-Ansatzes.

Schichten des konzeptionellen Entwurfs. Die Aufteilung des konzeptionellen Schemas in Schichten wird in [Saa91a, EGH+92] erläutert. Vergleichbare Ansätze sind in [SFNC84, CS88] beschrieben. In [EGH+92] wird basierend auf dem erweiterten ER-Modell ein Sprachvorschlag für alle vier Komponenten vorgestellt. In [Saa91b] steht der dynamische Anteil des konzeptionellen Entwurfs im Vordergrund. Der Bezug zu objektorientierten Entwurfsansätzen wird in [Saa93] diskutiert. Die erwähnte Sprache TROLL wird unter anderem in [Saa93, Jun93, JSHS96] vorgestellt; eine kurze Einführung bietet [LL95, Kapitel 16].

Die Erweiterung um eine fünfte Komponente zur Prozessbeschreibung wird ebenfalls in [Saa91a, Saa93] diskutiert. Das als Beispielformalismus herangezogene ConTract-Modell von Reuter und Wächter wird u.a. in [WR92] beschrieben. Der Artikel [WR92] ist in einem Sammelband erschienen, der mehrere erweiterte Transaktionsmodelle vorstellt, die zur Prozessmodellierung geeignet sind [Elm92].

EER-Kalkül. Der EER-Kalkül stammt von Hohenstein und Gogolla [HG88, GH91, Hoh93, Gog94].

B.2.7 Übungsaufgaben

Übung B-1 Geben Sie alternative Formulierungen der Modellierungsaufgaben aus dem ER-Kapitel im EER-Modell an. Welche Änderungen haben sich ergeben, welche neuen semantischen Aspekte konnten modelliert werden? □

Übung B-2 Geben Sie ein weiteres Beispiel für eine Spezialisierung oder Generalisierung einer *Relationship* an (dieses Konzept ist in den vorgestellten Modellen – bis auf HERM – nicht vorgesehen!). Wie kann dieses Konzept im ER- bzw. EER-Modell simuliert werden? □

Übung B-3 Vergleichen Sie die Begriffe Spezialisierung, Generalisierung, Klassen- und Typhierarchie aus dem EER- und den objektorientierten Modellen. Wie unterscheiden sich die einzelnen Semantiken? Bei welchen Anwendungsbeispielen wirken sich diese Unterschiede aus? □

Übung B-4 Erläutern Sie die grundlegenden Konzepte des objektorientierten Ansatzes! Worin besteht der Unterschied bzw. der Zusammenhang zwischen einem Objekt und einer Klasse? Welche Möglichkeiten zur Abbildung von Beziehungen zwischen Klassen/Objekten gibt es? □

Übung B-5 Geben Sie für das Weinbeispiel jeweils eine Anwendung für die Konzepte der verschiedenen Schichten des konzeptionellen Entwurfs an. □

Übung B-6 Gegeben ist der folgende reale Weltausschnitt:

- Person, Fahrzeughalter, Fußgänger,
- Fahrzeug, LKW, PKW, Schiff,
- Motor, Rad, Karosserie.

Bezüglich dieses Weltausschnittes lassen sich folgende Einschränkungen bzw. Bedingungen angeben:

- jedes Fahrzeug hat *genau einen* Fahrzeughalter,
- jeder PKW besitzt *genau* 4 Räder und *genau ein* Reserverad,
- ein LKW besitzt *mehr als* 4 Räder und
- LKWs und PKWs haben *genau einen* Motor und Schiffe *genau 1, 2 oder 4* Motoren.

Gesucht sind für diesen Weltausschnitt das UML-Klassendiagramm, *zu einer* Klasse mögliche Attribute und *zu einer* Klasse mögliche Operationen. Wie würden Sie eine Klasse „Taxi“ in das Klassendiagramm mit aufnehmen? □

Übung B-7 Formulieren Sie die Beispiel-Anfragen aus der Aufgabe 10-4 auch im EER-Kalkül. Behandeln Sie dafür Relationen wie Entity-Typen.