IT-Systeme für Verkehrsunternehmen

Informationstechnik im öffentlichen Personenverkehr

von
Gero Scholz

1. Auflage

Thematische Gliederung:

Wirtschaftsinformatik

dpunkt.verlag 2011

Verlag C.H. Beck im Internet:
www.beck.de
ISBN 978 3 89864 770 0

14 Methodik der Modellierung

Ausführungen zur Methodik geraten oft recht trocken – ganz besonders, wenn sie Musterbeispiele heranziehen, die mit dem eigentlichen Thema nichts zu tun haben. Wir haben uns daher entschlossen, Beispiele aus unserem Branchenmodell aufzugreifen. Auch wenn Sie mit UML bereits vertraut sind, empfehlen wir, das Kapitel zumindest zu überfliegen, da es etliche individuelle Festlegungen enthält.

14.1 Bildung von Teilmodellen

Das Modell umfasst ca. 230 Klassen, welche zu rund 20 Paketen zusammengefasst sind. Auf der obersten Ebene unterteilen wir das Modell in neun Bereiche.

Darstellungen des Gesamtmodells

In der Darstellung unterscheiden wir zwischen dem Gesamtmodell und den Teilmodellen. Zum Gesamtmodell gehören folgende Darstellungen:

- *Paketübersicht*:
  alle Pakete, jeweils mit Name und Liste der Klassen; optische Anordnung nach Modellbereichen
- *Kernmodell*:
  die wichtigsten Klassen (ca. 25) und ihre Beziehungen (ca. 40)
- *Hauptmodell*:
  ca. 70 Klassen (inkl. Attribute, Funktionen, Legende) und deren Beziehungen; die restlichen Klassen sind pauschal als Pakete aufgelistet; das Kernmodell ist im Hauptmodell enthalten
- *Kompaktmodell*:
  gleicher Umfang wie das Hauptmodell, jedoch wird von jeder Klasse nur der Name angezeigt

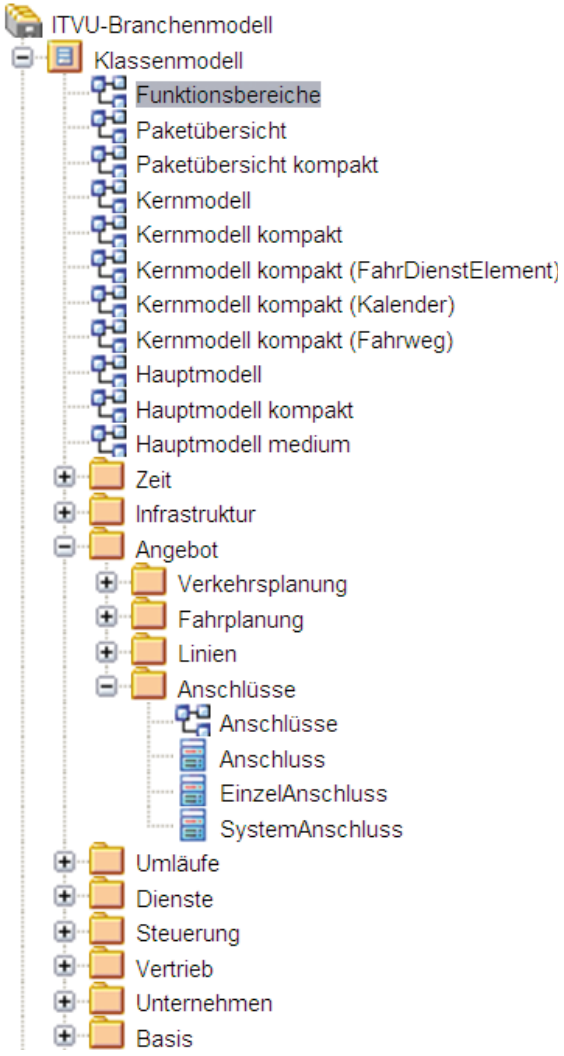
Darstellung der Teilmodelle

Teilmodelle beschreiben jeweils die Klassen eines Pakets. Zusätzlich enthalten sie diejenigen Klassen anderer Pakete, zu denen irgendeine Klasse des betreffenden Pakets eine direkte Beziehung hat. Die Darstellungsinhalte der Teilmodelle überschneiden sich daher. Einige Teilmodelle halten wir für weniger wichtig als andere – sie sind in einer eigenen Gruppe als »Zusatzmodelle« zusammengefasst. Insgesamt gibt es genauso viele Teilmodelle wie Pakete, also ca. 20.

Baumstruktur

Es ergibt sich die in der Abbildung dargestellte Baumstruktur[1]. Sie findet sich so auch auf der ITVU-Website. Wir haben beispielhaft das Paket *Anschlüsse* innerhalb des Funktionsbereichs *Angebot* aufgeklappt dargestellt.

Abb. 14-1
Baumstruktur des Modells



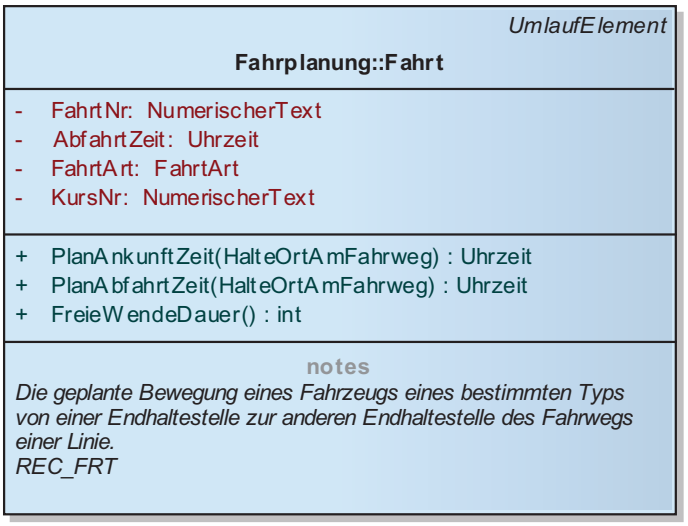
1. Die unterschiedlichen Piktogramme symbolisieren *Modelle*, *Funktionsbereiche/Pakete*, *Klassen* und *grafische Darstellungen*.

14.2 Darstellung von Klassen

Betrachten wir die Klasse `Fahrt`. Wir finden sie in mehreren Darstellungen:

Darstellung von Klassen

Abb. 14-2
Grafische Darstellung einer Klasse



- In der *Paketübersicht*[2] finden wir sie innerhalb des Modellbereichs *Angebot*. Sie tritt dort als zweiter Eintrag im Paket `Fahrplanung` auf.
- Im *Kernmodell*[3] erfahren wir mehr: Die Fahrt ist als ein Rechteck dargestellt, das neben dem Namen der Klasse auch einen erläuternden Text enthält. Am Ende des Textes steht ein einzelnes Wort in Großbuchstaben (REC_FRT). Es verweist auf eine Datensatzart des VDV-Modells[4]. Der Paketname steht vor dem Klassennamen (`Fahrplanung::Fahrt`). Ferner sehen wir im Kernmodell die Beziehungen der `Fahrt`. Beispielsweise verweist eine `Linie` auf den in der Planung vorgesehenen Typ des Transportmittels (`TM-TypPlanung`), und ein Pfeil zeigt zur Elternklasse[5] der Fahrt, dem `UmlaufElement`.
- Im *Hauptmodell* finden wir zusätzlich wichtige Attribute und Funktionen der Fahrt, jeweils mit ihrem Ergebnistyp. Als besonders wichtige Klasse ist die Fahrt optisch durch einen roten Rand hervorgehoben.
- Im *Teilmodell* `Fahrplan` ist die Fahrt auf die gleiche Weise dargestellt. Im erklärenden Text findet man hier einen Hyperlink (»more...«), der auf die entsprechende Webseite im Wiki verweist.

2. Die Paketübersicht befindet sich auf der vorderen Innenseite des Buchumschlags.
3. Das Kernmodell befindet sich auf der hinteren Innenseite des Buchumschlags.
4. Hier: VDV-Schrift 452, Abschnitt 9.8.1; nicht immer gibt es jedoch eine solche eindeutige Entsprechung.
5. Zur sog. Vererbung siehe Abschnitt 14.14.

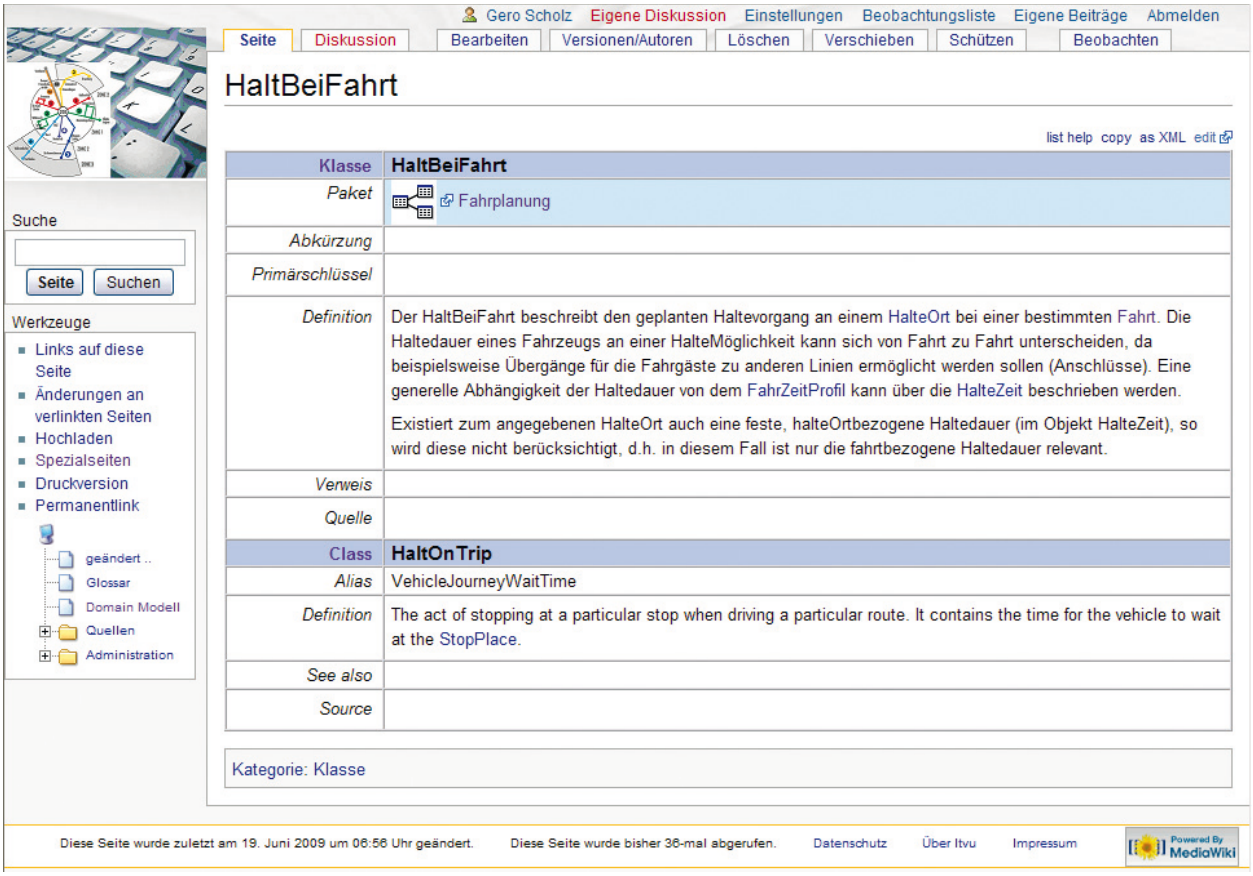


Abb. 14-3 Wiki-Seite zu einer Klasse (`HaltBeiFahrt`)

- Auf der *Wiki-Seite* zu einer Klasse (Abb. 14-3) findet man ausführliche Texte zur Definition, ergänzende Anmerkungen, Quellenverweise sowie einen Abschnitt mit der Darstellung in englischer Sprache. Klickt man in der *Online-Version* des Teilmodells auf das Piktogramm einer Klasse, so öffnet sich ein Informationsfenster mit mehreren Reitern. Hier kann man sich über Attribute, Funktionen, Beziehungen und Invarianten (»Constraints«) näher informieren (Abb. 14-4).

Darstellung von Beziehungen

Beziehungen sind in der Regel gemäß UML als Linien dargestellt. Manchmal würde eine Linie quer durch das ganze Bild verlaufen. In solchen (Einzel-)Fällen verwenden wir im Hauptmodell eine Konvention: Anstelle einer Linie tragen wir ein Attribut ein, das mit dem Präfix *Zugeordnete*(*r*) beginnt. Beispielsweise legen viele Verkehrsunternehmen für ihre `Fahrzeuge` jeweils einen bestimmten Heimatort (Depot, `Betriebshof`, Garage) fest. Das Modell hat daher eine Beziehung vom `Fahrzeug` zum `Betriebshof`. Diese Beziehung ist in den Teilmodellen als Linie dargestellt, im Hauptmodell jedoch als Attribut (*ZugeordneterBetriebshof*) beim `Fahrzeug`, da eine entsprechende Linie quer durch das Bild verlaufen würde.

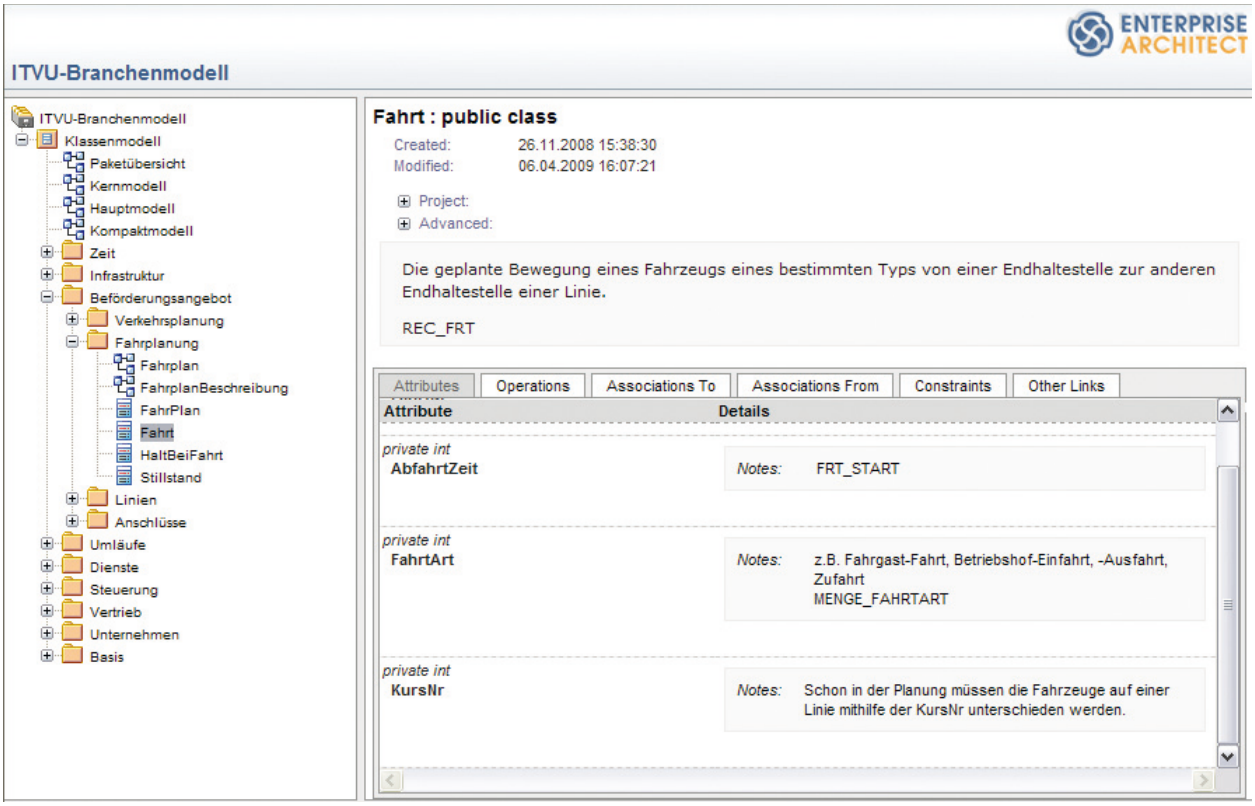


Abb. 14-4 Details zu einer Klasse im Webdialog

Farbschema

Wir ordnen jedem Modellbereich eine Grundfarbe zu und verwenden für die zugehörigen Pakete diese Farben mit unterschiedlichem Sättigungsgrad. Jede Klasse trägt die Farbe ihres Pakets. So kann man insbesondere in den Teilmodellen schnell erkennen, welche Klassen im Mittelpunkt stehen.

Layout

Das Layout unserer Diagramme richtet sich nicht nach der bestmöglichen grafischen Entflechtung von Linienkreuzungen – obwohl dieser Punkt natürlich auch eine Rolle spielt. Wir legen der Paketübersicht ein semantisches Ordnungsprinzip zugrunde und verwenden dieses auch in allen anderen Grafiken. So steigt der Wiedererkennungseffekt: Die Infrastruktur ist z. B. immer links, die Abwicklung von Fahrten und Diensten befindet sich immer auf der rechten Seite (*vgl. Kap. 3*).

14.3 Allgemeine Namensregeln

Es gibt kaum etwas Wichtigeres als die präzise und konsistente Benennung von Modellelementen. Es mag kleinlich erscheinen – aber es hilft enorm, wenn man hier detaillierte Regeln aufstellt.

Worte ohne Leerzeichen
- Um Pakete, Klassen, Attribute oder Funktionen zu benennen, verwenden wir immer ein einziges Wort, d. h., diese Begriffe enthalten keine Leerzeichen.

Groß-/Kleinschreibung
- Pakete, Klassen, Attribute und wertliefernde Funktionen beginnen mit Großbuchstaben. Funktionen, die keinen Wert liefern, beginnen mit Kleinbuchstaben.

CamelCase
- In Anlehnung an Konventionen der objektorientierten Softwareentwicklung setzen wir bei der Benennung von Klassen, Attributen und Funktionen gemischte Groß- und Kleinschreibung ein. Wir schreiben daher `DienstElement` oder `DienstBestandteil`. Eine Ausnahme bilden zusammengesetzte Worte, die sprachlich sehr eng verbunden sind. Wir schreiben daher nicht `Dienst-BestandTeil` oder gar `FahrZeug`.

Umlaute
- Anders als die meisten Programmiersprachen lassen wir der besseren Lesbarkeit wegen Umlaute zu. Daher schreiben wir `StörungsManagement` und nicht `StoerungsManagement`.

Wortverbindungen
- Aus dem gleichen Grund lassen wir das verbindende »s« zu. Wir schreiben also `StörungsManagement` und nicht `StörungManagement`.

Abkürzungen
- Abkürzungen verwenden wir nur ungern. Wir beschränken uns auf wenige, allgemein verbreitete Abkürzungen, um überlange Namen bei Klassen zu vermeiden; Abkürzungen bestehen aus Großbuchstaben und werden mit einem Bindestrich[6] abgetrennt. Also z. B. `DFI-Anzeige` anstelle von `Dynamische-FahrgastInformationsAnzeige`.

14.4 Pakete, Klassen, Attribute, Funktionen

Benennung von Paketen
- Paketnamen sind abstrakte Begriffe, sie stehen oft im Plural (`Linien`, `Basis-Typen`) oder enden auf »ung« (`Planung`, `Ausführung`).
- Sie beschreiben eine Gruppe von Dingen (`Transportmittel`) oder einen Prozess (`Planung`).
- Es soll keine Klasse geben, die exakt genauso heißt wie ein Paket.

Benennung von Klassen
- Wir verwenden ein Substantiv im Singular.
- Manchmal tritt ein erläuterndes Adjektiv oder eine Präposition hinzu: `KonfigurierbaresMerkmal`, `OrtAufVerbundEbene`, `HaltBeiFahrt`.
- Wir berücksichtigen, dass in vielen Listen alphabetisch sortiert wird. Zwei Bezeichnungen der Art `AdressRaumFachlich` und `AdressRaumTechnisch` können daher zweckmäßiger sein als die an sich etwas schöneren Bezeichnungen `FachlicherAdressRaum` und `TechnischerAdressRaum`.

6. Unterstriche (»_«) verwenden wir nicht.

- Klassennamen sind modellweit eindeutig.[7]
- Bei der Benennung von Attributen verwenden wir Substantive und Adjektive. Attributnamen aus reinen Präpositionen (»von« oder »bis«) sind unzweckmäßig. Reine Adjektivkonstrukte (»*GültigAb*«) verwenden wir dann, wenn sie deutlich kürzer als ein Substantiv sind (»*GültigkeitsBeginnZeitPunkt*«).

Benennung von Attributen

- Im Attributnamen deuten wir den Typ an, ohne jedoch eine starre Konvention zu benutzen[8]. Bei der grafischen Darstellung lassen wir den Typ weg. Ein Attributname wie *Abfahrt* wäre zu allgemein, da offen bleibt, ob ein Zeitpunkt oder ein Ort gemeint ist. Ein Name wie *HalteZeit* ist unklar, da man nicht weiß, ob ein Zeitpunkt oder ein Zeitraum gemeint ist. Wir verwenden daher bestimmte Wortzusätze wie *Ort*, *Zeit* (Uhrzeit ohne Tagesbezug), *ZeitPunkt* (Uhrzeit und Datum) und *Dauer* (Zeitspanne in Sekunden). Handelt es sich um eine Menge, so verwenden wir den Plural (*AusgeführteFahrten*, nicht *ListeDerAusgeführtenFahrten*).
- Wir vermeiden Wiederholungen des Klassennamens im Attributnamen: also nicht *FahrzeugTyp*, sondern einfach nur *Typ*. Im Schreib- und Denkzusammenhang `Fahrzeug.`*Typ* ist der kurze Name ausreichend; `Fahrzeug.`*FahrzeugTyp* wäre zu schwerfällig.
- Als Konsequenz daraus ergibt sich, dass Attributnamen nur innerhalb ihrer eigenen Klasse eindeutig sein müssen.
- Wertliefernde Funktionen enthalten kein Verb im Namen – also nicht `Fahrt.`*berechnenVerspätung* sondern einfach nur `Fahrt.`*Verspätung*. Verben haben den Nachteil, dass sie keine Information über den Ergebnistyp liefern. Welche Art von Ergebnis sollte man z. B. von einer Funktion namens `Fahrt.`*vergleichenMit(AndereFahrt)* erwarten?

Benennung von Funktionen

- Funktionen, die eine Zustandsänderung bewirken, sollen nicht gleichzeitig einen fachlichen Wert zurückliefern. Sie beginnen per Konvention mit einem Verb im Infinitiv: z. B. `Fahrt.`*verlängernBisHaltestelle(ZielHaltestelle)*.
- Namen von Funktionsparametern wählen wir so, dass daraus ihr Typ möglichst klar hervorgeht (also z. B. *ZielHaltestelle* und nicht einfach *Ziel*. Bei der Darstellung von Funktionssignaturen zeigen wir nur den Namen der Parameter, nicht deren Typ.
- Gibt es keine Parameter, so lassen wir auch die Klammern weg, also nicht `Fahrt.`*Verspätung()*, sondern `Fahrt.`*Verspätung*. Den Unterschied zwischen einem Attribut und einer parameterlosen Funktion kann man daher nur an der Definitionsstelle erkennen, nicht jedoch im Benutzungskontext.

7. Bei einer objektorientierten Realisierung könnten in verschiedenen Paketen Klassen gleichen Namens vorkommen.
8. Ein strikter Formalismus, der immer ein typgebundenes Suffix wie »Zahl«, »Text« oder »Datum« an die Attributnamen anhängt, ist uns zu schwerfällig.

14.5 Zeitliche Muster und Ereignisse

Ein besonderes Problem bei der Namensgebung ist die Unterscheidung zwischen Begriffen, die ein wiederkehrendes zeitliches Muster beschreiben, und solchen, die sich auf ein konkretes, historisch einmaliges Ereignis beziehen. Hierzu ein Beispiel:

> Versteht man unter einer `Fahrt`, dass um 6:53 an jedem Werktag ein Fahrzeug der Linie 17 am Hauptbahnhof in Richtung Messeplatz halten soll? Oder ist mit `Fahrt` die Aussage gemeint, dass am Donnerstag, den 17.12.2009 ein Fahrzeug der Linie 17 in Richtung Messeplatz am Hauptbahnhof um 6:55 gehalten hat?

Homonyme

Im täglichen Sprachgebrauch sprechen Verkehrsbetriebe in beiden Fällen von *der Fahrt*, da aus dem Zusammenhang heraus jeweils klar ist, worum es sich handelt: Der Mitarbeiter in der Planungsabteilung meint eine geplante Fahrt[9], Leitstelle und Fahrer meinen die aktuelle Fahrt des laufenden Tages, und der Mitarbeiter im Qualitätsmanagement meint die historisch durchgeführte Fahrt, die sich in seiner Pünktlichkeitsstatistik niedergeschlagen hat.

Darstellungsvarianten für Klassennamen mit Zeitbezug

In unserem Modell müssen wir zwei unterschiedliche Bezeichnungen finden, denn zu jeder geplanten `Fahrt` gibt es viele `durchgeführte Fahrten`.

Die folgende Tabelle zeigt die Schwierigkeiten der Namensgebung. Sie skizziert fünf Varianten für die Benennung von *Fahrten*, wobei in der zweiten Spalte jeweils die *geplante* Fahrt gemeint ist, in der dritten Spalte die *konkrete* Fahrt, die an einem bestimmten Kalendertag stattfindet.

Tab. 14-1 Namensgebung für temporale Klassen

Variante	zeitlich wiederkehrend	historisch einmalig	Vor- und Nachteile der Benennung
1	`Fahrt`	`Fahrt`	▪ Verwechslungsgefahr, Mehrdeutigkeit an Schnittstellen ▪ unzulässig für ein übergreifendes Modell
2	`PlanFahrt`, `GeplanteFahrt`	`Fahrt`	▪ korrekt, aber etwas umständlich ▪ wenig Akzeptanz bei Planern
3	`Fahrt`	`WirklicheFahrt`, `AusgeführteFahrt`, `TatsächlicheFahrt`, `DatierteFahrt`	▪ bombastisch ▪ für die Leitstelle nicht zumutbar
4	`FahrtP`	`FahrtA`, `FahrtT`	▪ knapper, aber das Suffix A für »ausgeführt« ist nicht intuitiv; Das »T« für »Time« spricht sich schlecht; das Suffix ist sprachabhängig ▪ den Planer stört immer noch das angehängte »P«
5	`Fahrt`	`FahrtX`	▪ ungewohnt, »X« klingt nach Mathematik, sprachunabhängiges Suffix ▪ aus unserer Sicht die akzeptabelste Variante

9. Er meint also die Definition eines Ereignisses, das an zahlreichen konkreten Tagen »planmäßig« stattfinden soll.

Für das ITVU-Modell haben wir folgendermaßen entschieden:

- Ein Suffix ist besser als ein Präfix, denn es zieht weniger Aufmerksamkeit auf sich und ordnet bei der Sortierung korrespondierende Elemente hintereinander ein (`Fahrt`, `FahrtX`, `Umlauf`, `UmlaufX`).
- Es wäre schade, auf den natürlichen Begriff (ohne Suffix) ganz zu verzichten. Da ein erheblicher Teil der Modellkomplexität im Bereich der Planung liegt, bevorzugen wir hier den natürlichen Begriff ohne Suffix.
- Ein Suffix, das phonetisch mit einem Vokal beginnt (»iX«) ist besser geeignet als ein Konsonant (»T«). Das X steht für »Beliebigkeit« und bildet daher eine passende Assoziation, denn zu einer (geplanten) `Fahrt` kann es eben »x-beliebig« viele konkrete Fahrten geben. Außerdem hat das »X« den Vorteil, dass es in der englischen Fassung unverändert benutzt werden kann.

Darstellungsvarianten für Attribute mit Zeitbezug

Auch bei der Benennung der Attribute müssen wir aufpassen: Im Planungsumfeld kann es sich nur um geplante Zeiten handeln; bei der ausgeführten Fahrt gibt es Sollvorgaben und tatsächliche Ist-Zeiten. Wie soll man dies sprachlich unterscheiden? Tabelle 14-2 zeigt am Beispiel der Abfahrtszeiten mehrere Alternativen:

Tab. 14-2 Namensgebung für Attribute

	Fahrt	FahrtX	Vor- und Nachteil der Benennung
1	*AbfahrtZeitSoll*	*AbfahrtZeitSoll* *AbfahrtZeitIst*	▪ gut lesbar, aber Verstoß gegen Normalformen
2	*AbfahrtZeitSoll*	*AbfahrtZeitIst* →`Fahrt`.*AbfahrtZeitSoll*[3]	▪ saubere Modellierung, aber semantische Doppelung bei der `Fahrt`
3	*AbfahrtZeit*	*AbfahrtZeitX* →`Fahrt`.*AbfahrtZeit*	▪ saubere Modellierung, aber semantische Doppelung bei der `FahrtX`
4	*AbfahrtZeit*	*AbfahrtZeit* →`Fahrt`.*AbfahrtZeit*	▪ schlank, einfach, aber nur lokal eindeutig

Die erste Variante legt nahe, dass innerhalb jeder `FahrtX` die geplante *AbfahrtZeit* als eine Kopie des originalen Werts aus der Fahrt verfügbar ist. In realen Softwaresystemen mag es gute Gründe zur Erzeugung solcher Redundanzen geben[10]. In einem konzeptionellen Modell hingegen dürfen wir dies nicht zulassen. Wir werden die geplante *AbfahrtZeit* für eine konkrete Fahrt daher ausdrücken, indem wir durch das Modell navigieren.[11] Variante 1 scheidet deshalb aus. Welche der drei anderen Möglichkeiten ist die beste?

Navigation entlang von Verknüpfungen

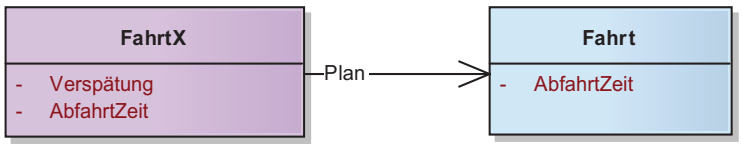
Jeder konkreten Fahrt liegt ein Fahrplan zugrunde, d. h., jede `FahrtX` hat einen Bezug zu einer `Fahrt`. Umgekehrt gibt es zu einer `Fahrt` normalerweise zahlreiche Fahrt-Ausführungen. Wenn wir von einer `FahrtX` ausgehen, so kommen wir eindeutig zu der zugehörigen geplanten `Fahrt`. Die übliche Notation für das »Navigieren in eindeutiger Richtung« verwendet Punkte; man schreibt also z. B. `FahrtX`.*Fahrt*.*AbfahrtZeitSoll*.

10. Hierfür könnten z. B. Performance-Aspekte oder der Wunsch nach einer technischen Entkopplung zwischen Planungssystem und Leitstellensystem sprechen.
11. Dies hindert uns natürlich nicht daran, eine Funktion namens *AbfahrtZeitSoll* bei der `FahrtX` zu definieren, in der wir die Navigation »verstecken«.

Bezeichnerverkettung

Beim Aufschreiben dieser Wortkette fällt auf, dass der Zusatz »Soll« eigentlich verzichtbar ist, da durch den vorangestellten Verweis auf die `Fahrt` bereits klar ist, dass es nur um geplante Zeiten gehen kann. Mit dem gleichen Argument kann man das Suffix »Ist« oder »X« bei der *AbfahrtZeit* der `FahrtX` weglassen. Wenn es sich um eine *AbfahrtZeit* handelt, die spezifisch für eine `FahrtX` ist, dann kann es nur um die tatsächliche, konkrete *AbfahrtZeit* gehen, die historisch eingetreten ist. Wir meinen, dass die vierte Möglichkeit aus Tabelle 14-2 die beste ist.

Abb. 14-5
Benennung von Beziehungen



Zusätzlich bietet es sich an, anstelle des Wortes »Fahrt« die spezifische *Rolle* zu benutzen, die die Fahrt aus Sicht der FahrtX innehat – in diesem Falle also das Wort »*Plan*«. Die Ermittlung der Verspätung einer `FahrtX` wird daher von uns wie folgt notiert:

```
FahrtX.Verspätung =   FahrtX.AbfahrtZeit
                    - FahrtX.Plan.AbfahrtZeit
```

Ein positiver Wert entspricht einer Verspätung, ein negativer Wert drückt eine Verfrühung aus.

Lokale Namensräume

Nicht immer hat man es so schwer mit der Namensgebung wie in diesem Fall, aber das Beispiel zeigt noch einmal, wie viel Sorgfalt nötig ist, wenn man ein Modell entwirft, das alle Bereiche abdecken soll. Würde man getrennte Modelle für ein Leitstellensystem und ein Fahrplanungssystem entwickeln, so könnte man im Planungssystem den Begriff *Fahrt* mit seiner lokalen Semantik verwenden. Erst wenn man dann an der Schnittstelle Daten austauscht, kommt es zu sprachlichen Verrenkungen.

14.6 Fachlich eindeutige Schlüsselbegriffe

Bei der klassischen Datenmodellierung legte man Wert darauf, für jede Klasse mindestens ein Attribut (oder eine Attributkombination) zu definieren, das eine Instanz dieser Klasse (= ein Objekt) eindeutig identifiziert.

Primärschlüssel

Bei einem `Mitarbeiter` könnte man z. B. *Name*, *GeburtsDatum* und *Geburts-Ort* festlegen. Da man aber oft nicht absolut sicher sein kann, dass die gewählte Kombination zwingend eindeutig ist, geht man zu künstlichen Schlüsselsystemen über, etwa indem man eine »nichtssagende« *MitarbeiterNummer* vergibt und sicherstellt, dass diese eindeutig ist. Man muss sich auch dann noch mit Widrigkeiten im Detail herumschlagen: Was soll z. B. passieren, wenn ein Mitarbeiter das Unternehmen verlässt und später wieder eingestellt wird? Es ist zweifellos noch dieselbe Person, aber in vielen Systemen würde man sich wohl dafür entscheiden, ihm eine neue Nummer zuzuordnen.