

## Lernziele

Der Kurs „Objektorientierte Systemanalyse“ behandelt die objektorientierte Analyse zur Entwicklung betrieblicher Anwendungssysteme und vermittelt darüber hinaus allgemeine Grundlagen des Software Engineering. Nach dem Durcharbeiten des Kurses sollen Sie einen Überblick des Software Engineering erworben haben sowie die Methode der objektorientierten Analyse einschließlich ihrer UML-Notation kennen und selbständig anwenden können.

### allgemeine Lernziele

Hiermit sollen Sie eine solide Basis für die eigenständige Abrundung Ihrer Kenntnisse und Fähigkeiten in der objektorientierten Analyse erworben haben, nach der Sie zur objektorientierten Modellierung von betrieblichen Anwendungssystemen in der Lage sind.

Aus dieser allgemeinen Zielsetzung leiten sich Teilziele für die einzelnen Kurseinheiten ab. Nach der Bearbeitung der zweiten Kurseinheit sollen Sie im einzelnen folgende kenntnisorientierten und verhaltensorientierten Lernziele erreicht haben:

- Sie kennen Techniken der Anforderungsermittlung und Darstellungskonzepte für funktionale Anforderungen, die zur Erstellung des Pflichtenheftes verwendet werden können. Der Zweck, die Bedeutung und die inhaltlichen Schwerpunkte einer fachlichen Vorstudie und eines Pflichtenheftes sind Ihnen bekannt, und Sie sind in der Lage, diese Dokumente für einfachere betriebliche Anwendungen zu erarbeiten.
- Sie haben die Konzepte der objektorientierten Analyse sowie ihre Notation in der UML kennengelernt. Sie können diese Konzepte innerhalb der wichtigsten Diagramme der UML anwenden und haben die Fähigkeit erworben, statische und dynamische Modelle für betriebliche Anwendungen zu verstehen und selbständig zu erstellen. Insbesondere sind Sie auch mit der im Kurs vermittelten methodischen Vorgehensweise bei der objektorientierten Analyse vertraut und kennen strategische und taktische Regeln, die bei der Modellierung betrieblicher Anwendungen zu berücksichtigen sind.
- Sie kennen Grundbegriffe und grundlegende Anforderungen an grafische Benutzungsoberflächen betrieblicher Anwendungssysteme sowie Grundzüge des GUI-Systems von *MS Windows*. Sie sind mit Grundlagen der Gestaltung von Dialogabläufen, Fenstern und Menüs vertraut, die für die Erstellung des Prototyps einer grafischen Benutzungsoberfläche benötigt werden. Anhand der Programmierumgebung *MS Visual C++* haben Sie gelernt, einfache Prototypen selbständig zu programmieren und dabei einige Elemente der Klassenbibliothek der *MS Foundation Classes* einzusetzen.

### Teilziele für die zweite Kurseinheit

### 3 Erstellung des Pflichtenheftes

Die Erarbeitung des Pflichtenheftes erfolgt in zwei Schritten. Sie wird eingeleitet mit der fachlichen Voruntersuchung, die zur Vorstudie führt. In einem zweiten Schritt wird ausgehend von der Vorstudie das Pflichtenheft selbst erstellt.

**Schritte**

Beide Schritte sind methodisch nicht weit voneinander entfernt. Es sind jeweils Anforderungen an die Anwendung einerseits zu erheben und zu sammeln, andererseits zu formulieren und darzustellen. Die für die Vorstudie und das Pflichtenheft benötigte Methodik sei daher vorab beschrieben. Zur Erhebung und Sammlung von Anforderungen können herkömmliche Techniken der Organisationsanalyse verwendet werden (siehe Kap. 3.1). Eine semiformale Darstellung der Funktionalität der zu entwickelnden Anwendung wird durch Anwendungsfälle unterstützt (siehe Kap. 3.2).

**Methodik**

Die Voruntersuchung beschreibt zunächst den Anwendungsbereich des zu entwickelnden Softwareprodukts aus fachlicher Sicht. Der Anwendungsbereich bezeichnet dabei einen abgegrenzten betrieblichen Bereich, in dem das Softwareprodukt später eingesetzt werden soll. Weiterhin sollte die derzeitige Informationsverarbeitung im Anwendungsbereich untersucht werden, um Schwachstellen zu identifizieren, die mit der neuen Anwendung beseitigt werden sollen. Schließlich werden die Anforderungen an das neue Softwareprodukt in einem groben Lösungskonzept umrissen (siehe Kap. 3.3).

**Voruntersuchung**

Basierend auf der fachlichen Vorstudie kann nun einerseits die Durchführbarkeit und Vorteilhaftigkeit der zu erstellenden Anwendung untersucht und die Projektdurchführung geplant werden (vgl. Kap. 1.7).

Die Vorstudie, darunter vor allem das Lösungskonzept, bildet zum anderen den Ausgangspunkt für die folgende Erstellung des Pflichtenheftes. In diesem werden die Produktanforderungen vollständig, im Vergleich zur Vorstudie bereits detaillierter und teilweise formalisiert niedergelegt (siehe Kap. 3.5).

**Pflichtenheft**

Nicht selten erfordern betriebliche Anwendungen auch die Lösung komplexerer quantitativer Probleme, darunter betrieblicher Entscheidungsprobleme. Dies sollte gegebenenfalls bereits während der Erstellung des Pflichtenheftes Berücksichtigung finden (siehe Kap. 3.7).

**Entscheidungsprobleme**

Die vorstehend skizzierten Schritte der Erarbeitung des Pflichtenheftes werden jeweils anhand des Kursprojekts „Tourenplanungssystem“ demonstriert (siehe Kap. 3.4, 3.6 und 3.8). Die Kap. 3.4 und 3.8 basieren auf entsprechenden Darstellungen in GEHRING (1993).

**Tourenplanungssystem (TPS)**

#### 3.1 Techniken der Anforderungsermittlung

Zur Ermittlung von Anforderungen und relevanten Sachverhalten des Anwendungsbereiches eines zu entwickelnden Softwaresystems werden in der Praxis verschiedene Techniken verwendet. Eine Auswahl wird nachfolgend kurz vorgestellt (vgl. z. B. COAD und YOURDON 1994, OESTEREICH 1998).

### (1) Dokumenten- und Literaturstudium

Das Dokumentenstudium dient der Analyse schriftlicher Unterlagen, die relevante Informationen über den Anwendungsbereich enthalten. In Betracht kommen z.B. Dokumente wie

- relevante Dokumente**
- Organisationspläne und Stellenbeschreibungen,
  - Arbeitsanleitungen und Ablaufbeschreibungen,
  - Formulare, Drucklisten und Erfassungsbelege,
  - Produktbeschreibungen, Stücklisten, Kunden- und Lieferantenverzeichnisse,
  - Programmdokumentationen und Dateibeschreibungen.

#### Fachliteratur

Das Studium operativer Dokumente ist eine wichtige Technik für den Systemanalytiker, um sich in den Anwendungsbereich einzuarbeiten. Darüber hinaus ist oft ein Studium einschlägiger Fachliteratur sinnvoll oder gar zwingend erforderlich.

### (2) Interviews

#### Interviews: wozu und wer?

Bei einem Interview befragt der Systemanalytiker seinen Interviewpartner zu Gegebenheiten des Anwendungsbereiches oder zu gewünschten Merkmalen und Eigenschaften des zu entwickelnden Softwaresystems. Die Interviewpartner werden meist fachliche Experten, Mitarbeiter des Anwendungsbereiches oder spätere Systembenutzer sein.

#### wie?

Grundlage eines Interviews ist ein sorgfältig vorbereiteter Fragenkatalog. Wichtig ist eine aufgeschlossene und konstruktive Gesprächsatmosphäre. So sollte der Systemanalytiker z.B. keine voreilige Kritik an der bisherigen Gestaltung der Informationsverarbeitung äußern. Ebenso sollten dem Interviewpartner gegenüber unverständliche Fachausdrücke der Informationsverarbeitung vermieden werden. Zur Auswertung eines Interviews ist ein schriftlicher Bericht anzufertigen.

#### Arten

Neben Einzelinterviews können auch Gruppeninterviews mit mehreren Interviewpartnern durchgeführt werden. Ferner sind strukturierte und freie Interviews zu unterscheiden. Bei einem strukturierten Interview liegen die Fragen dem Partner schon vor dem Interview in schriftlicher Form vor. Der Ablauf eines freien Interviews kann durch die Gesprächspartner während des Interviews bestimmt werden.

### (3) Fragebogen

Ein Fragebogen dient der schriftlichen Befragung meist mehrerer Mitarbeiter zu denselben Sachverhalten. Mit einem Fragebogen sollen Tatsachen aus dem Anwendungsbereich ermittelt werden, die durch knappe und eindeutige Fragestellungen umrissen werden können. Beispielsweise können Mengengerüste, d.h. mengenmäßige Anforderungen wie etwa die Anzahl zu bedienender Kunden durch Fragebögen erfasst werden. Ein Fragebogen sollte übersichtlich strukturiert sein und nur präzise formulierte und eindeutig beantwortbare Fragen enthalten.

### (4) Beobachtung

Durch die unmittelbare Beobachtung des Anwendungsbereiches und seiner Arbeitsabläufe gewinnt der Systemanalytiker eigene, „ungefilterte“ Eindrücke und Erfahrungen.

Eine Beobachtung kann kurzfristig oder längerfristig erfolgen. Eine längerfristige kontinuierliche Beobachtung wird z.B. durch eine zeitweilige Mitarbeit eines Systemanalytikers im Anwendungsbereich ermöglicht.

### (5) Glossar relevanter Fachbegriffe

Empfehlenswert ist die Einrichtung und Pflege eines Glossars der relevanten fachlichen Begriffe. Eine von den Projektmitarbeitern und den Mitarbeitern der Auftraggeberseite übereinstimmend verwendete Terminologie ist eine wesentliche Bedingung einer funktionierenden Kommunikation.

## Übungsaufgaben zu Kapitel 3.1

### Übungsaufgabe 3.1.1

Die beschriebenen Techniken zur Anforderungsermittlung werden innerhalb der Analyse generell kombiniert genutzt. In welchem Umfang und wann einzelne Techniken zum Einsatz kommen, hängt natürlich von den konkreten Bedingungen eines Projekts und nicht zuletzt von den persönlichen Erfahrungen und Neigungen der Entwickler ab. Jedoch bestehen auch allgemeine Abhängigkeitsbeziehungen zwischen den genannten Techniken. Umreißen Sie einige solche Beziehungen.

## 3.2 Konzepte zur Darstellung funktionaler Anforderungen

Zur systematischen Darstellung der Funktionalität eines gewünschten Software-systems innerhalb des Pflichtenheftes können verschiedene Konzepte verwendet werden. Im objektorientierten Paradigma sind vor allem die von JACOBSON et al. (1992) eingeführten use cases (deutsch: Anwendungsfälle) gebräuchlich. Ergänzend wird hier auch das herkömmliche Konzept des Funktionsbaumes vorgestellt.

**Konzepte**

Ein Anwendungsfall eines zu entwickelnden Softwaresystems beschreibt eine bestimmte Variante der Benutzung des Systems. Die Gesamtheit der Anwendungsfälle eines Systems definiert die gesamte Funktionalität der Software.

**Anwendungsfall**

Die Beschreibung der Funktionalität eines Softwaresystems mittels Anwendungsfällen besitzt vor allem folgende Merkmale:

**Merkmale**

- Anwendungsfälle stellen Arbeitsabläufe dar. Den Kern eines Anwendungsfalls bildet eine Sequenz von Aktionen zur Bearbeitung einzelner Aufgaben. Wenigstens ein Teil, jedoch nicht unbedingt alle Aufgaben werden dabei von einem oder mehreren Benutzern mit Hilfe des Softwaresystems erledigt. Die Funktionalität eines Systems wird also dynamisch beschrieben.
- Die innerhalb eines Anwendungsfalls zu erledigenden Aufgaben führen zu einem aus der Benutzerperspektive in sich abgeschlossenen Resultat. Mit den Worten von JACOBSON et al. (1994) soll ein Anwendungsfall für den oder die Benutzer ein „Ergebnis von meßbarem Wert“ liefern.
- Ein Anwendungsfall stellt eine Variante der Systembenutzung auf einem hohen Abstraktionsniveau dar. Zum einen wird von allen Aspekten der konkreten Umsetzung des jeweiligen Arbeitsablaufes mittels der Software, so etwa von der Gestaltung der Benutzungsoberfläche, abstrahiert. Zum anderen werden der

Anwendungsfall wie auch die zu ihm gehörenden Aktionen inhaltlich nur grob beschrieben, worauf noch konkreter einzugehen ist.

Erwähnt sei, dass JACOBSON et al. neben den Anwendungsfällen eines Softwaresystems auch Anwendungsfälle in einem Unternehmen betrachten, die „eine Sequenz von Transaktionen“ im Unternehmen definieren (vgl. JACOBSON et al. 1994). Derartige Anwendungsfälle umfassen oft die gleichzeitige Benutzung mehrerer Systeme, sind auf einem noch höheren Abstraktionsniveau als die Anwendungsfälle eines einzelnen Softwaresystems angesiedelt, stellen komplette Geschäftsprozesse dar und werden hier nicht betrachtet (vgl. z.B. GEHRING 1999).

#### Akteur

Das Konzept des Anwendungsfalls wird durch das Konzept des Akteurs ergänzt. Ein Akteur definiert eine gewisse Rolle, die von einem Benutzer eines Softwaresystems bei der Bearbeitung eines Anwendungsfalls ausgeübt wird. Akteure gehören selbst nicht zum System, sondern zu seiner Umwelt. Derselbe Benutzer kann bei einem Anwendungsfall mehrere Rollen übernehmen, d.h. mehrere Akteure repräsentieren. Bei den Akteuren eines Softwaresystems kann es sich auch um Organisationseinheiten oder weitere Softwaresysteme handeln. Ein Beispiel für einen Akteur bildet etwa ein Sachbearbeiter des Verkaufsbereiches einer Firma, der mittels eines Softwaresystems eintreffende Kundenaufträge erfasst.

#### Anwendungsfallsschablone

Ein Anwendungsfall wird umgangssprachlich oder auch semiformal beschrieben. Er kann semiformal mittels einer Anwendungsfallsschablone dargestellt werden, deren Aufbau in Abb. 3.1 gezeigt wird (vgl. COCKBURN 1997, BALZERT 1999).

##### **Anwendungsfall:**

Name des Anwendungsfalls

##### **Ziel:**

Allgemeine Zielstellung des Anwendungsfalls.

##### **Akteure:**

Rollen der am Anwendungsfall beteiligten Personen, Organisationseinheiten oder externen Systeme.

##### **Kategorie:**

Primär oder sekundär.

##### **Initiierendes Ereignis:**

Ereignis, welches den Anwendungsfall auslöst.

##### **Vorbedingung:**

Notwendige Bedingung für Bearbeitung des Anwendungsfalls.

##### **Nachbedingung bei Erfolg:**

Ergebnis bei erfolgreicher Bearbeitung des Anwendungsfalls.

##### **Nachbedingung bei Misserfolg:**

Ergebnis bei erfolgloser Bearbeitung des Anwendungsfalls.

##### **Standardspezifikation:**

Aktion 1: Beschreibung der ersten Aktion.

Aktion 2: Beschreibung der zweiten Aktion.

...

##### **Erweiterungen der Standardspezifikation:**

Aktion 1a: Erweiterung der ersten Aktion.

Aktion 1b: Erweiterung der ersten Aktion.

...

Aktion 2a: Erweiterung der zweiten Aktion.

...

**Alternativen zur Standardspezifikation:**

Aktion 1a: Alternative Ausführung der ersten Aktion.

...

**Abb. 3.1.** Schablone eines Anwendungsfalls.

Einige Komponenten der Schablone seien näher erläutert:

- Die Kategorie eines Anwendungsfalls dient der Differenzierung zwischen wichtigeren und weniger wichtigen Anwendungsfällen.
- Mit Vorbedingungen und Nachbedingungen können u.a. mögliche Reihenfolgen der Bearbeitung von Anwendungsfällen erzwungen werden. Betrachtet seien etwa die Anwendungsfälle „Liefere Ware“ und „Erstelle Rechnung“. Die Nachbedingung des ersten Anwendungsfalls, nämlich „Ware geliefert“, sei zugleich die Vorbedingung des zweiten. Dann kann bei der Bearbeitung derselben Lieferung der Anwendungsfall „Erstelle Rechnung“ nur nach dem Anwendungsfall „Liefere Ware“ ausgeführt werden.
- Die Aktionen eines Anwendungsfalls werden umgangssprachlich und knapp beschrieben. Erweiterungen und Alternativen der Standardspezifikation zielen ab auf Aktionen, die – zusätzlich oder alternativ – in Sonderfällen erforderlich werden. Sind z.B. Kunden bei der Auftragserteilung normalerweise bereits im System vorhanden und stellt sich bei der Erfassung eines Kundenauftrages heraus, dass der Kunde selbst noch unbekannt ist, müssen zusätzlich seine Daten erfasst werden.

Nicht benötigte Komponenten der Schablone können jeweils unberücksichtigt bleiben. Das folgende Beispiel 3.1 zeigt den Anwendungsfall „Kfz-zurücknehmen“ des im Beispiel 1.4 des Kap. 1.4.1 skizzierten Informationssystems einer Autoverleihfirma.

**Beispiel  
Anwendungsfall**

**Beispiel 3.1**

**Anwendungsfall:** Kfz-zurücknehmen

**Ziel:** Vermietetes Kfz zurücknehmen und Rechnung erstellen.

**Akteure:** Mitarbeiter Kfz-Vermietung.

**Kategorie:** Primär.

**Initiierendes Ereignis:** Kunde erscheint und möchte Kfz zurückgeben.

**Vorbedingung:** Kfz an Kunde vermietet.

**Nachbedingung bei Erfolg:** Kfz zurückgenommen und Rechnung erstellt.

**Nachbedingung bei Misserfolg:** Beschädigtes Kfz zurückgenommen, keine Rechnung erstellt; Weiterbehandlung mit Anwendungsfall Kfz-Schadenbehandeln.

**Standardspezifikation:**

- 1 Mietvertrag anhand Mietvertragsnummer suchen.



- 2 Einhaltung Mietvertrag prüfen, darunter:
  - vollständige Rückgabe aller vermieteten Gegenstände,
  - Termineinhaltung,
  - Kfz-Schäden.
- 3 Abrechnungsdaten erfassen (km-Stand und Kraftstoffverbrauch).
- 4 Rechnung erstellen, ausdrucken und an Kunde übergeben.

**Erweiterungen der Standardspezifikation:**

- 2a Schadensprotokoll bei Kfz-Schäden und beschädigten bzw. fehlenden Zubehörteilen aufnehmen.
- 3a Terminverzug zusätzlich unter Abrechnungsdaten erfassen.

**Alternativen zur Standardspezifikation:**

- 1a Mietvertrag über Kundennamen und zugehörige Vertragsliste suchen.
- 4a Im Schadensfall wird (zunächst) keine Rechnung erstellt, sondern Vereinbarung mit Kunden über weiteres Vorgehen getroffen und protokolliert.

**Anwendungsfall-  
diagramm**

Ein Anwendungsfalldiagramm dokumentiert alle gewünschten Anwendungsfälle eines Softwaresystems in einer überblicksartigen grafischen Darstellung. Im Einzelnen werden folgende Sachverhalte veranschaulicht:

1. Die Gesamtheit der Anwendungsfälle des Softwaresystems.
2. Die Gesamtheit der Akteure und ihre Kommunikation mit dem System. Pro Anwendungsfall sind alle beteiligten Akteure anzugeben.
3. Beziehungen zwischen Anwendungsfällen oder auch Akteuren. Hierzu gehören Beziehungen der Generalisierung von Anwendungsfällen bzw. Akteuren und ferner Inklusions- und Erweiterungsbeziehungen zwischen Anwendungsfällen. Da die genannten Beziehungen hier von untergeordneter Bedeutung sind, werden sie nicht näher betrachtet (vgl. z.B. OESTERICH 1998).

**Elemente**

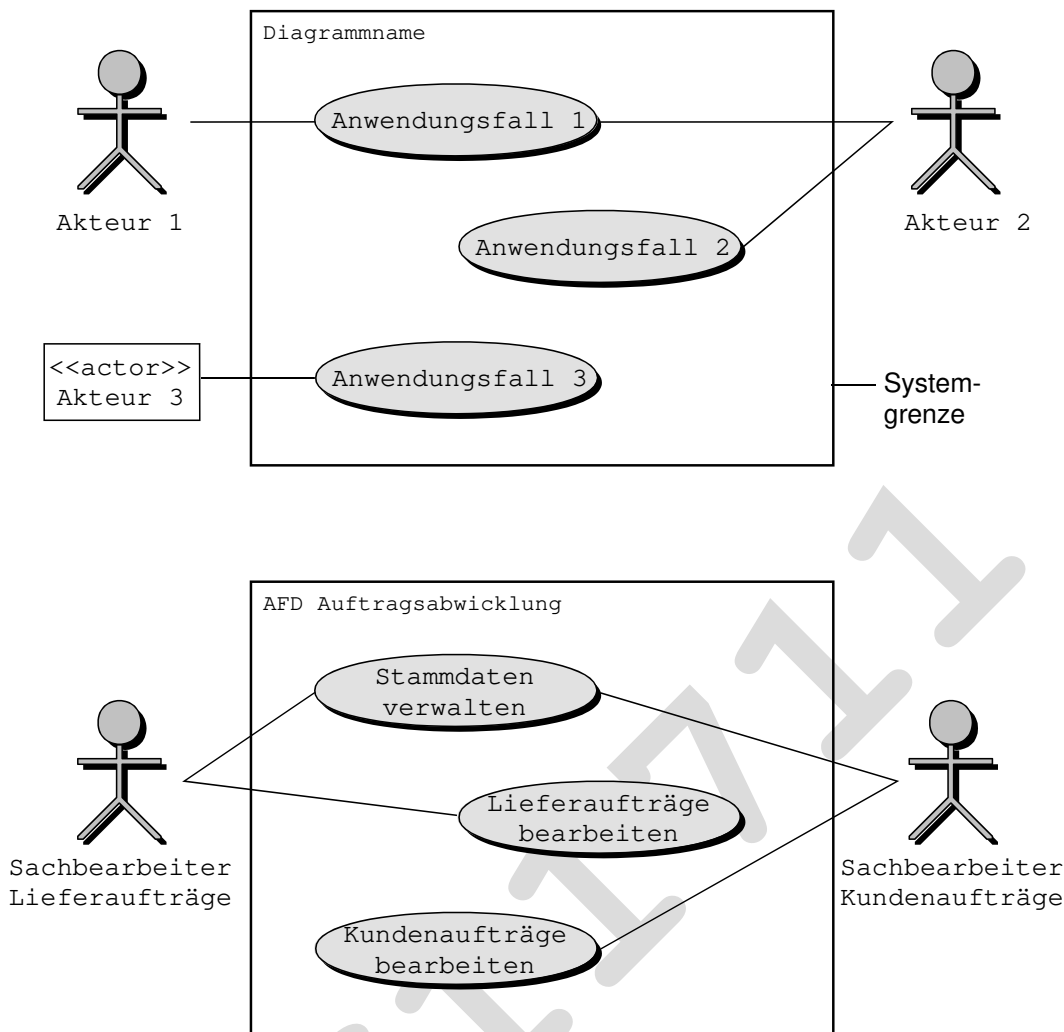
In der UML werden die Anwendungsfälle eines Anwendungsfalldiagramms durch Ellipsen dargestellt, die den Namen des Anwendungsfalls umschließen. Akteure werden durch Strichmännchen oder Rechtecke wiedergegeben, die jeweils mit dem Namen des Akteurs beschriftet sind. Die Kommunikation eines Akteurs mit dem System bei der Ausführung eines Anwendungsfalls wird durch eine Verbindungslinie zwischen Akteur und Anwendungsfall veranschaulicht.

**Stereotypen**

Ein Akteur kann zusätzlich durch das Stereotyp <<actor>> gekennzeichnet werden. Stereotypen sind ein Sprachkonzept der UML, das zur Klassifizierung verschiedener Modellelemente wie z.B. Klassen genutzt werden kann. Sie werden stets in französische Anführungszeichen (<<, >>) eingeschlossen. Zusätzlich zu den in der UML vordefinierten Stereotypen – wie dem gerade vorgestellten – lassen sich weitere Stereotypen einführen. Beispielsweise kann das Stereotyp <<Stammdaten>> für Klassen eingeführt werden, die der Verwaltung von Stammdaten einer Anwendung dienen.

Die folgende Abb. 3.2 veranschaulicht die Notation und zeigt zugleich ein Beispiel für ein Anwendungsfalldiagramm.

**Schema und Beispiel**



**Abb. 3.2.** Notation und Beispiel für ein Anwendungsfalldiagramm.

Das Anwendungsfalldiagramm und die zugehörigen Konzepte lassen sich nicht ausschließlich dem objektorientierten Entwicklungsansatz zurechnen und können ebenso im strukturierten Entwicklungsansatz verwendet werden. Bereits dies spricht für ihre Einführung an dieser Stelle, also noch vor der Behandlung der eigentlichen objektorientierten Konzepte. Ferner beschreiben Anwendungsfälle und das Anwendungsfalldiagramm die Funktionalität eines Systems auf hoher Abstraktionsebene, aus der Perspektive des Auftraggebers sowie verbal bzw. gemäß einer hinreichend einfachen semiformalen Notation. Sie erscheinen daher für die Darstellung der Funktionalität innerhalb des Pflichtenheftes gut geeignet. Während einzelne Anwendungsfälle das System aus einer dynamischen Sicht betrachten, enthält das Anwendungsfalldiagramm als überblicksartige Darstellung aller Anwendungsfälle sowie der Akteure keine dynamischen Aspekte und wird daher der funktionalen Sicht zugeordnet (vgl. Kap. 2.2).

Im Folgenden werden einige methodische Hinweise zusammengefasst, die bei der Ermittlung und Darstellung von Anwendungsfällen (einschließlich der zugehörigen Akteure) beachtet werden sollten. Anwendungsfälle und Anwendungsfalldiagramme können auch für die Beschreibung des Ist-Zustandes einer Informationsverarbeitung genutzt werden (vgl. Kap. 3.3).

**Anwendungsfälle im  
Pflichtenheft**

**Methodische  
Vorgehensweise**



## Methodische Hinweise und Regeln zum Konzept Anwendungsfall

1. Anwendungsfälle und das Anwendungsfalldiagramm sind gleichermaßen für die Entwickler wie für den Auftraggeber bestimmt. Unbedingt erforderlich ist eine Orientierung an der Terminologie des Auftraggebers. Zu achten ist auf aussagekräftige Bezeichnungen von Anwendungsfällen und Akteuren. Erstere sollten durch ein Verb und ein Substantiv bezeichnet werden.
2. Abstraktionsniveau und Granularität von Anwendungsfällen und Aktionen müssen geeignet gewählt werden. Wird nur ein einziger Anwendungsfall identifiziert, ist die Beschreibung fast immer zu grob. Andererseits sollten Anwendungsfälle ihrerseits stets noch verschiedene Varianten der Ausführung, sogenannte Szenarios, zulassen (vgl. Kap. 4.2.2). Nach JACOBSON (1995) umfassen kleinere Anwendungen (zwei bis fünf Mitarbeiterjahre) ca. 3 – 20 Anwendungsfälle, mittlere Anwendungen (10 bis 100 Mitarbeiterjahre) können 10 bis 60 Anwendungsfälle enthalten. Allerdings schwanken entsprechende Angaben in der Literatur stark (vgl. OESTEREICH 1998). Hier werden der Einfachheit der Fallbeispiele entsprechend eher kleine Anwendungsfälle spezifiziert. Anwendungsfälle sollten etwa auf ein bis zwei Seiten, Aktionen in ein bis drei Sätzen beschrieben werden. Die Granularität verschiedener Anwendungsfälle sowie der Aktionen eines Anwendungsfalls sollte möglichst übereinstimmen.
3. Anwendungsfälle und Akteure können mit Hilfe von Interviews der Mitarbeiter der Auftraggeberseite, anhand von Vorgängersystemen und durch ein Dokumentenstudium identifiziert werden. Wichtige Fragen sind:
  - Welche Anwendungsfälle können aus einer ersten, vom Auftraggeber stammenden Beschreibung der Funktionalität eines neuen Systems abgeleitet werden?
  - Welche Mitarbeiter sind derzeit bzw. sollen künftig für welche Arbeitsabläufe verantwortlich sein?
  - Welche Aufgaben eines Arbeitsablaufes werden derzeit bzw. sollen künftig durch das neue System unterstützt werden?
  - Welche Ereignisse externer oder zeitlicher Art lösen Arbeitsabläufe aus? Z.B. können nachts automatisch Verarbeitungen angestoßen werden (Batchbetrieb).
  - Wie ist die Schnittstelle des Systems zu ziehen, was gehört zum System und was zu seiner Umwelt?
4. Zur näheren Beschreibung eines Anwendungsfalls können u.a. die vorher genannten Informationsquellen genutzt werden. Die Anwendungsfallschablone (vgl. Abb. 3.1) stellt bereits eine Grundlage der zielgerichteten Informationsgewinnung dar. Weitere Fragen können sich z.B. auf Reihenfolgebedingungen für Arbeitsschritte, Eingabedaten und Zwischenergebnisse beziehen.
5. Bei der Beschreibung eines Anwendungsfalls sollte man sich zunächst immer auf den Standardablauf konzentrieren. Sonderfälle (Alternativen, Erweiterungen) sollten erst danach behandelt werden.
6. Für kleinere Anwendungen bietet es sich an, einem Top-Down-Ansatz folgend zunächst alle Anwendungsfälle und Akteure zu identifizieren und ein Anwendungsfalldiagramm zu zeichnen, bevor einzelne Anwendungsfälle komplett spezifiziert werden. Bei größeren Anwendungen kommt wegen der

Abhängigkeiten der Anwendungsfälle auch ein Bottom-Up-Ansatz in Betracht.

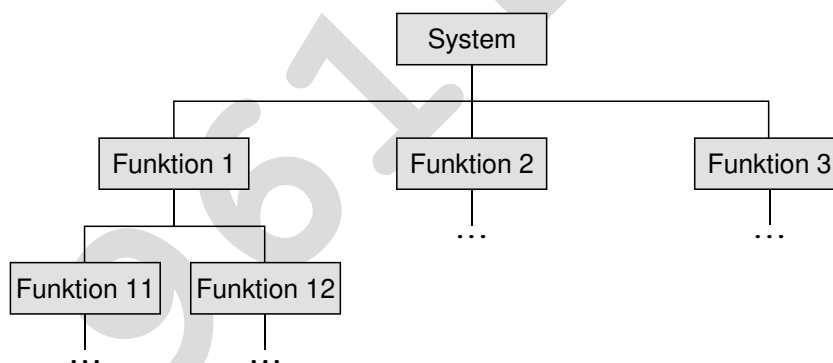
## Funktionsbaum

## Funktionsbaum

Ein weiteres bewährtes Mittel zur Darstellung der Funktionalität einer Anwendung stellt der Funktionsbaum dar. Anders als die dynamisch orientierten Anwendungsfälle gibt der Funktionsbaum die Funktionalität der Anwendung aus statischer Sicht wieder.

Mit einem Funktionsbaum wird die gesamte Funktionalität einer Anwendung hierarchisch strukturiert. Wie bei Baumdarstellungen in der Informatik üblich, wird der Funktionsbaum mit der Wurzel nach oben gezeichnet. Diese repräsentiert die Funktionalität der gesamten Anwendung. Die Funktionsbereiche jeder folgenden Ebene werden jeweils aus einer an inhaltlichen Aspekten ausgerichteten, vollständigen und überschneidungsfreien Untergliederung eines Funktionsbereiches der unmittelbar übergeordneten Ebene gewonnen. So kann etwa der Funktionsbereich Stammdatenverwaltung einer kommerziellen Anwendung nach den verschiedenen zu verwaltenden Stammdaten aufgefächert werden. Jeder Funktionsbereich ist im Funktionsbaum durch einen aussagekräftigen Namen darzustellen. Gegebenenfalls können die Funktionsbereiche eines Funktionsbaums zusätzlich verbal erläutert werden.

Die folgende Abb. 3.3 veranschaulicht das Prinzip eines Funktionsbaumes. Für ein konkretes Beispiel sei auf Kap. 3.6 verwiesen.



Schema

Abb. 3.3. Prinzipieller Aufbau eines Funktionsbaumes.

## Übungsaufgaben zu Kapitel 3.2

### Übungsaufgabe 3.2.1

Der in Beispiel 3.1 dargestellte Anwendungsfall „Kfz-zurücknehmen“ ist offenbar noch unvollständig beschrieben. Nicht alle vorkommenden Sonderfälle wurden beachtet. Nennen Sie zwei Sonderfälle, die im Anwendungsfall noch nicht berücksichtigt wurden.

### Übungsaufgabe 3.2.2

Beschreiben Sie den Anwendungsfall „Kfz-reservieren“ des im Beispiel 1.4, Kap. 1.4.1 skizzierten Informationssystems für eine Autoverleihfirma mittels Anwendungsfallschablone. Unterscheiden Sie zwischen neuen und bereits im System

bekannten Kunden. Der letztgenannte Fall sei der Standardfall, bei welchem der Kunde bereits eine Kundennummer besitzt. Es kann vorkommen, dass ein Reservierungswunsch nicht befriedigt werden kann.

### **Übungsaufgabe 3.2.3**

Zeichnen Sie ein Anwendungsfalldiagramm für das im Beispiel 1.4, Kap. 1.4.1 skizzierte Informationssystem für eine Autoverleihfirma. Allerdings soll die Kfz-Wartung hier unbeachtet bleiben. Treffen Sie auf der Grundlage Ihnen bereits bekannter Anwendungsfälle des Informationssystems sinnvolle Annahmen zu Akteuren und Anwendungsfällen. Ist auch bei der Kundenberatung ein Zugriff auf die Stammdaten sinnvoll?

### **Übungsaufgabe 3.2.4**

Betrachtet sei das als Lösung der vorigen Aufgabe angegebene Anwendungsfalldiagramm. Fahrzeuge können nur im Rahmen eines (zu unterschreibenden) Vertrages gemietet werden. Ein Vertrag kann durch den Kunden erst innerhalb des Anwendungsfalls „Kfz-verleihen“ abgeschlossen werden, in dessen Verlauf das gemietete Fahrzeug auch übergeben wird. Dies zieht eine gewisse Einschränkung der Flexibilität bei der Übergabe von Fahrzeugen nach sich. Welche Einschränkung ist gemeint und wie könnte die Flexibilität bei der Kfz-Übergabe erhöht werden?

### **Übungsaufgabe 3.2.5**

Als Akteure werden hier stets nur die unmittelbaren Systembenutzer (oder externe kooperierende Systeme) identifiziert. Kunden oder allgemeiner Personen, auf die sich eine Systembenutzung zwar bezieht, die das System jedoch selbst nicht nutzen, werden dagegen nicht als Akteure betrachtet. Nennen Sie ein Beispiel, in dem ein Kunde ein Anwendungssystem selbst nutzt und folglich Akteur ist.

### **Übungsaufgabe 3.2.6**

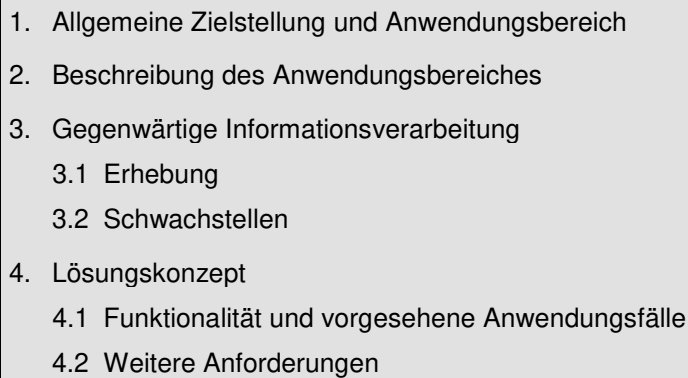
Ein Reisebüro ist spezialisiert auf Pauschalreisen und verkauft innerhalb eines Auftrages mindestens eine Hin- und Rückbeförderung sowie eine Unterbringung. Ein Informationssystem soll alle direkt kundenbezogenen Funktionen eines Reisebüros unterstützen. Hierzu gehören die Kundenberatung, die Verwaltung der Stammdaten, das Buchen von Reisen und deren Abrechnung sowie das Erstellen der Reisedokumente. Identifizieren Sie Akteure und Anwendungsfälle des Informationssystems und zeichnen Sie ein Anwendungsfalldiagramm. Nur unmittelbare Benutzer des Systems sollen Akteure sein; auch eine evtl. Kooperation mit weiteren Systemen ist nicht zu berücksichtigen.

### **Übungsaufgabe 3.2.7**

Beschreiben Sie den Anwendungsfall „Reisedokumente-erstellen“ für das in der vorigen Aufgabe skizzierte Informationssystem eines Reisebüros mittels Anwendungsfallschablone. Treffen Sie sinnvolle Annahmen zu Sonderfällen.

### 3.3 Fachliche Vorstudie

Der Umfang der fachlichen Voruntersuchung wurde bereits knapp umrissen. Einen Gliederungsvorschlag für die Vorstudie zeigt Abb. 3.4. Die Bestandteile der Vorstudie werden anschließend genauer erläutert.

- 
- Das Diagramm zeigt den Aufbau einer Vorstudie in einer nummerierten Liste. Die Punkte sind: 1. Allgemeine Zielstellung und Anwendungsbereich, 2. Beschreibung des Anwendungsbereiches, 3. Gegenwärtige Informationsverarbeitung (mit Unterpunkten 3.1 Erhebung und 3.2 Schwachstellen), 4. Lösungskonzept (mit Unterpunkten 4.1 Funktionalität und vorgesehene Anwendungsfälle und 4.2 Weitere Anforderungen).
1. Allgemeine Zielstellung und Anwendungsbereich
  2. Beschreibung des Anwendungsbereiches
  3. Gegenwärtige Informationsverarbeitung
    - 3.1 Erhebung
    - 3.2 Schwachstellen
  4. Lösungskonzept
    - 4.1 Funktionalität und vorgesehene Anwendungsfälle
    - 4.2 Weitere Anforderungen

**Schema**

**Abb. 3.4.** Aufbau einer Vorstudie.

Die Vorstudie steht ganz am Anfang der Analyse. Daher ist zunächst die allgemeine Zielstellung und der vorgesehene Anwendungsbereich des zu entwickelnden Softwaresystems genauer abzugrenzen. Da hier betriebliche Anwendungssysteme im Vordergrund stehen, ist unter einem Anwendungsbereich ein Komplex von zusammenhängenden Aufgaben zu verstehen, für dessen Wahrnehmung Mitarbeiter bzw. Organisationseinheiten eines Unternehmens oder einer öffentlichen Verwaltung zuständig sind. Zu dem Aufgabenkomplex gehören insbesondere auch Aufgaben der Informationsverarbeitung.

**Zielstellung,  
Anwendungsbereich**

Allgemeine Zielstellung und Anwendungsbereich eines Softwaresystems ergeben sich grundsätzlich bereits aus dem ursprünglich geäußerten Bedarf oder im Verlauf erster Kontakte mit dem Auftraggeber, wobei spätere Erweiterungen oder Einschränkungen allerdings nicht auszuschließen sind. Sinnvoll ist eine explizite Abgrenzung des Anwendungsbereiches sowohl hinsichtlich der betroffenen Organisationseinheiten und Mitarbeiter wie auch anhand der in den Anwendungsbereich fallenden Aufgabengebiete.

Die Beschreibung des Anwendungsbereiches erfolgt unmittelbar aus der Sicht der betroffenen Organisationseinheiten und Mitarbeiter. Sie zielt auf eine präzise Feststellung der fachlich-inhaltlichen Gegebenheiten ab, die bei der Entwicklung einer neuen Anwendung zu berücksichtigen sind. Die in die Beschreibung einzubeziehenden Sachverhalte und Tatsachen hängen natürlich stark von der konkreten Situation ab. Einer groben Orientierung mögen folgende Stichworte dienen (vgl. z.B. OESTEREICH 1998):

**Beschreibung des  
Anwendungsbereiches**

- Aufbauorganisation innerhalb des Anwendungsbereichs,
- Mitarbeiter und ihre Aufgabengebiete,
- bisherige Arbeitsabläufe bzw. Anwendungsfälle des Vorgängersystems und deren bereits besprochene Aspekte (Arbeitsschritte, Standardfälle und Sonderfälle, auslösende Ereignisse, Vorbedingungen, benötigte und erzeugte Informationen usw.),

### gegenwärtige Informations- verarbeitung

- betroffene Personen wie etwa Kunden, benutzte Gegenstände und Arbeitsmittel, erzeugte Produkte und erbrachte Leistungen, Arbeitsorte,
- Mengengerüste, d.h. quantitative Angaben über Umfänge von Produkten, Leistungen, Informationen usw.

Die gegenwärtige Informationsverarbeitung im Anwendungsbereich wird vor allem im Hinblick auf die Aufdeckung von Schwachstellen untersucht. Sie sollte daher etwa anhand folgender Aspekte betrachtet werden:

- Differenzierung in computergestützt und manuell durchgeführte Aufgaben,
- Angaben zum verwendeten Computersystem, der benutzten System- und Anwendungssoftware und deren Funktionsumfang,
- gegebenenfalls formalisierte Darstellung der bisherigen Informationsverarbeitung im Anwendungsbereich z.B. mit dem Ziel, redundante Datenerfassungen zu erkennen,
- Beschreibung der verarbeiteten Daten sowie der zugehörigen Datenträger (Belege, Formulare, Dateien, Bildschirmmasken, Drucklisten usw.).

Natürlich wird man bei der Darstellung der derzeitigen Informationsverarbeitung möglichst auf die Dokumentation bisher eingesetzter Software zurückgreifen.

Neben der Aufdeckung von Schwachstellen bildet die Untersuchung der bisherigen Informationsverarbeitung eine Grundlage für folgende Entscheidungen:

- Kann der Bedarf des Auftraggebers durch eine Weiterentwicklung (Wartung) vorhandener Software befriedigt werden oder ist eine Neuentwicklung erforderlich?
- Können im Falle einer Neuentwicklung Entwicklungsdokumente bzw. Komponenten des alten Softwaresystems wiederverwendet werden?

Während die erste Frage bereits innerhalb der Voruntersuchung selbst entschieden werden sollte, kann die zweite Frage später beantwortet werden. Hier wird im Weiteren ausschließlich der Fall einer Neuentwicklung betrachtet.

Die Analyse der Schwachstellen der bisherigen Informationsverarbeitung gestattet, gezielte Anforderungen an die neue Anwendung zur Ausschaltung bisheriger Defizite festzulegen. In der Praxis der betrieblichen Informationsverarbeitung treten die unterschiedlichsten Schwachstellen auf. Einige typische Beispiele für Schwachstellen sind:

- fehlende Computerunterstützung für gewisse Arbeitsaufgaben,
- unvollständige oder verspätete Bereitstellung benötigter Informationen, fehlende Abfrage- und Auswertungsmöglichkeiten,
- redundante, d.h. mehrfache Erfassung von Daten,
- Medienbrüche, z.B. Wechsel von maschinell lesbaren und nicht maschinell lesbaren Datenträgern,
- veraltete, wenig komfortable Benutzungsoberflächen,
- fehleranfällige, unzuverlässige Programme, häufige Systemabstürze,
- wartungsunfreundliche, kaum noch erweiterbare Programme.

### Lösungskonzept

Das Lösungskonzept umreißt abschließend die Anforderungen an das zu entwickelnde Softwaresystem. Zu diesem frühen Zeitpunkt der Analyse können und sollen die Anforderungen nur so detailliert beschrieben werden, dass die Durch-

föhrbarkeit und Wirtschaftlichkeit der Anwendung – im Rahmen der anschlieöenden Durchführbarkeitsuntersuchung – geprüft werden kann.

Mit einer Skizze der vorgesehenen Anwendungsfälle des neuen Softwaresystems wird dessen Funktionalität umrissen. Hatte der Auftraggeber ursprünglich vielleicht nur einen Hauptzweck der gewünschten Anwendung benannt, so werden jetzt die wichtigsten Funktionen abgegrenzt. Die vorgesehenen Anwendungsfälle können hier noch informell, also umgangssprachlich und grob aufgeföhrt werden. Eine vollständige und formalisierte Darstellung der Anwendungsfälle bleibt dem Pflichtenheft vorbehalten. Den Ausgangspunkt für die Abgrenzung der Anwendungsfälle bilden die zuvor festgestellten Aufgaben und Arbeitsabläufe im Anwendungsbereich.

**vorgesehene  
Anwendungsfälle**

Weitere Anforderungen beziehen sich etwa auf:

- quantitative Leistungen wie z.B. maximale Dialogantwortzeiten und zu bewältigende Mengengerüste,
- die Produkt- bzw. Systemumgebung (Hardware und Systemsoftware),
- grundsätzliche Merkmale der Benutzungsoberfläche (z.B. vorgegebenes GUI-System, vgl. Kap. 5),
- die Art der Datenverwaltung (z.B. relationales Datenbanksystem).

**weitere  
Anforderungen**

## Übungsaufgaben zu Kapitel 3.3

### Übungsaufgabe 3.3.1

Ein Vorgängersystem soll durch ein neues Anwendungssystem abgelöst werden. Nennen Sie zwei Vorteile, die eine Ist-Analyse des alten Systems verspricht.

## 3.4 Projekt Tourenplanungssystem: Fachliche Vorstudie und Durchführbarkeitsstudie

Herr Schulze, Leiter der Abteilung Verkauf der Metallgroöhandelsfirma „Schraube und Partner“ wendet sich an die unternehmenseigene Abteilung Informationsverarbeitung. Im Gespräch mit Frau Hesse, Leiterin der Systemanalysegruppe, erläutert er den Bedarf nach einer verbesserten und erweiterten Computerunterstützung der Auftragsabwicklung in der Abteilung Verkauf. Im Kern gehe es darum,

- die bisher nur manuell durchgeföhrt Tourenplanung bei der Kundenbelieferung computergestützt abzuwickeln und
- für eine verbesserte Transparenz der Auftragsabwicklung durch geeignete periodische Auswertungen zu sorgen.

Zu prüfen sei ferner, ob das bereits vorhandene Softwaresystem lediglich zu erweitern oder ob eine Neuentwicklung erforderlich ist.

Man einigt sich, dass zunächst eine fachliche Vorstudie sowie eine Durchführbarkeitsstudie für ein entsprechendes Projekt erstellt werden sollen.



Die von Mitarbeitern der Gruppe Systemanalyse in der Abteilung Verkauf durchgeführte Voruntersuchung führt zu folgendem Ergebnis.

### **Vorstudie zum Projekt „Tourenplanungssystem“**

#### **Allgemeine Zielstellung und Anwendungsbereich**

#### **1 Allgemeine Zielstellung und Anwendungsbereich**

Das Handelsunternehmen „Schraube und Partner“ beliefert Kunden mit Metallwaren. Die Belieferung erfolgt mit eigenen Fahrzeugen von einem Depot aus. Dieses ist zugleich Warenlager- und Fahrzeugstandort. Die täglich eintreffenden Kundenaufträge sind computergestützt abzuwickeln und kostengünstig zu disponieren.

Der Anwendungsbereich umfasst die Abteilungsgruppe „Auftragsabwicklung und Tourenplanung“ der Abteilung Verkauf. Zur Abteilungsgruppe gehören zwei Mitarbeiter. Für das Aufgabengebiet „Auftragsabwicklung“ ist ein Sachbearbeiter, für das Teilgebiet „Tourenplanung“ ist ein Disponent verantwortlich.

#### **Beschreibung des Anwendungsbereiches**

#### **2 Beschreibung des Anwendungsbereiches**

##### **(2.1) Aufgabengebiet Auftragsabwicklung**

Durchzuführen sind folgende Aufgaben:

- (AA1) Verwaltung aller Stammdaten der Auftragsabwicklung,
- (AA2) Annahme von Kundenaufträgen,
- (AA3) Rechnungserstellung für durchgeführte Aufträge,
- (AA4) periodische Auswertung der Auftragsabwicklung.

##### **(2.2) Aufgabengebiet Tourenplanung**

Durchzuführen sind folgende Aufgaben:

- (TP1) Ermittlung kostengünstiger Touren für die pro Planungsperiode (ein Tag) zur Kundenbelieferung einzusetzenden Fahrzeuge,
- (TP2) Erstellung von Lieferscheinen und Fahrtanweisungen,
- (TP3) Kurzfristige Information der Kunden über Lieferdatum und Lieferzeit,
- (TP4) Einweisung der Lagermitarbeiter und Fahrer in die pro Planungsperiode durchzuführenden Lieferungen.

##### **(2.3) Kunden**

Das Unternehmen beliefert zurzeit etwa 500 Kunden. Pro Planungsperiode gehen etwa 50 bis 100 Aufträge ein. Die Kundenorte liegen bis zu 180 km vom Depot entfernt, wobei Häufungen in Großstadtbereichen auftreten.

Ein Kunde kann entweder Laufkunde oder Stammkunde sein. Ein Auftrag eines Laufkunden wird unmittelbar nach Auftragsabwicklung abgerechnet und vom Kunden bezahlt. Bei einem Stammkunden erfolgen hingegen Rechnungserstellung und Bezahlung zu Beginn des Folgemonats für alle Aufträge des Vormonats. Als Gegenleistung für den pro Monat quasi eingeräumten zinslosen Kredit verpflichtet sich ein Stammkunde dazu, ein individuell und wertmäßig festgelegtes Auftragsvolumen pro Monat nicht zu unterschreiten. Stammkunden können ferner Daueraufträge erteilen (vgl. 2.7). Ein Kunde kann seinen Kundenstatus in beiden Richtungen wechseln.

## (2.4) Artikel

Das Artikelsortiment besteht aus etwa 300 verschiedenen Metallwaren. Für jeden Artikel ist eine minimale Abgabemenge (Verkaufseinheit) festgelegt, auf die sich der Verkaufspreis bezieht. Zur Verpackung einer Verkaufseinheit eines Artikels dienen Kartons und Kisten unterschiedlicher Abmessungen. Die zu befördernden Metallwaren sind durchweg von relativ hoher Dichte. Die Kartons und Kisten besitzen stets kleinere Abmessungen, stellen also keine sperrigen Güter dar.

## (2.5) Fuhrpark

Der Fuhrpark umfasst acht Fahrzeuge gleicher Ladekapazität und gleicher technischer Ausstattung. Laut Auskunft der Firmenleitung soll der Fuhrpark auch künftig homogen gehalten werden. Die technischen Fahrzeugdaten gehen aus folgender Tab. 3.1 hervor.

Wegen der kleinen Abmessungen und der hohen Dichte der Metallwaren stellt die volumenmäßige Begrenzung des Laderaums keine gesonderte Restriktion dar. Überschreitet das Gewicht einer zu verstauenden Gütermenge die vorgegebene Nutzlast nicht, können die Güter auch volumenmäßig im Laderaum des Fahrzeugs verstaut werden.

Techn. Fahrzeuggröße	Wert
Nutzlast [t]	2,8
Gesamtgewicht [t]	6
Laderaumtiefe [cm]	500
Laderaumbreite [cm]	235
Laderaumhöhe [cm]	220
Laderaumvolumen [m <sup>3</sup> ]	25,85
Hebebühne	nein

**Tab. 3.1.** Technische Fahrzeuggrößen.

Der Einsatz der Fahrzeuge verursacht fixe und variable Kosten. Sie sind nach Kostenarten gegliedert in den beiden folgenden Tab. 3.2 und Tab. 3.3 angegeben:

Fixkostenart (Angaben in Euro/Monat)	Wert
Abschreibung	582,88
Kalkulatorische Zinsen	71,54
Steuern	50,08
Versicherungen	151,94
Löhne / Sozialabgaben	2081,72
Sonstiges	40,88
Fixkosten [Euro/Monat]	2979,03
Fixkosten [Euro/Tag]	99,30

**Tab. 3.2.** Angaben zu Fixkosten des Fuhrparks.

Variable Kostenart (Angaben in Euro/km)	Wert
Kraftstoffe	0,106
Schmierstoffe	0,008
Verschleißreparaturen	0,041
Reifen	0,008
Sonstiges	0,004
Variable Kosten [Euro/km]	0,168

**Tab. 3.3.** Angaben zu variablen Kosten des Fuhrparks.

Für die Fahrtgeschwindigkeiten aller Fahrzeuge gelten einheitlich folgende Daten der Tab. 3.4:

Durchschnittsgeschwindigkeit [km/h]	55
Fahrzeitaufschlag für Nebel [%]	30
Fahrzeitaufschlag für Schnee [%]	50
Fahrzeitaufschlag für Glatteis [%]	70

**Tab. 3.4.** Angaben zu Geschwindigkeiten.

#### (2.6) Fahrer

Jedem Fahrzeug ist ein Fahrer fest zugeordnet. Die anfallenden Personalkosten sind in der Fixkostentabelle erfasst. Bei der Tourenplanung sind die gesetzlichen Vorschriften zu maximalen Lenkzeiten und Pausenzeiten für die Fahrer zu beachten.

#### (2.7) Auftrag

Zu unterscheiden sind einmalig und periodisch abzuwickelnde Aufträge, die als Einzel- bzw. Daueraufträge bezeichnet werden. Letztere kommen nur für Stammkunden in Betracht.

Ein Einzelauftrag umfasst einen oder mehrere Artikel und pro Artikel eine oder mehrere Verkaufseinheiten. Die mit einem Einzelauftrag bestellte Gütermenge ist stets vollständig an den Kunden auszuliefern, d.h. ein Splitting von Einzelaufträgen kommt nicht in Betracht.

Ein Dauerauftrag umfasst über die Angaben eines Einzelauftrages hinaus eine Vereinbarung, an welchen Wochentagen, z.B. am Montag und am Donnerstag, der Auftrag abzuwickeln ist. Solange ein Dauerauftrag nicht widerrufen bzw. gelöscht wurde, wird er an den festgelegten Tagen wie ein Einzelauftrag durchgeführt.

Die Aufgabengebiete und zugehörigen Arbeitsabläufe werden bei der Betrachtung der derzeitigen Informationsverarbeitung näher beschrieben.

### 3 Gegenwärtige Informationsverarbeitung

#### 3.1 Erhebung

##### (3.1) Computerunterstützte und manuell durchgeführte Aufgaben

Derzeit werden folgende Arbeitsaufgaben computerunterstützt abgewickelt:

- (AA1) Stammdatenverwaltung,
- (AA2) Auftragsannahme,
- (AA3) Rechnungserstellung,
- (AA4) periodische Auswertung der Auftragsabwicklung,
- (TP2) Erstellung von Lieferscheinen und Fahrtanweisungen.

Die Tourenplanung (TP1) wird bisher ausschließlich manuell durchgeführt, während für die übrigen Aufgaben (TP3, TP4) eine Computerunterstützung nicht in Betracht kommt.

### (3.2) Angaben zum Computersystem und der benutzten Software

Sämtliche computerunterstützten Arbeiten werden an einem Personal Computer (PC) mit einem Bildschirmarbeitsplatz sowie einem angeschlossenen Drucker erledigt. Als Betriebssystem wird Windows 95 verwendet. Das bisherige Softwaresystem ist ein COBOL-Programm, das im Einbenutzerbetrieb (single user) genutzt wird und eine zeilenorientierte Benutzungsoberfläche besitzt. Die Anwendungsdaten (Stamm-, Auftrags- und Tourenplanungsdaten) werden mit einem Dateiverwaltungssystem verwaltet.

Im Folgenden werden die bisherigen Arbeitsabläufe beschrieben. Deren formalisierte Darstellung – etwa als Anwendungsfälle – erweist sich im Rahmen der Vorstudie als entbehrlich. Auf eine nähere Beschreibung der Anwendungsdaten sei an dieser Stelle ebenfalls verzichtet.

### (3.3) Aufgabe (AA1): Stammdatenverwaltung

Die Stammdaten der Auftragsabwicklung beinhalten:

- die Artikeldaten,
- die Fuhrparkdaten,
- die Kundendaten,
- das Entfernungswerk.

Die Stammdatenverwaltung umfasst die üblichen Verwaltungsoperationen (Erfassen, Ändern, Löschen und Ausgeben). Sie erfolgt fallweise bzw. bei Bedarf.

Das Artikelsortiment wird in regelmäßigen Abständen aktualisiert. Das neue Artikelsortiment wird dann als gedruckter Katalog (Liste) an alle Kunden verschickt.

Aufträge eines Kunden können nur bearbeitet werden, nachdem sich ein Kunde angemeldet und eine Kundennummer erhalten hat, mit der er sich künftig bei der Auftragserteilung ausweist. Mit der Aufnahme eines Kunden wird zugleich das Entfernungswerk erweitert, in welchem die gegenseitigen Entfernungen der Kunden unter Berücksichtigung aller Adressbestandteile (einschließlich Straße) erfasst werden.

### (3.4) Aufgabe (AA2): Annahme von Kundenaufträgen

Kundenaufträge gehen telefonisch oder per E-Mail ein und werden arbeitstäglich von 6.00 bis 8.00 Uhr erfasst. Ein Kunde kann pro Tag auch mehrere Aufträge erteilen. Neben den täglich eintreffenden Einzelaufträgen sind gegebenenfalls auch die Daueraufträge von Stammkunden zu berücksichtigen. Der Kunde gibt neben den Bestelldaten seinen Namen und die Kundennummer an. Gegebenenfalls werden die Auftragsdaten korrigiert (Kunden-Nr. nicht „im System“, Kun-

den-Nr. und -name nicht konsistent, bestellte Artikel nicht im Sortiment usw.). Hierzu können auch telefonische Rückfragen bei Kunden erforderlich sein. Die pro Auftrag erfassten Daten werden durch Daten aus dem Kundenstamm und dem Artikelstamm ergänzt und in der Auftragsdatei abgelegt. Nach Erfassung und Korrektur aller Aufträge wird eine aktuelle Auftragsliste erstellt. Jeder Auftrag sollte möglichst am Annahmetag ausgeführt werden. Jedoch kann die Auftragsliste auch noch nicht erledigte Aufträge der Vortage umfassen.

#### (3.5) Aufgabe (AA3): Rechnungserstellung für durchgeführte Aufträge

Ab 9.00 Uhr werden am Vortag ausgeführte Kundenaufträge abgerechnet. Grundlage der Abrechnung sind vom Kunden quittierte Lieferscheine, die die Fahrer nach Tourende abgeben. Für Laufkunden werden Rechnungen erstellt und verschickt. Bei Stammkunden wird eine Monatsrechnung üblicherweise am ersten Montag des Folgemonats erstellt und verschickt. Abgerechnete Aufträge werden in der Auftragsdatei entsprechend gekennzeichnet. Kopien aller Rechnungen gehen an die Buchhaltung zur Prüfung der Zahlungseingänge.

#### (3.6) Aufgabe (AA4): periodische Auswertung der Auftragsabwicklung

Bisher wird nur eine Tagesauswertung der durchgeführten Belieferungen vorgenommen. Diese umfasst Angaben wie beförderte Mengen, gefahrene Kilometer und angefallene Kosten der Belieferung. Außerdem prüft der Sachbearbeiter manuell und in größeren Abständen, ob seitens der Stammkunden die vorgesehenen monatlichen wertmäßigen Auftragsvolumina eingehalten werden.

#### (3.7) Aufgabe (TP1): Ermittlung von Touren für eine Planungsperiode

Die Tourenplanung erfolgt von 8.00 bis 9.00 Uhr. Anhand der aktuellen Auftragsliste werden zuerst die abzuwickelnden Touren manuell festgelegt. Eine Tour wird durch eine Auswahl von Kunden bzw. Aufträgen sowie die Reihenfolge der zu bedienenden Kunden bestimmt. Dann werden die ermittelten Touren den einzusetzenden Fahrzeugen zugeordnet. Einem Fahrzeug können auch mehrere Touren zugewiesen werden.

#### (3.8) Aufgabe (TP2): Erstellung von Lieferscheinen und Fahrtanweisungen

Im Anschluss an die Tourenplanung werden die festgelegten Touren eingegeben und terminiert, d.h. mit Zeitangaben wie Abfahrtszeiten, Ankunftszeiten bei den Kunden usw. versehen. Die für die Terminierung benötigten Entfernungsdaten werden dem Entfernungswerk entnommen. Außerdem wird jede terminierte Tour um Auftrags- und Fahrzeugdaten aus den betreffenden Stammdateien ergänzt und dann in der Tourenplandatei abgelegt.

Während ein Lieferschein u.a. die pro Fahrzeug und Auftrag beförderten Güter enthält, beschreibt eine Fahrtanweisung den Ablauf einer Tour. Disponierte Aufträge werden in der Auftragsdatei entsprechend gekennzeichnet.

#### (3.9) Aufgabe (TP3): Kurzfristige Information der Kunden

Die heute zu beliefernden Kunden werden telefonisch über das Lieferdatum und die voraussichtliche Lieferzeit informiert.

**(3.10) Aufgabe (TP4): Einweisung der Lagermitarbeiter und Fahrer**

Im Anschluss werden die Fahrer und Lagermitarbeiter in die im Tagesablauf abzuwickelnden Aufträge anhand der Fahrtanweisungen und Lieferscheine eingewiesen. Die weitere Auftragsabwicklung umfasst die Beladung der Fahrzeuge und die Auslieferung.

**3.2 Schwachstellen**

Die derzeitige Informationsverarbeitung weist vor allem folgende Schwachstellen auf, die teils schon bei der Projektinitialisierung benannt wurden:

- die manuelle Tourenplanung ist zeitaufwendig und führt bei größeren täglichen Auftragsanzahlen nicht zu qualitativ hochwertigen Tourenplänen, folglich zu überhöhten Lieferkosten;
- die zum Teil ebenfalls manuelle Auswertung der Auftragsabwicklung ist mühsam und unzureichend; so weist die nur lückenhafte Auswertung der Tourenplanung derzeit z.B. keine Terminverzögerungen bei der Belieferung aus, durch deren Abstellung die Kundenzufriedenheit verbessert werden könnte;
- die Benutzungsoberfläche ist unkomfortabel und veraltet;
- künftig ist eine Vernetzung der betrieblichen DV mittels eines PC-LAN vorgesehen; das zur Datenhaltung benutzte Dateiverwaltungssystem gestattet dann nicht, eine einheitliche und gemeinsam benutzbare Datenbasis für die Auftragsabwicklung und die Buchhaltung einzurichten;
- das bisher benutzte COBOL-Programm ist sehr schlecht wartbar, zumal der Autor nicht mehr Mitarbeiter der Firma ist.

Nach den Ergebnissen der Schwachstellenanalyse ist nur eine Neuentwicklung sinnvoll. Hierfür können die Beschreibungen der bisherigen Dateien genutzt werden.

**4 Lösungskonzept****4.1 Funktionalität und vorgesehene Aufgaben**

Für die neue Anwendung werden die folgenden bereits abgegrenzten Aufgaben vorgesehen:

- (AA1) Stammdatenverwaltung,
- (AA2) Auftragsannahme,
- (AA3) Rechnungserstellung,
- (AA4) periodische Auswertung der Auftragsabwicklung,
- (TP1) Ermittlung von Tourenplänen für eine Planungsperiode,
- (TP2) Erstellung von zugehörigen Lieferscheinen und Fahrtanweisungen.

Die Aufgaben (AA1) bis (AA3) sind durch die bisherige Beschreibung im Rahmen der Vorstudie bereits hinreichend definiert. Die Aufgaben (AA4), (TP1) und (TP2) werden wie folgt umgestaltet:

Aufgabe (AA4):

- vorzusehen ist zum einen eine automatisierte quartalsweise Auswertung der Tourenplanung; zu berücksichtigen sind hierbei einerseits Aufwandsgrößen (transportierte Mengen, gefahrene Kilometer, sämtliche für die Belieferung an-

**Schwachstellen****Lösungskonzept****Funktionalität und vorgesehene Aufgaben**



gefallene Kosten), andererseits Ertragsgrößen (wertmäßiges Volumen der gelieferten Aufträge); schließlich sind auch aufgetretene Verzögerungen bei der Auftragsabwicklung in die Auswertung einzubeziehen;

- vorzusehen ist ferner eine automatisierte quartalsweise Auswertung des Kundenverhaltens, die den kundenbezogenen Umfang der Auftragserteilung ausweisen soll; geht es bei Stammkunden vor allem darum, die Einhaltung des minimalen monatlichen Auftragsvolumens zu prüfen, so soll die Auswertung der Laufkunden eine Basis für gezielte Marketing-Maßnahmen bilden.

Aufgaben (TP1) und (TP2):

- die Tagestouren werden computergestützt geplant; zugrunde gelegt werden jeweils alle noch offenen Aufträge; zu beachten sind sowohl die Ladekapazität wie auch die maximale Tourdauer (maximale Lenkzeit der Fahrer); die Zuordnung der Touren zu den verfügbaren Fahrzeugen wird ebenfalls systemseitig unterstützt („Zuordnung am Bildschirm“); entsprechende Entscheidungen werden aber durch den Disponenten vorgenommen; dabei wird die Einhaltung zeitlicher Einschränkungen (maximale Lenkzeit, Pausenzeiten zwischen zwei Touren) automatisch geprüft;
- für die Terminierung der Touren sind lediglich Abfahrtszeiten vom Depot einzugeben; alle anderen Zeitangaben werden systemseitig ermittelt.

#### 4.2 Weitere Anforderungen

Folgende weitere Anforderungen werden an das System gestellt:

- Abwicklung von bis zu 200 Aufträgen pro Tag, Berechnung von Tourenplänen innerhalb von ein bis zwei Minuten;
- Einbenutzersystem mit grafischer Windows-Benutzeroberfläche, Umstellung der Datenverwaltung auf ein relationales Datenbanksystem.

weitere  
Anforderungen

Auf der Grundlage der Vorstudie erfolgt eine Durchführbarkeitsuntersuchung. Diese kommt zu dem Ergebnis, dass das Projekt technisch und personell realisierbar ist. Als Alternative wird der Einkauf einer Standardsoftware für die Tourenplanung erwogen. Diese Variante wird jedoch verworfen, weil keine Software im Angebot ist, welche die betriebsspezifische Auftragsabwicklung (Stammkunden mit monatlicher Abrechnung, Daueraufträge) sinnvoll berücksichtigt. Die absehbaren ökonomischen Vorteile sprechen eindeutig für die Wirtschaftlichkeit des Projekts. Es wird ein Auftrag für die Durchführung des Projekts „Tourenplanungssystem“ erteilt.

### Übungsaufgaben zu Kapitel 3.4

#### Übungsaufgabe 3.4.1 (Projekt Bibliotheksausleihsystem)

Für eine Universitätsbibliothek soll ein neues Informationssystem erstellt werden, das ein unzureichendes Vorgängersystem ablöst. Im Rahmen dieser und weiterer Übungsaufgaben sollen für das neue Bibliotheksausleihsystem (kurz BAS) alle Schritte einer objektorientierten Analyse durchlaufen werden. Zunächst ist in Anlehnung an die Vorstudie des Tourenplanungssystems eine fachliche Vorstudie für das BAS zu erarbeiten. Erstellen Sie die Vorstudie auf der Grundlage der folgen-

den Informationssammlung. Treffen Sie zusätzlich sinnvolle Annahmen etwa über das abzulösende Altsystem.

*Die Bibliothek wird von Studenten und Mitarbeitern der Universität genutzt. Jeder Benutzer wird durch eine Identifikationsnummer (ID) identifiziert und besitzt ein Konto, in dem die aktuell ausgeliehenen und vorgemerkten Medien gespeichert werden.*

*Die Bibliothek verwaltet Bücher und Zeitschriften. Bücher werden durch den Titel, Autor, Verlag, Jahrgang und eine ID beschrieben. Zeitschriften haben keinen Autor, dafür aber eine Ausgabennummer. Häufig ausgeliehene Medien werden im Lesesaal ausgestellt und können unmittelbar ausgeliehen werden. Selten verwendete Medien werden im Magazin gelagert und müssen für eine Ausleihe erst bestellt und von einem Mitarbeiter der Bibliothek in den Lesesaal gebracht werden. Wenn das Medium dann nach zwei Tagen nicht abgeholt wurde, wird es in das Magazin zurückgebracht.*

*Ausgeliehene Medien können von einem anderen Benutzer vorgemerkt werden. Nach der Rückgabe eines vorgemerkten Mediums, verbleibt es generell für mindestens 10 Tage im Lesesaal, bis es ggf. ins Magazin zurückgebracht wird.*

*Ein Benutzer kann ein Medium bestellen, ausleihen, vormerken lassen, zurückgeben und die Vormerkung aufheben. Bei der Ausleihe wird das Ausleihdatum gespeichert. Die Ausleihfrist beträgt 4 Wochen für Studenten und 3 Monate für Mitarbeiter. In einem Abstand von 14 Tagen werden die Benutzerkonten in einem Batch-Job überprüft. Wenn das späteste Rückgabedatum mehr als 7 Tage überschritten ist, erfolgt eine Mahnung (schriftlich bei Studenten, telefonisch bei Mitarbeitern).*

### 3.5 Aufbau eines Pflichtenheftes

Das Pflichtenheft dokumentiert die Gesamtheit der Anforderungen an eine gewünschte Anwendung aus der Sicht des Auftraggebers. Da es im Allgemeinen die verbindliche Grundlage der weiteren Anwendungsentwicklung darstellt, müssen spätere Änderungen am Pflichtenheft stets mit dem Auftraggeber abgestimmt werden.

**Pflichtenheft**

Die zu berücksichtigenden Anforderungen sind verschiedener Natur. So können etwa fachliche und technische, ferner funktionale und nicht funktionale Anforderungen differenziert werden. Die Abb. 3.5 zeigt ein Gliederungsschema eines Pflichtenheftes, das anschließend näher erläutert wird.

Die globalen Produktziele beschreiben in knapper Form die mit dem Produkt zu erreichenden allgemeinen Ziele des Auftraggebers. Dabei sind die unbedingt zu realisierenden Funktionen der Anwendung zu benennen. Zur Abgrenzung des Projekts trägt bei, wenn zusätzlich angegeben wird, welche naheliegenden weiteren Funktionen im Rahmen des Projekts nicht realisiert werden sollen.

**Produktziele**

Festlegungen zum Produkteinsatz betreffen den Anwendungsbereich des Produkts, die Benutzergruppen sowie die Betriebsbedingungen der Anwendung. Die Festlegung der Benutzergruppen umfasst auch Aspekte wie vorausgesetzte Bedienungskenntnisse sowie die Rechte der Benutzergruppen. Die Betriebsbedingungen

**Produkteinsatz**

bestimmen vor allem die Betriebsformen sowie weitere Bedingungen wie z.B. die tägliche Einsatzdauer. Übliche Betriebsformen stellen u.a. die Dialog- oder Batchverarbeitung, der Einbenutzer- (single user) oder Mehrbenutzerbetrieb (multi user) dar.

#### Schema

1. Globale Produktziele
2. Produkteinsatz
  - 2.1 Anwendungsbereich
  - 2.2 Benutzergruppen
  - 2.3 Betriebsbedingungen
3. Produktumgebung
  - 3.1 Hardware
  - 3.2 Systemsoftware
  - 3.3 Orgware
  - 3.4 Schnittstellen zu weiteren Anwendungen
4. Produktfunktionen
  - 4.1 Übersicht der Produktfunktionen
  - 4.2 Anwendungsfälle
5. Produktdaten
6. Produktleistungen
7. Qualitäts- und nichtfunktionale Anforderungen
8. Benutzungsoberfläche
9. Entwicklungsumgebung
10. Abnahme- und Einführungsfestlegungen
11. Projektrahmenfestlegungen
12. Anhang

**Abb. 3.5.** Aufbau eines Pflichtenheftes.

#### Produktumgebung

Die Produktumgebung beschreibt zum einen die Zielplattform, d.h. die Hardware und die Systemsoftware, auf und mit der das Produkt genutzt werden soll. Dabei kann es sich auch um verschiedene Plattformen handeln. Hierzu gehört auch, ob die Anwendung als sequentielles, auf einem Prozessor ablaufendes System oder etwa als verteiltes System in einem Computer-Netzwerk zu realisieren ist. Die Produktumgebung umfasst weiterhin die Orgware, d.h. erforderliche organisatorische und technische Randbedingungen für den Produkteinsatz wie z.B. elektronische Post (E-Mail). Schließlich gehören zur Produktumgebung gegebenenfalls Schnittstellen zu weiteren Anwendungssystemen, etwa einer Finanzbuchhaltung, mit denen das entwickelte System kooperieren soll.

#### Produktfunktionen

Die Produktfunktionen geben die mit dem Produkt zu bewältigenden komplexeren Teilaufgaben und zusammenhängenden Arbeitsabläufe an. Diese werden hier zum einen überblicksartig in einem Anwendungsfalldiagramm sowie als Funktionsbaum dargestellt. Ferner werden die einzelnen Anwendungsfälle mittels der oben vorgestellten Anwendungsfallschablone beschrieben.

#### Produktdaten

Die Produktdaten beschreiben die dauerhaft von der Anwendung zu speichernden Daten (z.B. Kundendaten, Artikeldaten). Hierzu kann ein Datenlexikon (Data Dictionary) angelegt werden, das jedoch für die einzelnen Datenarten jeweils nur die

wichtigsten Merkmale umfassen sollte (z.B.: Kundendaten = Kunden-Nr. + Kundenname + Kundenadresse usw.). Infrage kommt auch eine tabellarische Darstellung der zu verwaltenden Daten.

Produktleistungen geben die quantitativen Anforderungen an das System an. Darunter fallen Mengengerüste (z.B. Verwaltung von maximal 20.000 Kunden), zeitliche Anforderungen (z.B. maximale Dialogantwortzeiten) sowie Genauigkeitsanforderungen an zu berechnende Größen (z.B. Angabe auf drei Dezimalstellen genau).

**Produktleistungen**

Qualitätsanforderungen werden oft hinsichtlich der Zuverlässigkeit der Anwendung vorgegeben wie eine auf einen gewissen Zeitraum bezogene maximale Fehlerrate. Neben Qualitätsanforderungen, deren Einhaltung – wie in dem eben angegebenen Beispiel – gemessen und überprüft werden kann, können hier auch zu beachtende Qualitätsschwerpunkte wie etwa die Forderung nach einer besonders bedienungsfreundlichen Benutzungsoberfläche oder einer wartungsfreundlichen Anwendung aufgenommen werden. Weitere nicht funktionale Anforderungen können sich etwa auf die Unterstützung des internationalen Produkteinsatzes, z.B. die Berücksichtigung verschiedener Landeswährungen, beziehen.

**Qualitätsanforderungen**

**nicht funktionale Anforderungen**

Unter dem Gliederungspunkt Benutzungsoberfläche werden produktbezogene spezifische Anforderungen an die Benutzungsoberfläche spezifiziert wie etwa das Bildschirm- und Drucklistenlayout oder Tastaturbelegungen. Ferner wird ggf. das zu verwendende GUI-System (vgl. Kap. 5) festgelegt.

**Benutzungsoberfläche**

Die Entwicklungsumgebung spezifiziert analog zur Produktumgebung die Hardware, Systemsoftware, Orgware und Schnittstellen zu anderen Systemen während der Softwareentwicklung.

**Entwicklungsumgebung**

Die Abnahme- und Einföhrungsfestlegungen enthalten bereits detailliert festgelegte Testfälle für die Produktabnahme oder spezifizieren diese zumindest grob. Die Testfälle orientieren sich an den oben erläuterten Produktfunktionen. Weitere Festlegungen betreffen z.B. Dokumentations- und Installationsrichtlinien oder notwendige Schulungsmaßnahmen für die Benutzer.

**Abnahme- und Einföhrungsfestlegungen**

Die Projektrahmenfestlegungen dokumentieren Rahmenbedingungen der Realisierung des Entwicklungsvorhabens. Dazu gehören die geplante Entwicklungsdauer, Termine und Kosten. Diese Angaben werden innerhalb der Durchführbarkeitsuntersuchung (vgl. Kap. 1.7.1) sowie in Verhandlungen mit dem Auftraggeber ermittelt bzw. festgelegt.

**Projektrahmenfestlegungen**

Im Anhang können ein Glossar relevanter Begriffe, Verweise auf relevante Normen und Standards sowie Verweise auf weitere Dokumente erscheinen.

**Anhang**

Eine abschließende Bemerkung bezieht sich auf die Punkte Produktumgebung bzw. Entwicklungsumgebung und weitere technische Festlegungen des Pflichtenheftes. Der Mindestumfang, in dem diesbezügliche Festlegungen im Pflichtenheft zu vereinbaren sind, ergibt sich aus den vom Auftraggeber zum Zeitpunkt der Erstellung des Pflichtenheftes vorgegebenen Anforderungen. Alle dann noch ausstehenden technischen Realisierungsentscheidungen lassen sich auf den Beginn des Entwurfs verschieben.

## Übungsaufgaben zu Kapitel 3.5

### Übungsaufgabe 3.5.1

Grenzen Sie die Dialog- und die Batchverarbeitung voneinander ab. Wozu setzt man die Batchverarbeitung ein? Wie können Sie bei der Analyse eines vorhandenen Systems Batchverarbeitungen identifizieren?

### Übungsaufgabe 3.5.2

Für die Darstellung der langfristig zu speichernden bzw. wesentlichen Anwendungsdaten im Pflichtenheft kann bereits ein Data Dictionary benutzt werden. In diesem werden die Anwendungsdaten in ihre Bestandteile gegliedert. Allerdings sind Datenstrukturen in vielen Fällen nicht einfach additiv aus Datenelementen, d.h. atomaren Elementen oder einfacheren Datenstrukturen, zusammengesetzt. Können Sie einige Varianten nennen, die bei der Zusammensetzung von Datenstrukturen aus Datenelementen vorkommen?

## 3.6 Projekt Tourenplanungssystem: Pflichtenheft

Auf der Grundlage der Vorstudie wird folgendes Pflichtenheft erstellt.

### Pflichtenheft zum Projekt „Tourenplanungssystem“

#### 1 Globale Ziele

Mit dem Projekt „Tourenplanungssystem“ soll ein Software-System für eine Neugestaltung der Auftragsabwicklung mit integrierter Tourenplanung (kurz: Tourenplanungssystem) in der Abteilung Verkauf entwickelt werden. Hiermit werden folgende allgemeine Ziele verbunden:

- Reduzierung der variablen Kosten der Kundenbelieferung durch eine qualitativ verbesserte Tourenplanung,
- Reduzierung des personellen Aufwands der Tourenplanung,
- größere Kostentransparenz der Auftragsabwicklung durch periodische Auswertung der anfallenden Kosten für die Belieferung,
- Unterstützung von Kontroll- und Marketingmaßnahmen durch periodische Auswertung der Auftragserteilung der Kunden,
- Unterstützung der künftigen Integration der betrieblichen Datenverarbeitung durch Umstellung der Datenverwaltung auf ein relationales Datenbanksystem.

Das Tourenplanungssystem sollte unbedingt folgende Funktionen abdecken:

- Verwaltung aller Stammdaten der Auftragsabwicklung,
- Erfassung, Verwaltung und Abrechnung von Aufträgen,
- Automatisierte Tourenplanung inkl. Erzeugung von Begleitdokumenten,
- periodische Auswertung der Tourenplanung und der Auftragserteilung.

#### Globale Ziele

## 2 Produkteinsatz

### 2.1 Anwendungsbereich

Der Anwendungsbereich umfasst die Abteilungsgruppe „Auftragsabwicklung und Tourenplanung“ der Abteilung Verkauf mit den Aufgabengebieten „Auftragsabwicklung“ und „Tourenplanung“.

### 2.2 Benutzergruppen

Vorgesehene Benutzer des Tourenplanungssystems sind:

- Sachbearbeiter Auftragsabwicklung,
- Disponent Tourenplanung.

Beide Benutzer verfügen über gute Kenntnisse in der Benutzung des alten Auftragsystems und gängiger Arbeitsplatzsoftware des Bürobereichs.

Beide Benutzer sollen alle Rechte erhalten, um die gegenseitige Vertretung zu erleichtern.

### 2.3 Betriebsbedingungen

Das Tourenplanungssystem wird als Einbenutzersystem ausgelegt und ausschließlich tagsüber im Dialogbetrieb genutzt.

## 3 Produktumgebung

### 3.1 Hardware

- Handelsüblicher PC,
- Prozessor mit mindestens 200 MHz Taktfrequenz, 16 MB Hauptspeicher und 1 Gigabyte Plattenspeicher,
- 1 Laserdrucker.

### 3.2 Systemsoftware

- Windows 95 oder Nachfolger oder Windows NT 4.0,
- MS Access oder vergleichbares relationales Datenbanksystem (DBS) mit ODBC-Schnittstelle.

### 3.3 Orgware

- Telefon mit Anrufbeantworter, E-Mail-System.

### 3.4 Schnittstellen zu weiteren Anwendungen

- Zunächst keine.
- Nach der Einrichtung eines PC-LAN soll die Datenbank des Tourenplanungssystems von der Buchhaltung mitbenutzt werden. Vgl. Punkt 5, Produktdaten.

## 4 Produktfunktionen

### 4.1 Übersicht der Produktfunktionen

Nachfolgend werden die Funktionen des Tourenplanungssystems im Überblick

- als Funktionsbaum sowie
- als Anwendungsfalldiagramm

dargestellt (vgl. Abb. 3.6 und Abb. 3.7).

**Produkteinsatz**

**Anwendungsbereich**

**Benutzergruppen**

**Betriebsbedingungen**

**Produktumgebung**

**Hardware**

**Systemsoftware**

**Orgware**

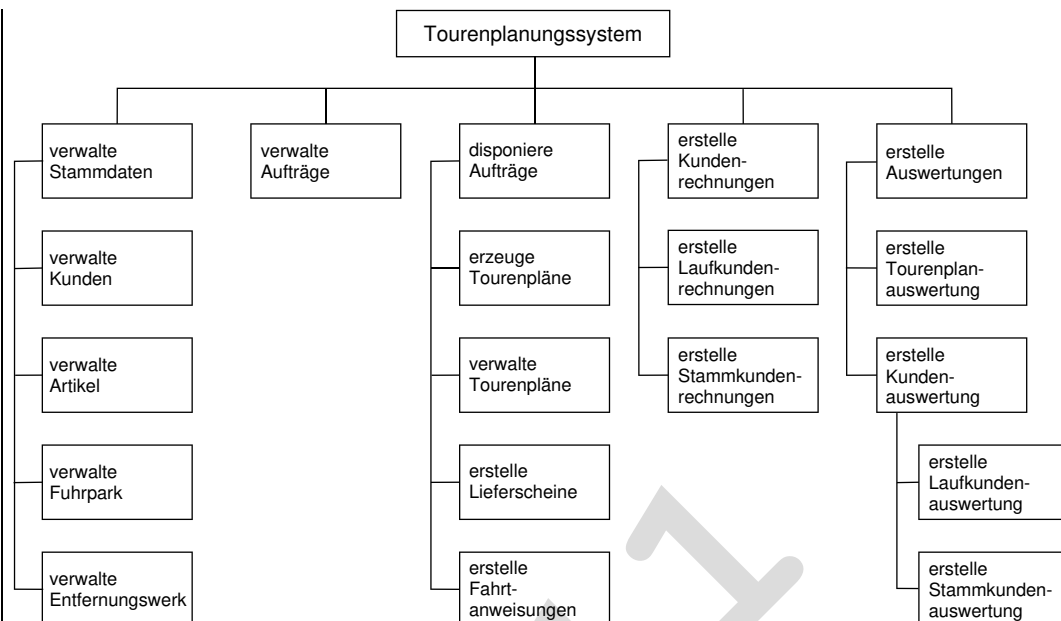
**Schnittstellen zu  
weiteren  
Anwendungen**

**Produktfunktionen**

**Übersicht der  
Produktfunktionen**



## Funktionsbaum

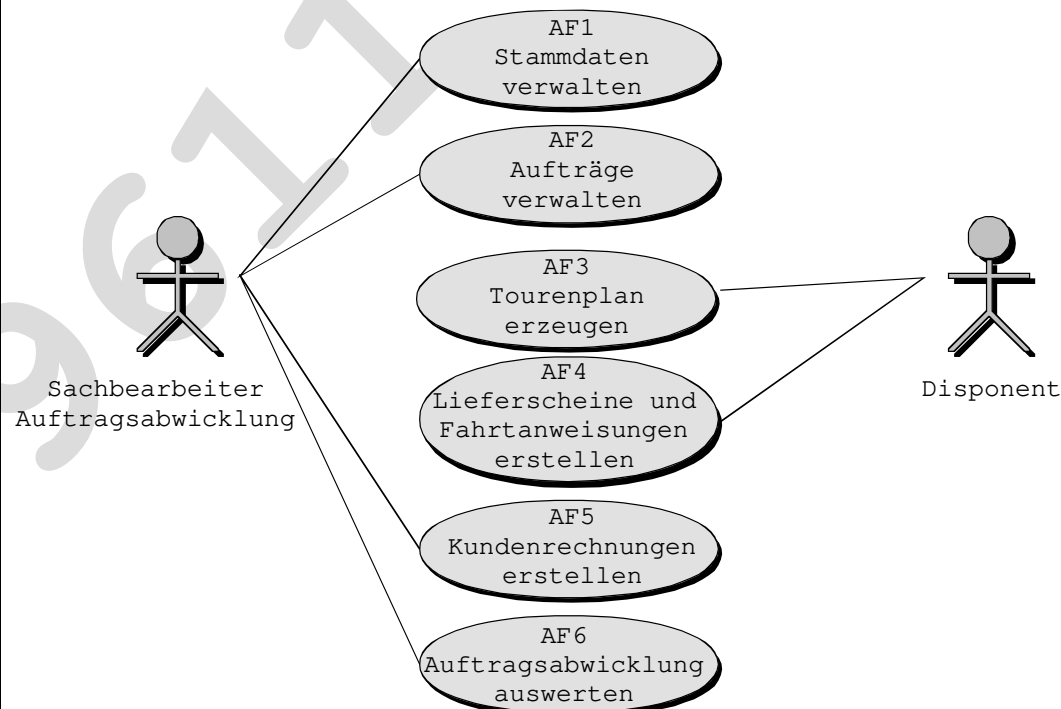


Anmerkungen:

- Die Verwaltungsfunktionen für Stammdaten, Aufträge und Tourenpläne umfassen jeweils übliche Verwaltungsfunktionen für das Erfassen, Ändern, Löschen einzelner Objekte sowie die Ausgabe (Anzeige/Druck) einzelner Objekte bzw. von Objektlisten.
- Auf eine weitere Verfeinerung wird daher verzichtet.

**Abb. 3.6.** Funktionsbaum für das Tourenplanungssystem.

## Anwendungsfall-diagramm



**Abb. 3.7.** Anwendungsfalldiagramm für das Tourenplanungssystem.

## 4.2 Anwendungsfälle

### Anwendungsfälle

Nachfolgend werden die in Abb. 3.7 vorkommenden Anwendungsfälle im Einzelnen beschrieben.

#### AF1, Stammdaten verwalten

**Anwendungsfall: AF1, Stammdaten verwalten****Ziel:**

Die Stammdaten sollen auf aktuellem Stand sein. Zu den Stammdaten gehören:

- die Artikeldaten,
- die Kundendaten,
- die Fuhrparkdaten,
- die Entfernungswerkdaten.

**Akteure:**

Sachbearbeiter Auftragsabwicklung.

**Kategorie:**

Primär.

**Initiierendes Ereignis:**

Jegliche Änderung der Stammdaten.

Änderungen umfassen hinzukommende, veränderte oder entfallende Daten der oben genannten Stammdatenarten. Hingewiesen sei insbesondere auf die notwendige Erweiterung bzw. Änderung des Entfernungswerks bei hinzukommenden Kunden bzw. veränderten Kundenadressen.

**Vorbedingung:**

–

**Nachbedingung bei Erfolg:**

Aktualisierte Stammdaten.

**Nachbedingung bei Misserfolg:**

–

**Standardspezifikation:**

- 1 Der Sachbearbeiter überprüft die Vollständigkeit und Widerspruchsfreiheit der neuen Stammdaten.
- 2 Der Sachbearbeiter gibt die neuen bzw. geänderten Stammdaten ein oder löscht ggf. Stammdaten.

**Erweiterungen der Standardspezifikation:**

- 1a Der Sachbearbeiter hält bei fehlenden oder widersprüchlichen Daten telefonisch Rücksprache mit geeigneten Ansprechpartnern (u.a. Kunden, Vertrieb).

**Alternativen zur Standardspezifikation:**

- 1a Der Sachbearbeiter gibt die Artikelliste aus und versendet diese an Kunden.
- 1b Der Sachbearbeiter gibt die Kundenliste zu Kontrollzwecken aus.

**AF2, Aufträge  
verwalten**

**Anwendungsfall: AF2, Aufträge verwalten****Ziel:**

Auftragsdaten sollen auf aktuellem Stand sein. Insbesondere sollen eintreffende Kundenaufträge für die Auftragsabwicklung erfasst werden.

**Akteure:**

Sachbearbeiter Auftragsabwicklung.

**Kategorie:**

Primär.

**Initiierendes Ereignis:**

Neuer Kundenauftrag ist telefonisch oder per E-Mail eingegangen oder Kundenauftrag wurde geändert bzw. storniert.

**Vorbedingung:**

Der Kunde hat bereits eine Kundennummer.

**Nachbedingung bei Erfolg:**

Aktualisierte Auftragsdaten. Aufträge erhalten Bearbeitungsstand „angenommen“.

**Nachbedingung bei Misserfolg:**

–

**Standardspezifikation:**

- 1 Der Sachbearbeiter prüft die Kundendaten.
- 2 Der Sachbearbeiter überprüft, ob die bestellten Artikel im Sortiment sind.
- 3 Der Sachbearbeiter erfasst den Auftrag in der Auftragsdatei.

**Erweiterungen der Standardspezifikation:**

- 1a Neukunden erfassen.
- 1b Die vorliegenden Auftragsdaten sind nicht vollständig. Daher muss der Sachbearbeiter die fehlenden Informationen beim Kunden telefonisch erfragen.
- 2a Es wird festgestellt, dass der bestellte Artikel derzeit nicht im Sortiment ist. Der Sachbearbeiter schlägt dem Kunden telefonisch Alternativen vor.

**Alternativen zur Standardspezifikation:**

- 1a Änderung der Daten eines Auftrags.
- 1b Stornierung eines Auftrags.
- 1c Ausgabe einer Auftragsliste zu Kontrollzwecken.

**Anwendungsfall: AF3, Tourenplan erzeugen****Ziel:**

Tourenplan für tägliche Belieferung von Kunden mit angenommenen Aufträgen.

**Akteure:**

Disponent.

**Kategorie:**

Primär.

**Initiierendes Ereignis:**

Regelmäßig, arbeitstäglich ca. 8.00 Uhr.

**Vorbedingung:**

Einsetzbare Fahrzeuge sind vorhanden. Es wurden alle Kundenaufträge für den Planungstag eingegeben.

**Nachbedingung bei Erfolg:**

Die Kundenaufträge für den Planungstag sind ganz oder teilweise disponiert und erhalten den Bearbeitungsstand „geliefert“.

**Nachbedingung bei Misserfolg:**

–

**Standardspezifikation:**

- 1 Der Disponent gibt die Planungsdaten (Lieferdatum, witterungsabhängige Fahrzeuggeschwindigkeit) ein und initiiert die Tourenplanung.
- 2 Der Disponent komplettiert den generierten Tourenplan:
  - er ordnet den einzelnen Touren Fahrzeuge zu und
  - er legt die Startzeiten der Touren fest.

**Erweiterungen der Standardspezifikation:**

–

**Alternativen zur Standardspezifikation:**

–

**Anwendungsfall: AF4, Lieferscheine und Fahrtanweisungen erstellen****Ziel:**

Lieferscheine und Fahrtanweisungen als Begleitdokumente für die tägliche Auftragsbelieferung.

**Akteure:**

Disponent.

**Kategorie:**

Primär.

**Initiierendes Ereignis:**

Regelmäßig, arbeitstäglich ca. 8.15 Uhr.

AF4, Lieferscheine  
und Fahrtanweisungen  
erstellen

**Vorbedingung:**

Ein oder mehrere Tourenpläne für die tägliche Auftragsbelieferung wurden erzeugt.

**Nachbedingung bei Erfolg:**

Disposition der täglichen Auftragsbelieferung ist komplett durchgeführt.

**Nachbedingung bei Misserfolg:**

–

**Standardspezifikation:**

- 1 Der Disponent wählt die für den Arbeitstag erzeugten Tourenpläne aus.
- 2 Der Disponent initiiert die Erstellung der Lieferscheine und druckt diese aus.
- 3 Der Disponent initiiert die Erstellung der Fahrtanweisungen und druckt diese aus.
- 4 Der Disponent prüft, ob alle Tourenpläne des Arbeitstages berücksichtigt wurden.

**Erweiterungen der Standardspezifikation:**

–

**Alternativen zur Standardspezifikation:**

–

AF5,  
Kundenrechnungen  
erstellen

**Anwendungsfall: AF5, Kundenrechnungen erstellen****Ziel:**

Rechnungen für einzelne gelieferte Aufträge für Laufkunden;  
Rechnungen für im Vormonat gelieferte Aufträge für Stammkunden.

**Akteure:**

Sachbearbeiter Auftragsabwicklung.

**Kategorie:**

Primär.

**Initiierendes Ereignis:**

Regelmäßig, arbeitstäglich ca. 9.00 Uhr.

**Vorbedingung:**

Von Kunden quitierte Lieferscheine liegen vor.

**Nachbedingung bei Erfolg:**

Aufträge erhalten Bearbeitungsstand „abgerechnet“.

**Nachbedingung bei Misserfolg:**

–

**Standardspezifikation:**

- 1 Sachbearbeiter erstellt und druckt Rechnungen für die an Laufkunden gelieferten Aufträge.
- 2 Sachbearbeiter verschickt die Rechnungen an die Laufkunden.
- 3 Sachbearbeiter gibt Kopien der Laufkundenrechnungen an die Buchhaltung.

**Erweiterungen der Standardspezifikation:**

- 1a Sachbearbeiter erstellt und druckt zu Monatsbeginn Rechnungen für die an Stammkunden im Vormonat gelieferten Aufträge.
- 2a Sachbearbeiter verschickt die Rechnungen an die Stammkunden.
- 3a Sachbearbeiter gibt Kopien der Stammkundenrechnungen an die Buchhaltung.

**Alternativen zur Standardspezifikation:**

- 1a Einzelne bereits disponierte Aufträge konnten nicht ausgeliefert werden. Protokollierung der näheren Umstände. Die betreffenden Aufträge erhalten erneut den Bearbeitungsstand „angenommen“.

**Anwendungsfall: AF6, Auftragsabwicklung auswerten**

**AF6,  
Auftragsabwicklung  
auswerten**

**Ziel:**

- Quartalsauswertung des Aufwands und der Kosten der Tourenplanung sowie des Auftragsumsatzes.
- Quartalsauswertung der Auftragserteilung der Kunden.

**Akteure:**

Sachbearbeiter Auftragsabwicklung.

**Kategorie:**

Sekundär.

**Initiierendes Ereignis:**

Regelmäßig zu Quartalsbeginn für das abgelaufene Quartal.

**Vorbedingung:**

Auswertungsquartal ist abgelaufen.

**Nachbedingung bei Erfolg:**

Oben genannte Auswertungen liegen als Listen vor.

**Nachbedingung bei Misserfolg:**

—

**Standardspezifikation:**

- 1 Der Sachbearbeiter gibt das Auswertungsquartal ein.
- 2 Der Sachbearbeiter initiiert die Auswertung der Tourenplanung und der Auftragserteilung der Laufkunden sowie der Stammkunden.



- 3 Der Sachbearbeiter überprüft anhand der Auswertung der Auftragserteilung, ob die Stammkunden die vereinbarten monatlichen Mindestvolumina einhalten und versendet ggf. Mahnschreiben an Stammkunden.

#### Erweiterungen der Standardspezifikation:

- 3a Der Disponent schlägt Maßnahmen zur Verbesserung der Auftragsabwicklung und zur Werbung neuer Stammkunden vor.

#### Alternativen zur Standardspezifikation:

–

### 5 Produktdaten

#### Produktdaten

Das Tourenplanungssystem verwaltet dauerhaft folgende Daten:

- Stammdaten, darunter:  
    Artikeldaten, Fahrzeugdaten, Kundendaten, Entfernungswerkdaten,
- Auftragsdaten,
- Tourenplandaten.

Zur Verwaltung der Daten ist ein relationales DBS zu nutzen (vgl. 3.2).

Die Auftragsdaten sind derart zu gestalten, dass die Kontrolle der Zahlungseingänge durch die Buchhaltung bereits datenseitig unterstützt wird (vgl. 3.4).

Aufbewahrungszeiträume und Archivierung der Daten sind noch zu vereinbaren.

Die dauerhaft zu verwaltenden Daten werden in der folgenden Tab. 3.5 näher beschrieben.

Datenart	Zu verwaltende Merkmale
Kunden	Kunden-Nr., Firmenname, Kundenadresse, Kundentyp (Lauf- oder Stammkunde)
Artikel	Artikel-Nr., Artikelbezeichnung, Verkaufseinheit, Preis pro Verkaufseinheit, Gewicht pro Verkaufseinheit
Fuhrpark	max. Nutzlast, Durchschnittsgeschwindigkeit bei normaler Witterung, Nebel, Schnee und Glatteis, Fixkosten pro Monat, variable Kosten pro km, max. Lenkzeit, Pausenzeit zwischen zwei Touren, Anzahl Fahrzeuge, pro Fahrzeug: Kfz-Nr.
Entfernungswerk	Entfernung zwischen je zwei Kundenorten (gegeben durch vollständige Kundenadresse einschl. Straße) sowie Entfernungen aller Kundenorte zum Depot
Auftrag	Auftrags-Nr., Kunden-Nr., Auftragswert, Auftragsgewicht, Anzahl Auftragspositionen; pro Auftragsposition: Artikel-Nr. und Anzahl bestellter Verkaufseinheiten, Auftragsstyp (Einzel- oder Dauerauftrag); bei Einzelaufträgen zusätzlich: Bearbeitungsstand des Auftrags und Datumsangaben zu einzelnen Bearbeitungsständen, z.B. Annahmedatum; bei Daueraufträgen zusätzlich: Wochentage der Auftragsausführung
Tourenplan	Anzahl Touren, Lieferdatum, pro Tour: Kfz-Nr., Strecke, Startzeit Depot, Ankunftszeit Depot, Tourfrachtwert, Tourfrachtgewicht, Tourroute unter Angabe aller zu beliefernden Kunden und auszuliefernden Aufträge in Lieferreihenfolge

**Tab. 3.5.** Dauerhaft zu verwaltende Daten des Tourenplanungssystems.

## 6 Produktleistungen

- Verwaltung von ca. 300 Artikeln,
- Verwaltung von ca. 500 Kunden, entsprechender Umfang des Entfernungswerks,
- Erfassung von Entfernungen in km mit 2 Nachkommastellen, d.h. bis auf 0,01 km genau, Ermittlung von Tourstrecken mit gleicher Genauigkeit,
- Abwicklung von bis zu 200 Aufträgen pro Tag,
- Erzeugung eines Tourenplans innerhalb von ein bis zwei Minuten,
- sonstige Dialogantwortzeiten im Sekundenbereich.

**Produktleistungen**

## 7 Qualitäts- und nichtfunktionale Anforderungen

- Bedienungsfreundliche komfortable Benutzungsoberfläche,
- leichte Erweiterbarkeit bzw. Änderbarkeit.

**Qualitäts- und nichtfunktionale Anforderungen**

## 8 Benutzungsoberfläche

- Graphische Benutzungsoberfläche basierend auf GUI-System von Windows 95.

**Benutzungsoberfläche**

## 9 Entwicklungsumgebung

- Handelsüblicher PC, Windows 95, DBS MS Access.

**Entwicklungsumgebung**

## 10 Abnahme und Einführungsfestlegungen

- Testfälle für den Abnahmetest:
  - je ein Testfall pro Stammdatenart, wobei jeweils alle Verwaltungsoperationen zu testen sind (Erfassen, Ändern, Löschen, Anzeige bzw. Ausdruck),
  - ein Testfall für die Auftragsverwaltung analog zu Stammdatentestfällen unter Berücksichtigung von Einzel- und Daueraufträgen,
  - ein Testfall für Erzeugung von Tourenplänen und Begleitdokumenten, Prüfung der Zeitrestriktion bei ca. 200 Aufträgen sowie der gedruckten Begleitdokumente,
  - je ein Testfall für die Rechnungserstellung für Lauf- und Stammkunden, Prüfung der gedruckten Rechnungen,
  - je ein Testfall für die Auswertung der Tourenplanung und der Auftragserteilung durch Lauf- und Stammkunden, Prüfung der Listenausdrucke.
- Schulung: halbtägige Benutzerschulung.
- Dokumentation: Benutzerhandbuch und Systemdokumentation gemäß Firmenstandard.

**Abnahme und Einführungs-festlegungen**

## 11 Projektrahmenfestlegungen

- Zunächst keine.

**Projektrahmen-festlegungen**

## 12 Anhang

- Verweise:
  - (1) Fachliche Vorstudie zum Projekt „Tourenplanung“,
  - (2) Durchführbarkeitsstudie zum Projekt „Tourenplanung“,

**Anhang**

- (3) Dokument „Modell und Lösungsverfahren für das Tourenplanungsproblem“;
- (4) Firmenstandard zur Dokumentation von DV-Anwendungen.
- Hinweis: Auf die unter (1) bis (3) genannten Dokumente kann kurz unter dem Stichwort „Pflichtenheft“ verwiesen werden.

## Übungsaufgaben zu Kapitel 3.6

### Übungsaufgabe 3.6.1 (Projekt Bibliotheksausleihsystem)

Erstellen Sie auf der Grundlage der Musterlösung der Aufgabe 3.4.1, d.h. der fachlichen Vorstudie für das BAS auf der CD-ROM, ein Pflichtenheft für das Bibliotheksausleihsystem. Achten Sie besonders auf die Spezifizierung der Anwendungsfälle und der Anwendungsdaten. Treffen Sie ggf. sinnvolle zusätzliche Annahmen.

## 3.7 Behandlung integrierter betrieblicher Entscheidungsprobleme

**quantitativ-mathematisches Problem**

Oftmals beinhaltet die Entwicklung eines betrieblichen Anwendungssystems auch die Lösung eines komplexen quantitativ-mathematischen Problems. Dabei kann es sich beispielsweise um ein Optimierungsproblem, ein Prognoseproblem oder ein Simulationsproblem handeln.

**betriebliche Entscheidungsprobleme**

In diesem Kurs kann die Behandlung quantitativ-mathematischer Probleme im Rahmen der Entwicklung betrieblicher Anwendungen natürlich nur skizziert werden. Für weitergehende Ausführungen wird auf die einschlägige Literatur des Operations Research verwiesen, etwa auf DOMSCHKE und DREXL (1995). Hier werden betriebliche Entscheidungsprobleme unterstellt, denen Optimierungsprobleme zugrunde liegen. Ein Optimierungsproblem liegt bekanntlich vor, wenn für eine oder mehrere unabhängige Variablen einer (mathematischen) Funktion Werte derart zu bestimmen sind, dass die Funktion einen minimalen oder einem maximalen Wert annimmt. Fast immer sind zusätzliche Nebenbedingungen zu beachten, welche die Mengen der zulässigen Werte der Variablen einschränken.

**Behandlung – ab wann?**

Ein integriertes komplexeres mathematisches Problem sollte bereits bei der Erstellung der fachlichen Vorstudie bzw. des Pflichtenheftes berücksichtigt werden. Einerseits deshalb, weil derartige Probleme die Schwierigkeit, den Gesamtaufwand und den Ablauf des Projekts maßgeblich beeinflussen können. Ferner setzt eine fachliche Modellierung der Anwendung voraus, dass das Lösungsverfahren für das jeweilige mathematische Problem bereits spezifiziert wurde.

**Behandlung, zwei Schritte**

Die Behandlung eines integrierten mathematischen Problems sollte in zwei Schritten erfolgen. In einem ersten Schritt ist eine präzise quantitative Problemformulierung zu erarbeiten, die als Modellierung oder Quantifizierung bezeichnet wird. Im zweiten Schritt ist das Lösungsverfahren auszuwählen und zu spezifizieren. Beide Schritte seien nachfolgend etwas näher betrachtet.

**Modellierung**

Eine quantitative Problemformulierung kann durch die modellanalytische Vorgehensweise gewonnen werden. Gemäß dem Konzept der Modellanalyse bietet sich im Fall eines Optimierungsproblems eine Quantifizierung in folgenden Schritten an:

- (1) Präzisierung der Problembeschreibung.
- (2) Formulierung der Modellprämissen.
- (3) Definition der Modellgrößen.
- (4) Mathematische Formulierung der Zielfunktion und der Restriktionen.

Bei der Auswahl eines Lösungsverfahrens ist zu beachten, dass zur Lösung von Optimierungsproblemen grundsätzlich optimierende und suboptimierende Verfahren in Frage kommen. Sofern Lösungen überhaupt existieren, ermitteln optimierende Verfahren stets (exakt-)optimale Lösungen. Suboptimierende Verfahren führen dagegen zu mehr oder minder guten Lösungen und zumindest für konkrete Problemfälle größeren Umfangs kaum zu optimalen Lösungen.

**Verfahrensauswahl**

In der Regel ist die Ermittlung von optimalen Lösungen mit einem höheren Rechenaufwand verbunden als die Erzeugung suboptimaler Lösungen. Für eine umfangreiche Klasse von Optimierungsproblemen – die sogenannten NP-harten Probleme – wächst die erforderliche Rechenzeit exponentiell mit der Problemgröße. Daher ist die Anwendung exakter Verfahren auf größere Problemfälle mit einem nicht mehr tolerierbaren Rechenaufwand verbunden. Doch auch für suboptimierende Verfahren stellt sich meist die Situation eines Trade-off („Schere“) zwischen Lösungsqualität und Lösungsaufwand.

Welchen Verfahrenstyp und welches Verfahren man in einem konkreten Fall wählt und ob man ein Verfahren selbst entwickelt oder ob man Problemlösungssoftware erwirbt, hängt u.a. von folgenden Faktoren ab:

- Verfügbarkeit, Kosten und Eigenschaften kommerzieller Problemlösungssoftware.
- Verfügbarkeit des für die Eigenentwicklung eines Verfahrens erforderlichen Knowhow.
- Erforderliche Qualität erzeugter Problemlösungen.
- Erwünschtes Rechenzeitverhalten in Abhängigkeit von der Problemgröße.

**Auswahl-Faktoren**

Die weitere Vorgehensweise ist durch die genannten Faktoren bedingt. Bei einem Zukauf von Problemlösungssoftware sind vor allem deren Schnittstellen im weiteren Verlauf der Analyse und nachfolgenden Entwicklungsphasen zu berücksichtigen. Ferner sollte das Leistungsverhalten der zugekauften Software frühzeitig experimentell überprüft werden.

Nach einer Entscheidung für eine Eigenentwicklung ist das ausgewählte Lösungsverfahren genauer zu spezifizieren. Hierbei wird man oft auch auf einschlägige Literatur zurückgreifen und Standardvarianten des Lösungsverfahrens an vorliegende Bedürfnisse und Besonderheiten anpassen. Die Verfahrensspezifikation kann etwa in folgenden Schritten erfolgen:

**Verfahrensspezifikation**

- (1) Beschreibung der Lösungsidee.
- (2) Verbale Beschreibung des Verfahrens.
- (3) Formalisierte Beschreibung des Verfahrens als Flussdiagramm oder Struktogramm oder mit Hilfe eines Pseudocodes.

Bei kritischen Leistungsanforderungen hinsichtlich der Lösungsqualität oder der Rechenzeit ist es ratsam, das Lösungsverfahren möglichst bald als Prototyp zu realisieren und anhand praxisnaher Fallbeispiele zu erproben.

## Übungsaufgaben zu Kapitel 3.7

### Übungsaufgabe 3.7.1

Betriebliche Entscheidungsprobleme sind oft innerhalb von operativen Dispositionssystemen oder innerhalb von Systemen für längerfristige Planungen zu lösen (vgl. Kap. 1). Nennen Sie für beide Systemarten je ein typisches zu bearbeitendes Entscheidungsproblem.

## 3.8 Projekt Tourenplanungssystem: Modell und Lösungsverfahren für das betrachtete Tourenplanungsproblem

Im Folgenden wird das im Rahmen der Anwendung zu lösende Tourenplanungsproblem zunächst modelliert. Anschließend wird ein Lösungsverfahren spezifiziert.

Modell und  
Lösungsverfahren

Modellierung

### Modell und Lösungsverfahren für das Tourenplanungsproblem

#### 1 Modellierung

##### (1) Präzisierung der Problemstellung

Das betrachtete Tourenplanungsproblem stellt ein Auslieferungsproblem bei einem Depot dar. Zunächst ist zu klären, unter welcher Zielsetzung und unter welchen Restriktionen dieses Problem zu lösen ist.

Als Zielkriterien kommen grundsätzlich in Frage:

- (1) Die Zeitminimierung, d.h. die Minimierung der gesamten Einsatzzeit der eingesetzten Fahrzeuge.
- (2) Die Entfernungsminimierung, d.h. die Minimierung der von den eingesetzten Fahrzeugen insgesamt zurückgelegten Wegstrecke.
- (3) Die Kostenminimierung, d.h. die Minimierung der durch die eingesetzten Fahrzeuge insgesamt verursachten Kosten.

Zur Güterbeförderung werden nur eigene Fahrzeuge eingesetzt. Daher sind nur die variablen Kosten entscheidungsrelevant und die beiden letztgenannten Zielsetzungen sind äquivalent. Vernachlässigt man fahrzeugabhängige Entladezeiten bei den Kunden und geht man zusätzlich von wege- und fahrzeugunabhängigen Durchschnittsgeschwindigkeiten aus, so sind alle drei Zielsetzungen äquivalent. Hier sei die Minimierung der Entfernung als Zielsetzung unterstellt.

Auch bei der Formulierung der Restriktionen existieren in Abhängigkeit von der Realitätsnähe der Modellierung gewisse Freiheitsgrade. Gefordert werde hier die Einhaltung der folgenden Restriktionen:

- (R1) Ein Splitten der auszuliefernden Sendungen ist nicht gestattet. Erteilt ein Kunde nur einen Auftrag, so muss er genau einmal angefahren werden, vgl. Prämisse (P6).
- (R2) Die Fahrt eines jeden Fahrzeugs beginnt und endet am Depot; es sind also geschlossene Touren zu planen.
- (R3) Subtouren, die das Depot nicht enthalten, sind nicht zulässig.
- (R4) Die Ladekapazitäten der Fahrzeuge sind sowohl gewichtsmäßig als auch volumenmäßig beschränkt. Im vorliegenden Fall genügt jedoch die Berücksichtigung einer Gewichtsrestriktion.
- (R5) Die maximale Einsatzzeit der Fahrzeuge ist beschränkt. Die entsprechenden gesetzlichen Vorschriften sind zu beachten. Bezogen auf einen einzelnen Tourenplan bedeutet dies, dass die Tourdauer eine vorgegebene Obergrenze nicht überschreiten darf.

In der Praxis können noch weitere Restriktionen von Bedeutung sein, u.a.:

- weitere kundenbezogene Restriktionen wie etwa Belieferungszeitfenster,
- artikelbezogene Restriktionen wie das Verbot des gemeinsamen Transports von giftigen Gütern und Lebensmitteln,
- Umweltrestriktionen wie im Winter nicht von allen Fahrzeugen befahrbare Zufahrtswege.

## (2) Formulierung von Modellprämissen

Modellprämissen sind Annahmen über betrachtete reale Zusammenhänge. Sie werden u.a. eingeführt, um Modelle möglichst einfach und übersichtlich zu halten. Im vorliegenden Fall seien folgende Prämissen unterstellt:

- (P1) Der Fuhrpark ist homogen. Daher weisen alle Fahrzeuge die gleiche maximale Nutzlast auf.
- (P2) Der Fuhrpark ist unbeschränkt, d.h. er umfasst beliebig viele Fahrzeuge.
- (P3) Die Fahrzeiten der Fahrzeuge lassen sich hinreichend genau über eine für alle Fahrzeuge als gleich anzunehmende Durchschnittsgeschwindigkeit ermitteln.
- (P4) Die durch Belade- oder Entladevorgänge bedingten Verweilzeiten bei Kunden sind vernachlässigbar klein; außerdem fallen keine Wartezeiten an.
- (P5) Weitere kundenbezogene, artikelbezogene und umweltbezogene Restriktionen treten nicht auf.
- (P6) Pro Kunde liegt nur ein Auftrag vor.
- (P7) Das Gewicht eines Auftrages überschreitet nicht die maximale Nutzlast.

Bemerkt sei, dass die Prämissen (P2) und (P6) bei der späteren Auswahl bzw. Gestaltung eines Lösungsverfahrens revidiert werden müssen.

## (3) Definition von Modellgrößen

Neben Größen wie Variablen, Konstanten und Parametern enthält ein mathematisches Modell in der Regel auch Indexgrößen. Im vorliegenden Fall werden folgende Indizes benötigt:

- $i$  - Index zur Bezeichnung von Kunden,  $i = 1, \dots, NK$ ;  
 $i = 0$  bezeichnet das Depot.



- $j$  - Index zur Bezeichnung von Kunden,  $j = 1, \dots, NK$ ;  
 $j = 0$  bezeichnet das Depot.
- $k$  - Index zur Bezeichnung von Touren,  $k = 1, \dots, NT$ .

Für die angegebenen Indexobergrenzen gilt:

- $NK$  - Anzahl der in einer Planungsperiode zu beliefernden Kunden.
- $NT$  - Anzahl der Touren eines Tourenplans.

Man beachte, dass  $NK$  vorgegeben ist, während  $NT$  eine Entscheidungsvariable darstellt. In das Modell gehen ferner folgende Größen ein:

- $Bedarf(i)$  - Bedarf des  $i$ -ten Kunden in Kilogramm,  $i = 1, \dots, NK$ .
- $DGeschw$  - Durchschnittsgeschwindigkeit eines Fahrzeugs in Kilometern pro Stunde (km/h).
- $Entf(i,j)$  - Entfernung zwischen dem  $i$ -ten und  $j$ -ten Kunden in Kilometern,  $i = 0, \dots, NK, j = 0, \dots, NK$ . Für  $i = j$  gilt stets  $Entf(i,j) = 0$ .
- $Fahrzeit(i,j)$  - Fahrzeit zwischen dem  $i$ -ten und  $j$ -ten Kunden,  $i = 0, \dots, NK, j = 0, \dots, NK$ , in Stunden. Für  $i = j$  gilt stets  $Fahrzeit(i,j) = 0$ .
- $Ladekap$  - Ladekapazität eines Fahrzeugs in Kilogramm.
- $MaxTourZeit$  - Maximale Tourdauer in Stunden.
- $GesStrecke$  - Gesamtstrecke eines Tourenplans in Kilometern.
- $u(i)$  - Ganzzahlige nichtnegative Hilfsvariablen für Restriktion (8),  $i = 1, \dots, NK$ .

Aus der Entfernung und aus der Durchschnittsgeschwindigkeit ermittelt sich die Fahrzeit zwischen dem  $i$ -ten und dem  $j$ -ten Kunden wie folgt:

$$Fahrzeit(i,j) = Entf(i,j) / DGeschw, \quad i \neq j, \quad i = 0, \dots, NK, \quad j = 0, \dots, NK.$$

Benötigt werden außerdem Entscheidungsvariablen  $x(i, j, k)$ ,  $i = 0, \dots, NK, j = 0, \dots, NK, k = 1, \dots, NT$ :

$$x(i, j, k) = \begin{cases} 1, & \text{falls } i \neq j \text{ und das auf der } k\text{-ten Tour eingesetzte Fahrzeug vom } \\ & i\text{-ten zum } j\text{-ten Kunden fährt,} \\ 0, & \text{andernfalls.} \end{cases}$$

#### (4) Mathematische Modellformulierung

Zu minimieren ist die Gesamtstrecke  $GesStrecke$  eines Tourenplans. Das Zielkriterium lautet also:

$$\text{Min } GesStrecke = \sum_{i=0}^{NK} \sum_{j=0}^{NK} \sum_{k=1}^{NT} Entf(i,j) \cdot x(i, j, k). \quad (1)$$

Insgesamt, d.h. über alle Touren, wird jeder Kundenort genau einmal angefahren, formal ausgedrückt:

$$\sum_{i=0}^{NK} \sum_{k=1}^{NT} x(i, j, k) = 1, \quad j = 1, \dots, NK. \quad (2)$$

Jeder angefahrene Kundenort muss auch wieder verlassen werden:

$$\sum_{i=0}^{NK} x(i, j, k) - \sum_{i=0}^{NK} x(j, i, k) = 0, \quad j = 1, \dots, NK, \quad k = 1, \dots, NT. \quad (3)$$

Jede Tour muss am Depot beginnen und enden sowie mindestens einen Kunden enthalten:

$$\sum_{j=1}^{NK} x(0, j, k) = 1, \quad k = 1, \dots, NT. \quad (4)$$

$$\sum_{i=1}^{NK} x(i, 0, k) = 1, \quad k = 1, \dots, NT. \quad (5)$$

Pro Tour darf die Summe der Gewichte der auszuliefernden Sendungen die Ladekapazität des eingesetzten Fahrzeugs nicht überschreiten; es muss also gelten:

$$\sum_{i=0}^{NK} \sum_{j=1}^{NK} \text{Bedarf}(j) \cdot x(i, j, k) \leq \text{Ladekap}, \quad k = 1, \dots, NT. \quad (6)$$

Ebenso darf die Einsatzzeit eines Fahrzeugs die maximale Tourzeit nicht überschreiten:

$$\sum_{i=0}^{NK} \sum_{j=0}^{NK} \text{Fahrzeit}(i, j) \cdot x(i, j, k) \leq \text{MaxTourZeit}, \quad k = 1, \dots, NT. \quad (7)$$

Schließlich müssen Subtouren, die nicht das Depot einschließen, ausgeschlossen werden. Dies kann durch folgende Modellrestriktion geschehen:

Es existieren ganzzahlige, nichtnegative Werte  $u(i)$ ,  $i = 1, \dots, NK$ , derart, dass

$$u(i) - u(j) + NK \cdot \sum_{k=1}^{NT} x(i, j, k) \leq NK - 1, \quad i = 1, \dots, NK, \quad j = 1, \dots, NK. \quad (8)$$

Mit der Formulierung des Zielkriteriums (1) und der Restriktionen (2) bis (8) ist die Problemquantifizierung abgeschlossen. Als Ergebnis liegt ein ganzzahliges Optimierungsmodell vor. Bezüglich der Restriktion (8) sei der interessierte Leser auf RÖSCHER (1993) verwiesen.

Im Anschluss an die Modellierung des Tourenplanungsproblems kann nun ein Lösungsverfahren ausgewählt und spezifiziert werden.

## 2 Auswahl und Spezifikation eines Lösungsverfahrens

### 2.1 Vorbemerkungen

Das vorliegende Tourenplanungsproblem gehört zu den NP-harten Problemen. Daher scheiden exakt-optimierende Verfahren aufgrund der zu erwartenden Problemgrößen und des gewünschten Rechenzeitverhaltens von vornherein aus.

Ein in der Praxis weit verbreitetes suboptimierendes Verfahren für das Ein-Depot-Tourenplanungsproblem ist das sogenannte Savingsverfahren. Es wird auch hier zur Lösung der beschriebenen Problemstellung gewählt. In seiner ursprünglichen, auf CLARK und WRIGHT (1964) zurückgehenden Fassung geht das Savingsverfahren von Transportmitteln gleicher Kapazität aus. Dies entspricht der vorliegenden

**Auswahl und  
Spezifikation eines  
Lösungsverfahrens**

**Vorbemerkungen**

Situation eines homogenen Fuhrparks. Außerdem berücksichtigt es keine Tourdauerrestriktionen und geht von einer unbegrenzten Fahrzeugzahl aus. Da hier eine Tourdauerrestriktion zu beachten ist und Fahrzeuge nur in begrenzter Zahl zur Verfügung stehen, muss das ursprüngliche Verfahren diesbezüglich angepasst werden. Zu beachten ist ferner, dass pro Kunde mehrere Aufträge vorliegen können. Jedoch wird zunächst weiterhin von der Modellprämisse ausgegangen, dass nur ein Auftrag pro Kunde vorliegt; vgl. aber Abschnitt 2.2, (4). Unterstellt werden nachfolgend ferner symmetrische Entfernungen, d.h. die Entfernung zwischen zwei Kunden ist richtungsunabhängig.

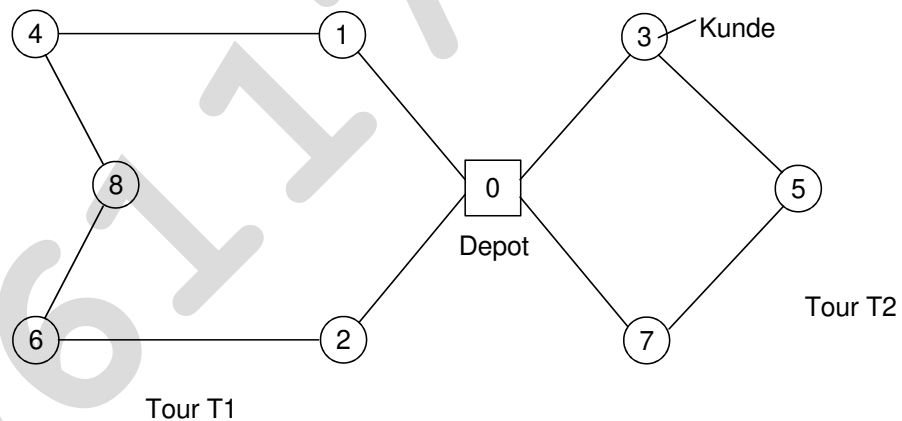
Bevor das Savingsverfahren im Einzelnen vorgestellt wird, sei eine formale Schreibweise für einige Begriffe eingeführt. Zur Demonstration diene das in Abb. 3.8 gezeigte Beispiel eines Tourenplans.

Bezeichne  $T1$  die erste Tour und  $T2$  die zweite Tour des in Abb. 3.8 dargestellten Tourenplans  $TP$ . Dann lässt sich  $TP$  als Menge von Touren wie folgt angeben:

$$TP = \{T1, T2\}.$$

Die Touren  $T1$  und  $T2$  stellen Mengen von zu beliefernden Kunden dar; laut Abb. 3.8 gilt:

$$T1 = \{1,2,4,6,8\} \text{ und } T2 = \{3,5,7\}.$$



**Abb. 3.8.** Beispiel für einen einfachen Tourenplan.

Im Unterschied zu einer Tour beschreibt eine Route die Reihenfolge, in der die Kunden einer Tour zu beliefern sind. Zu den Touren  $T1$  und  $T2$  gehören demnach folgende Routen  $R1$  und  $R2$ :

$$R1 = \{0,2,6,8,4,1,0\} \text{ und } R2 = \{0,3,5,7,0\}.$$

Als vollständige Wegdarstellungen enthalten  $R1$  und  $R2$  auch das Depot und zwar einmal als Ausgangspunkt und einmal als Endpunkt.

## 2.2 Savingsverfahren für das zu lösende Tourenplanungsproblem

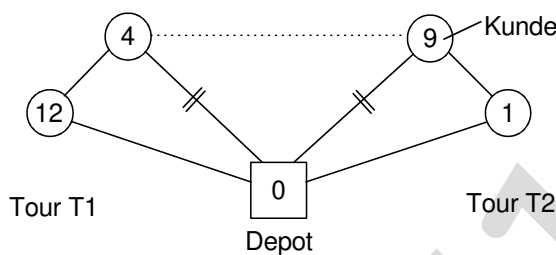
### (1) Verfahrensidee

Das Verfahren startet mit einer Ausgangslösung, die nur aus Pendeltouren für alle Kunden besteht. Eine Pendeltour enthält genau einen Kunden. Die Route für die  $i$ -te Pendeltour,  $i = 1, \dots, NK$ , lautet also:  $\{0, i, 0\}$ .

Im weiteren Verfahrensablauf werden jeweils zwei Pendeltouren und später auch Touren zu einer Tour verschmolzen. Allerdings nur unter folgenden Voraussetzungen:

- Die Verschmelzung führt zu einer Lösung mit geringerem Zielfunktionswert.
- Die entstandene Tour ist zulässig, d.h. sie genügt der Kapazitäts- und der Tourdauerrestriktion.

Eine Verschmelzung ist nur an den durch Endkunden gegebenen Stellen der zu den Touren gehörigen Routen zulässig. Endkunden einer Route sind der erste und der letzte Kunde. Folgende Abb. 3.9 veranschaulicht den Vorgang der Verschmelzung:



**Abb. 3.9.** Verschmelzung zweier Touren.

Verschmolzen werden die Routen  $R1 = \{0, 12, 4, 0\}$  und  $R2 = \{0, 9, 1, 0\}$ . Die Verschmelzung findet bei den Endkunden 4 und 9 statt (siehe gestrichelte Verbindung). Zwei Verbindungen zum Depot fallen weg (siehe doppelt durchgestrichene Verbindungen). Das Ergebnis der Verschmelzung ist die Route  $R = \{0, 12, 4, 9, 1, 0\}$ .

Die Verschmelzung zweier Touren über zwei Endkunden  $i, i = 1, \dots, NK$ , und  $j, j = 1, \dots, NK, i \neq j$ , führt zu einer Wegeinsparung  $S(i, j)$ . Diese Einsparung (engl. saving) ermittelt sich zu:

$$S(i, j) = \text{Entf}(0, i) + \text{Entf}(0, j) - \text{Entf}(i, j). \quad (9)$$

$S(i, j)$  fällt umso höher aus, je geringer die Entfernung zwischen den beiden Kunden ist und je größer die Entfernungen der beiden Kunden zum Depot sind.

Im Verfahrensablauf werden jeweils die beiden Routen als Kandidaten für die nächste Verschmelzung gewählt, für die die Einsparung am größten ist.

## (2) Verfahrensbeschreibung

Das Savingsverfahren besteht aus drei Teilen, die hier mit „Initialisierung“, „Iteration“ und „Abschluss“ bezeichnet seien. Der Verfahrensteil Iteration wird wiederholt ausgeführt, wobei eine Abbruchbedingung das Iterationsende steuert.

Im Einzelnen gilt für die Verfahrensteile:

- Initialisierung: Nach dem Einlesen der Problemdaten wird eine Ausgangslösung mit  $NK$  Routen  $\{0, i, 0\}, i = 1, \dots, NK$ , entsprechend den vorliegenden  $NK$  Aufträgen, generiert. Es folgt die Berechnung der Savings  $S(i, j)$  für alle Kundenpaare  $(i, j), i < j$ . Die Savings werden nach abnehmenden Werten in einer linearen Liste, der sogenannten Savingsliste, abgelegt.
- Iteration: Das jeweils erste bzw. größte Element der Savingsliste sei  $S(i, j)$ ; es wird aus der Liste entfernt. Sind die durch das größte Element gegebenen Kunden  $i$  und  $j$  Endkunden, so werden die durch diese Indizes definierten Routen

als Kandidaten für eine Verschmelzung verwendet. Eine Verschmelzung findet statt, falls die Kunden  $i$  und  $j$  Endkunden zweier verschiedener Touren sind, und falls die neue Tour der Kapazitätsrestriktion und der Tourdauerrestriktion genügt.

- Abbruchbedingung: Die im Iterationsteil beschriebenen Operationen werden solange wiederholt, bis die Savingsliste leer ist.
- Abschluss: Nach Beendigung des Iterationsteils liegt ein suboptimaler Tourenplan mit  $NT$  Touren vor. Die verfügbare Fahrzeuganzahl werde durch  $NFzg$  bezeichnet. Gilt  $NT > NFzg$ , d.h. enthält der Tourenplan mehr Touren als Fahrzeuge bereitstehen, so werden  $NFzg$  Touren nach einem geeigneten Auswahlkriterium selektiert und die übrigen Touren verworfen. Abschließend wird die Gesamtstrecke des Tourenplans berechnet und dieser ausgegeben.
- Tourauswahlkriterium: es kann entweder das Frachtwertkriterium oder das Terminverzugskriterium benutzt werden. Bei Wahl des Frachtwertkriteriums werden die  $NFzg$  Touren mit den höchsten beförderten Frachtwerten selektiert. Bei Anwendung des Terminverzugskriteriums werden diejenigen  $NFzg$  Touren selektiert, für welche die Summe aus den Terminverzügen aller zugehörigen Aufträge am größten ist. Der Terminverzug eines Auftrags bestimmt sich dabei als Differenz zwischen dem Auslieferungsdatum und dem Annahmedatum eines Auftrags. Das Motiv für dieses Kriterium ist, diejenigen Aufträge vorrangig zu disponieren, die bereits am längsten auf die Abwicklung warten.

### (3) Struktogramm

Das folgende Struktogramm in Abb. 3.10 stellt das Savingsverfahren mit den oben beschriebenen Teilen dar. Dabei wird unterstellt, dass die vorliegenden Aufträge aus einer Auftragsdatei und die Entfernungen aus einer Entfernungswerkdatei einzulesen sind, während der fertige Tourenplan in einer Tourenplandatei abzulegen ist. Als interne Datenstrukturen werden benutzt:

- Eine Auftragsliste, welche sämtliche Kundenaufträge der Planungsperiode einschließlich der zugehörigen Kundenindizes enthält.
- Eine Savingsliste, welche die Savings für alle Kundenpaare nach abnehmenden Werten enthält.
- Eine Tourenliste, welche maximal  $NK$  Touren enthält; jede Tour ist (zunächst) eine Pendeltour.

### (4) Verfahrensmodifikation bei mehreren Aufträgen pro Kunde

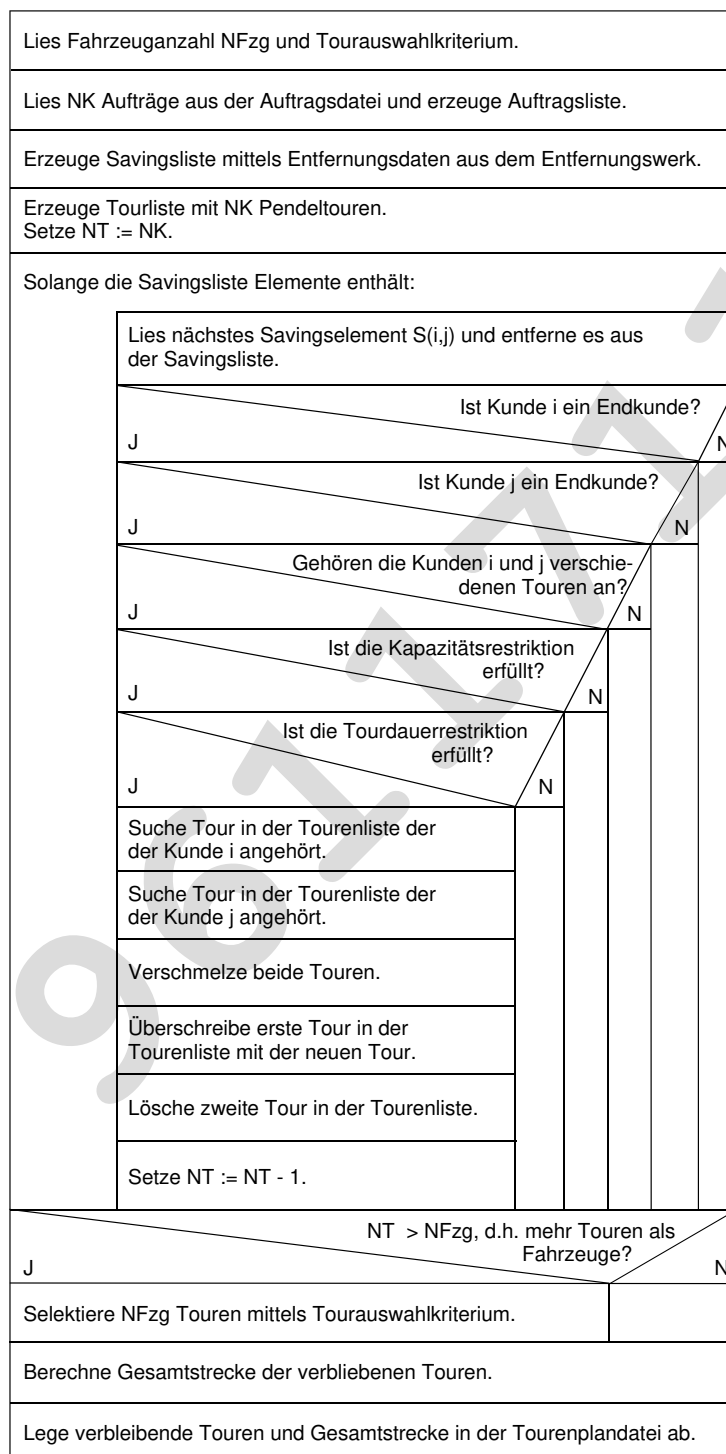
Das Verfahren wird abschließend auf einfachste Weise modifiziert, um mehrere Aufträge pro Kunde berücksichtigen zu können. Für jeden Auftrag eines Kunden wird quasi eine gesonderte Instanz des Kunden eingeführt, wobei natürlich die Orte aller Instanzen eines Kunden übereinstimmen. Dementsprechend wird anfangs pro Auftrag eine Pendeltour erzeugt, wobei nun mehrere Pendeltouren zu demselben Kunden führen können.

Ferner wird die Erzeugung der Savingsliste wie folgt modifiziert:

- ermittelt werden nun auch alle Savings  $S(i, i)$  für übereinstimmende Kunden  $i$ ,  $i = 1, \dots, NK$ ,
- haben die Kunden  $i$  und  $j$   $n_i$  bzw.  $n_j$  Aufträge erteilt, so wird der Saving  $S(i, j)$   $n_i \cdot n_j$ -mal in die Savingsliste aufgenommen;  $i, j = 1, \dots, NK$ ;  $n_i, n_j \geq 1$ .

Bei der Suche nach verschmelzbaren Touren zu einem Saving ist schließlich zu beachten, dass Kunden in mehreren Touren zugleich Endkunden sein können. Da Savingswerte  $S(i, j)$ ,  $i = 1, \dots, NK$ , meist relativ groß ausfallen, tendiert das Verfahren dazu, mehrere Aufträge eines Kunden gemeinsam in eine Tour aufzunehmen.

### Struktogramm Savingsverfahren



**Abb. 3.10.** Savings-Algorithmus für die Tourenplanung.

Die komplexen Operationen und Datenstrukturen des spezifizierten Lösungsverfahrens sind im weiteren Entwicklungsprozess zu verfeinern und zu konkretisieren.



## Übungsaufgaben zu Kapitel 3.8

### Übungsaufgabe 3.8.1

Begründen Sie kurz, warum es sinnvoll ist, vor der Auswahl und Spezifizierung des Lösungsverfahrens für ein betriebliches Entscheidungsproblem dieses zunächst zu modellieren.

961171

## 4 Fachliche Modellierung

### 4.1 Konzepte und Diagramme für die statische Modellierung

Die der statischen Modellierung dienenden Konzepte wurden bereits überblicksartig umrissen (vgl. Kap. 2.1). Das zentrale Diagramm der statischen Modellierung ist das Klassendiagramm, während das Objektdiagramm von untergeordneter Bedeutung ist. Das Klassendiagramm sowie das Klassenlexikon einer Anwendung werden am Ende dieses Abschnitts ausführlicher betrachtet.

**statische Modellierung**

#### 4.1.1 Objekt

Den Ausgangspunkt einer objektorientierten Analyse bilden die realen Objekte des jeweiligen Anwendungsbereiches. Diese können z.B. Personen oder Dinge, jedoch auch Vorgänge wie ein Fußballspiel, Ereignisse wie der Start eines Flugzeuges oder Begriffe wie „Informatik“ sein. Objekte des Anwendungsbereiches stellen also Gegenstände im weitesten Sinne des Wortes dar.

**reale Objekte und  
Modellobjekte**

Die OOA bildet reale Objekte des Anwendungsbereiches in spezifischer Weise auf Objekte des Fachkonzepts der Anwendung ab. Jedes modellierte Objekt besitzt folgende vier Komponenten:

**Komponenten von  
Objekten**

1. Attribute und deren Werte,
2. Operationen,
3. Verbindungen zu anderen Objekten und
4. eine Objektidentität.

Wird einfach von Objekten gesprochen, so sind nachfolgend stets modellierte Objekte gemeint.

Attribute legen Merkmale des Objekts fest; die Attributwerte geben Merkmalsausprägungen an. Während die Auswahl der Attribute unveränderlich ist, können sich die Attributwerte im Laufe der Zeit ändern. Attribute werden durch einfache oder komplexe Datentypen definiert.

**Attribute**

Objekte können Operationen ausführen, die vor allem der Manipulation ihrer Attributwerte dienen. Die Menge der Operationen eines Objekts ist unveränderlich. Operationen werden durch einfache oder komplexere Algorithmen realisiert.

**Operationen**

Ein Objekt X kann Verbindungen zu anderen Objekten Y, Z, ... besitzen. Dies bedeutet, dass sich das Objekt X und die Objekte Y, Z, ... kennen. Die Verbindungen eines Objekts sind veränderlich (vgl. Kap. 4.1.5).

**Objektverbindungen**

Schließlich besitzt jedes Objekt eine Objektidentität. Die Eigenschaft der Objektidentität gewährleistet, dass jedes Objekt von allen anderen unterscheidbar ist. Dies gilt auch, wenn zwei oder mehrere Objekte einer Klasse zufällig dieselben Attributwerte besitzen. Die Objektidentität wird also nicht durch ein Attribut, sondern unabhängig von diesen gewährleistet. Im Unterschied zu den Attributwerten ist die Identität eines Objekts unveränderlich. Unterstellt wird, dass jedes Ob-

**Objektidentität**

jekt von selbst eine Identität erhält; ihre Realisierung ist nicht Gegenstand der Systemanalyse.

#### Zustand, Verhalten

Die momentanen Werte der Attribute sowie die aktuellen Verbindungen eines Objekts bilden gemeinsam den Zustand des Objekts. Die Gesamtheit der ausführbaren Operationen eines Objekts wird auch als Verhalten des Objekts bezeichnet.

#### UML-Notation

Einzelne Objekte werden in der UML lediglich durch eine Bezeichnung sowie optional durch ihre Attribute und deren Werte dargestellt. Operationen und Objektidentität werden nicht notiert. Die Verbindungen eines Objekts zu anderen Objekten werden ggf. außerhalb des Objekts dargestellt. Im Einzelnen gelten folgende Festlegungen:

- Ein Objekt wird als ein Rechteck mit ein oder zwei Feldern präsentiert. In dem oberen Feld wird eine Objektbezeichnung eingetragen, die stets unterstrichen wird. In dem unteren optionalen Feld werden gegebenenfalls Angaben zu allen oder einer Auswahl der Attribute des Objekts aufgeführt.
- Für die Bezeichnung eines Objekts sowie die Attributangaben existieren jeweils mehrere Alternativen, über die Tab. 4.1 Auskunft gibt (vgl. auch Kap. 2.1).

Format	Umfang der Angaben	Bemerkung/Zweck
<b>Objektbezeichnungen</b>		
Objekt: Klasse	Objektname und Klassenname	Benanntes Objekt, d.h. ein bestimmtes Objekt der Klasse.
Objekt	nur Objektname	Wie vorher, Klasse aus Kontext ersichtlich.
:Klasse	nur Klassenname	Anonymes Objekt, d.h. irgendein Objekt der Klasse.
<b>Attributangaben</b>		
Attribut: Typ = Wert	Attributname, Typ und Wert	Typangabe redundant, da in Klasse vereinbart.
Attribut = Wert	nur Attributname und Wert	
Attribut	nur Attribut	Sinnvoll, wenn Wert uninteressant.

Tab. 4.1. Notation von Objektbezeichnungen und -attributen.

#### Beispiel 4.1

Objekte einer Klasse Buch können z.B. wie in Abb. 4.1 notiert werden:

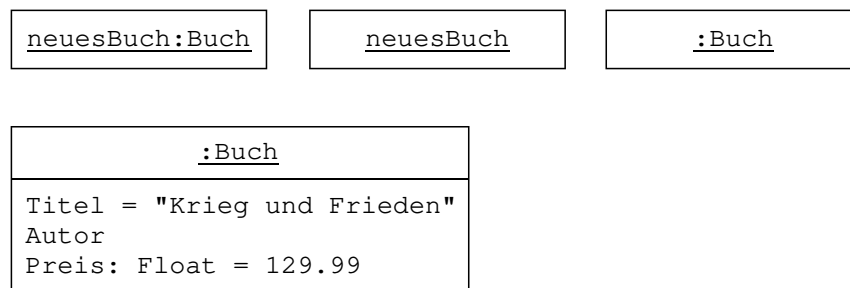


Abb. 4.1. Beispiele für die Notation von Objekten.

#### Objektdiagramme

Ein Objekt wird gewöhnlich nicht isoliert, sondern gemeinsam mit anderen Objekten in einem Objektdiagramm notiert. Objektdiagramme stellen Momentauf-

nahmen einer Anwendung oder eines Ausschnitts der Anwendung dar. Sie geben neben einer Menge von Objekten und deren Attributwerten vor allem die Verbindungen von Objekten zu einem gewissen Zeitpunkt an.

### Beispiel 4.2

Ein Objektdiagramm mit Objekten der Klasse Lieferant und der Klasse Artikel kann wie in Abb. 4.2 notiert werden:

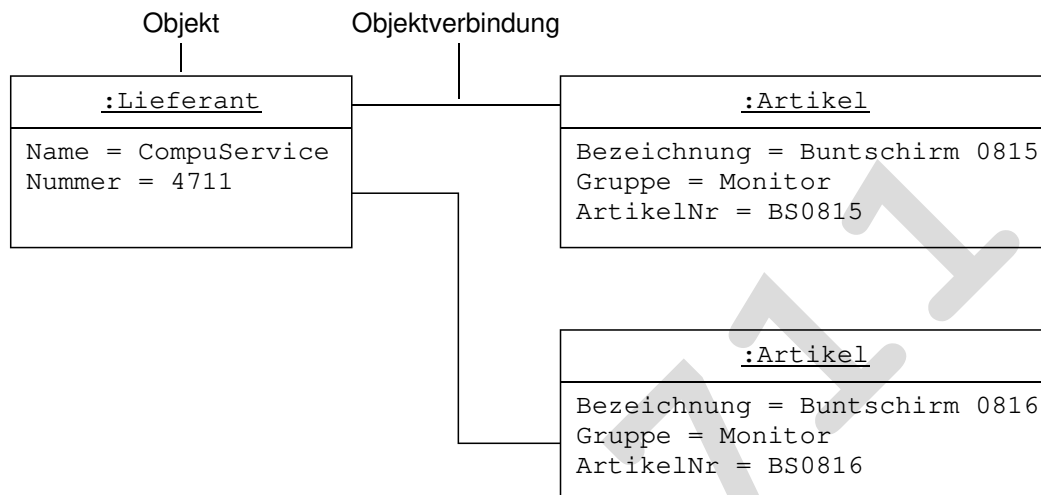


Abb. 4.2. Notation eines Objektdiagramms.

Objekte werden in Objektdiagrammen meist als anonyme Objekte aufgeführt. Erscheinen in einem Objektdiagramm konkrete, also benannte Objekte, so muss der Name des Objekts nur in dem betreffenden Diagramm eindeutig sein.

Aus der Situationsabhängigkeit der Objektdiagramme ergibt sich ihre beschränkte Aussagefähigkeit. Objektdiagramme können mitunter sinnvoll zur Veranschaulichung von Assoziationen herangezogen werden (vgl. Kap. 4.1.5); sie spielen insgesamt jedoch eine untergeordnete Rolle in der Analyse.

Objekte und ihre Klassen realisieren die Kapselung von Daten und Funktionen sowie die Datenkapselung gemäß dem Geheimnisprinzip. Da dies in Kap. 2 bereits ausführlich besprochen wurde, erübrigen sich an dieser Stelle weitere Erläuterungen.

**Kapselung und Datenkapselung**

### 4.1.2 Klasse

Jedes Objekt einer Anwendung gehört zu einer Klasse. Eine Klasse definiert einen gewissen Objekttyp, mit anderen Worten einen „Bauplan“ für gleichartige Objekte. Die Objekte einer Klasse werden auch als ihre Exemplare oder Instanzen bezeichnet. Eine Klasse wird unmittelbar durch folgende Komponenten festgelegt:

**Begriff der Klasse**

1. Attribute und
2. Operationen.

Eine Klasse besitzt ferner meist strukturelle Beziehungen zu anderen Klassen, nämlich Assoziationen und Vererbungsbeziehungen. Jedoch erscheinen diese Be-

ziehungen in einem OOA-Modell als separate Modellelemente und nicht unmittelbar als Klassenkomponenten.

Eine Klasse definiert zum einen den Typ ihrer Objekte. Sie hat ferner im Allgemeinen die Fähigkeit, Objekte der Klasse zu generieren. Klassen werden daher auch als Objektfabriken charakterisiert.

## UML-Notation

Eine Klasse wird in der UML durch ihren Namen, ihre Attribute und ihre Operationen notiert. Im Einzelnen gilt:

- Eine Klasse wird bei vollständiger Darstellung als ein Rechteck mit drei Feldern präsentiert. In dem oberen Namensfeld erscheint fettgedruckt, zentriert und mit einem Großbuchstaben beginnend der Klassenname. Im mittleren Attributfeld wird eine Liste der Attribute der Klasse aufgeführt. Das untere Operationenfeld enthält eine Liste der Operationen der Klasse.
- Als Klassenname ist stets ein Substantiv im Singular zu wählen. Es kann durch ein vorangestelltes Adjektiv ergänzt werden. Der Klassenname sollte innerhalb der Anwendung eindeutig sein (vgl. aber Kap. 4.1.7). Nur erwähnt sei, dass das Namensfeld durch ein Stereotyp (vgl. Kap. 3.2) und zusätzliche Merkmale zur Gruppierung bzw. Kommentierung der Klasse ergänzt werden kann.
- Attribute und Operationen können über ihre Benennung hinaus näher beschrieben werden. Hiermit beschäftigen sich die beiden folgenden Abschnitte. Die Attribute einer Klasse besitzen – anders als ihre Objekte und dem Typcharakter der Klasse gemäß – im Allgemeinen keine Werte.
- Auch reduzierte Notationen einer Klasse sind möglich, bei der auf das Attributfeld und/oder das Operationenfeld verzichtet wird. Reduzierte Darstellungen kommen u.a. bei noch unvollständigen Informationen in Betracht.

### Beispiel 4.3

Verschiedene Formen der Notation von Klassen zeigt anhand einer Klasse Konto die Abb. 4.3:

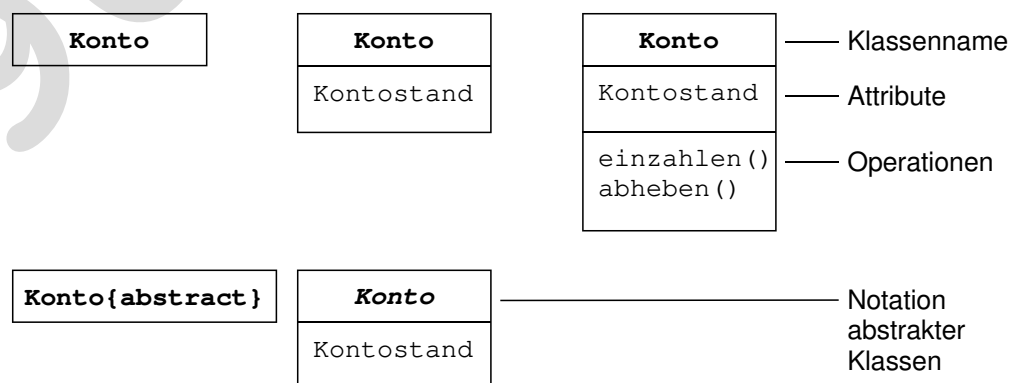


Abb. 4.3. Beispiele für die Notation von Klassen.

## abstrakte Klassen

Neben Klassen, die Objekte erzeugen können, gibt es auch sogenannte abstrakte Klassen. Von einer abstrakten Klasse können keine Exemplare generiert werden. Eine abstrakte Klasse wird im Namensfeld durch das dem Klassennamen folgende Wort {abstract} (in geschweiften Klammern) oder durch Kursivschreibung des Klassennamens gekennzeichnet (vgl. Abb. 4.3). Abstrakte Klassen spielen bei Vererbungsbeziehungen eine Rolle und werden in diesem Zusammenhang näher

betrachtet (vgl. Kap. 4.1.6). Ist eine Klasse nicht abstrakt, wird sie auch als konkrete Klasse bezeichnet.

In dem Klassendiagramm einer Anwendung werden ihre Klassen sowie die Assoziationen und Vererbungsbeziehungen zwischen den Klassen dargestellt. Da es alle statischen, d.h. zeitunabhängigen Aspekte der Anwendung modelliert, ist das Klassendiagramm der Anwendung das zentrale Dokument ihres statischen Modells. Zugleich dienen Klassendiagramme auch zur Visualisierung von Teilmengen der Klassen einer Anwendung sowie ihrer Beziehungen; für Beispiele sei auf folgende Abschnitte verwiesen. Details der Klassen einer Anwendung bzw. ihrer Komponenten werden in einem Klassenlexikon niedergelegt, worauf ebenfalls weiter unten eingegangen wird.

**Klassendiagramm,  
Klassenlexikon**

Die Beziehungen zwischen einer Klasse und ihren Objekten seien noch näher betrachtet. Die Attribute und Operationen eines Objekts werden innerhalb seiner Klasse definiert. Insbesondere werden die von den Objekten einer Klasse ausführbaren Operationen als Algorithmen bzw. als vollständige Funktionen in der Klasse abgelegt. Die innerhalb der Klasse als Algorithmen definierten Operationen werden in der UML als Methoden bezeichnet.

**Klassen und Objekte**

**Methode**

Jedes Objekt kennt seine Klasse. Dies bedeutet, dass es auf die in der Klasse abgelegten Informationen zu Attributen und Operationen zugreifen kann. Da alle Objekte einer Klasse hinsichtlich der ausführbaren Operationen übereinstimmen, müssen die zugehörigen Methoden nur einmal – nämlich innerhalb der Klasse – vorhanden sein. Soll ein Objekt eine gewisse Operation ausführen, so kommt die zugehörige, in der Klasse hinterlegte Methode zur Anwendung. Für jedes Objekt muss allerdings ein separater Satz von Attributwerten vorgesehen werden, da diese für verschiedene Exemplare der Klasse im Allgemeinen nicht übereinstimmen. In der UML werden Objekte daher ohne Operationen, aber mit Attributen und Attributwerten notiert.

Kennt ein Objekt seine Klasse, so kennt deshalb eine Klasse ihre – zu einem gewissen Zeitpunkt existierenden – Objekte noch nicht. Doch wird auch dies im Rahmen der Analyse vorausgesetzt. Jede Klasse führt also eine Liste ihrer Objekte, die stets aktualisiert wird, wenn ein Objekt erzeugt oder gelöscht wird. Dies wird als Objektverwaltung bezeichnet. Die in der OOA vorausgesetzte implizite Verwaltung aller Objekte einer Klasse durch ihre Klasse vereinfacht die Modellierung von Operationen, die mehrere oder alle Objekte einer Klasse betreffen (vgl. Kap. 4.1.4). Einfach deshalb, weil die Objektverwaltung selbst nicht modelliert werden muss. Die Abb. 4.4 verbildlicht die Beziehungen zwischen einer Klasse und ihren Objekten.

**Objektverwaltung**

Objekte sind temporärer Natur. Sie werden im Ablauf einer Anwendung erzeugt, ihre Attributwerte werden verändert, und sie werden spätestens dann zerstört, wenn die Ausführung der Anwendung endet. Welche Objekte bei einer konkreten Ausführung erzeugt werden und welche Werte diese annehmen, hängt von den jeweiligen Eingabedaten ab. Klassen sind dagegen permanente Einheiten, die von einer speziellen Ausführung der Anwendung nicht tangiert werden. Daher sind auch die Klassen einer Anwendung, nicht ihre Objekte, der eigentliche Gegenstand der Modellierung.

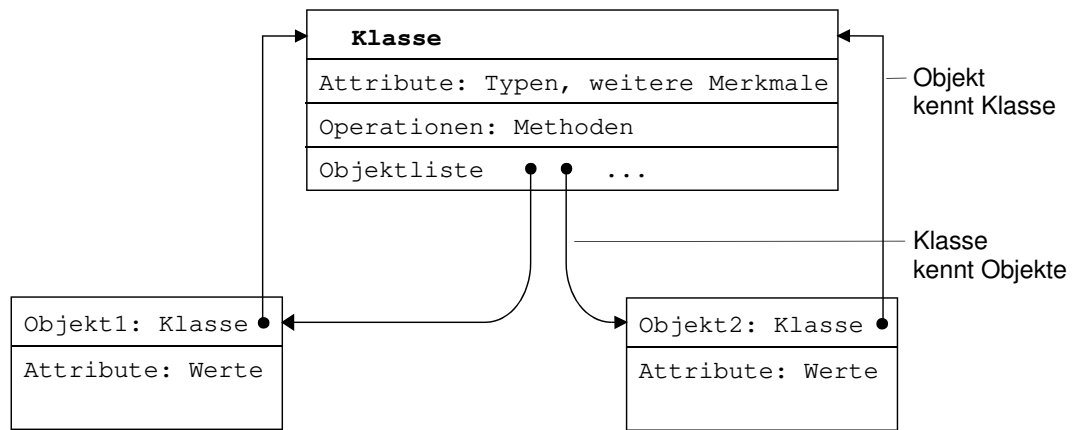


Abb. 4.4. Beziehungen zwischen einer Klasse und ihren Objekten (Objektverwaltung).

### 4.1.3 Attribut

#### Begriff des Attributs

Attribute legen die Daten der Objekte einer Klasse fest. Jedes Attribut besitzt einen Namen, einen Typ und optional weitere Merkmale, die im Folgenden vorgestellt werden.

Als Attributname wird meist ein Substantiv, manchmal auch ein Adjektiv gewählt. Die UML sieht kleine Anfangsbuchstaben für Attributnamen vor. Hier wird jedoch für Substantive ein großer Anfangsbuchstabe gewählt. Attributnamen müssen innerhalb einer Klasse eindeutig sein. Außerhalb einer Klasse kann ein Attributname durch den vorangestellten und mit einem Punkt abgetrennten Klassennamen qualifiziert, d.h. erweitert werden, um die Eindeutigkeit zu wahren, z.B.: Artikel.Bezeichnung.

#### Attributtypen

Der Attributtyp beschreibt die möglichen Werte eines Attributs. Attributwerte können aus einem oder auch mehreren elementaren Werten, z.B. aus mehreren ganzen Zahlen bestehen. Attribute können also auch komplexe Datenobjekte darstellen. Die Definition von Attributtypen wird von der UML nicht festgelegt. Im Sinne einer einheitlichen Beschreibung von Attributtypen wird hier folgende Auswahl möglicher Typen von Attributen vereinbart:

- Standardtypen, die üblichen vordefinierten Datentypen von Programmiersprachen entsprechen, wobei hier Typnamen angelehnt an C++ verwendet werden.
- Aufzählungstypen, die eine endliche Menge verschiedener, als Namen angegebener Werte zusammenfassen. Ein Attribut eines Aufzählungstyps kann jeweils nur einen dieser Werte annehmen.
- Auch Klassen sind als Attributtypen zugelassen. Es sei z.B. eine Klasse Adresse mit entsprechenden Attributen wie Postleitzahl, Strasse usw. definiert worden. Dann kann in einer Klasse Person ein Attribut PersonenAdresse eingeführt werden und für dessen Typ die Klasse Adresse gewählt werden. Ein Objekt der Klasse Person besitzt in diesem Fall mit seinem Attribut PersonenAdresse ein Teilobjekt, das zur Klasse Adresse gehört. Record- bzw. Strukturtypen gelten in der OOA als spezielle Klassen ohne Operationen. Damit sind Recordtypen ebenfalls als Attributtypen zugelassen.
- Listentypen beschreiben Listen mit beliebig vielen Elementen eines einheitlichen Typs.



Die Tab. 4.2 enthält eine Aufstellung der hier verwendeten Attributtypen.

Typ	Erläuterung
<b>Standardtypen</b>	
Int	Beliebige ganze Zahl.
UInt	Beliebige nicht negative ganze Zahl.
Float	Beliebige reelle Zahl.
Boolean	Wahrheitswert, mögliche Werte: true (wahr), false (falsch).
Date	Datum.
Time	Zeit.
String, String(Länge)	String ohne bzw. mit spezifizierter maximaler Zeichenanzahl.
<b>Weitere Typen</b>	
Aufzählung	Aufzählungstyp. Seine möglichen Werte müssen in einer separaten Typdefinition festgelegt werden. Diese wird notiert gemäß: Typname = enum(Wert_1, Wert_2,..., Wert_n); z.B.: Ampelfarbe = enum(rot, gelb, gruen).
Klasse	Beliebige Klasse, die separat definiert wird. Recordtypen gelten ebenfalls als Klassen.
list of Typ	Listentyp, der durch die Angabe des Typs der Listenelemente als definiert gilt. <i>Typ</i> kann einer der zuvor aufgeführten Typen oder wiederum ein Listentyp sein.

**Tab. 4.2.** Zulässige Attributtypen für die Analyse.

Einem Attribut kann ein Anfangswert zugeordnet werden. Dieser wird dem Attribut bei der Erzeugung eines Objekts automatisch zugewiesen.

**Anfangswert**

Die bisher behandelten Merkmale von Attributen werden wie folgt notiert:

Attribut [:Typ] [=Anfangswert]

Dabei steht Attribut für den Attributnamen; optionale Bestandteile der Attributnotation sind in eckigen Klammern eingeschlossen.

Weitere optionale Merkmale eines Attributs werden anschließend in geschweiften Klammern notiert:

**weitere  
Attributmerkmale**

{Merkmal-1, Merkmal-2,...}

Als weitere optionale Merkmale von Attributen werden hier berücksichtigt:

- Muss-Attribute müssen stets einen Wert besitzen, während Kann-Attributen kein Wert zugeordnet sein muss. So wird etwa das Attribut Rechnungsdatum bei der Erzeugung eines Bestells-Objekts noch keinen Wert besitzen, während das Attribut Bestellnummer ein Muss-Attribut ist. Da Muss-Attribute überwiegen, werden nur Kann-Attribute durch die Merkmalsangabe {optional} besonders gekennzeichnet.
- Ein Attribut kann als Schlüsselattribut ausgezeichnet werden, wenn es – allein oder gemeinsam mit anderen Attributen (zusammengesetzter Schlüssel) – ein Objekt eindeutig identifiziert. Als Schlüsselattribut für Artikel kommt z.B. die Artikelnummer in Frage. Eine Kennzeichnung eines Schlüsselattributs wird durch die Merkmalsangabe {key} vorgenommen.

**Muss-/Kann-Attribut**

**Schlüsselattribut**

**konstante Attribute**

	<ul style="list-style-type: none"> <li>- Konstante Attribute dürfen ihren Wert nicht ändern, nachdem er einmal zugewiesen wurde. Dies gilt z.B. für die Artikelnummer eines Artikels. Die einmalige Wertzuweisung erfolgt meist bei der Erzeugung des Objekts mittels einer Konstruktoroperation (vgl. Kap. 4.1.4). Ein konstantes Attribut wird durch die Merkmalsangabe {readonly} notiert.</li> <li>- Stellt ein Attribut eine quantitative Größe dar, ist oft die Angabe einer Einheit erforderlich. So wird etwa für das Attribut Verkaufspreis einer Klasse Artikel die Einheit Euro angegeben. Eine Einheit wird gemäß {Einheit} notiert.</li> </ul>
<b>Restriktionen</b>	<ul style="list-style-type: none"> <li>- Attribute können mit Restriktionen versehen werden. Diese beschreiben Bedingungen, denen Attributwerte genügen müssen. Restriktionen können ein oder mehrere Attribute einer Klasse einbeziehen. Die Restriktion {Verkaufspreis <math>\geq</math> 10 Euro} fordert z.B., dass ein Attribut Verkaufspreis den Betrag von 10 Euro nicht unterschreitet. Die Restriktion {Rechnungsdatum <math>\geq</math> Lieferdatum} verlangt, dass das Datum einer Rechnungserstellung nicht vor dem Lieferdatum liegt. Beide Attribute sollen derselben Klasse Bestellung angehören.</li> </ul>
<b>Klassenattribute</b>	<p>Schließlich seien zwei Sonderformen von Attributen betrachtet. Klassenattribute beschreiben Merkmale, die nicht einzelnen Objekten, sondern ihrer Klasse zukommen. Sie werden folglich nicht den Objekten, sondern der Klasse selbst zugeordnet. Daher besitzt ein Klassenattribut auch nur einen einzigen Wert für alle Objekte der Klasse und existiert unabhängig von diesen. Es kann bereits mit einem Wert initialisiert werden, wenn noch kein Objekt der Klasse existiert. Ein Klassenattribut wird durch die Unterstreichung des Attributnamens gekennzeichnet. In einer Klasse Artikel könnten als Klassenattribute etwa die Attribute <u>Artikelanzahl</u> und <u>Rabatt</u> eingeführt werden, wobei <u>Artikelanzahl</u> vor der Erzeugung von Artikel-Objekten mit dem Wert 0 zu initialisieren ist.</p>
<b>abgeleitete Attribute</b>	<p>Einen weiteren Sonderfall stellen abgeleitete Attribute dar. Der Wert eines abgeleiteten Attributs lässt sich aus anderen Attributwerten derselben Klasse bestimmen. So kann etwa für eine Klasse Person das abgeleitete Attribut Alter aus dem Attribut Geburtsdatum berechnet werden. Ein unabhängiger Zugriff auf abgeleitete Attribute ist dagegen nicht möglich. Diese stellen daher zugleich konstante Attribute dar. Abgeleitete Attribute werden durch einen Schrägstrich vor dem Attributnamen notiert wie z.B. /Alter.</p>

#### Beispiel 4.4

Die Attributbeschreibung einer Klasse Person könnte wie folgt aussehen:

Name: String

Vorname: String {optional}

PersonenNr: UInt {key, readonly, PersonenNr > 0}

Geburtsdatum: Date

/Alter: UInt

Familienstand: TFamilienstand = ledig;

TFamilienstand = enum (ledig, verheiratet, geschieden, verwitwet)

LetzteAenderungFamilienstand: Date

{optional, LetzteAenderungFamilienstand > Geburtsdatum}

KoerperGroesse: Float {m}

Hobbies: list of String

AnzahlPersonen: UInt

MaximaleKoerperGroesse: Float {m}

#### 4.1.4 Operation

Operationen stellen Aktivitäten dar, die im Allgemeinen von Objekten ausgeführt werden. Alle Objekte einer Klasse können dieselben Operationen ausführen. Die Operationen einer Klasse beziehen sich vor allem auf die Attributwerte von Objekten dieser Klasse. Die Ausführung einer Operation durch ein Objekt wird durch eine Botschaft ausgelöst. Man spricht auch von dem Aufruf einer Operation.

##### Begriff Operation

Die Attributwerte eines Objekts können entsprechend der Datenkapselung unmittelbar nur durch die Operationen der zugehörigen Klasse bearbeitet werden. Jedoch können innerhalb einer Operation weitere Operationen aufgerufen werden, die sich auf andere Objekte derselben oder einer anderen Klasse beziehen. Eben hierdurch wird eine Interaktion verschiedener Objekte, d.h. eine sich über beliebig viele Objekte mehrerer Klassen erstreckende Verarbeitung, ermöglicht (vgl. hierzu nochmals das Beispiel 2.1).

Eine Operation besitzt eine Schnittstelle für ihren Aufruf, die als Signatur bezeichnet wird. Der Algorithmus einer Operation wird durch die zugehörige Methode definiert, die in der Klasse abgelegt ist.

##### Signatur

Die Notation einer Operation innerhalb der UML-Darstellung einer Klasse bzw. im Klassendiagramm beinhaltet nur die Signatur der Operation. Im Einzelnen gilt:

##### Notation der Signatur

- Die Signatur umfasst den Namen der Operation, eine Parameterliste, ggf. den Typ des Rückgabewerts der Operation sowie eine optionale Merkmalsliste. Außerdem kann die Sichtbarkeit der Operation angegeben werden, die jedoch erst im Entwurf relevant ist und daher hier nicht berücksichtigt wird. Liefert eine Operation keinen Wert zurück, entfällt die Angabe des Ergebnistyps. Für den Aufbau einer Signatur gilt folgendes Schema:

Operationsname(Parameterliste) : Ergebnistyp {Merkmalsliste}

- Als Operationsname wird meist ein Verb gewählt. Dieses kann zwecks genauere Umschreibung der Operation durch ein Substantiv ergänzt werden. Der Operationsname muss innerhalb der Klasse eindeutig sein. Außerhalb der Klasse kann der Operationsname wie bei Attributen durch den Klassennamen qualifiziert werden, z.B. Artikel.aenderePreis().
- Die Parameterliste umfasst alle Parameter der Operation und entspricht der Parameterliste von Funktionen (vgl. Kap. 1.3 und 2.4.1). Zwei aufeinanderfolgende Parameter werden durch ein Komma getrennt; eine leere Parameterliste wird durch ein Paar runder Klammern () gekennzeichnet. Jeder Parameter wird wie folgt spezifiziert:

##### Parameterliste

[Art] Parametername : Typ [= Defaultwert],

wobei Angaben in eckigen Klammern optional sind. Für die Art eines Parameters kommen die Varianten in (Eingabeparameter), out (Ausgabeparameter) und inout (Parameter für Eingabe und Ausgabe) in Frage. Wird die Art eines Parameters nicht angegeben, so wird hier von einem Eingabeparameter ausgegangen. Als Typen von Parametern – wie auch des Rückgabewerts einer Operation – kommen hier alle in Tab. 4.1 aufgeführten Typen in Betracht. Ein Parameter kann auch mit einem voreingestellten Wert, genannt Defaultwert, für das bei einem Aufruf übergebene Argument versehen werden.

- Mit der in geschweifte Klammern eingeschlossenen Merkmalsliste kann eine Operation noch um weitere Merkmale ergänzt werden. Ein Beispiel für weitere Merkmale wird weiter unten angeführt.
- Die Signatur kann vereinfachend nur durch die Angabe des Operationsnamens ergänzt durch ein Paar runder Klammern notiert werden.

#### Beispiel 4.5

Nachfolgend werden Signaturen von Operationen einer Klasse Konto aufgelistet.

Vollständige Signaturen:

druckeKontostand(in Tag : Date)

einzahlen(in Betrag : Float, in Tag : Date)

abheben(inout Betrag : Float = 100.00, in Tag : Date) : Boolean

Verkürzte Signaturen:

druckeKontostand()

einzahlen()

abheben()

#### Aufgaben von Operationen

Die Operationen einer Klasse seien nun aus inhaltlicher Perspektive betrachtet.

Hierzu werden zunächst verschiedene mögliche und auch kombiniert ausführbare Aufgaben von Operationen zusammenfassend aufgelistet:

- (1) Veränderung des Objektbestands durch Objekterzeugung bzw. -vernichtung.
- (2) Lesender Zugriff auf Attribute von Objekten bzw. Klassenattribute.
- (3) Schreibender Zugriff auf Attribute von Objekten bzw. Klassenattribute.
- (4) Bereitstellung und Veränderung der Verbindungen zwischen Objekten.
- (5) Durchführung von Berechnungen.
- (6) Aufruf weiterer Operationen derselben oder einer anderen Klasse.
- (7) Bearbeitung aller Objekte oder einer Teilmenge der Objekte einer Klasse.

#### Arten von Operationen

Die folgenden Differenzierungen der Operationen betreffen teils ebenfalls ihre Aufgaben, berücksichtigen jedoch auch andere Differenzierungskriterien.

#### Basisoperationen

Aus der Gesamtheit der Operationen einer Klasse lassen sich gewisse Operationen herausheben, die eine stets erforderliche Basisfunktionalität repräsentieren. Es wird daher in der OOA angenommen, dass diese Basisoperationen immer vorhanden sind. Aus diesem Grund werden sie in die Notation von Klassen bzw. in ein Klassendiagramm nicht aufgenommen. Zu den Basisoperationen zählen:

- die Operation new(), mit der ein neues Objekt der Klasse erzeugt wird;
- die Operation delete(), mit der ein existierendes Objekt der Klasse gelöscht wird;
- Operationen getAttributname(), mit der der aktuelle Wert des Attributs Attributname gelesen wird, z.B. getKontonummer();
- Operationen setAttributname(), mit der der Wert des Attributs Attributname modifiziert wird, z.B. setKontostand();
- die Operation link(), mit der eine Verbindung zu einem anderen Objekt (derselben oder einer anderen Klasse) dauerhaft eingerichtet wird; die Operation kann auch gemäß setlinkKlassenname() notiert werden;

- die Operation `unlink()`, mit der eine bestehende Verbindung zu einem anderen Objekt eliminiert wird; die Operation kann auch gemäß `unlinkKlassenname()` notiert werden.

Die Operationen `getAttributname()` bzw. `setAttributname()` werden im allgemeinen für alle Attribute einer Klasse benötigt, um die Datenkapselung zu realisieren.

Die übrigen Operationen einer Klasse werden als komplexe Operationen bezeichnet. Diese gliedern sich wiederum in verschiedene, nicht immer überschneidungsfreie Untergruppen (vgl. BOOCH 1994, COAD und YOURDON 1994):

- Konstruktor- und Destruktoroperationen erzeugen bzw. zerstören Objekte. Im Unterschied zur Basisoperation `new()` initialisiert eine Konstruktoroperation (kurz: Konstruktor) auch die Attribute eines generierten Objekts. Anders als die Basisoperation `delete()` führt eine Destruktoroperation (kurz: Destruktor) vor der Objektzerstörung oft noch weitere Verarbeitungsschritte durch. Deshalb rechnen Konstrukturen und Destruktoren zu den komplexen Operationen.
- Verwaltungsoperationen für das Erfassen, Ändern, Löschen oder die Ausgabe einzelner Objekte wie auch die Ausgabe von Objektlisten sind typische komplexe Operationen, die hier meist in Klassendiagramme aufgenommen werden. Das Erfassen stellt zugleich eine Konstruktor-, das Löschen eine Destruktoroperation dar. In beiden Fällen sind meist zusätzliche Datenprüfungen erforderlich.
- Zu den komplexen Operationen gehören ferner Operationen zur Berechnung von Werten sowie für die Überwachung von Zuständen. Man denke etwa an die Überwachung von Zuständen eines externen Geräts.

**komplexe Operationen**

**Konstrukturen/  
Destruktoren**

**Verwaltungs-  
operationen**

**Wertberechnung,  
Zustandsüberwachung**

Eine weitere Differenzierung von Operationen betrifft die Aktivierung der Operation. Unterschieden werden diesbezüglich externe und interne Operationen. Externe Operationen werden von der Benutzungsschnittstelle aus aufgerufen. Interne Operationen werden dagegen nur innerhalb anderer Operationen bzw. von anderen Objekten aus aufgerufen.

**externe und interne  
Operationen**

Schließlich können analog zu den Attributen in einer Klasse Objektoperationen und Klassenoperationen abgegrenzt werden. Die wichtigeren und häufigeren Objektoperationen werden einfach Operationen genannt und von einem einzelnen Objekt ausgeführt.

**Objektoperationen  
und...**

Klassenoperationen sind der Klasse zugeordnet und werden von ihr selbst, also nicht von einem Objekt ausgeführt. Sie dienen dem Zugriff auf Klassenattribute und der gleichzeitigen Verarbeitung aller oder mehrerer Objekte der Klasse (vgl. die Aufgaben (1), (2) und (7) in der obigen Auflistung). Klassenoperationen werden durch eine Unterstreichung des Operationsnamens gekennzeichnet. Eine Klasse `Artikel` mit dem Klassenattribut `Artikelanzahl` kann z.B. die Klassenoperationen `aktualisiereArtikelanzahl()`, `druckeArtikelliste()` sowie `druckeSaisonartikelliste()` enthalten. Die erste Operation dient dem Zugriff auf das Klassenattribut. Die zweite Operation gibt Daten aller existierenden Objekte aus. Die dritte Operation gibt nur Daten einer Teilmenge aller Objekte aus, nämlich der Saisonartikel. Sie greift also selektiv nach einem gewissen Kriterium auf eine Teilmenge aller Objekte der Klasse `Artikel` zu. Für die Bearbeitung mehrerer Objekte einer Klasse innerhalb einer Klassenoperation wird die in Kap. 4.1.2 besprochene implizite Verwaltung der Objekte durch ihre Klasse in Anspruch genommen.

**Klassenoperationen**

Die Algorithmen bzw. Methoden komplexer Operationen bedürfen der Spezifikation. Diese kann in der OOA überwiegend informell, durch eine textuelle Be-

**Spezifikation von  
Methoden**

schreibung erfolgen. Für komplizierte Operationen, z.B. umfangreiche Berechnungsoperationen, bieten sich semiformale Notationen (Pseudocode, Struktogramm) an (vgl. Abb. 3.10). Zu beachten ist, dass es in der Analyse nur um eine rein fachliche Spezifikation der Funktionalität, nicht um Implementierungsdetails geht. Viele Beispiele zur Spezifikation von Operationen finden sich in Kap. 4.4.

### 4.1.5 Assoziation

#### Begriff Assoziation

Bei der Ausführung einer Anwendung kooperieren Objekte durch den Austausch von Botschaften. Für das Versenden von Botschaften werden Verbindungen zwischen den beteiligten Objekten benötigt. Unterscheiden lassen sich dauerhafte und temporäre Objektverbindungen (vgl. Kap. 2.1). Zur Übermittlung von Botschaften werden meist dauerhafte Verbindungen genutzt. In der statischen Modellierung einer Anwendung werden dauerhafte Objektverbindungen als Assoziationen abgebildet. Eine einzelne Assoziation beschreibt eine Gesamtheit von logisch-inhaltlich zusammengehörigen Objektverbindungen.

#### binäre Assoziation

Betrachtet sei zunächst nur der wichtigste Typ von Assoziationen, nämlich binäre Assoziationen zwischen zwei verschiedenen Klassen. Die als binäre Assoziation modellierten Objektverbindungen bestehen jeweils zwischen zwei Objekten, die zu zwei verschiedenen Klassen gehören.

#### Beispiel 4.6

Gegeben seien etwa die beiden Klassen Kunde und Auftrag. Zu einem gewissen Zeitpunkt mögen die Kunden Müller und Schulze existieren. Der Kunde Müller hat die Aufträge „1001“ und „1002“, der Kunde Schulze den Auftrag „1003“ erteilt. Die Situation wird im Objektdiagramm der Abb. 4.5 veranschaulicht, während das zugehörige Klassendiagramm die Klassen Kunde und Auftrag sowie die zwischen ihnen bestehende Assoziation zeigt. Die Assoziation erweist sich z.B. als erforderlich, wenn nach der Erfassung von Aufträgen für einen Kunden eine Liste aller Aufträge dieses Kunden erstellt werden soll.

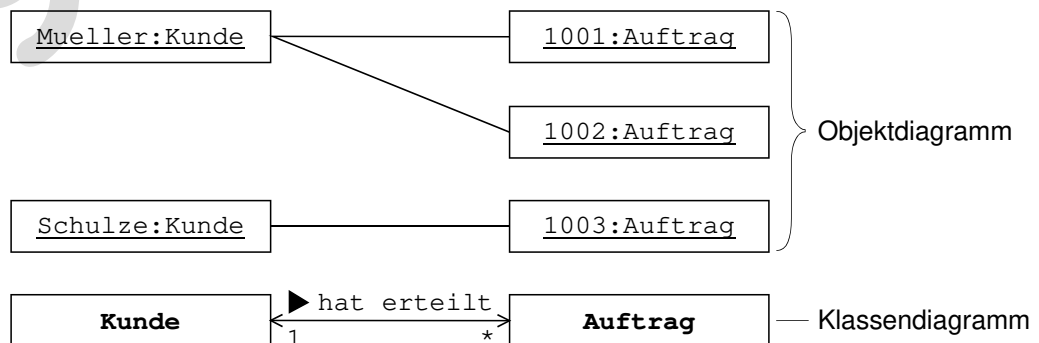


Abb. 4.5. Assoziation zwischen den Klassen Kunde und Auftrag.

#### Aspekte einer Assoziation

Eine Assoziation erfasst vier Aspekte einer Gesamtheit von Objektverbindungen:

- (1) Die Klassen, zwischen deren Objekten Verbindungen existieren. Im Beispiel 4.6 sind die Klassen Kunde und Auftrag assoziierte Klassen.
- (2) Die Richtung der Objektverbindungen. In einer bidirektionalen Assoziation kennen sich die Objekte gegenseitig. Im Beispiel 4.6 gibt die mit zwei Pfeil-



spitzen versehene Verbindungslinie zwischen den Klassen Auftrag und Kunde an, dass eine bidirektionale Assoziation vorliegt. Jedes Kunden-Objekt kennt die zugehörigen Auftrags-Objekte. Ein Auftrags-Objekt kennt auch das zugehörige Kunden-Objekt. In einer unidirektionalen Assoziation von Klasse A nach Klasse B kennen nur Objekte der Klasse A Objekte der Klasse B.

- (3) Die Semantik, d.h. die fachliche Bedeutung der Objektverbindungen. Diese ergibt sich in vielen Fällen bereits aus den assoziierten Klassen. Die Semantik einer Assoziation kann jedoch durch einen Namen der Assoziation explizit angegeben werden. Im obigen Beispiel wird die fachliche Bedeutung der Assoziation durch den Namen „hat erteilt“ ausgedrückt.
- (4) Die minimale und die maximale Anzahl von Verbindungen eines Objekts zu Objekten der assoziierten Klasse. Im obigen Beispiel gibt der Wert 1 bei der Klasse Kunde an, dass jeder Auftrag von genau einem Kunden erteilt wird, während das Jokerzeichen \* bei der Klasse Auftrag bedeutet, dass ein Kunde beliebig viele (auch gar keinen) Aufträge erteilt haben kann.

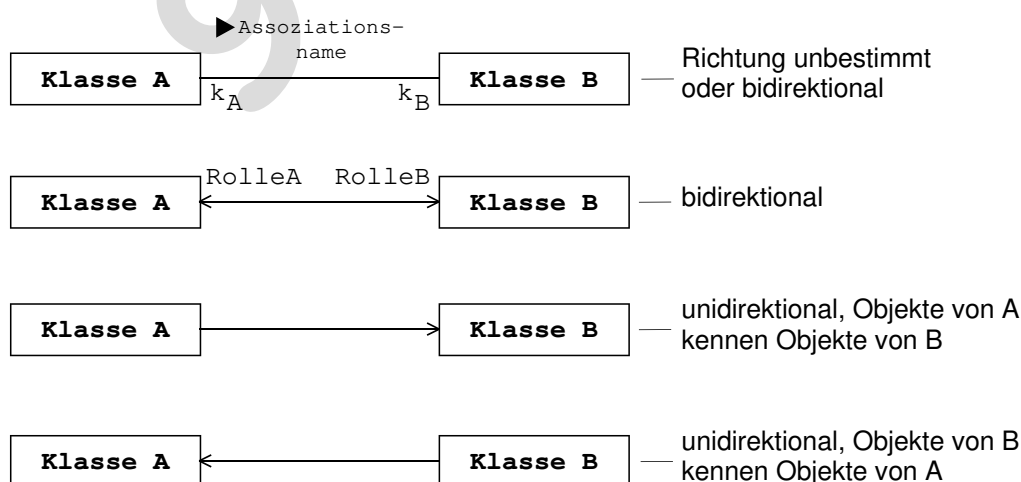
Zusammengefasst gilt: eine binäre Assoziation modelliert eine Gesamtheit von Objektverbindungen zwischen zwei bestimmten Klassen mit einheitlicher Richtung und Semantik.

Assoziationen bilden Verbindungen ab, die unmittelbar zwischen Objekten, nicht zwischen Klassen bestehen. Doch ebenso wie der Objektbestand einer Klasse verändert sich die Menge von Objektverbindungen zwischen zwei Klassen während der Ausführung einer Anwendung. Mit einem Objektdiagramm kann folglich stets nur die zu einem gewissen Zeitpunkt vorhandene Menge von Objektverbindungen zwischen zwei Klassen veranschaulicht werden. Permanent gültige Information bieten nur die zuvor erläuterten Unter- und Obergrenzen der Anzahlen der Objektverbindungen. Daher wird die Gesamtheit der Objektverbindungen zwischen zwei Klassen als Assoziation im statischen Klassendiagramm abgebildet und von Assoziationen zwischen Klassen sowie assoziierten Klassen gesprochen.

Die Abb. 4.6 stellt die UML-Notation von Assoziationen zwischen zwei Klassen allgemein dar.

**Assoziation zwischen Klassen**

**UML-Notation**



**Abb. 4.6.** Notation von Assoziationen zwischen zwei Klassen.

Die Notation sei wie folgt erläutert:

- Werden zwei Klassen durch eine einfache Linie verbunden, so liegt entweder eine bidirektionale Assoziation vor oder die Richtung der Assoziation ist noch

**Richtung**



unspezifiziert. Während die UML beide Interpretationen zulässt, wird hier die letztere bevorzugt (vgl. OESTERREICH 1998). Eine bidirektionale Assoziation liegt vor, wenn die Verbindungslinie an beiden Enden mit (offenen) Pfeilspitzen versehen ist. Bei unidirektionalen Assoziationen ist nur eine Pfeilspitze vorhanden. Kennen nur Objekte von Klasse A Objekte von Klasse B, so weist der Pfeil von Klasse A nach Klasse B. Eine bidirektionale Assoziation ist stets als eine Zusammenfassung zweier unidirektionaler Assoziationen mit entgegengesetzter Richtung aufzufassen. Wird zu einer unidirektionalen Assoziation ein Objektdiagramm erstellt, dann sollten auch die Objektverbindungen mit entsprechenden Pfeilen versehen werden.

- Name**
  - Der Name einer Assoziation ist im Allgemeinen optional. Da er auch bei bidirektionalen Assoziationen meist nur eine Richtung der Assoziation ausdrückt, kann die durch den Namen ausgedrückte Richtung zusätzlich durch ein schwarzes Dreieck markiert werden. So zeichnet im Beispiel 4.6 der Assoziationsname „hat erteilt“ die Richtung von der Klasse Kunde zur Klasse Auftrag aus, die durch das nach rechts weisende Dreieck hervorgehoben wird.
- Rollen**
  - Bei jeder der beiden assoziierten Klassen kann eine Rolle eingetragen werden. Sie gibt die Bedeutung an, die die betreffende Klasse in der Assoziation besitzt. Auch Rollen sind optionale Angaben; möglich ist auch, dass nur für eine assoziierte Klasse eine Rolle angegeben wird. Rollen werden ggf. alternativ zum Namen einer Assoziation benutzt. Sie erhöhen oft die Lesbarkeit einer Assoziation. Beispiele folgen weiter unten.
- Kardinalitäten**
  - Für beide assoziierten Klassen ist eine Kardinalität anzugeben. Die Kardinalität  $k_A$  definiert, mit wie vielen Objekten von Klasse A ein Objekt von Klasse B verbunden sein kann bzw. muss. Die Kardinalität  $k_B$  spezifiziert umgekehrt, mit wie vielen Objekten von Klasse B ein Objekt von Klasse A verbunden sein kann bzw. muss. Eine Kardinalität gibt zugleich den minimalen und den maximalen Umfang der jeweiligen Objektverbindungen an.

Die möglichen Kardinalitäten einer Assoziation und ihre Notation gehen aus der Tab. 4.3 hervor.

Kardinalität $k_A$ einer Klasse A	Anzahl der Objekte der Klasse A, die mit je einem Objekt einer assoziierten Klasse B verbunden sind
1	genau 1
0..1	0 oder 1
*	0 bis beliebig viele
2..*	2 bis beliebig viele
0..3	0 bis 3
4	genau 4
1,3,5	1 oder 3 oder 5
1..2,4, 6..*	mehr als 0, nicht 3 oder 5

**Tab. 4.3.** Varianten der Kardinalität und ihre Notation.

#### Assoziation als selbständiges Modellelement

Obwohl die Assoziationen einer Klasse diese zweifellos charakterisieren, erscheinen sie nicht als Bestandteile der Klasse selbst, sondern vielmehr als selbständige Modellelemente.

Das Wissen über die Verbindungen der Objekte einer Klasse steckt ausschließlich in ihren Assoziationen. Verbindungen eines Objekts zu anderen Objekten werden im Rahmen der Analyse also nicht durch Attribute wie Zeiger oder Fremdschlüssel repräsentiert. So wird im Beispiel 4.6 die Tatsache, dass der Kunde Müller die Aufträge „1001“ und „1002“ erteilt hat, nicht etwa derart ausgedrückt, dass die beiden Auftragsnummern in dem Kunden-Objekt Müller „vergraben“ werden.

Die selbständige Modellierung von Objektverbindungen erweist sich aus mehreren Gründen als vorteilhaft. Sie wird zum einen der Tatsache gerecht, dass Objektverbindungen Informationen darstellen, die wesentlich „von zwei oder mehreren Klassen abhängen“ (vgl. RUMBAUGH et al. 1993). Ferner erhöht die selbständige Darstellung der Objektverbindungen als Assoziationen die Lesbarkeit des Klassendiagramms. Schließlich sind Assoziationen in der OOA realisierungsneutral, d.h. sie nehmen keine bestimmte Form der späteren Implementierung von Objektverbindungen vorweg. Innerhalb der Analyse wird lediglich festgestellt, dass Verbindungen zwischen Objekten gewisser Paare von Klassen zu realisieren sind; das „Wie“ bleibt dem Entwurf vorbehalten.

Die Richtung von Assoziationen sollte soweit wie möglich bereits innerhalb der Analyse festgelegt werden. Die fachlichen Anforderungen bilden hierfür überwiegend eine hinreichende Grundlage. Außerdem wird die Richtung von Assoziationen bereits bei der Erstellung eines Prototyps der Benutzungsoberfläche benötigt.

Bei der Modellierung von Objektverbindungen ist zwischen temporären und dauerhaften Objektverbindungen zu differenzieren. Hier werden nur dauerhafte Objektverbindungen als Assoziationen modelliert. Werden beispielsweise nur im Moment der Erzeugung eines Objekts einer Klasse A Informationen von Objekten einer zweiten Klasse B benötigt, so wird keine Assoziation zwischen den Klassen A und B vorgesehen. Für konkrete Beispiele sei auf Kap. 4.4 verwiesen. Alternativ können temporäre Objektverbindungen jedoch auch als temporäre Assoziationen abgebildet werden, die mit dem Stereotyp <<temp>> versehen werden (vgl. OESTERICH 1998).

Über den bisher behandelten Haupttyp hinaus besitzt das Konzept der Assoziation zahlreiche Erweiterungen und Varianten. Einige von ihnen werden nachfolgend vorgestellt; auf die Spezifikation der Richtung wird dabei nicht eingegangen.

### (1) Reflexive Assoziationen

Die Objektverbindungen einer reflexiven (binären) Assoziation verknüpfen je zwei Objekte derselben Klasse. Die Assoziationslinie verbindet hier die Klasse mit sich selbst. Anstelle eines Assoziationsnamens sind im Interesse der Lesbarkeit stets die Rollen einzutragen, die beide Objekte einer Verbindung innerhalb der Assoziation spielen.

**reflexive Assoziation**

#### **Beispiel 4.7**

Betrachtet sei eine hierarchisch aufgebaute Stückliste der Bauteile eines Erzeugnisses. Auf der Klasse Bauteil besteht eine reflexive Assoziation. Sie verbindet jedes Bauteil mit seinem übergeordneten Bauteil und wird in Abb. 4.7 dargestellt. Während das eine Bauteil als Oberteil (oder Baugruppe) fungiert, ist das andere Bauteil innerhalb der Verbindung ein Unterteil (oder Einzelteil). Die Rollennamen wurden entsprechend gewählt. Ein Bauteil kann als Oberteil beliebig viele Unter-

teile besitzen; es besitzt kein Unterteil, wenn es nicht zerlegt werden kann (Kardinalität \*). Ein Bauteil besitzt als Unterteil genau ein Oberteil; eine Ausnahme bildet das komplette Erzeugnis, das kein Oberteil mehr besitzt (Kardinalität 0..1).

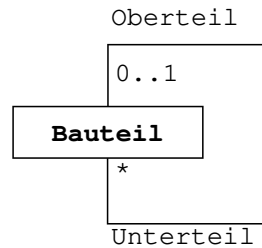


Abb. 4.7. Reflexive Assoziation auf der Klasse Bauteil.

## (2) Mehrere Assoziationen zwischen zwei Klassen

### mehrere Assoziationen

Bestehen zwischen zwei Klassen mehrere Assoziationen, so sind diese stets mit einem Namen oder Rollen zu versehen, um sie semantisch differenzieren zu können.

### Beispiel 4.8

Für die Studierenden eines Fachbereichs gelte, dass sie im Allgemeinen bei mehreren Lehrstühlen Vorlesungen hören, jedoch nur bei einem Lehrstuhl eine Diplomarbeit anfertigen. Die Abb. 4.8 zeigt die beiden Klassen Lehrstuhl und Student mit den beiden zwischen ihnen bestehenden Assoziationen „hoert bei“ und „ist Diplomand“ (vgl. auch die folgende Erweiterung (3)). Anstelle der Assoziationsnamen können auch die bei der Klasse Student einzutragenden Rollen „Hoerer“ bzw. „Diplomand“ angegeben werden.

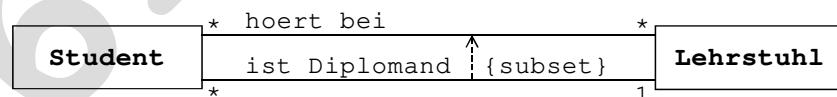


Abb. 4.8. Zwei Assoziationen zwischen den Klassen Student und Lehrstuhl.

## (3) Assoziationen mit Restriktionen

### Assoziationen mit Restriktionen

Assoziationen lassen sich um Restriktionen, d.h. zusätzlich einzuhaltende Bedingungen erweitern. Diese können sich auf eine einzelne Assoziation oder auch auf mehrere Assoziationen zugleich beziehen. Generell werden Restriktionen frei formuliert und in geschweifte Klammern gesetzt.

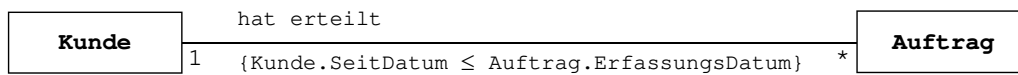
Für die beiden Restriktionen „hoert bei“ und „ist Diplomand“ des vorigen Beispiels besteht eine Teilmengenrestriktion, die durch den gestrichelten Pfeil sowie das Schlüsselwort {subset} notiert wird (vgl. Abb. 4.8). Die Restriktion fordert, dass jeder Diplomand eines Lehrstuhls auch Vorlesungen des Lehrstuhls hört.

### Beispiel 4.9

Für die Assoziation „hat erteilt“ zwischen den Klassen Kunde und Auftrag des Beispiels 4.6 kann z.B. folgende Restriktion gefordert werden:

$\text{Kunde.SeitDatum} \leq \text{Auftrag.ErfassungsDatum}$ .

Die Restriktion besagt, dass ein Kunde erst einen Auftrag erteilen kann, nachdem er selbst Kunde wurde. Ihre Notation zeigt Abb. 4.9.



**Abb. 4.9.** Assoziation zwischen den Klassen Kunde und Auftrag mit Restriktion.

#### (4) Geordnete Assoziationen

Als Spezialfall von Assoziationen mit Restriktionen lassen sich geordnete Assoziationen verstehen. Diese sind allerdings nur dann sinnvoll, wenn mindestens eine Kardinalität der Assoziation größer als 1 ist. Eine geordnete Assoziation wird gefordert, indem eine Kardinalität (größer als 1) um das Schlüsselwort `{ordered}` ergänzt wird. Die genaue Definition der Ordnung bzw. des Ordnungskriteriums sollte im Klassenlexikon notiert werden (vgl. Kap. 4.1.8).

**geordnete  
Assoziationen**

##### Beispiel 4.10

Betrachtet sei erneut die Assoziation „hat erteilt“ des Beispiels 4.6 zwischen den Klassen Kunde und Auftrag. Die Erweiterung der Kardinalität „\*“ der Klasse Auftrag um den Eintrag `{ordered}` kennzeichnet zunächst ganz allgemein, dass die Objektverbindungen eines Kunden mit seinen Aufträgen in irgendeiner Weise geordnet sind – vgl. Abb. 4.10. Eine nähere Spezifikation der Ordnung steht also noch aus. In Betracht kommt z.B. eine Ordnung aufsteigend nach dem Erfassungsdatum der Aufträge.



**Abb. 4.10.** Geordnete Assoziation zwischen den Klassen Kunde und Auftrag.

#### (5) Assoziative Klassen

Mitunter ist es sinnvoll, einer Assoziation bzw. ihren Objektverbindungen selbst Attribute und Operationen zuzuordnen. In diesem Fall wird eine Assoziation als assoziative Klasse realisiert. Jede Objektverbindung der ursprünglichen Assoziation wird in diesem Fall zu einem Objekt der assoziativen Klasse (vgl. RUMBAUGH et al. 1993).

**assoziative Klassen**

##### Beispiel 4.11

Zwischen den Klassen Lehrstuhl und Student des Beispiels 4.8 sei diesmal nur die Assoziation „hoert bei“ betrachtet. Nun soll erfasst werden, seit welchem Zeitpunkt ein Student Hörer bei einem gewissen Lehrstuhl ist. Hierzu wird die Assoziation „hoert bei“ als assoziative Klasse „Lehrstuhlhoerer“ mit dem Attribut `HoererSeitDatum` modelliert. Die Abb. 4.11 zeigt die Notation der assoziativen Klasse Lehrstuhlhoerer.

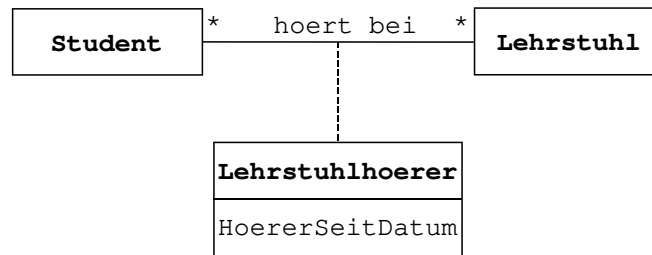


Abb. 4.11. Assoziative Klasse Lehrstuhlhoerer.

## (6) Aggregation und Komposition

Aggregation und Komposition sind Sonderformen der Assoziation.

### Aggregation

Das charakteristische Merkmal einer Aggregation ist, dass zwischen den Objekten der assoziierten Klassen eine Enthaltenseins-Beziehung besteht. Die beiden Klassen werden daher auch als Aggregatklasse und als Teilklasse bezeichnet. Ein Objekt der Aggregatklasse (Aggregatobjekt) ist ein übergeordnetes Ganzes, die mit ihm verknüpften Objekte der Teilklasse (Teilobjekt) bilden seine Teile. Für die Wahl der Richtung einer Aggregation gilt die Bedingung, dass ein Aggregatobjekt stets seine Teilobjekte kennen muss.

Allerdings sind die Begriffe der Enthaltenseins-Beziehung sowie des Ganzen und seiner Teile hier in einem weiten Sinne zu verstehen. COAD und YOURDON (1994) geben z.B. folgende Interpretationsmuster an:

- eine Baugruppe und ihre Bauteile,
- ein Behälter und sein Inhalt,
- eine Sammlung und ihre Elemente.

Eine Aggregation ist eine asymmetrische Beziehung. Ist ein Teilobjekt Bestandteil eines gewissen Aggregatobjekts, so kann das Aggregatobjekt nicht zugleich Bestandteil des Teilobjekts sein. Jedoch kann ein Teilobjekt Bestandteil mehrerer Aggregatobjekte zugleich sein. Es kann ferner seine Aggregatobjekte „überleben“.

### UML-Notation

Im Klassendiagramm wird eine Aggregation im Unterschied zu einer einfachen Assoziation durch eine leere Raute bei der Aggregatklasse gekennzeichnet. Wird die Richtung einer Aggregation spezifiziert und müssen die Teilobjekte das Aggregatobjekt kennen, so wird hier bei der Raute noch ein Pfeil eingetragen. Analog wird bei Kompositionen verfahren.

### Beispiel 4.12

Zwischen den Klassen PKW-Typ und PKW-Komponententyp besteht eine Aggregation, die in Abb. 4.12 gezeigt wird. Jeder PKW-Typ enthält diverse Komponenten wie z.B. Motor, Getriebe usw., die jeweils von einem gewissen Typ sind. Zugleich kann ein Komponententyp, etwa ein Motortyp, durchaus für mehrere PKW-Typen Verwendung finden.

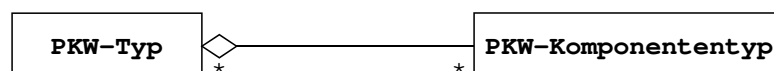


Abb. 4.12. Aggregation zwischen den Klassen PKW-Typ und PKW-Komponententyp.

### Komposition

Eine Komposition ist eine Aggregation, für die folgende zusätzliche Bedingungen erfüllt sind:

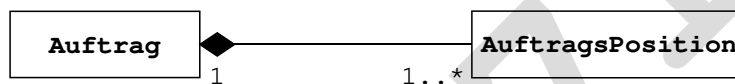
- Jedes Objekt der Teilklasse ist mit genau einem Objekt der Aggregatklasse verbunden, kann also nicht Teil mehrerer Aggregatobjekte zugleich sein. Ein Teilobjekt kann allerdings sein Aggregatobjekt im Allgemeinen wechseln.
- Die Existenz eines Teilobjekts hängt von der Existenz seines Aggregatobjekts ab. Wird das Aggregatobjekt gelöscht, so werden auch alle seine Teile gelöscht.

Im Klassendiagramm wird eine Komposition durch eine ausgefüllte Raute auf der Seite der Aggregatklasse gekennzeichnet.

**UML-Notation**

#### Beispiel 4.13

Die Beziehungen zwischen Aufträgen und ihren Auftragspositionen bilden ein typisches Beispiel einer Komposition, die in Abb. 4.13 notiert wird. Denn eine Auftragsposition kann nur Bestandteil eines Auftrags sein. Wird ferner ein Auftrag gelöscht, so ist auch das Fortbestehen einzelner Auftragspositionen sinnlos und diese sind ebenso zu entfernen.



**Abb. 4.13.** Komposition zwischen den Klassen Auftrag und AuftragsPosition.

Die Konzepte der Aggregation und der Komposition dienen der Modellierung asymmetrischer Enthaltenseins-Beziehungen. Dagegen werden diese Sonderformen der Assoziation nicht benötigt, wenn es nur darum geht, Existenzabhängigkeiten zwischen den Objekten zweier Klassen zu modellieren. Im Beispiel 4.6 kann ein Auftrag nicht ohne einen Kunden existieren, der den Auftrag erteilt. Dies wird bereits durch die Kardinalität 1 bei der Klasse Kunde ausgedrückt. In diesem Fall wurde eine einfache Assoziation verwendet, weil keine Enthaltenseins-Beziehung zwischen Aufträgen und Kunden besteht. In den beiden letzten Beispielen liegt eine Enthaltenseins-Beziehung vor, weshalb eine Aggregation bzw. eine Komposition modelliert wurde.

**Anwendungshinweis**

### (7) Höherwertige Assoziationen

Neben binären Assoziationen kommen auch Assoziationen vor, die Verbindungen zwischen je drei Objekten (ternäre Assoziationen) oder allgemeiner zwischen je  $k$  ( $k > 2$ ) Objekten ( $k$ -näre bzw. höherwertige Assoziationen) darstellen. Da meist nur binäre Assoziationen benötigt werden, wird auf höherwertige Assoziationen nicht näher eingegangen. Für Beispiele sei auf RUMBAUGH et al. (1993) und OESTERREICH (1998) verwiesen.

**höherwertige  
Assoziationen**

## 4.1.6 Vererbung

Eine Vererbungsbeziehung besteht meist zwischen einer übergeordneten und mehreren untergeordneten Klassen. Sie bewirkt, dass Eigenschaften der übergeordneten Klasse wie Attribute und Operationen an alle untergeordneten Klassen weitergegeben werden. Eine übergeordnete Klasse wird auch als Oberklasse, un-

**Begriff Vererbung,  
Ober- und  
Unterklassen**



tergeordnete Klassen werden auch als Unterklassen oder abgeleitete Klassen bezeichnet. Die Tab. 4.4 fasst die Eigenschaften zusammen, die von einer Oberklasse an ihre Unterklassen vererbt werden.

was wird vererbt?

Vererbte Eigenschaft	Erläuterung
Attribut	Jede Unterklasse besitzt alle Attribute der Oberklasse. Jedes Objekt der Unterklasse besitzt Attributwerte für alle Attribute der Oberklasse.
Klassenattribut	Jede Unterklasse verfügt über alle Klassenattribute der Oberklasse. Besitzt ein Klassenattribut der Oberklasse einen gewissen Wert, dann hat dieses Klassenattribut in den Unterklassen denselben Wert.
Operation/ Klassenoperation	Jede Unterklasse bietet die Operationen der Oberklasse an. Ein Objekt einer Unterklasse kann also alle Operationen der Oberklasse ausführen. Auch Klassenoperationen werden vererbt.
Assoziation	Besitzt die Oberklasse eine Assoziation, so besitzen auch alle Unterklassen diese Assoziation. Sie besitzt für die Unterklassen die gleichen Merkmale (z.B. Kardinalitäten) wie für die Oberklasse.

**Tab. 4.4.** Übertragene Eigenschaften bei der Vererbung.

Zusätzlich zu den von der Oberklasse ererbten Eigenschaften besitzen Unterklassen eigene Attribute, Operationen oder Assoziationen. Diese differieren von Unterklasse zu Unterklasse und stellen daher die speziellen Eigenschaften der Unterklassen dar.

Generalisierung/  
Spezialisierung

Die Vererbung unterstützt also die Generalisierung (Verallgemeinerung) und die Spezialisierung. Das Gemeinsame mehrerer Spezialfälle oder Varianten kann in einem allgemeinen Fall zusammengefasst werden, ein allgemeiner Sachverhalt kann in Sonderfälle untergliedert werden. Da die Verallgemeinerung und Spezialisierung zur Erfassung realer Situationen ständig benötigt werden, stellt die Vererbung in der objektorientierten Systemanalyse ein wesentliches Konzept für eine adäquate Modellierung dar. Gemeinsame Eigenschaften mehrerer Klassen müssen nur einmal, nämlich in der Oberklasse definiert werden. Die Vererbung leistet daher einen wichtigen Beitrag zu übersichtlichen und redundanzarmen Modellen. Ferner können auf der Basis vorhandener Klassen für allgemeine Fälle mit reduziertem Aufwand weitere abgeleitete Klassen für Spezialfälle gebildet werden.

UML-Notation

Die UML-Notation der Vererbung wird in Abb. 4.14 gezeigt. Eine Vererbungsbeziehung wird demnach durch einen Pfeil mit einem unausgefüllten Dreieck als Spitze dargestellt, der von jeder Unterklasse zur Oberklasse weist. Beide angegebenen Notationen sind gleichwertig.

Diskriminator

Die Spezialisierung einer Oberklasse in mehrere Unterklassen wird häufig nach einem einheitlichen Spezialisierungskriterium vorgenommen, das in der UML als Diskriminator bezeichnet wird. Ein Diskriminator kann, wie in Abb. 4.14 gezeigt, an die Vererbungspfeile der betreffenden Unterklassen angetragen werden. Darüber hinaus kann es weitere Unterklassen geben, die nicht einem bestimmten Diskriminator entsprechend abgeleitet wurden.



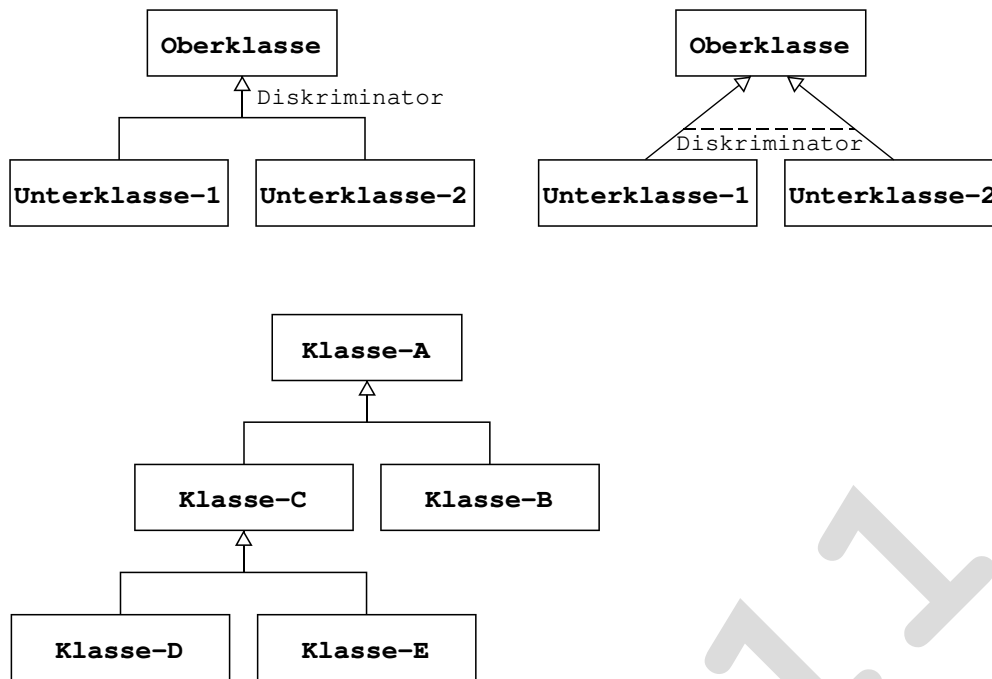


Abb. 4.14. Notation von Vererbungsbeziehungen.

**Beispiel 4.14**

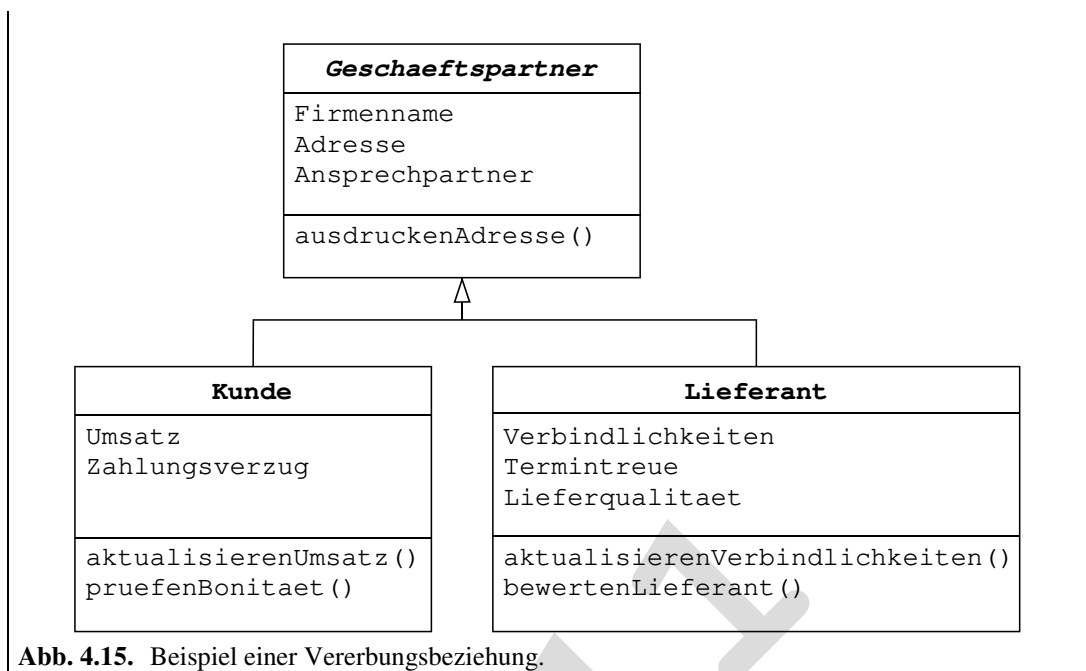
Eine Spezialisierung der Studenten der FernUniversität Hagen könnte etwa einerseits nach der Fachrichtung, andererseits nach dem Studiengang (Zusatzstudiengang, Diplomstudiengang) erfolgen. Entsprechende Vererbungsbeziehungen können als Diskriminatoren die Fachrichtung bzw. den Studiengang ausweisen.

Eine Oberklasse und ihre Unterklassen bilden eine Vererbungshierarchie. Eine Platzierung der Oberklasse oberhalb ihrer Unterklassen ist daher üblich, jedoch nicht zwingend. Die Abb. 4.14 veranschaulicht darüber hinaus, dass eine Unterklasse zugleich Oberklasse weiterer abgeleiteter Klassen sein kann. Auf diese Weise entstehen Vererbungshierarchien, die sich über mehr als eine Stufe erstrecken. Jede Unterklasse erbt dann die Eigenschaften ihrer direkten Oberklasse wie auch ihrer indirekten, in der Hierarchie höherstehenden Oberklassen.

Eine einstufige Vererbungsbeziehung mit zwei Unterklassen enthält das nächste Beispiel.

**Vererbungshierarchie****Beispiel 4.15**

Jeder Geschäftspartner eines Unternehmens sei entweder ein Kunde oder ein Lieferant. Im folgenden Klassendiagramm der Abb. 4.15 werden gemeinsame Eigenschaften von Kunden und Lieferanten in der Oberklasse Geschäftspartner zusammengefasst, während spezifische Eigenschaften von Kunden und Lieferanten in den Unterklassen erscheinen.

**abstrakte Oberklasse**

Die Klasse *Geschaeftspartner* wurde als abstrakte Klasse modelliert, von der keine Objekte erzeugt werden können (vgl. Kap. 4.1.2). Dies geschieht im gegebenen Fall, weil es keine Geschäftspartner an sich, sondern nur Kunden oder Lieferanten gibt. Abstrakte Klassen werden ausschließlich als Oberklassen in Vererbungsbeziehungen zur Weitergabe ihrer Eigenschaften an Unterklassen benutzt. Wie das Beispiel zeigt, können sie immer dann eingesetzt werden, wenn sich eine Generalisierung anbietet, während zugleich Objekte der Oberklasse – wie in diesem Fall *Geschaeftspartner*-Objekte – unvollständig und als solche nicht sinnvoll sind.

**Verwendung abstrakter Klassen**

Zusammenfassende Angaben zur möglichen Verwendung abstrakter Klassen in Vererbungsbeziehungen und als assoziierte Klassen enthält die folgende Tab. 4.5.

Beziehungstyp: Assoziation			
assozierte Klasse 1	assozierte Klasse 2	möglich	Bemerkung
abstrakt	konkret	ja	Möglich, weil (abstrakte) Klassen Assoziationen vererben.
abstrakt	abstrakt	ja	Siehe vorher.
Beziehungstyp: Vererbung			
Oberklasse	Unterklasse	möglich	Bemerkung
abstrakt	konkret	ja	Bereits erläutert.
abstrakt	abstrakt	ja	Abstrakte Klassen müssen (direkt oder indirekt) konkrete Unterklassen besitzen.
konkret	abstrakt	ja	Eher ungebräuchlich, siehe vorher.

**Tab. 4.5.** Abstrakte Klassen und ihre möglichen Beziehungen.

**Redefinition von Operationen**

In einer Unterklasse kann eine Operation erneut definiert werden, die den gleichen Namen und darüber hinaus die gleiche Signatur wie eine Operation der Oberklasse besitzt. In diesem Fall sagt man, dass die Operation der Unterklasse die Operation der Oberklasse redefiniert bzw. überschreibt. Natürlich wird eine Operation in einer Unterklasse ggf. derart redefiniert, dass ihre speziellen Erfordernisse berücksichtigt werden. Innerhalb der redefinierten Operation wird häufig die gleich-

namige Operation der Oberklasse aufgerufen. Die Redefinition einer Operation wird in Beispiel 4.16 betrachtet.

#### Beispiel 4.16

Studenten einer Universität können studentische Mitarbeiter sein und besitzen in diesem Fall zusätzliche Eigenschaften. Beim Ausdruck der Daten eines studentischen Mitarbeiters müssen neben den Attributwerten eines Studenten auch Mitarbeiterattribute beachtet werden. Daher wird die Operation `ausdruckenDaten()` der Oberklasse `Student` in der Unterklasse `StudentischerMitarbeiter` redefiniert, wie Abb. 4.16 zeigt. Das Beispiel zeigt darüber hinaus, dass eine Oberklasse auch eine einzige Unterklasse besitzen kann und dass Oberklassen auch konkrete Klassen sein können.

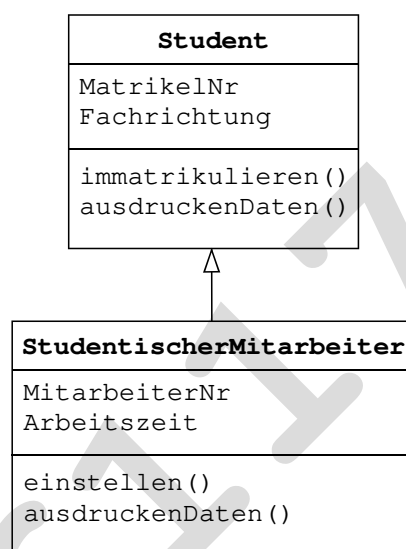


Abb. 4.16. Beispiel einer Vererbungsbeziehung mit Redefinition.

In der Analyse werden Attribute, Operationen wie auch Assoziationen stets so hoch wie möglich in einer Vererbungshierarchie angesiedelt. Existieren etwa zwei Unterklassen zu einer Oberklasse, die beide eine Assoziation zu einer dritten Klasse besitzen, so wird die Assoziation bei der Oberklasse eingetragen. Differenzen der Assoziation bezüglich beider Unterklassen können dann ggf. als Restriktionen formuliert werden; für Beispiele sei auf die Übungsaufgaben verwiesen. Redefinierte Operationen können wie gezeigt in den Unterklassen notiert werden.

Redefinitionen bilden eine unverzichtbare Voraussetzung für eine polymorphe Verarbeitung. Die Umsetzung des Polymorphismus-Konzepts ist vor allem eine Aufgabe des Entwurfs (vgl. Kap. 2.3). Jedoch kann die Analyse hierfür bereits eine Vorarbeit leisten. Dies geschieht eben dadurch, dass für inhaltlich analoge Operationen verschiedener Klassen einer Vererbungshierarchie bereits Redefinitionen vorgesehen werden.

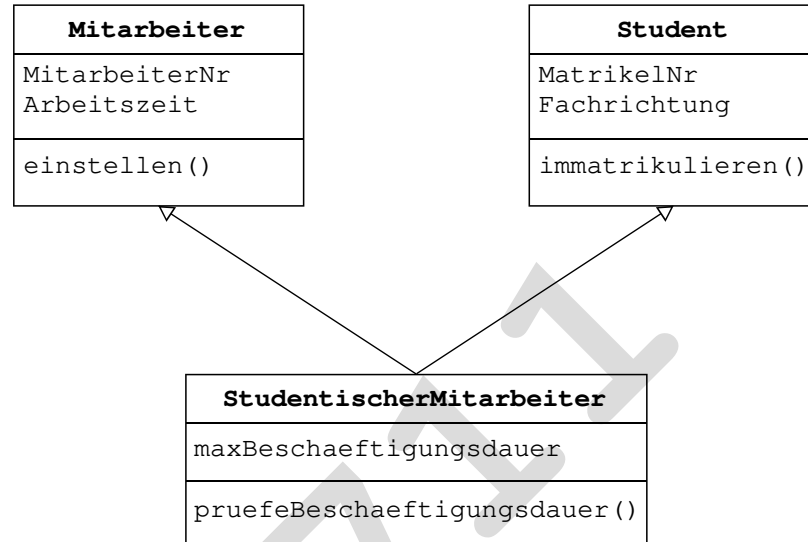
Eine einfache Vererbung liegt vor, wenn Unterklassen nur eine Oberklasse besitzen. Da in der OOA meist die einfache Vererbung genutzt wird, wurde sie bisher ausschließlich betrachtet. Daneben gibt es jedoch auch die Mehrfachvererbung, bei der Unterklassen mehrere Oberklassen besitzen. Bei einer Mehrfachvererbung verfügen die Unterklassen bzw. ihre Objekte dementsprechend zugleich über Eigenschaften mehrerer Oberklassen.

**Gestaltung einer Vererbungshierarchie**

**einfache und Mehrfachvererbung**

**Beispiel 4.17**

Für studentische Mitarbeiter bietet sich eine Mehrfachvererbung an, wobei als Oberklassen zugleich die Klassen Student und Mitarbeiter in Betracht kommen. Dies wird in Abb. 4.17 dargestellt.



**Abb. 4.17.** Beispiel für eine Mehrfachvererbung.

### 4.1.7 Paket

#### Begriff Paket

Ein Paket stellt eine Zusammenfassung mehrerer Modellelemente dar. Es wird daher auch als Subsystem bezeichnet. Mit Hilfe von Paketen kann ein Gesamtmodell in überschaubare Teilmodelle gegliedert und die Struktur eines Anwendungssystems auf höherer Abstraktionsebene dargestellt werden.

Als Modellelemente, die in Paketen zusammengefasst werden, kommen vor allem Klassen und ihre strukturellen Beziehungen (Assoziationen, Vererbungsbeziehungen) in Betracht. So können die Elemente eines Klassendiagramms in mehrere Pakete zerlegt werden. Jedes Paket enthält einige logisch-fachlich zusammengehörende Klassen sowie ihre Beziehungen und wird durch ein eigenes Klassendiagramm repräsentiert. Bei umfangreichen Klassendiagrammen trägt die Paketbildung dazu bei, dass das statische Modell übersichtlich bleibt.

#### Pakethierarchie

Auch Pakete können als Modellelemente innerhalb umfangreicherer Pakete erscheinen. Pakete können also geschachtelt werden, wodurch eine Pakethierarchie entsteht. Das Gesamtmodell einer Anwendung kann als das oberste Paket der Pakethierarchie aufgefasst werden.

#### UML-Notation

Pakete werden in der UML, wie in Abb. 4.18 gezeigt, als Rechtecke mit einem Reiter dargestellt. Werden die zugehörigen Modellelemente im Paket abgebildet, so trägt der Reiter den Paketnamen. Andernfalls wird der Paketname innerhalb des Pakets selbst eingetragen und der Reiter bleibt leer. In dem Klassendiagramm einer kompletten Anwendung bzw. dem obersten Paket einer Pakethierarchie wird hier teils auf einen Reiter verzichtet.

#### Abhängigkeits- beziehung

Die Abb. 4.18 beinhaltet auch eine Pakethierarchie, wobei das Paket Anwendung das komplette Anwendungssystem bezeichnet. Zwischen Paket-A und Paket-B besteht außerdem eine Abhängigkeitsbeziehung. Diese wird durch einen gestrichelten Pfeil notiert. Die Abhängigkeitsbeziehung bedeutet, dass eine Änderung des Pakets an der Pfeilspitze, hier Paket-A, eventuell auch eine Änderung des Pakets am anderen Ende des Pfeils, hier Paket-B, erfordert. Dies muss jedoch nicht der Fall sein und ist daher ggf. gesondert zu prüfen.

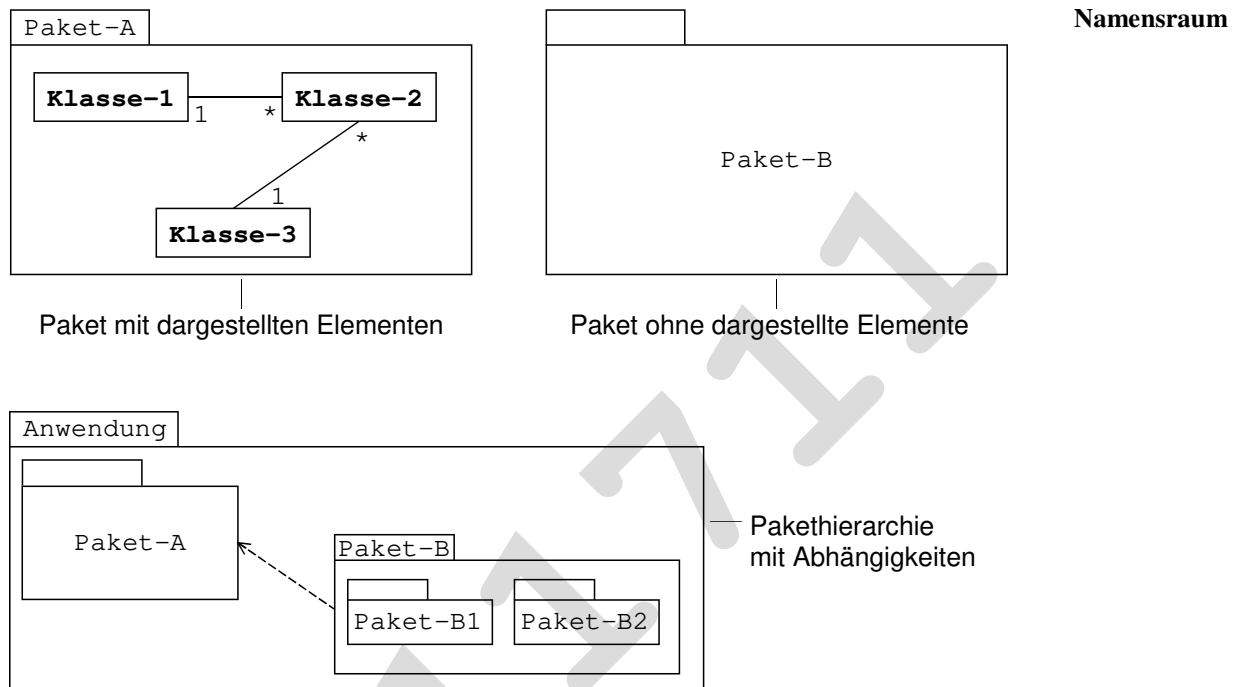


Abb. 4.18. Notation von Paketen.

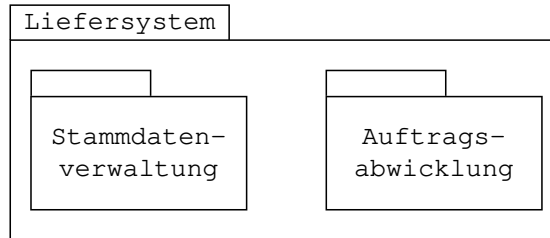
Jedes Paket definiert einen eigenen Namensraum. Der Name einer Klasse oder eines anderen Modellelements muss daher nur innerhalb des Pakets eindeutig sein. Außerhalb eines Pakets kann eine zu diesem gehörende Klasse durch den Paketnamen gemäß Paketname::Klassenname qualifiziert werden. So kann z.B. bezogen auf die vorherige Abbildung Klasse-1 außerhalb von Paket-A durch Paket-A::Klasse-1 referenziert werden. Bei Schachtelungen von Paketen gelten entsprechende Regeln. Enthält z.B. Paket-B1 die Klasse K-2, so kann diese in Paket-A wie auch im Gesamtsystem durch Paket-B::Paket-B1::K-2 referenziert werden.

Ein Modellelement – insbesondere eine Klasse – kann in mehreren Paketen erscheinen. Jedoch ist jede Klasse bzw. Modellelement genau einem (Heimat-)Paket zuzuordnen, in dem es allein unter seinem Elementnamen angesprochen wird. In anderen Paketen darf das Element nur unter Benutzung der erläuterten Namensqualifizierung vorkommen, die auf das Heimatpaket verweist.

Das nächste Beispiel zeigt die Zerlegung eines Systems in zwei Pakete. Auf die vollständige Darstellung der Pakete durch gesonderte Klassendiagramme sei hier noch verzichtet (vgl. aber Kap. 4.4).

**Beispiel 4.18**

Ein Liefersystem soll in die Pakete Stammdatenverwaltung und Auftragsabwicklung zerlegt werden. Das Klassendiagramm des Systems kann dann ganz grob und abstrakt wie in Abb. 4.19 notiert werden.



**Abb. 4.19.** Klassendiagramm für ein Liefersystem mit zwei Paketen.

Pakete können auch zur Strukturierung von Anwendungsfalldiagrammen benutzt werden, worauf jedoch nicht näher eingegangen wird (vgl. OESTERREICH 1998).

#### 4.1.8 Klassendiagramm und Klassenlexikon

Nach der Einführung der für die statische Modellierung benötigten Konzepte sollte hinreichend deutlich geworden sein, dass das Klassendiagramm einer Anwendung ihre zeitunabhängigen Aspekte modelliert und hierbei die Funktionssicht und die Datensicht integriert.

#### Klassendiagramm und ERM

In Kap. 2.4 wurde das Entity-Relationship-Modell (ERM) kurz angesprochen. Es stellt bei seiner Nutzung innerhalb der strukturierten Softwareentwicklung eine Anwendung aus der Datensicht dar. Ein ERM lässt sich – in erster Näherung – als Projektion eines Klassendiagramms auf die Anwendungsdaten auffassen. Dabei korrespondieren die Klassen mit den Entitätsmengen und die Assoziationen mit den Beziehungen (relationships). Man beachte, dass Assoziationen im Wesentlichen Beziehungen zwischen Anwendungsdaten darstellen (vgl. Kap. 2.1). Klassendiagramm und ERM können ggf. gleichermaßen als Ausgangspunkt für einen Datenbankentwurf dienen, der jedoch erst bei dem Entwurf der Anwendung zu erstellen ist.

Anders als ein ERM enthält ein Klassendiagramm entsprechend der Kapselung von Daten und Funktionen auch die funktionale Sicht auf eine Anwendung. Darüber hinaus bestehen weitere Unterschiede zu einem ERM. So ist wegen der in der OOA vorausgesetzten Objektidentität die Einführung *künstlicher* Schlüsselattribute *ohne* fachliche Relevanz nicht erforderlich. Beispielsweise wird man in einer Klasse Artikel ein Schlüsselattribut Artikelnummer dann und nur dann einführen, wenn dies aus der Sicht des Auftraggebers erforderlich ist, also fachlich begründet ist. Ferner werden Assoziationen im Klassendiagramm im Hinblick auf den Austausch von Botschaften modelliert.

#### Begriff des Klassen- diagramms

Jedes Diagramm, das Klassen und ggf. weitere der zuvor eingeführten Konzepte der statischen Modellierung visualisiert, wird als Klassendiagramm bezeichnet. Dies gilt z.B. auch, wenn nur eine einzige Assoziation einer Anwendung gezeigt wird. Unter *dem* Klassendiagramm einer Anwendung (im Singular) wird das vollständige Klassendiagramm verstanden. Manchmal werden hier auch die bei einer

Paketbildung entstehenden und hierarchisch geordneten Klassendiagramme einer Anwendung zusammenfassend als ihr Klassendiagramm bezeichnet.

Das Klassendiagramm wird leicht unübersichtlich, wenn es allzu viele Details enthält. Eine „Überfrachtung“ des Klassendiagramms kann durch folgende Maßnahmen vermieden werden:

- In die Attributfelder der Klassen werden nur Attributnamen und allenfalls Attributtypen eingetragen.
- In die Operationenfelder der Klassen werden nur verkürzte Signaturen eingetragen.
- Auf eine vollständige Auflistung der Attribute und Operationen der Klassen wird verzichtet. Stattdessen erscheint pro Klasse nur eine repräsentative Auswahl von Attributen und Operationen.
- In übergeordneten Paketen werden Klassen nur durch ihre Namen repräsentiert.
- Details von Assoziationen wie komplexe Restriktionen werden nicht notiert.

Durch diese Maßnahmen wird erreicht, dass die Struktur des Klassendiagramms, d.h. die Beziehungen zwischen den Klassen, für seine Leser rascher und deutlicher erkennbar wird. Ihre Anwendung wird in Kap. 4.4.1 demonstriert.

Die im Klassendiagramm ausgesparten Details sowie die Methoden bzw. Algorithmen der Operationen werden pro Klasse in einer zusätzlichen textuellen Klassenspezifikation beschrieben. Für diese schlägt Abb. 4.20 eine Schablone vor.

**Umfang des  
Klassendiagramms**

**textuelle  
Klassenspezifikation**

```
//=====
Klasse: Klassenname
// ggf. Angabe, dass die Klasse abstrakt ist.
//-----
Beschreibung:
// Kurzbeschreibung der Klasse (reale Bezugsobjekte, Zweck, Aufgabe).
//-----
Attribute:
// detaillierte Beschreibung der Attribute, vgl. Kap. 4.1.3.
//-----
Operationen:
// Signaturen der Operationen und
// Beschreibung der zugehörigen Algorithmen bzw. Methoden, vgl. Kap. 4.1.4.
//-----
Assoziationen:
// Namen der assoziierten Klassen und
// ggf. Details einzelner Assoziationen, vgl. Kap. 4.1.5.
//-----
Oberklassen:
// Angabe der Oberklassen und
// ggf. Details einzelner Vererbungsbeziehungen, vgl. Kap. 4.1.6.
//-----
Kommentar:
// ggf. erläuternde Kommentare zur Klasse, die z.B. Zusammenhänge
// mehrerer Klassenelemente betreffen können.
//=====
```

**Schablone**

**Abb. 4.20.** Schablone für die textuelle Klassenspezifikation.



**Erläuterung der Schablone**

Der Erläuterung der Schablone dienen folgende Bemerkungen (für ein umfangreiches Beispiel vgl. Kap. 4.4.2):

- Die Kurzbeschreibung einer Klasse sollte nur wenige (maximal drei) Sätze umfassen.
- In die detaillierte Attributbeschreibung gemäß Kap. 4.1.3 kann auch erläuternder Text aufgenommen werden.
- Auch in der textuellen Klassenspezifikation können Signaturen von Operationen recht lax gehandhabt werden, da im Entwurf noch mit wesentlichen Änderungen der in der OOA identifizierten Operationen zu rechnen ist. Die Basisoperationen einer Klasse sind auch in ihrer textuellen Spezifikation überflüssig. Auf die Darstellung von Methoden wurde schon eingegangen (vgl. Kap. 4.1.4).
- Als Details von Assoziationen kommen z.B. Ordnungskriterien oder Restriktionen in Betracht, die hier auch ausführlicher erläutert werden können. Auch Kardinalitäten können begründet werden. Darüber hinaus kann die Assoziation als solche im Hinblick auf den von ihr unterstützten Botschaftsfluss begründet werden. Ebenso kann argumentiert werden, warum ggf. anstelle einer (permanenten) Assoziation nur temporäre Objektverbindungen genutzt werden.
- Für jede Klasse ist ggf. die direkte Oberklasse anzugeben. Liegt ausnahmsweise eine Mehrfachvererbung vor, so sind alle Oberklassen anzugeben.
- Als Details einer Vererbungsbeziehung sind Hinweise auf Redefinitionen aufzunehmen. Wird eine Operation der Oberklasse in einer oder mehreren Unterklassen redefiniert, dann wird die redefinierte Operation nicht nochmals im Klassendiagramm als Operation der Unterklassen eingetragen. Dies gilt allerdings nur unter der Voraussetzung, dass zusätzlich eine textuelle Klassenspezifikation erstellt wird. In diesem Fall wird die Operation in der Klassenspezifikation der Unterklasse erneut aufgeführt und dort auf eine erforderliche Redefinition sowie dabei zu beachtende Bedingungen hingewiesen.

**Beispiel 4.19**

Es liege eine Oberklasse Konto mit den Unterklassen Girokonto und Sparkonto vor. Für beide Unterklassen ist eine Operation buchen() sinnvoll, die deshalb in die Oberklasse „hochgezogen“ wird. Zugleich wird in der textuellen Spezifikation der Klasse Sparkonto die Operation buchen() mit dem Hinweis aufgeführt, dass eine Redefinition gewährleisten muss, dass keine negativen Kontostände auftreten. Die Spezifikation der Klasse Girokonto wird analog behandelt.

**Klassenlexikon**

Die Gesamtheit der textuellen Spezifikationen aller Klassen des Klassendiagramms einer Anwendung bildet ihr Klassenlexikon. Die Klassen sollten im Klassenlexikon ggf. nach Paketen und pro Paket alphabetisch sortiert werden. Klassendiagramm und Klassenlexikon repräsentieren nur gemeinsam das statische Modell der Anwendung. Das Klassenlexikon bildet also eine unverzichtbare Ergänzung des Klassendiagramms.

## Übungsaufgaben zu Kapitel 4.1

### Übungsaufgabe 4.1.1

Erstellen Sie zu der folgenden Beschreibung ein Objektdiagramm.

*Der Bankkunde Ludwig ist Inhaber der Konten 12345 und 12346. Das Konto 23456 gehört zum Bankkunden Meier, während der Bankkunde Otto derzeit kein Konto besitzt.*

### Übungsaufgabe 4.1.2

Erstellen Sie eine detaillierte Attributbeschreibung anhand der folgenden Angaben zu den Attributen einer Klasse Sparkonto.

*Ein Sparkonto-Objekt umfasst Werte für folgende Attribute: Kontonummer, Bankleitzahl, der optionale Bankname, Kontostand in Euro, Datum der Eröffnung, Liste der Zeitpunkte aller bisherigen Buchungen, Liste der Buchungsbeträge aller bisherigen Buchungen. Ferner wird das Sparkonto durch seinen Typ beschrieben, wobei einmalig bei der Kontoeröffnung eine der Varianten „Student“, „Standard“ und „Exklusiv“ ausgewählt werden kann und die Variante „Standard“ am meisten nachgefragt wird. Ferner sind negative Kontostände ausgeschlossen. Die Klasse Sparkonto soll auch die aktuelle Anzahl aller Sparkonten beinhalten.*

### Übungsaufgabe 4.1.3

Geben Sie unter Beachtung der folgenden Beschreibung eine mögliche Signatur der Operation buchen() einer Klasse Sparkonto an.

*Mit der Operation buchen() können Beträge eingezahlt oder abgehoben werden. Bei einer Ausführung für ein gewisses Sparkonto-Objekt werden der Typ der Buchung (Ein- oder Auszahlung) und der Betrag in Euro übergeben. Zurückgegeben wird, ob die Buchung erfolgreich durchgeführt werden konnte oder nicht. Ferner wird der neue bzw. im Falle eines Misserfolgs der alte Kontostand in Euro bereitgestellt.*

### Übungsaufgabe 4.1.4

Modellieren Sie die Objektverbindungen der Aufgabe 4.1.1 als Assoziation. Es soll sowohl von den Bankkunden ein direkter Zugriff auf ihre Konten wie auch von einem Konto ein Zugriff auf seinen einzigen Inhaber möglich sein. Für die beteiligten Klassen sind außerdem ein bis zwei Attribute anzugeben.

### Übungsaufgabe 4.1.5

Neben dem Inhaber eines Kontos kann es mehrere Bankkunden geben, die Zeichnungsberechtigte des Kontos sind. Natürlich ist der Kontoinhaber selbst auch Zeichnungsberechtigter. Modellieren Sie beide Beziehungen zwischen Bankkunden und Konten. Beachten Sie die Angaben der Aufgabe 4.1.4 und berücksichtigen Sie zusätzlich, dass sowohl von einem Zeichnungsberechtigten ein direkter Zugriff auf seine Konten wie auch von einem Konto ein Zugriff auf seine Zeichnungsberechtigten möglich sein soll.

#### Übungsaufgabe 4.1.6

Erstellen Sie zu der folgenden Beschreibung ein Klassendiagramm, das alle vorkommenden Daten und Tätigkeiten als Attribute und Operationen geeignet gewählter Klassen berücksichtigt.

*Eine Firma beschäftigt Angestellte und freie Mitarbeiter. Für freie Mitarbeiter werden die Tätigkeit, der Name, die Adresse, die geleisteten Stunden pro Monat und der Honorarsatz (Honorar pro Stunde) erfasst. Monatlich ist das Honorar zu berechnen, das entsprechend der Bankverbindung des freien Mitarbeiters anschließend überwiesen wird. Die Adresse kann gesondert auf Adressaufkleber ausgedruckt werden. Angestellte werden mit ihrem Namen, ihrer Adresse und einer Bankverbindung erfasst, die bei der monatlichen Gehaltsüberweisung genutzt wird. Für monatlich übermittelte Gehaltsbescheinigungen oder andere Firmenmitteilungen kann die Adresse eines Angestellten auf Adressaufklebern ausgedruckt werden. Für Angestellte kann auch ein Ausweis erstellt werden, auf dem u.a. die Personalnummer und das Einstellungsdatum erscheinen. Ferner können für Angestellte aus dem Tarifurlaub und dem Resturlaub die Urlaubstage im aktuellen Jahr berechnet werden.*

#### Übungsaufgabe 4.1.7

Für ein Informationssystem eines Handelsunternehmens sollen folgende Personengruppen modelliert werden: Kunden, Mitarbeiter, Sekretärinnen und Verkäufer. Stellen Sie eine entsprechende Vererbungshierarchie in einem Klassendiagramm dar. Wählen Sie pro Klasse einige sinnvolle Attribute und Operationen. Basisoperationen sind nicht zu verwenden.

#### Übungsaufgabe 4.1.8

Erstellen Sie zu der folgenden Beschreibung ein Klassendiagramm. Achten Sie auf die richtige Zuordnung der Assoziationen, deren Richtung unbestimmt bleiben kann. Auch auf Attribute und Operationen der einzelnen Klassen kann verzichtet werden.

*Eine Supermarktkette bietet neben Standardartikeln auch leicht verderbliche Saisonartikel an. Die Lieferanten liefern jeweils mindestens einen Artikel, der zu jeder der beiden Artikelgruppen gehören kann. Jeder Artikel wird von einem oder auch mehreren Lieferanten angeboten. Standardartikel werden zunächst in einem oder mehreren Zwischenlagern aufbewahrt, die auch zeitweilig leer sein können. Saisonartikel werden direkt vom jeweiligen Lieferanten bezogen.*

#### Übungsaufgabe 4.1.9

In der Aufgabe 3.2.6 wurde ein Informationssystem zur Unterstützung der direkt kundenbezogenen Funktionen eines Reisebüros umrissen. Erstellen Sie auf der Grundlage der Aufgabenstellung und der Musterlösung ein Klassendiagramm des Informationssystems, das lediglich Pakete enthält. Berücksichtigen Sie dabei auch offensichtliche Abhängigkeitsbeziehungen zwischen den Paketen.

### Übungsaufgabe 4.1.10

Das Berliner Immobilienbüro „NobelMaxe“ benötigt ein Informationssystem zur Unterstützung seiner kundenbezogenen Aktivitäten. Erstellen Sie anhand der folgenden Beschreibung ein Klassendiagramm. Alle Assoziationen können als bidirektional angenommen werden.

*Das Immobilienbüro verkauft und verpachtet Grundstücke an seine Kunden auf der Basis von Kauf- bzw. Pachtverträgen. Während der Laufzeit eines Pachtvertrages kann ein Grundstück nicht verkauft werden. Jeder Vertrag erhält eine identifizierende Nummer und kann ausgegeben werden (Anzeige/Druck). Ein Grundstück besitzt eine Grundstücksnummer und eine Adresse, ferner eine Größe und eine Lage, aus denen der Grundstückswert ermittelt werden kann. Grundstücke werden beim Ankauf erfasst. Außerdem können eine Grundstücksbeschreibung sowie eine Liste aller noch im Angebot befindlichen Grundstücke ausgegeben werden. Die Kunden des Büros werden mit ihrem Namen, ihrer Adresse und Telefonnummer geführt und erhalten bei ihrer Erfassung eine Kundennummer. Nicht mit jedem Kunden muss es zu einem Vertragsabschluss kommen. Natürlich können sich die Daten eines Kunden auch ändern. Auch eine Liste aller Kunden kann ausgegeben werden.*

*Zu einem Kaufvertrag gehören die Grundstücksnummer, die Kundennummer, der Kaufpreis und das Datum des Vertragsabschlusses. Der Kaufpreis ist auf einmal und in voller Höhe zu zahlen. Im System sollen nach der Zahlung das Zahlungsdatum und der gezahlte Betrag als separate Vertragsdaten erfasst werden.*

*Zu einem Pachtvertrag gehören ebenfalls die Grundstücksnummer, die Kundennummer und das Vertragsdatum. Ferner die Vertragslaufzeit in Jahren sowie die jährlich zu zahlende Pacht. Für die Zahlungsweise der jährlichen Pacht wird allerdings jeweils am Jahresanfang eine gesonderte Zahlungsvereinbarung abgeschlossen, die die Zeitpunkte und Beträge der pro Jahr zu leistenden Pachtzahlungen regelt. Die erste Zahlungsvereinbarung wird sofort bei Vertragsabschluss erstellt. Pro Pachtzahlung werden Zahlungstermin und Zahlungsbetrag erfasst. Am Ende des Jahres wird pro Pachtvertrag eine Übersicht der Zahlungen erstellt.*

## 4.2 Konzepte und Diagramme für die dynamische Modellierung

Das grundlegende dynamische Konzept der objektorientierten Analyse bildet die Botschaft. Wichtige dynamische Konzepte sind ferner das Szenario und der Zustandsautomat, die mit den verschiedenen Diagrammen zu ihrer Repräsentation vorgestellt werden.

**dynamische  
Modellierung**

### 4.2.1 Botschaft

Die Kooperation verschiedener Objekte wird durch den Austausch von Botschaften vermittelt. Eine von einem Sender-Objekt an ein Empfänger-Objekt gesendete Botschaft bewirkt, dass dieses eine durch die Botschaft spezifizierte Operation

**Begriff Botschaft**

ausführt. Nach der Durchführung der Operation fällt die Kontrolle wieder an das Sender-Objekt zurück, wo die Verarbeitung fortgesetzt wird.

Von der Übermittlung einer Botschaft an ein Objekt wird auch gesprochen, wenn dieses bei der Ausführung der angeforderten Operation erst erzeugt wird. Jedes Objekt kann ferner Botschaften an sich selbst senden, d.h. während einer Operation kann eine weitere Operation des Objekts ausgeführt werden.

Mit einer Botschaft können dem Empfänger-Objekt Werte von Eingabeparametern der auszuführenden Operation zur Verfügung gestellt werden. Nach der Operation gibt das Empfänger-Objekt ggf. Werte von Ausgabeparametern bzw. einen Wert des Ergebnistyps der Operation an das Sender-Objekt zurück.

Ein Empfänger-Objekt kann eine per Botschaft geforderte Operation nur dann ausführen, wenn diese von seiner Klasse angeboten wird. Die Botschaft muss mit der Signatur einer von der Klasse des Empfänger-Objekts angebotenen Operation korrespondieren. Es müssen sowohl die Operationsnamen wie auch die Anzahl und die Typen der Parameter übereinstimmen.

#### UML-Notation

Botschaften werden daher wie die Signaturen von Operationen notiert. Sie werden in der Analyse in Interaktionsdiagramme und Zustandsdiagramme eingetragen. Meist wird analog zur Kurzschreibweise von Signaturen nur der Operationsname gefolgt von paarigen runden Klammern notiert (vgl. Kap. 4.1.4).

#### Beispiel 4.20

Die Klasse Konto besitze die Operation buchen() mit der Signatur:

buchen(in Buchungstyp : Boolean, in Betrag : Float) : Boolean

Es folgen Beispiele für verarbeitbare und nicht verarbeitbare Botschaften an ein Konto-Objekt sowie die Kurzschreibweise einer Botschaft:

buchen(true, 99.99)	// ok
buchen(77.77)	// nicht ok
abbuchen(true, 99.99)	// nicht ok
buchen()	// Kurzschreibweise

Man beachte, dass im Kontext von Diagrammen der Empfänger einer Botschaft bekannt ist und daher nicht innerhalb der Botschaft angegeben werden muss.

#### sequentielle Anwendungen

Eine Botschaft wird ausschließlich innerhalb einer Operation versendet. Daraufhin wird diese Operation unterbrochen und die durch die Botschaft spezifizierte zweite Operation ausgeführt. Erst nach deren Ende wird die aufrufende Operation fortgesetzt. Betrachtet wird hier also nur eine strikt sequentielle Ausführung einer Anwendung.

#### Botschaft ist Operationsaufruf

Botschaften entsprechen den gewöhnlichen Funktionsaufrufen der prozeduralen Programmierung. Jedoch können Operationen jeweils nur für ein Objekt bzw. nicht unabhängig von Objekten aufgerufen werden. Ferner verarbeiten die Operationen einer Klasse gemäß der Datenkapselung unmittelbar nur die Daten von Objekten dieser Klasse.

Im Zusammenhang mit Vererbung, Klassenoperationen sowie abstrakten Klassen ergeben sich weitere Aspekte des Botschaftskonzepts, auf die kurz einzugehen ist.

Empfängt ein Objekt einer Unterklasse eine Botschaft, so wird zuerst geprüft, ob die Unterklasse selbst eine Operation mit passender Signatur anbietet. In diesem Fall wird diese Operation ausgeführt. Andernfalls wird in der (direkten) Oberklasse nach einer passenden Operation gesucht, die ggf. zur Anwendung kommt. Bei mehrstufigen Hierarchien erfolgt die Suche ggf. aufsteigend bis zur höchsten Oberklasse. Diese Vorgehensweise gewährleistet, dass eine in der Unterklasse redefinierte Operation vorzugsweise ausgeführt wird. Dies ist sinnvoll, weil die redefinierte Operation an spezielle Bedürfnisse der Unterklasse angepasst ist (vgl. Kap. 4.1.6).

#### Botschaften und Vererbung

Klassenoperationen werden von einer Klasse, nicht von ihren Objekten angeboten. Um sie auszulösen, kann ein Objekt eine Botschaft an eine andere oder seine eigene Klasse senden. Die Tab. 4.6 gibt verschiedene mögliche Varianten von Sendern und Empfängern einer Botschaft an.

#### Sender/Empfänger-Varianten

Sender	Empfänger	Bemerkung
Objekt-1	Objekt-2	Ausgelöst wird Objektoperation.
Objekt	Klasse	Ausgelöst wird Klassenoperation; Klasse kann auch abstrakt sein.
Klasse-1	Klasse-2	Ausgelöst wird Klassenoperation; eine oder beide Klassen können auch abstrakt sein.

**Tab. 4.6.** Mögliche Sender und Empfänger von Botschaften.

Alle Klassen eines OOA-Modells kennen einander von vornherein gegenseitig. Jedes Objekt einer Klasse kennt jede andere Klasse ebenfalls von vornherein, nicht jedoch ihre vorhandenen Objekte. Assoziationen wie temporäre Objektverbindungen werden also nur zur Versendung von Botschaften an Objekte benötigt.

## 4.2.2 Szenario und Interaktionsdiagramme

Ein Szenario stellt eine Variante der Benutzung einer Anwendung dar. Es gliedert sich in mehrere Verarbeitungsschritte, die von einem Akteur nacheinander ausgeführt werden. Die Anwendung soll alle oder zumindest einen Teil der Verarbeitungsschritte direkt durch externe Operationen unterstützen. Szenarios werden durch Spezialisierung aus den Anwendungsfällen eines Systems gewonnen. Im Vergleich zu einem Anwendungsfall besitzt ein Szenario daher ein niedrigeres Abstraktionsniveau.

#### Begriff Szenario

Ein Szenario kann beispielsweise aus einem Anwendungsfall abgeleitet werden, indem nur dessen standardmäßiger Ablauf betrachtet wird. Sonderfälle sind dann in weiteren Szenarios zu berücksichtigen. In dem folgenden Beispiel wird der Anwendungsfall „Kfz-zurücknehmen“ des Beispiels 3.1 (vgl. Kap. 3.2) zu dem Szenario „Kfz-zurücknehmen/Standard“ spezialisiert und zugleich eine Schablone für ein Szenario gezeigt. Diese wird auch später verwendet (vgl. Kap. 4.4). Dabei wird allerdings auf eine Auflistung der Verarbeitungsschritte verzichtet, wenn sich diese ohne weiteres aus dem Anwendungsfall und der Spezialisierung des Szenarios ergeben.

#### Ableitung aus Anwendungsfall



**Beispiel 4.21****Szenario:** Kfz-zurücknehmen/Standard**Zugehöriger Anwendungsfall:** Kfz-zurücknehmen.**Spezialisierung/Bedingungen:** Standardvariante des Anwendungsfalls:

- Kfz ohne Schäden zurückgebracht,
- kein beschädigtes oder verlorenes Zubehör,
- kein Unfall,
- Rückgabetermin eingehalten,
- Mietvertragsnummer vorhanden.

**Ergebnis:** Kfz zurückgenommen und Rechnung erstellt.**Verarbeitungsschritte:**

- 1 Mietvertrag anhand Mietvertragsnummer suchen.
- 2 Einhaltung Mietvertrag prüfen, darunter:
  - vollständige Rückgabe aller vermieteten Gegenstände,
  - Termineinhaltung,
  - Kfz-Schäden.
- 3 Abrechnungsdaten erfassen (km-Stand und Kraftstoffverbrauch).
- 4 Rechnung erstellen, ausdrucken und an Kunde übergeben.

**Interaktions-  
diagramme**

Szenarios werden in der dynamischen Modellierung durch verschiedene Interaktionsdiagramme repräsentiert. Diese veranschaulichen die durch Botschaften gesteuerte, kooperative Ausführung eines Szenarios durch Operationen von Objekten verschiedener Klassen.

**Sequenzdiagramm**

Das wichtigste Interaktionsdiagramm ist das Sequenzdiagramm. Charakteristisch für ein Sequenzdiagramm ist, dass die beteiligten Objekte in der Horizontalen des Diagramms erscheinen, während in der Vertikalen des Diagramms die Zeit von oben nach unten fortschreitet. Die Abb. 4.21 zeigt die wesentlichen Elemente und die Notation eines Sequenzdiagramms.

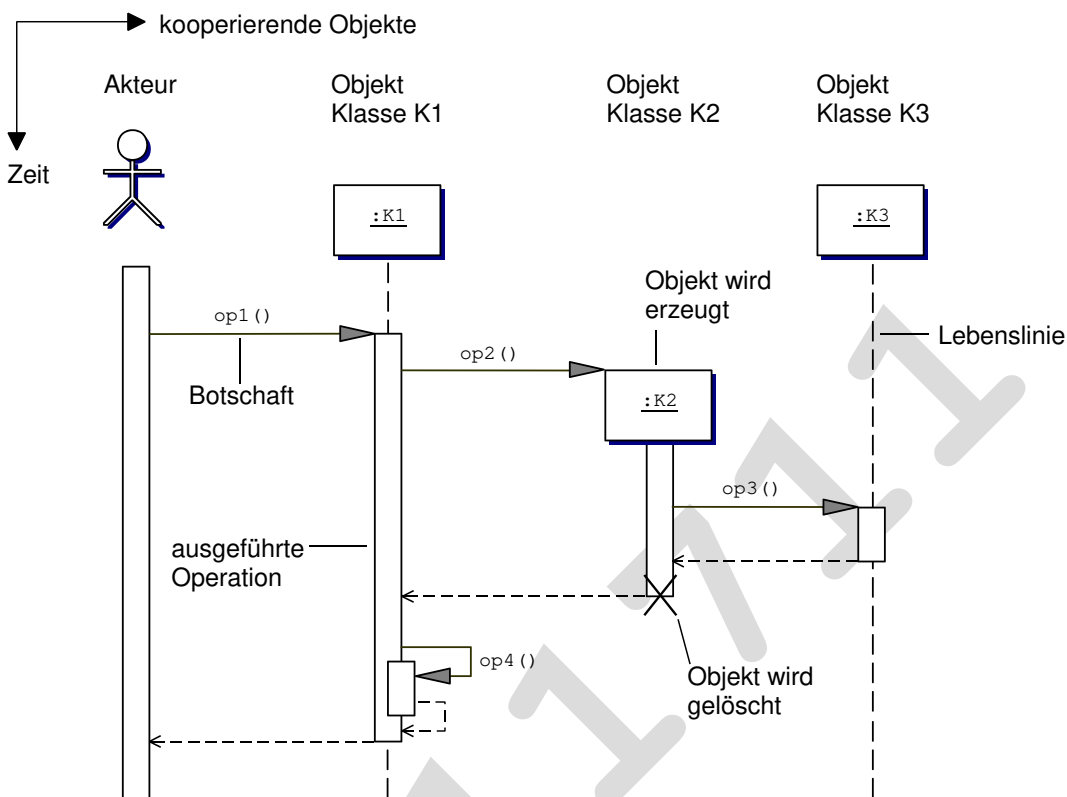
Die Elemente eines Sequenzdiagramms seien wie folgt erläutert:

**Erläuterung der  
Elemente**

- Links oben im Diagramm erscheint der Akteur des Szenarios, dessen jederzeitige Interaktionsbereitschaft der darunterliegende Balken ausdrückt.
- Rechts neben dem Akteur werden diejenigen beteiligten Objekte eingetragen, welche während des gesamten Szenarios vorhanden sind. In Abb. 4.21 sind dies die Objekte :K1 und :K3. Objekte werden anonym notiert, wenn sie – wie meist der Fall – stellvertretend für alle oder eine Teilmenge der vorhandenen Objekte ihrer Klasse stehen.
- Weitere beteiligte Objekte werden ggf. erst im Verlauf eines Szenarios erzeugt. Diese werden weiter unten im Diagramm rechts neben der Botschaft eingetragen, die ihre Erzeugung auslöst. In Abb. 4.21 trifft dies für das Objekt :K2 zu.
- Unter jedem Objekt beginnt seine gestrichelte Lebenslinie, die seine Existenz anzeigt und nur abbricht, wenn das Objekt gelöscht wird. Das Löschen eines Objekts wird gesondert durch ein Kreuz am Ende der jeweiligen Destruktorooperation markiert.



- Operationen eines Objekts werden durch Balken veranschaulicht, die seine Lebenslinie unterbrechen. Sie werden stets durch Botschaften ausgelöst. Zusätzlich zum Text einer Botschaft wird diese durch einen beschrifteten Pfeil dargestellt. Der Pfeil endet bei dem zugehörigen Operationsbalken.



#### Elemente und UML-Notation

Abb. 4.21. Elemente und Notation eines Sequenzdiagramms.

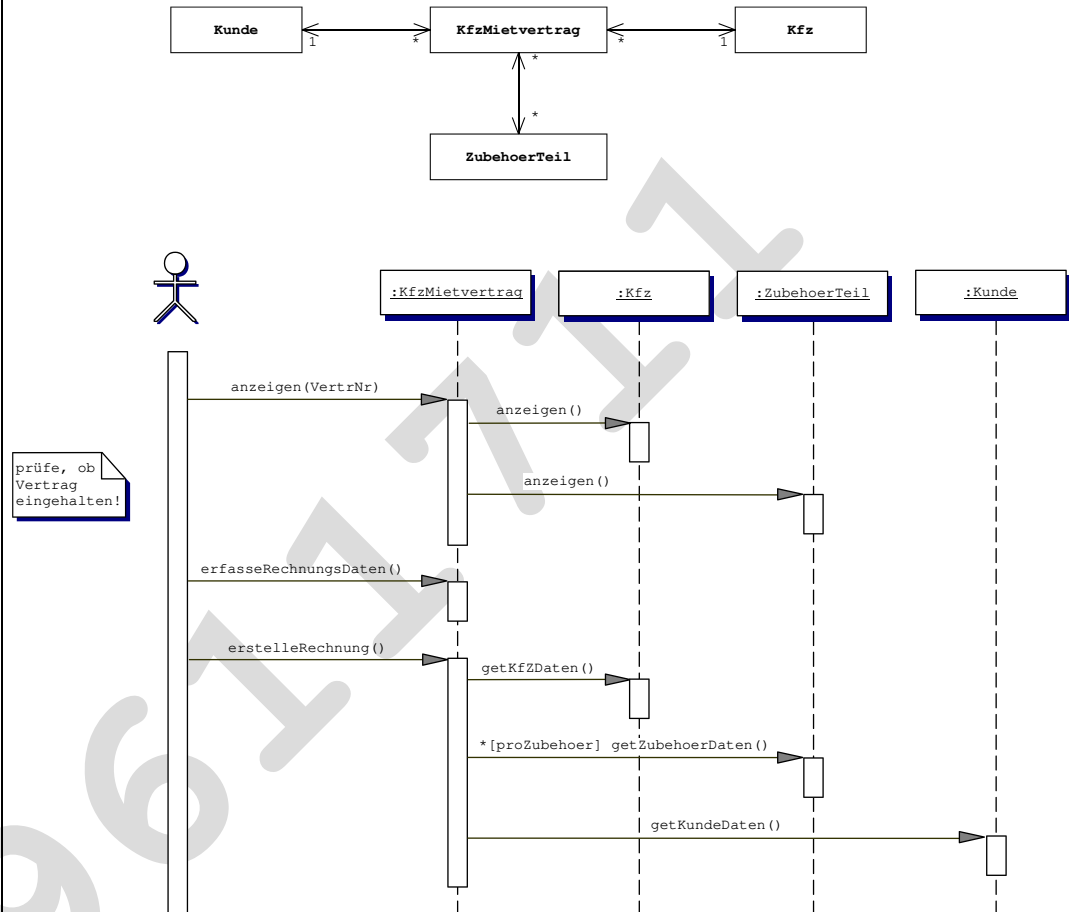
- Ruft eine Operation eines Objekts eine andere Operation desselben Objekts auf, so wird deren Balken innerhalb des Balkens der aufrufenden Operation, aber seitlich versetzt eingetragen (vgl. Operation op4()).
- Die Rückkehr der Kontrolle an die aufrufende Operation nach der Ausführung einer aufgerufenen Operation wird in Abb. 4.21 durch die gestrichelten Pfeile verdeutlicht, die aber in ein Sequenzdiagramm nicht eingetragen werden müssen, wenn – wie hier – nur sequentielle Anwendungen betrachtet werden.
- Zusätzlich kann eine bedingte und eine wiederholte Ausführung von Operationen ausgedrückt werden. Wird eine Botschaft gemäß [bedingung] operation() spezifiziert, so wird die Operation nur ausgeführt, wenn die in eckigen Klammern frei notierte Bedingung erfüllt ist. Wird eine Botschaft gemäß \* operation() oder \*[bedingung] operation() notiert, so wird die Operation wiederholt ausgeführt. Die Anzahl der Wiederholungen ist bei der ersten Variante unbestimmt und wird bei der zweiten Variante durch die angegebene Bedingung gesteuert (vgl. das folgende Beispiel 4.22).
- Sind in einem Szenario Klassenoperationen auszuführen, so wird als Ausführungseinheit anstelle eines anonymen Objekts die jeweilige Klasse selbst (durch Klassennamen oder ersatzweise durch all:Klassenname) angegeben. Für ein Beispiel siehe Kap. 4.4.3, Abb. 4.34.

Wird ein Sequenzdiagramm zur vollständigen Darstellung eines Szenarios benutzt, so werden alle Objektoperationen letztlich durch eine oder mehrere externe

Operationen angestoßen, die wiederum von einem Akteur ausgelöst werden. Daneben können mit Sequenzdiagrammen auch Auszüge von Szenarios abgebildet werden, wobei dann ein Akteur bzw. externe Operationen fehlen können.

### Beispiel 4.22

Die Abb. 4.22 zeigt ein Sequenzdiagramm zum Szenario „Kfz-zurücknehmen/Standard“ und vorab ein Klassendiagramm der relevanten Klassen.



**Abb. 4.22.** Sequenzdiagramms für das Szenario „Kfz-zurücknehmen/Standard“.

Unterstellt sei, dass die Klassen Kfz, KfzMietvertrag und ZubehörTeil je eine Operation anzeigen() zur Anzeige der Objektdaten besitzen. Ferner werden im Sequenzdiagramm mehrere Leseoperationen (get-Operationen) für Attribute der Klassen Kfz, Zubehörteil und Kunde jeweils zu einer Operation getxxxDaten() (xxx = Klassenname) zusammengefasst. Dieses abkürzende Vorgehen dient der Übersichtlichkeit und wird auch bei späteren Sequenzdiagrammen angewendet. Schließlich sollen die Abrechnungsdaten km-Stand und Benzinverbrauch im jeweiligen KfzMietvertrag-Objekt abgelegt werden.

#### UML-Element Notiz

Den Verarbeitungsschritten des Szenarios entsprechen im Sequenzdiagramm externe Operationen. Zusätzlich ist eine Notiz eingetragen, die auf die Prüfung der Vertragseinhaltung hinweist. Das bisher nicht erläuterte UML-Element Notiz wird durch ein Blatt mit Eselsohr dargestellt, das beliebigen Text beinhalten kann. Es wird ausschließlich zur Kommentierung in beliebigen Diagrammen verwendet.

Beim Aufruf der nicht externen Operationen werden generell die im Klassendiagramm ausgewiesenen Assoziationen der Klasse KfzMietvertrag benutzt. Bereits

bei der Erstellung eines Mietvertrag-Objekts werden die erforderlichen Verbindungen zu den anderen Objekten hergestellt, die dann wie oben gezeigt zur Botschaftsübermittlung genutzt werden können. Nur weil entsprechende Objektverbindungen bereits bestehen, können die Operationen der anderen Objekte ohne weiteres aufgerufen werden.

Die Umsetzung des vierten Szenarioschrittes sei stellvertretend näher betrachtet. Die externe Operation `erstelleRechnung()` ruft nacheinander die Operationen `getKfzDaten()`, `getZubehoerDaten()` und `getKundeDaten()` auf, die jeweils für eine Rechnungserstellung benötigte und nicht im `KfzMietvertrag`-Objekt selbst vorhandene Daten bereitstellen. Die Operation `getZubehoerDaten()` wird pro Zubehörteil (also evtl. auch gar nicht) einmal aufgerufen. Die erstellte Rechnung geht als Anzeige und/oder Ausdruck an den Akteur zurück. Dies könnte im obigen Diagramm durch eine Notiz am Ende der Operation `erstelleRechnung()`, evtl. in Verbindung mit einem gestrichelten Rückgabepfeil, explizit ausgedrückt werden.

In Sequenzdiagrammen sind verschiedene Konsistenzbedingungen einzuhalten:

- Es können nur Objekte von Klassen vorkommen, die im Klassendiagramm vorhanden sind.
- Pro Objekt bzw. Klasse dürfen nur Operationen bzw. Klassenoperationen aufgerufen werden, die im Klassendiagramm oder im Klassenlexikon eingetragen sind. Dies gilt allerdings nicht für die immer vorhandenen Basisoperationen, die teils explizit in Sequenzdiagrammen erscheinen. Dies war – wenn auch in modifizierter Weise – im zuvor gezeigten Diagramm der Fall.
- Eine Botschaft kann von einem Sender-Objekt A an ein Empfänger-Objekt B nur versendet werden, wenn eine Objektverbindung zwischen A und B, genauer von A nach B, besteht.

**Konsistenz-  
bedingungen**

Im Zusammenhang mit der letzten Konsistenzbedingung lassen sich verschiedene Fälle unterscheiden:

- Gemäß einer Assoziation wurde eine Objektverbindung von A nach B bereits früher hergestellt und aufbewahrt. Dann kann A ohne weiteres eine Botschaft an B senden. So basiert im letzten Beispiel das Lesen der Daten eines Kfz-Objekts durch ein Mietvertrag-Objekt auf der Assoziation zwischen den Klassen `Mietvertrag` und `Kfz` und einer früher erstellten Objektverbindung.
- Es besteht noch keine Objektverbindung von A nach B, aber das Objekt B existiert bereits. Dann kann es durch eine Suchoperation ermittelt werden, die eine Klassenoperation der Empfänger-Klasse ist. Das Objekt A sendet zuerst eine Botschaft zum Aufruf der Suchoperation an die Empfänger-Klasse, wobei das gesuchte Objekt B durch Parameter identifiziert wird. Ein Kfz-Objekt kann z.B. durch seine Kfz-Nummer identifiziert werden. Nach der Rückgabe einer Referenz auf das Objekt B kann A eine Botschaft an B senden.
- Alternativ kann eine Objektverbindung mittels einer externen Operation hergestellt werden, bei der der Akteur in einer Objektliste der Empfänger-Klasse das passende Objekt (in einer fertigen Anwendung etwa per Mausklick) identifiziert. Daraufhin wird die Referenz bereitgestellt. Im Sequenzdiagramm kann diese Variante durch Aufruf einer Klassenoperation der Empfänger-Klasse zur Ausgabe einer Liste aller Objekte notiert werden (vgl. Übungsaufgabe 4.2.1).
- Schließlich kann durch eine Botschaft ein Empfänger-Objekt B erst erzeugt werden. Der Aufruf eines Konstruktors der Empfänger-Klasse ist – analog zu

**Botschaften und  
Objektverbindungen**

### Kollaborationsdiagramme

einer Klassenoperation – ohne weiteres von dem Objekt A aus möglich (vgl. Übungsaufgabe 4.2.2). Ein Konstruktor gibt stets eine Referenz auf das erzeugte Objekt zurück.

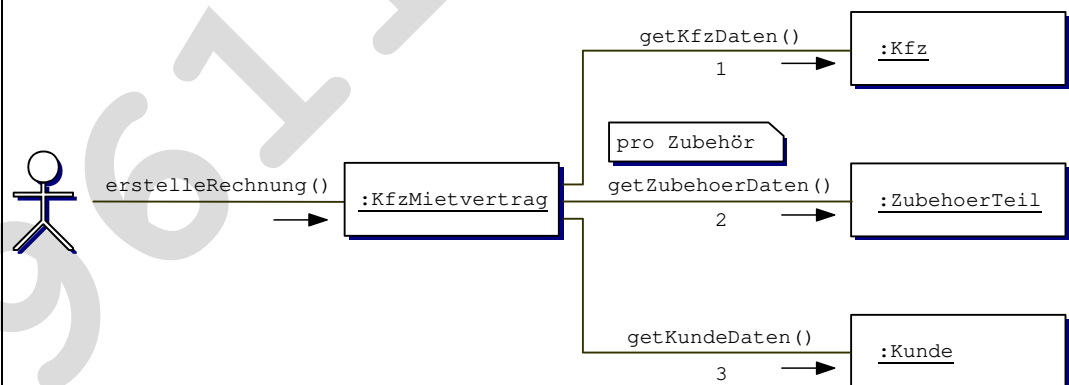
Die Varianten zur Herstellung einer Objektverbindung werden ebenso für temporäre wie für dauerhafte Objektverbindungen (Assoziationen) verwendet. Im letzteren Fall werden beschaffte Referenzen (mittels link-Operation) aufbewahrt.

Neben Sequenzdiagrammen bietet die UML auch Kollaborationsdiagramme zur Darstellung der Interaktion von Objekten an. Ein Kollaborationsdiagramm veranschaulicht vor allem die Verbindungen von Objekten. Der zeitliche Aspekt wird nicht betont. Die Reihenfolge ausgeführter Operationen wird lediglich durch eine Nummerierung verdeutlicht. Ein Kollaborationsdiagramm stellt nur die zu einem Verarbeitungsschritt eines Szenarios bzw. einer externen Operation gehörende Objektkooperation dar.

Ein Sequenzdiagramm mit nur einer externen Operation und ein korrespondierendes Kollaborationsdiagramm besitzen denselben Informationsgehalt. Daher können Sequenzdiagramme und Kollaborationsdiagramme auch werkzeuggestützt ineinander überführt werden. Im Folgenden wird nur das Sequenzdiagramm als der wichtigere Typ von Interaktionsdiagrammen verwendet.

#### Beispiel 4.23

Die Abb. 4.23 zeigt exemplarisch ein Kollaborationsdiagramm für den Schritt der Rechnungserstellung des Szenarios „Kfz-zurücknehmen/Standard“.



**Abb. 4.23.** Kollaborationsdiagramm für die Rechnungserstellung im Szenario „Kfz-zurücknehmen/Standard“.

### 4.2.3 Zustandsautomat und Zustandsdiagramm

#### zustandsabhängiges Verhalten

Eine Modellierung aus der Zustandssicht ist erforderlich, wenn Objekte ein zustandsabhängiges Verhalten besitzen. Dieses liegt vor, wenn die Ausführbarkeit oder Wirkung von Operationen von den Zuständen der Objekte abhängt.

#### Beispiel 4.24

Jedes Objekt einer Klasse Artikel soll eine gewisse Artikelart mit Attributen wie Bezeichnung und Preis abbilden. Nach seiner Erfassung kann ein Artikel-Objekt jederzeit auf gleiche Weise modifiziert werden. Eine Löschoperation beendet ggf.

seine Existenz. Während ein Artikel-Objekt existiert, besitzt es im Hinblick auf Ausführbarkeit und Wirkung von Operationen nur einen Zustand.

Anders verhält es sich bei Objekten einer Klasse Dienstreise. Eine Dienstreise wird zunächst beantragt und danach geprüft. Bei ihrer Bewilligung erfolgt nach der Reise eine Abrechnung. Schließlich werden abgerechnete Dienstreisen ausgewertet und archiviert. Offenbar liegt ein zustandsabhängiges Verhalten vor. So kann eine noch nicht bewilligte Dienstreise nicht abgerechnet und erst abgerechnete Dienstreisen können ausgewertet werden.

Zur Modellierung des zustandsabhängigen Verhaltens von Objekten wird in der OOA das Konzept des Zustandsautomaten verwendet. Dieses wurde außerhalb der Objektorientierung entwickelt und geht u.a. auf HAREL (1987) zurück; vgl. hierzu auch BALZERT (1996). Zustandsautomaten können auch unabhängig von einer objektorientierten Modellierung bzw. Anwendungsentwicklung zur Beschreibung technischer Systeme wie z.B. Geldautomaten dienen. Hier werden nur die Grundzüge des von der OOA adaptierten Konzepts des Zustandsautomaten betrachtet. Weitergehende Ausführungen finden sich in RUMBAUGH (1993) und OESTEREICH (1998).

Ein Zustandsautomat besteht aus einer endlichen Menge von Zuständen sowie aus einer endlichen Menge von Zustandsübergängen, die als Transitionen bezeichnet werden. Im OOA-Kontext beschreibt ein Zustandsautomat das zustandabhängige Verhalten von Objekten einer Klasse.

**Begriff**  
**Zustandsautomat**

Jedes Objekt befindet sich stets in genau einem Zustand. Ein Zustand besitzt eine bestimmte fachliche Bedeutung. Er währt eine gewisse Zeit, in der das Objekt auf ein Ereignis wartet. Dieses löst eine Transition, d.h. einen Übergang zu einem eindeutig bestimmten Folgezustand aus. Der Folgezustand ist meist vom Ausgangszustand verschieden; er kann aber auch der Ausgangszustand sein. Eine Transition und das auslösende Ereignis beanspruchen selbst keine Dauer. Die Lebenszeit eines Objekts ist lückenlos durch Zeitintervalle ausgefüllt, in denen das Objekt verschiedene Zustände annimmt.

**Zustand, Transition,**  
**Ereignis**

In einem Zustand kann das Objekt im Allgemeinen auf verschiedene Ereignisse reagieren. Jedem Ereignis, auf das reagiert werden kann, entspricht eine andere Transition. Die Menge der jeweils ausführbaren Transitionen wie auch die Wirkung einer Transition ist zustandsabhängig. Kann das Objekt im gegebenen Zustand auf ein Ereignis nicht reagieren, so wird es ignoriert. So kann etwa im obigen Beispiel die Operation auswerten() nur im Zustand „abgerechnet“ einer Dienstreise ausgeführt werden.

Ereignisse können von verschiedener Art sein. Ein Ereignis kann eine Botschaft an das Objekt sein. Ein Ereignis kann darin bestehen, dass eine zuvor nicht erfüllte Bedingung erfüllt wird. Ereignisse können ferner zeitlicher Natur sein. Ein Ereignis kann darin bestehen, dass ein gewisser Zeitpunkt erreicht wurde oder eine gewisse Zeitspanne abgelaufen ist. Ein Ereignis kann auch mehrere Teilereignisse umfassen, die durch logische Operatoren (and, or) verknüpft sind.

**Arten von Ereignissen**

Ein Zustandsautomat kennt zweierlei Arten von Tätigkeiten, nämlich Aktivitäten und Aktionen. Eine Aktivität beansprucht Zeit und kann nur während eines Zustands ausgeführt werden. Eine Aktion dauert nicht. Aktionen können sowohl während eines Zustands durchgeführt werden wie auch mit Transitionen verbunden sein. Operationen, die infolge einer Botschaft ausgeführt werden und eine

**Aktionen, Aktivitäten**

Transition begleiten, stellen Aktionen dar. Da sie keine Zeit erfordert, wird eine Aktion stets vollständig ausgeführt, kann also nicht unterbrochen werden. Aktivitäten können hingegen durch Ereignisse abgebrochen werden.

In der Tab. 4.7 werden einige weitere Begriffe im Zusammenhang mit Zustandsautomaten eingeführt.

#### weitere Begriffe zum Zustandsautomat

Begriff	Erläuterung
Startzustand	Jeder Zustandsautomat besitzt genau einen Startzustand. Dieser ist ein Pseudozustand, in dem das Objekt noch nicht existiert, sondern von dem aus es erzeugt wird und in seinen ersten „echten“ Zustand übergeht. Der Startzustand dient lediglich Darstellungszwecken (s. Abb. 4.24).
Endzustand	Aus einem Endzustand führen keine Transitionen heraus. Ein Zustandsautomat kann mehrere Endzustände besitzen. Darunter kann sich ein Endzustand befinden, der wiederum ein Pseudozustand ist und die Vernichtung des Objekts markiert (s. Abb. 4.24).
entry-Aktion	Eine entry-Aktion ist eine mit einem Zustand verbundene Aktion. Sie wird stets ausgeführt, unmittelbar nachdem der Zustand durch eine beliebige Transition erreicht wird.
exit-Aktion	Eine exit-Aktion ist eine mit einem Zustand verbundene Aktion. Sie wird stets ausgeführt, unmittelbar bevor der Zustand durch eine beliebige Transition verlassen wird.
implizites Ereignis	Pro Zustand kann es eine Transition geben, der kein Ereignis explizit zugeordnet ist. In diesem Fall stellt das autonom erreichte Ende der Verarbeitung (Aktivität) innerhalb des Zustands ein sogenanntes implizites Ereignis dar. Dieses löst die Transition (ohne explizites Ereignis) aus.
Ereignis mit Zusatzbedingung	Ein Ereignis kann mit einer Zusatzbedingung versehen werden. In diesem Fall löst das Ereignis eine Transition genau dann aus, wenn auch die Zusatzbedingung erfüllt ist, die quasi das Ereignis überwacht. Äquivalent ist eine and-Verknüpfung des Ereignisses mit der Zusatzbedingung.

**Tab. 4.7.** Einige Begriffe zum Konzept des Zustandsautomaten.

#### Objektlebenszyklus

Ein Zustandsautomat für eine Klasse wird auch als Objektlebenszyklus der Klasse bezeichnet. Er wird der Klasse und nicht ihren Objekten zugeordnet, weil die möglichen Zustände, Transitionen und deren auslösende Ereignisse für alle Objekte übereinstimmen. Alle Objekte besitzen insofern denselben Objektlebenszyklus. Allerdings können sich mehrere Objekte zu einem Zeitpunkt in verschiedenen Zuständen befinden. Im Allgemeinen muss auch die Folge der durchlaufenen Zustände während der Objektlebenszeit nicht übereinstimmen. Man denke etwa an Schleifen im Objektlebenszyklus, die infolge unterschiedlicher Ereignisse von einigen Objekten durchlaufen werden, von anderen nicht.

#### Typen von Objektlebenszyklen

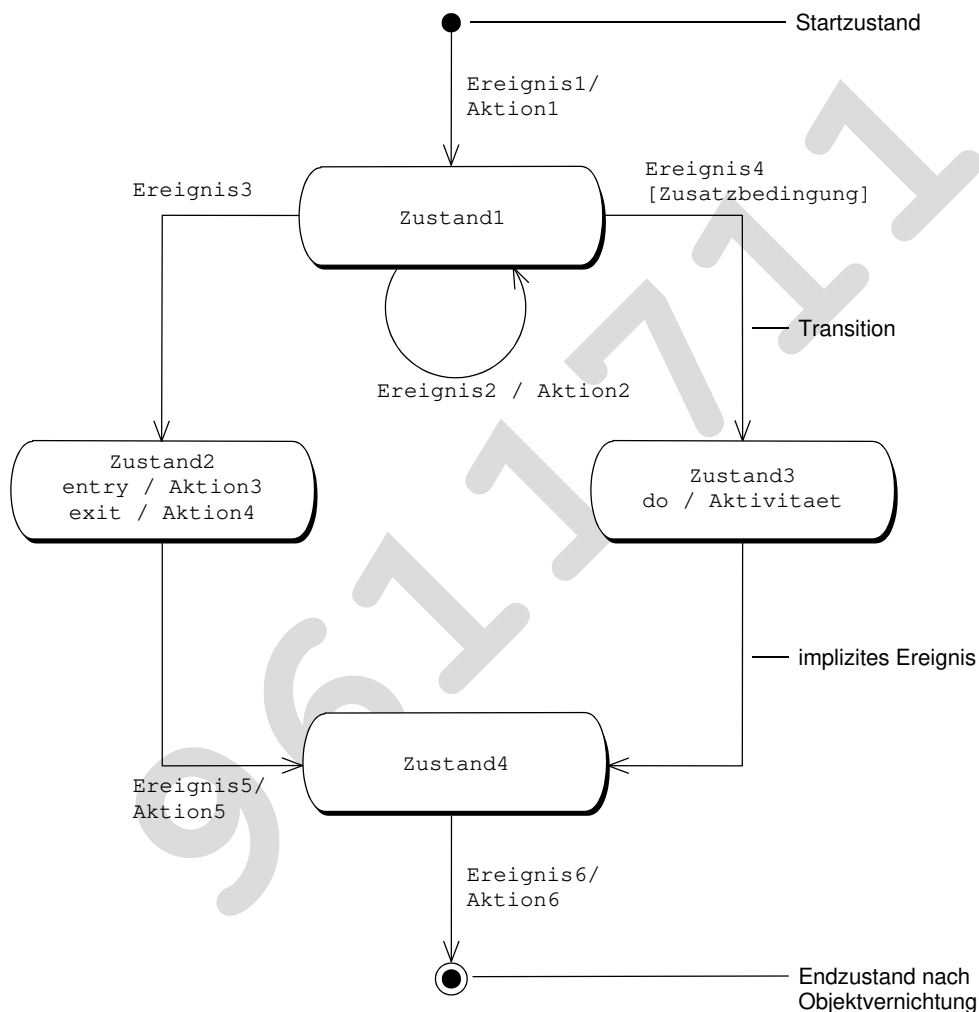
Unterscheiden lassen sich Objektlebenszyklen ohne und mit (mindestens einem) Endzustand. Ein Endzustand kann, muss aber nicht mit einer Vernichtung des Objekts gekoppelt sein (vgl. Tab. 4.7). Ferner muss – man denke wieder an Schleifen – ein vorhandener Endzustand nicht unbedingt erreicht werden. Ein besonders einfacher Lebenszyklus mit Endzustand liegt vor, wenn die möglichen Zustände eine sequentielle Zustandsfolge vom ersten Zustand bis zum Endzustand bilden.



Ein Zustandsautomat wird durch ein Zustandsdiagramm visualisiert. Die Abb. 4.24 zeigt die Notation eines Zustandsdiagramms. Aktivitäten und zuvor erläuterte besondere Aktionen sind mit den angegebenen Schlüsselworten zu verstehen. Zustände müssen keinen Namen besitzen. Anonyme Zustände sollten jedoch vermieden werden. Mehrere anonyme Zustände in einem Zustandsdiagramm gelten als verschieden. Mehrere gleich benannte Zustände sind hingegen identisch. Sind ein Ereignis und die transitionsbegleitende Aktion redundant, muss nur das Ereignis oder die Aktion notiert werden. Sind Aktionen Operationen einer Klasse, werden sie ggf. durch den Klassennamen qualifiziert. Gäbe es in Abb. 4.24 den Endzustand nach Objektvernichtung nicht, so wäre Zustand 4 ein Endzustand.

### Zustandsdiagramm

### UML-Notation Zustandsdiagramm



**Abb. 4.24.** Notation eines Zustandsdiagramms.

Der Zustand eines Objekts wurde in Kap. 4.1.1 als die Gesamtheit der Attributwerte sowie der Verbindungen des Objekts bestimmt. Die Klasse Artikel im Beispiel 4.24 besitzt nach dieser Definition beliebig viele Zustände, weil beliebig viele Kombinationen von Attributwerten möglich sind. Die Klasse besitzt aber im Sinne eines Zustandsautomaten nur einen Zustand. Jedoch besteht nur scheinbar ein Widerspruch. Zu beachten ist, dass bei einem Zustandsautomaten nur solche Zustände als verschieden gelten, die ein unterschiedliches Verhalten des Objekts nach sich ziehen. Ein Zustand eines Objektlebenszyklus sei hier auch als Verhaltenszustand, ein Zustand im Sinne des Objektbegriffs als Objektzustand bezeichnet. Dann entsprechen jedem Verhaltenszustand ein oder mehrere (evtl. auch un-

### Objektzustand vs. Verhaltenszustand



endlich viele) Objektzustände. Anders gesagt repräsentiert jeder Verhaltenszustand eine Klasse verhaltensäquivalenter Objektzustände.

#### Konsistenzregeln

Die Elemente des Objektlebenszyklus einer Klasse sollten soweit wie möglich durch Eigenschaften (Operationen, Attribute) der Klasse ausgedrückt werden. In einem Zustandsdiagramm sind folgende Konsistenzregeln bezüglich der Klasse bzw. des Klassendiagramms zu beachten (vgl. BALZERT 1999):

- Aktionen und Aktivitäten sollten Operationen der Klasse sein und werden entsprechend durch eine Signatur der Operation angegeben. Ausnahmen ergeben sich z.B., wenn Aktivitäten durch Personen („manuell“) abgewickelt werden (vgl. das folgende Beispiel). Für Ereignisse, die Botschaften sind, gilt eine analoge Forderung.
- Pro Zustand sind alle anwendbaren Operationen zu berücksichtigen. Dies gilt für die vom Zustand ausgehenden Transitionen ebenso wie für die zustandsinterne Verarbeitung. Erscheint eine Operation im Zusammenhang mit einem Zustand nicht, so gilt sie als nicht anwendbar, d.h. ihre Anwendung bleibt wirkungslos.
- Steht eine Operation für mehrere Zustände zur Verfügung, so können die Art und Weise ihrer Ausführung und ihr Resultat zustandsabhängig sein. Details sollten bei der Spezifikation der Operation im Klassenlexikon notiert werden.
- Ereignisse, die Bedingungen oder auch zeitlicher Natur sind, sollten möglichst unter Verwendung von Attributen der Klasse formuliert werden.

Allerdings werden Elemente eines Zustandsdiagramms oft noch nicht durch Eigenschaften der betrachteten Klasse beschrieben. So werden Ereignisse teils frei formuliert bzw. rein textuell angegeben. Verhaltenszustände werden meist erst im Entwurf auf Eigenschaften von Klassen abgebildet. In einfachen Fällen können Verhaltenszustände bereits in der OOA durch ein Statusattribut der Klasse ausgedrückt werden, dessen Wert den aktuellen Zustand eines Objekts angibt. Bei der Klasse Dienstreise kann ein Statusattribut z.B. angeben, ob eine Dienstreise erst beantragt oder bereits bewilligt ist (für ein Beispiel vgl. auch Kap. 4.4.2).

#### Verfeinerung von Zustandsdiagrammen

Nur erwähnt sei, dass Zustandsdiagramme verfeinert werden können. Eine Verfeinerung setzt bei einem einzelnen Zustand an, dessen interne Verarbeitung wiederum in einem zusätzlichen Zustandsdiagramm beschrieben wird. Eine Verschachtelung von Zustandsdiagrammen ist über mehrere Ebenen hinweg möglich.

#### Beispiel 4.25

In der Abb. 4.25 wird die Klasse Dienstreise und ihr Zustandsdiagramm gezeigt. Das Klassenattribut Limit gibt an, ab welcher Höhe der erwarteten Kosten eine Dienstreise zustimmungspflichtig ist. Der Zustand „in Prüfung“ wird durch ein implizites Ereignis – nämlich das autonom erreichte Ende der manuellen Prüfung – verlassen. Daher wird die anschließende Verzweigung mittels einer Zusatzbedingung realisiert. Auf die Einbeziehung weiterer manueller Aktivitäten, etwa bei der Antragstellung, wurde verzichtet. Neben durchgeführten Dienstreisen werden auch abgelehnte Anträge ausgewertet. Es liegt ein Objektlebenszyklus mit einem Endzustand vor, wobei ein Dienstreise-Objekt nicht gelöscht wird.

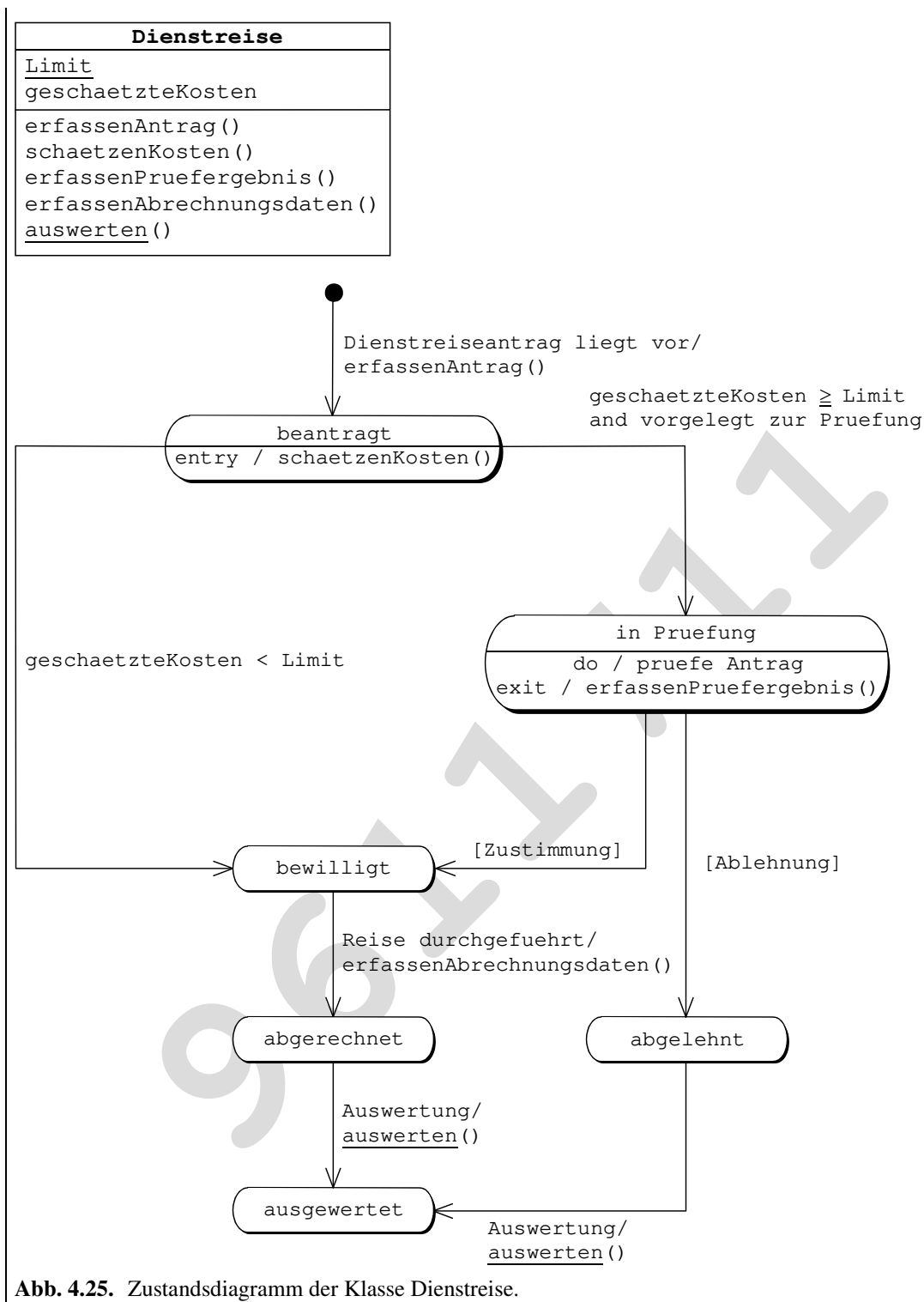


Abb. 4.25. Zustandsdiagramm der Klasse Dienstreise.

Aktivitätsdiagramme können als eine Sonderform von Zustandsdiagrammen aufgefasst werden. Sie gestatten die Spezifikation komplexer Verarbeitungsabläufe. Hervorzuheben ist, dass neben sequentiellen auch nebenläufige Verarbeitungen darstellbar sind. Aktivitätsdiagramme können in der OOA bereits bei der Spezifikation von komplexen Anwendungsfällen, in der dynamischen Modellierung wie auch im Entwurf verwendet werden. In diesem Kurs werden Aktivitätsdiagramme jedoch nicht benutzt.

#### Aktivitätsdiagramm

## Übungsaufgaben zu Kapitel 4.2

### Übungsaufgabe 4.2.1

Erstellen Sie auf der Grundlage der Aufgabenstellung und der Musterlösung der Aufgabe 4.1.10 ein Sequenzdiagramm für die Ausgabe eines Pachtvertrages. Jährliche Zahlungsvereinbarungen sind nicht auszugeben.

### Übungsaufgabe 4.2.2

Erstellen Sie auf der Grundlage der Aufgabenstellung und der Musterlösung der Aufgabe 4.1.10 ein Sequenzdiagramm für die Erfassung eines Pachtvertrages.

Setzen Sie zusätzlich voraus, dass die Klasse Vertrag eine Klassenoperation `getNaechsteVertragsNr()` bereitstellt, die für einen neu zu erfassenden Vertrag eine Nummer liefert. Beachten Sie, dass nach der Erstellung eines `PachtVertrag`-Objekts zunächst keine Objektverbindungen zum zugehörigen Kunden bzw. Grundstück bestehen. Richten Sie diese Objektverbindungen ein und nutzen Sie hierzu neben den Klassenoperationen der Klassen `Kunde` und `Grundstueck` die Basisoperationen `setlinkKunde()` und `setlinkGrundstueck()` der Klasse `Vertrag`.

### Übungsaufgabe 4.2.3

Erstellen Sie auf der Grundlage der Aufgabenstellung und der Musterlösung der Übungsaufgabe 4.1.10 ein Zustandsdiagramm für die Klasse `Grundstueck`.

## 4.3 Vorgehensweise der objektorientierten Analyse

### methodische Vorgehensweise

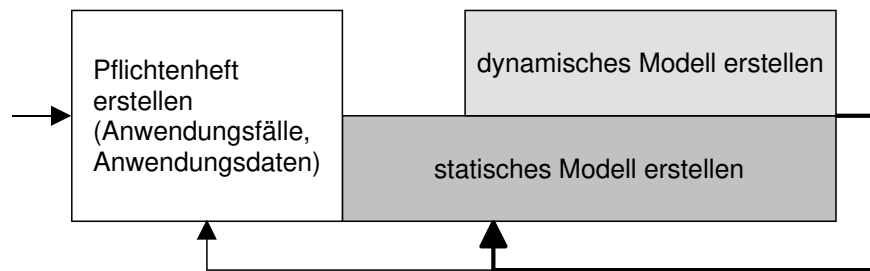
Die im Folgenden vorgestellte methodische Vorgehensweise der objektorientierten Analyse lehnt sich an BALZERT (1999) an. Sie umfasst einerseits eine Strategie, die den Ablauf der Entwicklung eines OOA-Modells grob beschreibt, und andererseits taktische Hinweise und Regeln zur Anwendung der einzelnen Konzepte. Interessierte seien auch auf HRUSCHKA (1998) hingewiesen.

### 4.3.1 Strategie der Erstellung eines OOA-Modells

#### Makroprozess ...

Das OOA-Modell einer Anwendung sollte möglichst systematisch und planmäßig entwickelt werden. Insbesondere sollte der grobe Ablauf einer Entwicklung einer definierten Strategie folgen, die auch als Makroprozess bezeichnet wird. Hier wird der in Abb. 4.26 veranschaulichte Makroprozess zugrunde gelegt.

#### Skizze und ...



**Abb. 4.26.** Makroprozess der Erstellung des OOA-Modells.

Im Einzelnen ist der Makroprozess durch folgende Aspekte charakterisiert:

- Die Entwicklung des OOA-Modells basiert auf der Spezifikation der Anwendungsfälle des geplanten Systems und der groben Beschreibung der Anwendungsdaten. Diese Bestandteile des Pflichtenheftes bilden den Ausgangspunkt für das OOA-Modell. Zur Spezifikation der Anwendungsfälle im Pflichtenheft gehört auch ein Anwendungsfalldiagramm.
- Die statische Modellierung bestimmt die Klassen der Anwendung. Ihre an den realen Objekten des Anwendungsbereiches orientierte Festlegung bildet die Grundlage für ein möglichst stabiles Modell (vgl. Kap. 2.4.2). Die dynamische Modellierung betrachtet eher das Zusammenwirken von Operationen als ihre Verteilung auf Klassen. Sie erscheint daher für eine Identifizierung der Klassen weniger geeignet. Aus diesem Grunde beginnt die statische Modellierung vor der dynamischen Modellierung, die bei ihrem Start bereits ein erstes Klassendiagramm vorfinden sollte.
- Andererseits werden die statische und die dynamische Modellierung zeitlich weitgehend parallel ausgeführt, weil sie sich inhaltlich ergänzen. Anhand des dynamischen Modells kann das statische Modell geprüft, korrigiert und erweitert werden. Häufig werden erst bei der dynamischen Modellierung erforderliche Operationen erkannt. Umgekehrt bildet das statische Modell den Rahmen für die dynamische Modellierung. Die zunehmende Reife und Detaillierung des statischen Modells ist Voraussetzung entsprechender Fortschritte der dynamischen Modellierung. Jede Weiterentwicklung eines Teilmodells sollte im anderen Teilmodell umgehend nachvollzogen werden, um die Konsistenz der Modelle zu bewahren.
- Sowohl die statische wie auch die dynamische Modellierung gliedern sich jeweils in mehrere Schritte. Pro Schritt stehen jeweils andere Konzepte und Modell Aspekte im Vordergrund (siehe unten). Der Makroprozess sieht von vornherein mehrere Iterationen bei der statischen und der dynamischen Modellierung vor. Eine neue Iteration kann bei dem ersten oder einem folgenden Schritt wieder aufsetzen. Modifikationen können sich auf alle Modellelemente beziehen. Möglich ist ferner auch eine Modifikation der Anwendungsfälle. Mehrere Iterationen sind die Regel, nicht die Ausnahme.

Bei der statischen wie der dynamischen Modellierung sind jeweils mehrere und teils bereits für sich genommen recht komplexe Konzepte zu berücksichtigen. Daher werden in objektorientierten Methoden verschiedene Schrittfolgen zur sequentiellen Einbeziehung der Konzepte vorgeschlagen. Hier wird für die Erstellung des statischen Modells die in Tab. 4.8 angeführte Schrittfolge empfohlen.

**Erläuterung zum  
Makroprozess**

**Ablauf der  
statischen  
Modellierung**

Schritt	Erläuterung
1. Klassen identifizieren	Jede Klasse wird nur anhand weniger charakteristischer Attribute und Operationen identifiziert.
2. Assoziationen identifizieren	Zunächst wird nur ermittelt, zwischen welchen Klassen überhaupt Assoziationen benötigt werden. Die Assoziationen werden nur als Linien im Klassendiagramm notiert. Vgl. Schritt 6.
3. Attribute identifizieren	Die Attribute der Klassen werden zwar möglichst vollständig ermittelt, aber nicht näher spezifiziert. Vgl. Schritt 8.
4. Operationen identifizieren	Die Operationen der Klasse werden zwar möglichst weitgehend ermittelt, aber nicht näher spezifiziert. Vgl. Schritt 9.
5. Vererbungsbeziehungen identifizieren	Ausgehend von den identifizierten Attributen, Operationen und Assoziationen werden Vererbungsbeziehungen ermittelt und im Klassendiagramm bzw. Klassenlexikon eingetragen.
6. Assoziationen komplettieren	Die Assoziationen werden vollständig bestimmt und im Klassendiagramm erweitert notiert bzw. im Klassenlexikon eingetragen. Zu ermitteln sind v.a. Richtung, Name, Kardinalitäten, ggf. Sonderform.
7. Pakete bilden	Das Klassendiagramm wird in Pakete zerlegt, wenn dies die Übersichtlichkeit des Modells unterstützt.
8. Attribute spezifizieren	Die Attribute werden im Klassenlexikon spezifiziert, vgl. Kap. 4.1.3.
9. Operationen spezifizieren	Die Operationen werden im Klassenlexikon spezifiziert, wozu auch eine Beschreibung der Methode gehört, vgl. Kap. 4.1.4.

**Tab. 4.8.** Schritte der Erstellung des statischen Modells.

Bei mittleren und größeren Anwendungen ist eine Paketbildung schon zu Beginn der statischen Modellierung erforderlich, um die Übersichtlichkeit des Modells zu bewahren und die statische Modellierung in mehreren Teams durchführen zu können. Zu diesem Zweck können auch bereits die Anwendungsfälle in Pakete eingeteilt werden.

In Tab. 4.9 sind die Schritte zur Erstellung des dynamischen Modells aufgeführt.

#### Ablauf der dynamischen Modellierung

Schritt	Erläuterung
1. Szenarios ableiten und beschreiben	Aus den Anwendungsfällen werden Szenarios abgeleitet, die in Sequenzdiagrammen beschrieben werden.
2. Objektlebenszyklen erstellen	Für Klassen mit einem zustandsabhängigen Verhalten werden Objektlebenszyklen als Zustandsdiagramme erstellt.

**Tab. 4.9.** Schritte der Erstellung des dynamischen Modells.

Nochmals betont sei, dass vor allem die Ermittlung von Operationen ein enges Zusammenwirken von statischer und dynamischer Modellierung erfordert.

Strategien zur Erstellung von OOA-Modellen sind nicht dogmatisch aufzufassen. So können Entwickler es vorziehen, Klasse für Klasse mehr oder minder komplett zu spezifizieren. Auch andere Schwerpunktsetzungen hinsichtlich der statischen

und dynamischen Modellierung sind möglich. In der Praxis werden zum Teil auch zunächst Szenarios ermittelt und in Interaktionsdiagrammen beschrieben, bevor das Klassendiagramm entwickelt wird. Dies kann bei Anwendungen mit hoher funktionaler Komplexität sinnvoll sein.

Für eine Anwendung gibt es nicht „das perfekte Modell“, sondern nur mehr oder weniger brauchbare Modelle. Diese entstehen nicht auf einmal, sondern entwickeln sich schrittweise. Daher sollte auch eine erste Modellversion recht zügig erstellt werden, um eine Grundlage für Diskussionen im Team zu schaffen, in deren Verlauf das Modell verbessert wird.

### 4.3.2 Taktische Hinweise zur Anwendung einzelner Konzepte

In den folgenden Teilabschnitten werden für alle eingeführten grundlegenden Konzepte der OOA jeweils einige methodische Hinweise und Regeln angegeben. Viele der hier aufgeführten sowie weitere methodische Regeln und Anregungen finden sich in BALZERT (1999), BOOCH (1994), COAD und YOURDON (1994) und RUMBAUGH (1993). Für Beispiele wird auf die anschließenden Übungsaufgaben sowie auf Kap. 4.4 verwiesen. Die Hinweise und Regeln können im Allgemeinen sowohl bei der Modellierung selbst als auch bei der Modellvalidierung genutzt werden. Sie eignen sich grundsätzlich auch für Reviews zum OOA-Modell (vgl. Kap. 1.8).

#### (1) Methodische Hinweise und Regeln zum Konzept Klasse

**Hinweise zum  
Konzept Klasse**

1. Die Klassen einer Anwendung sollten dauerhafte reale Objekte bzw. Objekttypen des Anwendungsbereiches widerspiegeln. Zugleich sollte das OOA-Modell insgesamt gewährleisten, dass die erforderlichen Anwendungsfunktionen angeboten und die relevanten Anwendungsdaten verarbeitet werden. Es geht also *nicht* um eine Modellierung des Anwendungsbereiches schlechthin. Die Klassen der Anwendung sind dementsprechend so zu bestimmen, dass die Anwendungsdaten und die -funktionen bzw. die aus ihnen resultierenden Operationen geeignet verteilt und gekapselt werden können.
2. Klassen können bottom up gewonnen werden, wobei man vor allem von den Anwendungsdaten, teils auch von den Anwendungsfunktionen ausgeht. Sie können zugleich top down, ausgehend von den realen Objekten des Anwendungsbereiches identifiziert werden. Empfohlen wird hier eine Kombination beider Zugänge.
3. Der Bottom-Up-Ansatz benutzt die Beschreibung der Anwendungsdaten und die spezifizierten Anwendungsfälle sowie Dokumente wie Formulare, Listen und Masken von Altsystemen. Das grundsätzliche Vorgehen besteht darin, im System zu verwaltende Daten nach ihrem fachlichen Zusammenhang in Gruppen zu bündeln. Aus diesen werden dann durch eine Zuordnung von Operationen die Klassen gewonnen.

**bottom up zu Klassen**



- |                                  |  |
|----------------------------------|--|
| <b>relevante Komplexität</b>     | 4. Klassen stellen komplexe, zusammengesetzte Einheiten dar. Ein bestimmter Typ von realen Objekten sollte nur als Klasse modelliert werden, wenn er im vorliegenden Anwendungskontext eine hinreichende Komplexität aufweist. So sollten mehrere Merkmale der realen Objekte für die Anwendung relevant sein. Ferner sollten die realen Objekte ein anwendungsrelevantes Verhalten besitzen. Wird von Mitarbeitern einer Firma nur ihr Name benötigt, so sind Mitarbeiter im gegebenen Anwendungskontext keine komplexen Objekte. Eine Klasse Mitarbeiter ist überflüssig. Ein Preis ist meist eine einfache Eigenschaft eines Produkts, der seinerseits weder weitere Merkmale noch ein Verhalten besitzt. Er ist als Attribut zu modellieren. Ein reales Objekt, das nur als Eigenschaft eines größeren Ganzen und nicht isoliert existieren muss, wird als Attribut, nicht als Klasse modelliert.  |
| <b>Klasse vs. Attribut</b>       |  |
| <b>top down zu Klassen</b>       | 5. Der Top-Down-Ansatz geht von den realen Objekten des Anwendungsbereiches aus. Diese werden anhand der gleichen Quellen wie beim Bottom-Up-Ansatz ermittelt. Dabei wird besonders auf Substantive, in zweiter Linie auf Verben geachtet. Es kommen nur reale Objekte in die engere Wahl, die im betrachteten Anwendungskontext komplexer Natur sind. Es handelt sich also keineswegs darum, <i>sämtliche</i> Substantive in Dokumenten zu sammeln. Um Klassen zu ermitteln, werden den ausgewählten realen Objekttypen jeweils einige charakteristische Attribute und Operationen zugeordnet.  |
| <b>reale Objekte und Klassen</b> | 6. Klassen bilden häufig folgende Typen realer Objekte ab: <ul style="list-style-type: none"> <li>- berührbare Dinge, z.B. Produkte,</li> <li>- Personen, wobei es sich meist nicht um Personen an sich, sondern um ihre Rollen handelt, die im Anwendungskontext relevant sind, z.B. Kunden,</li> <li>- Vorgänge wie z.B. eine Messung oder eine Veranstaltung, über die Daten zu speichern sind (deshalb können auch Verben auf Klassen hinweisen),</li> <li>- Ereignisse, deren Daten relevant sind,</li> <li>- Organisationen, Funktionsbereiche und Orte.</li> </ul> 7. Auch reale Objekte mit relevanter Komplexität werden nicht immer zu Klassen der Anwendung. So können z.B. in einer Auswertungsliste verschiedene Anwendungsdaten kombiniert erscheinen und ihre Erzeugung kann recht komplex sein. Dennoch können Auswertungen oder auch Rechnungen häufig auf der Basis von Attributen anderer Klassen erzeugt werden, so dass für sie keine separaten Klassen benötigt werden.         8. In der OOA werden keine Klassen eingeführt, deren Aufgabe die Verwaltung von Objekten einer anderen Klasse ist. Diese Aufgabe nimmt die implizite Objektverwaltung wahr, vgl. Kap. 4.1.2.         9. Klassen sollten einen aussagekräftigen Namen (Substantiv im Singular, evtl. ergänzt durch Adjektiv) erhalten. Von der konkreten Wahl der Klassennamen hängt die Verständlichkeit eines Modells wesentlich ab. Geht es in einer Anwendung um Personen in ihrer Rolle als Mitarbeiter, so wird man die entsprechende Klasse „Mitarbeiter“, nicht „Person“ nennen.         10. Aus Angaben von BOOCH (1996) lässt sich grob abschätzen, dass bei Anwendungen mittlerer Größe das OOA-Modell etwa 15 bis 35 Klassen umfasst. Es leuchtet ein, dass in dem jeweils modellierten Anwendungsbereich meist wesentlich mehr Typen realer Objekte vorkommen dürften. Hieraus ergibt sich, dass keineswegs alle realen Objekttypen des Anwendungsbereichs |



ches in der OOA auf Klassen abgebildet werden. Es besteht eher die Gefahr, zu viele als zu wenige Klassen zu ermitteln.

11. Die Attribute einer Klasse sollten vor allem durch Operationen dieser Klasse verarbeitet werden. Sind innerhalb einer Operation Attribute mehrerer Klassen zu verarbeiten, kann die Operation ihrer (Teil-)Aufgabe entsprechend derjenigen Klasse zugeordnet werden, zu deren Aufgabe bzw. Gesamtverantwortung die Operation am besten passt. Erstellt eine Operation etwa eine Rechnung, wird man sie eher einer Klasse Auftrag als einer Klasse Artikel zuordnen, obwohl die Operation Artikelpreise verarbeitet.

## (2) Methodische Hinweise und Regeln zum Konzept Attribut

### Hinweise zum Konzept Attribut

1. Attribute können anhand der Frage bestimmt werden, welche Informationen ein Objekt der betreffenden Klasse langfristig aufbewahren und bereitstellen muss (das Objekt fragt: „Was muss ich wissen?“).
2. Quellen für Attribute sind die Beschreibungen der Anwendungsdaten und der Anwendungsfälle im Pflichtenheft sowie Formulare, Listen, Masken von Altsystemen und weitere Dokumente des Anwendungsbereiches.
3. Als Attribute sollten in der OOA nur fachlich relevante Informationen gewählt werden, die im Zusammenhang mit der Aufgabe und Verantwortlichkeit der Klasse stehen. Attribute bilden keine Implementierungsdetails wie z.B. die Anzahl der Zeilen einer Listenseite ab.
4. Attribute sollten einen fachlich aussagekräftigen Bezeichner (Substantiv, Adjektiv) erhalten, der eindeutig in der Klasse sein muss.
5. Neben atomaren Attributen sind auch zusammengesetzte Attribute möglich. Attribute von einem Recordtyp bzw. Listentyp sollten zur Zusammenfassung mehrerer, inhaltlich zusammengehörender Merkmale benutzt werden. Ein einfaches Beispiel bildet ein Attribut Adresse. Auf die in diesem Zusammenhang relevante Abgrenzung von Attributen und Klassen wurde bereits eingegangen.
6. Schlüsselattribute werden zur Identifikation von Objekten nicht benötigt. Sie sollten nur eingeführt werden, wenn dies aus fachlicher Sicht erforderlich ist (vgl. Kap. 4.1.8).
7. Fremdschlüssel, die auf andere Objekte verweisen, sind zu vermeiden (vgl. Kap. 4.1.5).
8. Attribute, die nicht ein Objekt, sondern alle Objekte einer Klasse charakterisieren, sind als Klassenattribute auszuzeichnen.
9. Für jedes Attribut ist zu prüfen, ob es überhaupt einen Wert annehmen kann (vgl. Übungsaufgabe 1.3.5, e)). Ferner sind Muss-Attribute von Kann-Attributen (optional) sowie modifizierbare Attribute von konstanten Attributen (readonly) zu unterscheiden. Abgeleitete Attribute sind stets konstante Attribute, weil eine direkte Modifikation nicht möglich ist.
10. Pro Attribut ist zu prüfen, ob es ein Objekt allein charakterisiert oder nur in Verbindung mit einem zweiten Objekt sinnvoll ist. In diesem Fall ist das Attribut einer Assoziation bzw. einer assoziativen Klasse zuzuordnen.

### Hinweise zum Konzept Operation

### **(3) Methodische Hinweise und Regeln zum Konzept Operation**

1. Operationen können anhand der Frage bestimmt werden, welche Tätigkeiten ein Objekt der betreffenden Klasse anbieten sollte (das Objekt fragt: „Was muss ich tun?“).
2. Quellen für Operationen sind die Beschreibungen der Anwendungsfälle im Pflichtenheft, ferner Ablaufpläne, Menüs von Altsystemen und weitere Dokumente des Anwendungsbereiches.
3. Als Operationen kommen nur fachlich relevante Verarbeitungsschritte in Betracht. Operationen befassen sich nicht mit GUI-Aufgaben (Informationsdarstellung, Dialogabwicklung) oder der Datenspeicherung, die jeweils erst Gegenstand des OOD sind.
4. Operationen sollten einen fachlich aussagekräftigen Bezeichner (Verb, evtl. plus Substantiv) erhalten, der eindeutig in der Klasse sein muss.
5. Zu identifizieren sind vorrangig externe Operationen, die von der Benutzungsschnittstelle aufgerufen werden. Die Aufspaltung komplexer Operationen sollte so weit getrieben werden, dass die Assoziationen der Klasse benutzt werden. Operationen, die ausschließlich auf Attribute der Klasse selbst zugreifen bzw. keine Operationen anderer Klassen mehr aufrufen, müssen in der Regel nicht verfeinert werden.
6. Elementare Operationen erscheinen nicht im Klassendiagramm. Verwaltungsoperationen, darunter Konstruktor- und Destruktoroperationen werden hier aus Gründen der Verständlichkeit, aber in der Praxis überwiegend nicht eingetragen.
7. Meist genügen in der OOA Signaturen, die auf den Operationsnamen verkürzt sind. Offensichtliche Parameter können bereits spezifiziert werden, falls dies zur Verständlichkeit beiträgt.
8. Erste Operationen sollten bereits bei der Festlegung der Klassen identifiziert werden. Vielfach lassen sich Operationen erst im Zusammenhang mit Szenarios und Zustandsautomaten finden. Da Klassen dem Prinzip der Kapselung zusammengehöriger Daten und Funktionen entsprechen sollten, ist die Identifikation der Operationen jedoch auch Aufgabe der statischen Modellierung.
9. Operationen, die mehrere oder alle Objekte der Klasse betreffen, sind als Klassenoperationen auszuzeichnen. Die in der OOA angenommene Objektverwaltung gestattet ohne weiteres die Iteration über alle aktuell vorhandenen Objekte einer Klasse.
10. Die Spezifikation der Methoden von Operationen im Klassenlexikon sollte nicht fehlen. Sie kann meist informell und knapp erfolgen.

#### **Hinweise zum Konzept Assoziation**

### **(4) Methodische Hinweise und Regeln zum Konzept Assoziation**

1. Assoziationen einer Klasse können anhand der Frage bestimmt werden, Objekte welcher anderen Klassen einem Objekt der gegebenen Klasse bekannt sein sollten (das Objekt fragt: „Wen muss ich kennen?“).
2. Assoziationen werden in der OOA als separate Modellelemente spezifiziert und daher nicht durch Attribute wie Fremdschlüssel angegeben.
3. Assoziationen bilden dauerhafte Objektverbindungen ab, die einmal eingerichtet und wiederholt für direkte Objektzugriffe genutzt werden. In der Vermeidung mehrfacher „teurer“ (Such-)Operationen zur Identifizierung von

#### **Assoziation oder temporäre Verbindung**

Objekten besteht die Ökonomie von Assoziationen. Andererseits kann sich ihre spätere Implementierung relativ aufwendig gestalten. Daher ist jeweils zu fragen, ob Objektverbindungen tatsächlich dauerhaft sein müssen oder ob temporäre Verbindungen hinreichend sind. Sicherlich muss ein Auftrag stets „seinen“ Kunden kennen. Sind dagegen Operationen, die einen Zugriff von einem Kunden auf alle seine Aufträge verlangen, selten, so kann auf die umgekehrte Assoziation evtl. verzichtet werden und der Zugriff ggf. über eine Klassenoperation erfolgen.

4. Die Richtung einer Assoziation ist ein besonders wichtiges Merkmal und sollte aufgrund fachlicher Anforderungen schon während der OOA bestimmt werden. Es ist insbesondere – wie in dem vorigen Beispiel – stets zu prüfen, ob eine bidirektionale Assoziation erforderlich ist.
5. Die Semantik einer Assoziation kann durch einen Bezeichner (oft Verb) verdeutlicht werden. Häufig ist jedoch die Angabe der Rollen der Objekte in der Assoziation aussagekräftiger (vgl. BALZERT 1999).
6. Kardinalitäten müssen den minimalen und den maximalen Umfang der Objektverbindungen „zu jedem beliebigen Zeitpunkt“ (COAD und YOURDON 1994) erfassen. Dabei ist auf eventuelle Sonderfälle im Moment der Erzeugung oder Zerstörung von Objekten zu achten. Zu unterscheiden sind in erster Linie Kardinalitäten unter Einschluss der 0 (Kann-Beziehungen) von Kardinalitäten ab 1 (Muss-Beziehungen). Über Kardinalitäten kann nur aufgrund konkreter Angaben zum Anwendungsbereich entschieden werden, während allgemeine Erörterungen nicht hilfreich sind. Obwohl bei unidirektionalen Assoziationen für eine spätere Realisierung nur die Kardinalität auf der Zielseite der Assoziation notwendig ist, wird für die OOA die Bestimmung beider Kardinalitäten empfohlen.
7. Sonderformen der Assoziation (Aggregation und Komposition) sind nicht unbedingt erforderlich. Im Zweifelsfall ist stets die gewöhnliche Assoziation zu wählen. Beide Sonderformen sollten ggf. nur für die Modellierung asymmetrischer Enthaltenseins-Beziehungen genutzt werden (vgl. Kap. 4.1.5).

### Richtung

## Kardinalitäten

## Sonderformen

## **(5) Methodische Hinweise und Regeln zum Konzept Vererbung**

## Hinweise zum Konzept Vererbung

1. Modellierte Vererbungsbeziehungen sollten stets eine Generalisierungs- bzw. Spezialisierungsbeziehung ausdrücken. D.h. die Objekte jeder Unterklasse müssen inhaltlich gesehen auch Objekte der Oberklasse sein.
2. Vererbungsbeziehungen lassen sich ausgehend von mehreren vorhandenen (Unter-)Klassen mit gemeinsamen Eigenschaften durch Generalisierung finden. Deshalb steht die Identifizierung von Vererbungsbeziehungen auch eher am Ende der statischen Modellierung. Daneben lassen sich Vererbungsbeziehungen auch anhand von Fachbegriffen und Dokumentgliederungen des Anwendungsbereichs oder auch über Menüstrukturen von Altsystemen identifizieren. Man achte auf Spezialfälle, die sich oft hinter zusammengesetzten Substantiven wie z.B. Saisonartikel verbergen.
3. Vererbungsbeziehungen werden oft anhand gemeinsamer Attribute und Operationen identifiziert. Doch können auch gemeinsame Assoziationen mehrerer Klassen mit einer weiteren Klasse ein guter Grund für eine Generalisierung sein. Andererseits sollten Vererbungsbeziehungen nicht erzwungen

### Kriterium für Vererbung

## Auffindung von Vererbungsbeziehungen

werden. Liegt z.B. nur ein gemeinsames Attribut in zwei Klassen vor, sollte auf eine Generalisierung verzichtet werden.

#### **Gestaltung von Vererbungshierarchien**

4. Für eine Oberklasse sollte ein Name gefunden werden, der ihre Abstraktionshöhe möglichst genau ausdrückt. Sind Kochbücher und Bücher zum Thema Gartenbau zu verwalten, wäre Sachbuch als Name der Oberklasse geeigneter als Buch.
5. Oberklassen sind als abstrakte Klassen zu modellieren, wenn Objekte der Oberklasse als solche unvollständig bzw. nicht sinnvoll sind.
6. Vererbungshierarchien sollten in der OOA nur eine begrenzte Höhe, maximal etwa fünf Stufen umfassen.
7. Attribute, Operationen und Assoziationen werden so hoch wie möglich in der Vererbungshierarchie angeordnet. Dabei ist jedoch zu prüfen, ob jedes Element der Oberklasse in allen abgeleiteten Klassen benötigt wird. Dies gilt insbesondere für Assoziationen.
8. Kommen Operationen mit analogen Aufgaben in mehreren Unterklassen vor, so wird durch eine übereinstimmende Wahl der Operationsnamen und ihre „Beförderung“ in die Oberklasse die Anwendung des Polymorphismus im Entwurf vorbereitet. Zugleich ist im Klassenlexikon oder Klassendiagramm auf erforderliche Redefinitionen von Operationen hinzuweisen (vgl. Kap. 4.1.8).

#### **Vererbung vs. Klassifizierung**

9. Oft kann man eine Objektmenge anhand eines Merkmals und seiner Ausprägungen in mehrere Gruppen klassifizieren. Eine solche Klassifizierung rechtfertigt an sich keine Spezialisierung von Unterklassen. Diese sollte nur dann modelliert werden, wenn die Gruppen jeweils eigene Attribute, Operationen oder auch Assoziationen besitzen. Läufer kann man etwa nach der zurückzulegenden Distanz in Kurzstrecken-, Mittelstrecken- und Langstreckenläufer einteilen. Sind darüber hinaus keine Eigenschaften relevant, die nur jeweils eine einzelne Gruppe besitzt, so ist nur eine Klasse Läufer mit dem Attribut Distanz ohne Unterklassen vorzusehen.
10. Nicht immer lassen sich Spezialisierungen durch Vererbungsbeziehungen modellieren. Eine (reine) Vererbung impliziert, dass ein Objekt nur zu einer Unterklasse gehören kann. Doch werden z.B. Lieferanten eines Unternehmens nicht selten zugleich zu seinen Kunden gehören, so dass die Generalisierung zu einer Klasse Geschäftspartner nicht ausreicht. Bilden die realen Urbilder der Objekte von Unterklassen keine disjunkten Teilmengen, so sind andere bzw. zusätzliche Mechanismen zur Vererbung zu nutzen. Näheres findet sich z.B. bei OESTEREICH (1998).

#### **Hinweise zum Konzept Paket**

##### **(6) Methodische Hinweise und Regeln zum Konzept Paket**

1. Pakete sollten inhaltlich zusammenhängende Modellelemente enthalten, die gemeinsam der Erledigung einer übergeordneten fachlichen Aufgabe dienen. Pakete können oft ausgehend von den Anwendungsfällen eines Systems bestimmt werden.
2. Pakete erhalten einen aussagekräftigen fachlich orientierten Bezeichner. Lässt sich ein solcher nicht finden, weist dies auf eine nicht angemessene Paketgliederung hin.

3. Pakete sollten eine geeignete Granularität besitzen. Als grobe Anhaltspunkte seien genannt, dass ein Paket (auf unterster Ebene) etwa 7 bis 10 Klassen beinhalten und komplett auf einem A4-Blatt darstellbar sein sollte.
4. Vererbungsbeziehungen sowie auch Aggregationen und Kompositionen sollten durch Pakete nicht zerschnitten werden.
5. Bei größeren Anwendungen ist eine Pakethierarchie vorzusehen, die möglichst mit einer Hierarchie der Anwendungsfälle korrespondiert.

## **(7) Methodische Hinweise und Regeln zum Konzept Szenario**

**Hinweise zum  
Konzept Szenario**

**Umfang**

1. Szenarios sind aus Anwendungsfällen durch Spezialisierung abzuleiten. Dabei sollte man sich auf die Standardfälle oder wichtigeren Fälle beschränken. Booch (1994) empfiehlt, nur etwa 80% der möglichen Szenarios zu betrachten, vgl. auch HRUSCHKA (1998).
2. Bei der Erstellung von Sequenzdiagrammen greifen die Funktionen der Validierung und der Entdeckung von Modellelementen ineinander. Einerseits wird geprüft, ob vom statischen Modell angebotene Klassen, Assoziationen und Operationen für den Ablauf eines Szenarios zweckmäßig und bereits hinreichend sind. Das statische Modell wird unter dem Aspekt seiner „Durchgängigkeit“ validiert. Andererseits sollen bisher fehlende Modellelemente aufgefunden werden.
3. Wesentlich bei der Erstellung von Sequenzdiagrammen ist die Wahl einer geeigneten Granularität. Sie muss einerseits so fein gewählt werden, dass die Kooperation der beteiligten Objekte und ihrer Operationen hervortritt. Andererseits wird man aus Gründen des Aufwands und der Übersichtlichkeit meist auf Details, die etwa die Einrichtung von Objektverbindungen betreffen, verzichten. Ausgelassene Details können ersatzweise durch Kommentare angedeutet werden. Die Zusammenfassung elementarer Operationen wurde bereits vorgeschlagen (vgl. Kap. 4.2.2). Die Granularität sollte in einem Team explizit und je nach Bedarf vereinbart werden.
4. Ein Sequenzdiagramm sollte ausgehend von den externen Operationen eines Szenarios erstellt werden. Pro externer Operation sind die anschließenden Operationen und ihre Objekte Schritt für Schritt zu bestimmen. Die Existenz einer Verbindung zu dem Objekt, das als nächstes eine Operation ausführt, ist stets zu prüfen. Besonders geachtet werden sollte auf Konstruktoroperationen sowie die bedingte oder wiederholte Ausführung von Operationen.
5. Bedingungen sollten in Sequenzdiagrammen sparsam verwendet werden, um die Übersichtlichkeit nicht zu gefährden. Gegebenenfalls muss ein Szenario noch weiter spezialisiert werden.

**Granularität  
Sequenzdiagramm**

**Sequenzdiagramm  
wie erstellen?**

## **(8) Methodische Hinweise und Regeln zum Konzept Zustandsautomat**

**Hinweise zum Konzept  
Zustandsautomat**

**wann erstellen?**

1. Für alle Klassen ist zu prüfen, ob ein zustandsabhängiges Verhalten vorliegt. Zustandsdiagramme müssen nur für Klassen mit nicht trivialem Objektlebenszyklus erstellt werden.
2. Bei Zustandsdiagrammen geht es wie bei Sequenzdiagrammen um die Validierung vorhandener und die Auffindung fehlender Modellelemente. Dabei stehen hier Operationen und ihre Zustandsabhängigkeit im Vordergrund.

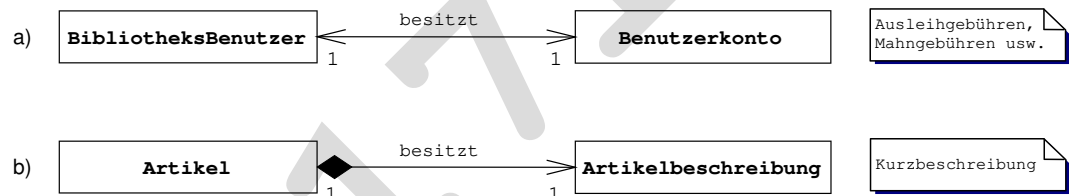
**wie erstellen?**

3. Zuerst sollten die möglichen Zustände und die pro Zustand wirksamen Ereignisse und Transitionen anhand der zugrundeliegenden realen Objekte bestimmt werden. Zu achten ist u.a. auf die möglichen Stati realer Objekte, Grenzwerte und zeitliche Ereignisse.
4. Danach sollten die Elemente des Zustandsdiagramms mittels der Eigenschaften der Klasse formuliert werden. Dabei sind die in Kap. 4.2.3 angegebenen Konsistenzregeln einzuhalten.
5. Zusätzliche Attribute, die lediglich den Status eines Objekts festhalten, können in einfachen Fällen eingeführt werden. Doch bleibt die Realisierung von Zuständen in komplizierteren Fällen dem Entwurf vorbehalten.

### Übungsaufgaben zu Kapitel 4.3

#### Übungsaufgabe 4.3.1

In den beiden folgenden Klassendiagrammen wurden jeweils zwei Klassen vorgegeben. Bewerten Sie die Modellierung.



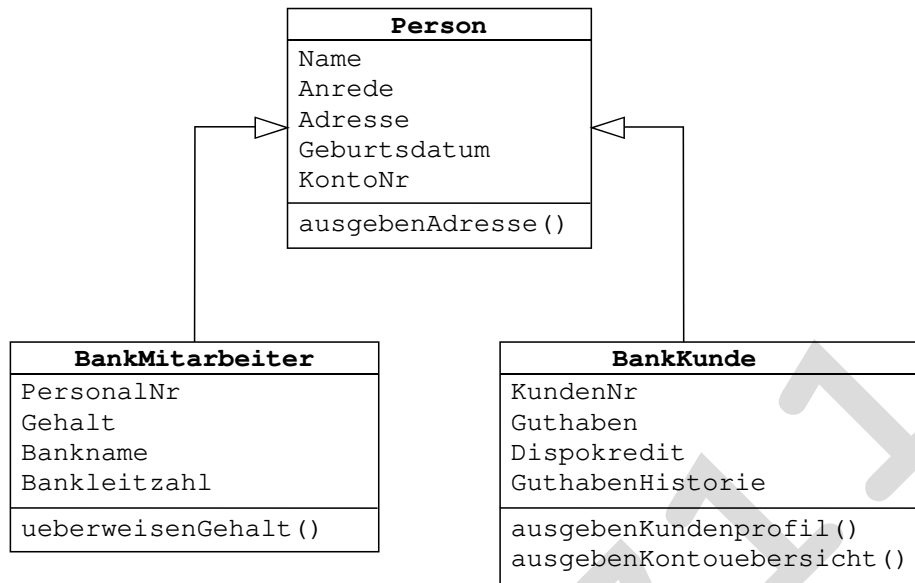
#### Übungsaufgabe 4.3.2

- a) Beurteilen Sie folgende Klasse BankKunde. Auf fehlende Attribute oder Operationen sollte dabei nicht eingegangen werden.

BankKunde
KundenNr
Name
Anrede
Geburtsdatum
Adresse
KontoNr
Guthaben
Dispokredit
GuthabenHistorie
ausgebenAdresse()
ausgebenKundenprofil()
ausgebenKontouebersicht()

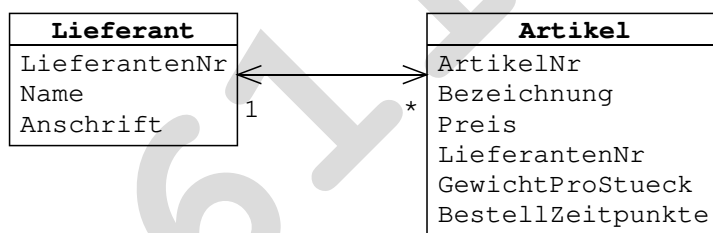


- b) Zusätzlich wird die Klasse BankMitarbeiter betrachtet und folgende Vererbungsstruktur gebildet. Bewerten Sie die Vererbungsbeziehung möglichst konkret.



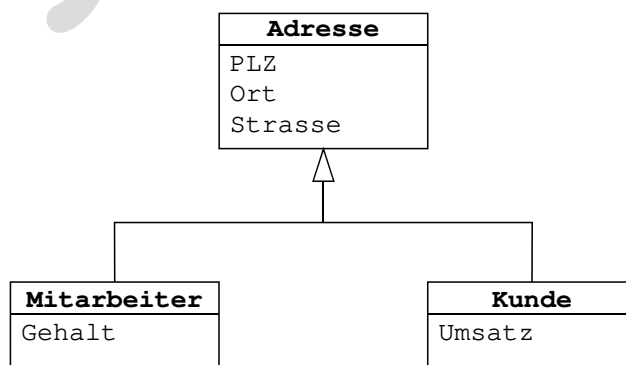
### Übungsaufgabe 4.3.3

Welche Einwände haben Sie gegen das folgende Klassendiagramm vorzubringen?



### Übungsaufgabe 4.3.4

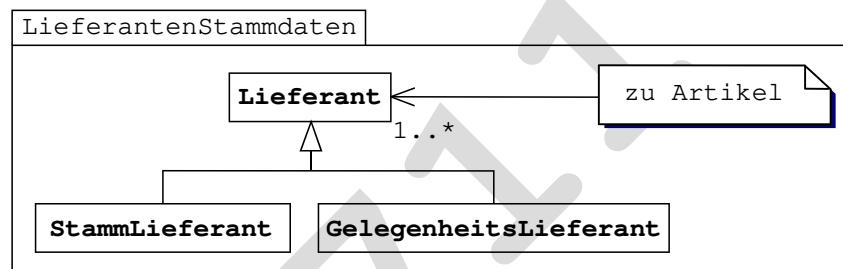
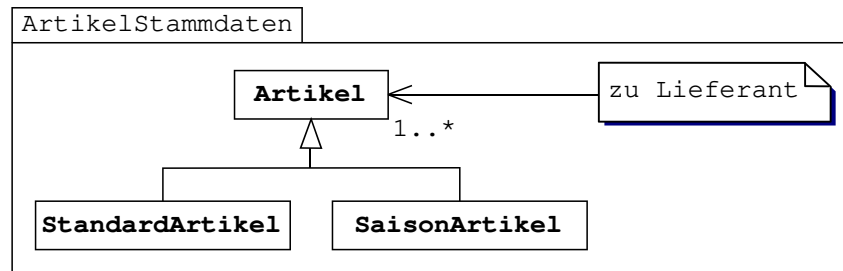
Wie beurteilen Sie folgende Vererbungsbeziehung?





### Übungsaufgabe 4.3.5

Sind Sie mit folgender Paketbildung einverstanden?

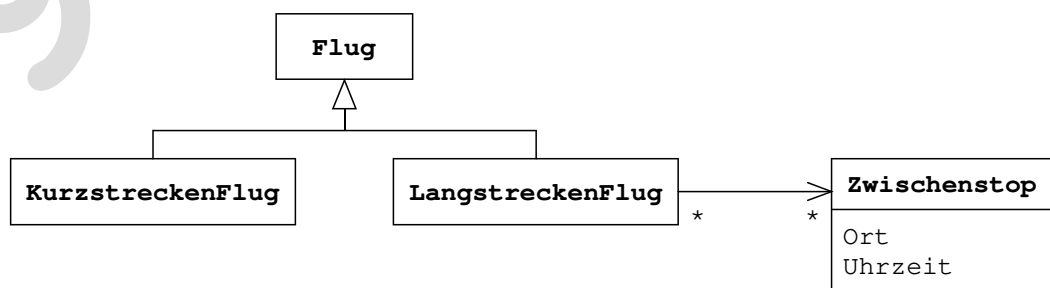


### Übungsaufgabe 4.3.6

Bei der Modellierung eines Flugreservierungssystems stehen zur Differenzierung von Flugtypen folgende Informationen zur Verfügung:

	Kurzstreckenflug	Langstreckenflug
Reichweite	≤ 600 km	> 2000 km
Zwischenstops	stets nonstop	meist vorhanden

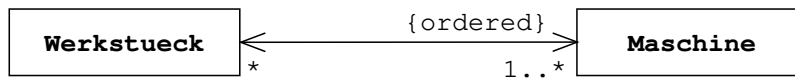
Beurteilen Sie das folgende zugehörige Klassendiagramm:



### Übungsaufgabe 4.3.7.

In einer Werkstatt werden Werkstücke mit verschiedenen Maschinen bearbeitet. Da Auswahl und Reihenfolge der eingesetzten Maschinen vom jeweiligen Werkstück abhängen, sollen diese Angaben pro Werkstück geführt werden. Ist folgende Assoziation zur Lösung dieser Aufgabe angemessen?

Die mit einem Werkstück verbundenen Maschinen seien nach der Bearbeitungsreihenfolge geordnet.



### Übungsaufgabe 4.3.8.

Würden Sie in den folgenden Fällen eine einfache Assoziation oder eine ihrer Sonderformen Aggregation bzw. Komposition wählen? Geben Sie ggf. auch die Sonderform an und begründen Sie Ihre Antwort stichwortartig.

- a) eine Abteilung eines Unternehmens und die zugehörigen Mitarbeiter des Unternehmens
- b) ein Musikfestival und seine Konzerte
- c) ein Gerätetyp und seine Komponenten

### Übungsaufgabe 4.3.9.

Spezifizieren Sie für die folgenden Assoziationen die Kardinalitäten.

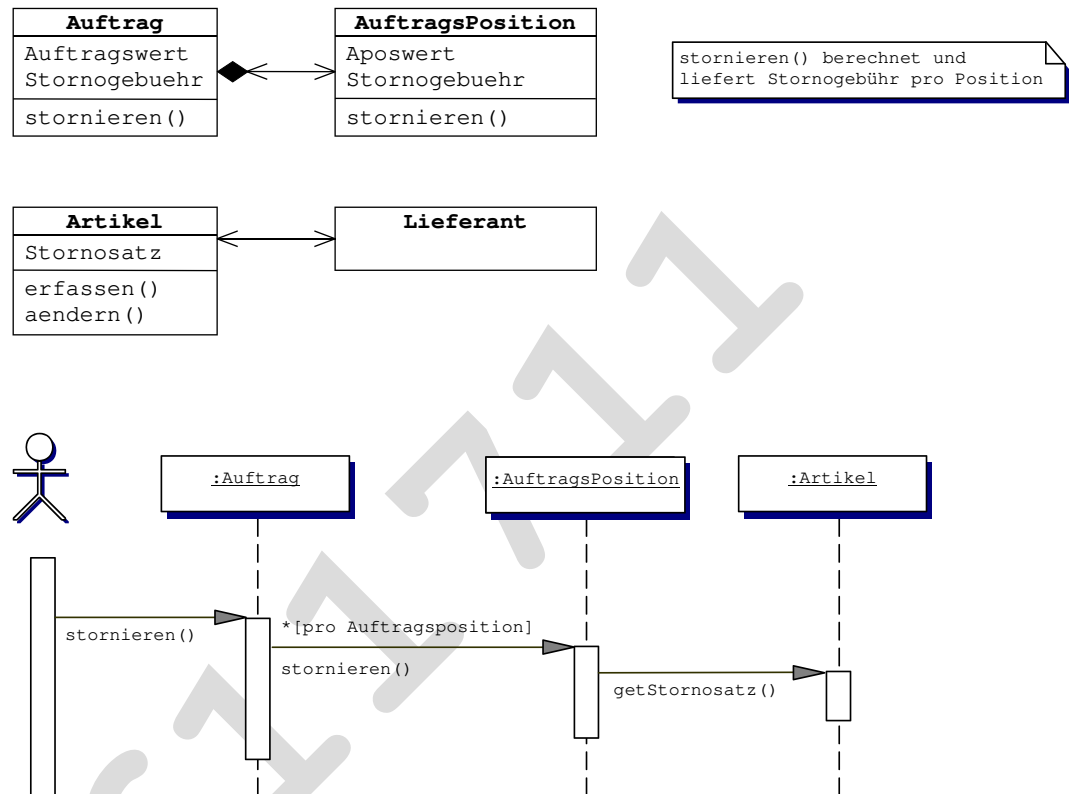
- a) BankKunde – BankKonto

Jedes Bankkonto gehört zu genau einem Kunden. Bei der Erfassung eines neuen Kunden wird sofort ein Konto eingerichtet. Ein Kunde kann mehrere Konten eröffnen und auch wieder auflösen. Er ist nicht mehr Kunde, nachdem das letzte Konto aufgelöst wurde.

- b) In einem Lager werden verschiedene Artikel aufbewahrt, wobei pro Lagerplatz nur Exemplare eines Artikels vorhanden sein können. Oft wird der Artikel bereits im Lagerverwaltungssystem erfasst, bevor erstmals Exemplare dieses Artikels zur Lagerung eintreffen.

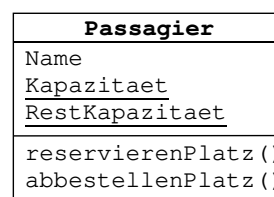
### Übungsaufgabe 4.3.10.

Das folgende Klassendiagramm beschreibt einen Ausschnitt eines Auftragsystems und ein Sequenzdiagramm für das Stornieren eines Auftrages. Die Stornogebühr des Auftrags ergibt sich als Summe der Stornogebühren seiner Positionen, die wiederum mittels eines individuellen Stornosatzes zu berechnen sind. Welches Problem ergibt sich beim Abgleich von Klassen- und Sequenzdiagramm?



### Übungsaufgabe 4.3.11.

Gegeben sei folgende Klasse Passagier eines Reservierungssystems für Schiffsreisen.



Erstellen Sie ein Zustandsdiagramm für die Reservierung. Beachten Sie neben der Kapazität, d.h. der angebotenen Platzanzahl im Schiff auch, dass die Reservierung nach einer gewissen Reservierungsfrist abgeschlossen wird.

## 4.4 Projekt Tourenplanungssystem: OOA-Modell

Im Folgenden werden die Klassendiagramme und das Klassenlexikon des Tourenplanungssystems dokumentiert. Da die Dokumentation recht ausführlich ist, seien vorab lediglich die Klassen des TPS kurz kommentiert.

Die Kundenorte wurden als Kundenadressen in die Klasse Kunde aufgenommen. Da es keine separate Klasse für Orte gibt, bezieht sich eine Entfernung auf je zwei Kunden.

**Klassen des TPS**

Sämtliche Daten, in denen alle Fahrzeuge übereinstimmen, werden in die Klasse Fuhrpark ausgelagert. Das einzige Objekt dieser Klasse ist dementsprechend als Aggregatobjekt mit den Fahrzeug-Objekten durch eine Komposition verbunden.

Die Klassen Kunde und Auftrag werden jeweils spezialisiert, da es hinreichende Differenzen der Unterklassen LaufKunde und StammKunde bzw. EinzelAuftrag und DauerAuftrag in Bezug auf Attribute, Operationen und Assoziationen gibt.

Die zentrale Aufgabe der Klasse Tourenplan ist die Erstellung von Tourenplänen. Erzeugte Tourenpläne sollen aufbewahrt werden, um sie beispielsweise später auswerten zu können. Daraus leiten sich der Name der Klasse und zusätzliche Aufgaben ab. Wäre ein Tourenplan lediglich zu erzeugen und als Liste auszugeben, könnte man die Klasse auch „Tourenplaner“ nennen. Um die Klasse Tourenplan nicht zu umfangreich werden zu lassen, wird jede Tour eines Tourenplans als gesondertes Objekt abgebildet.

Für Rechnungen und Auswertungen werden keine gesonderten Klassen benötigt, da alle erforderlichen Attribute in anderen Klassen vorhanden sind.

### 4.4.1 Klassendiagramme

Die folgenden Abbildungen 4.27 bis 4.29 enthalten drei OOA-Klassendiagramme des Tourenplanungssystems. Das Klassendiagramm in Abb. 4.27 zeigt alle TPS-Klassen ohne Attribute und Operationen, jedoch mit allen Klassenbeziehungen. Es sieht die Pakete „Stammdatenverwaltung“ und „Auftragsabwicklung“ vor. Die Klassendiagramme der Abb. 4.28 und 4.29 beziehen sich auf diese beiden Pakete. Zur Erläuterung der Klassendiagramme siehe das OOA-Klassenlexikon.

**oberstes  
Klassendiagramm**



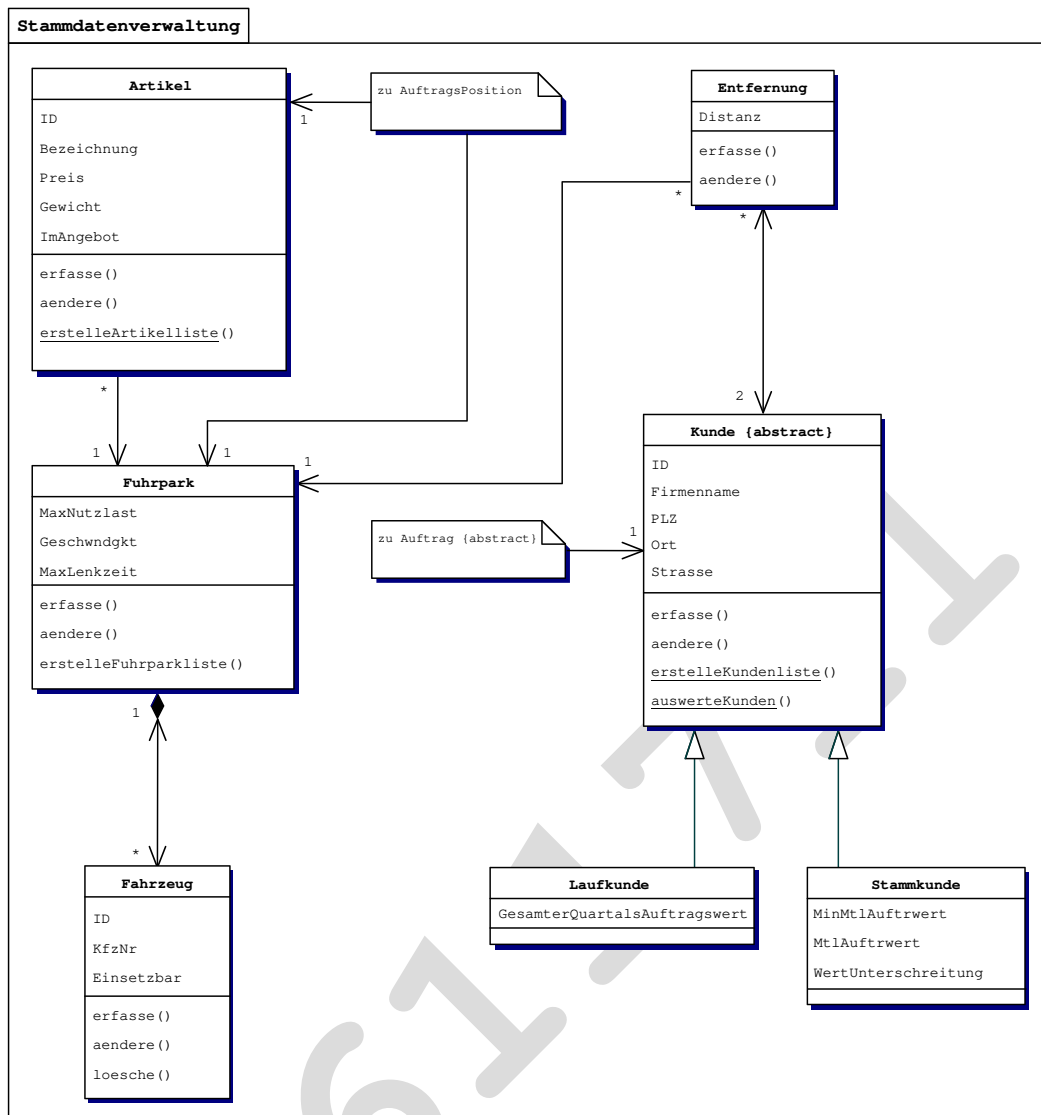


Abb. 4.28. Klassendiagramm des Pakets Stammdatenverwaltung.

**Paket  
Stammdaten-  
verwaltung**

**Paket  
Auftragsabwicklung**

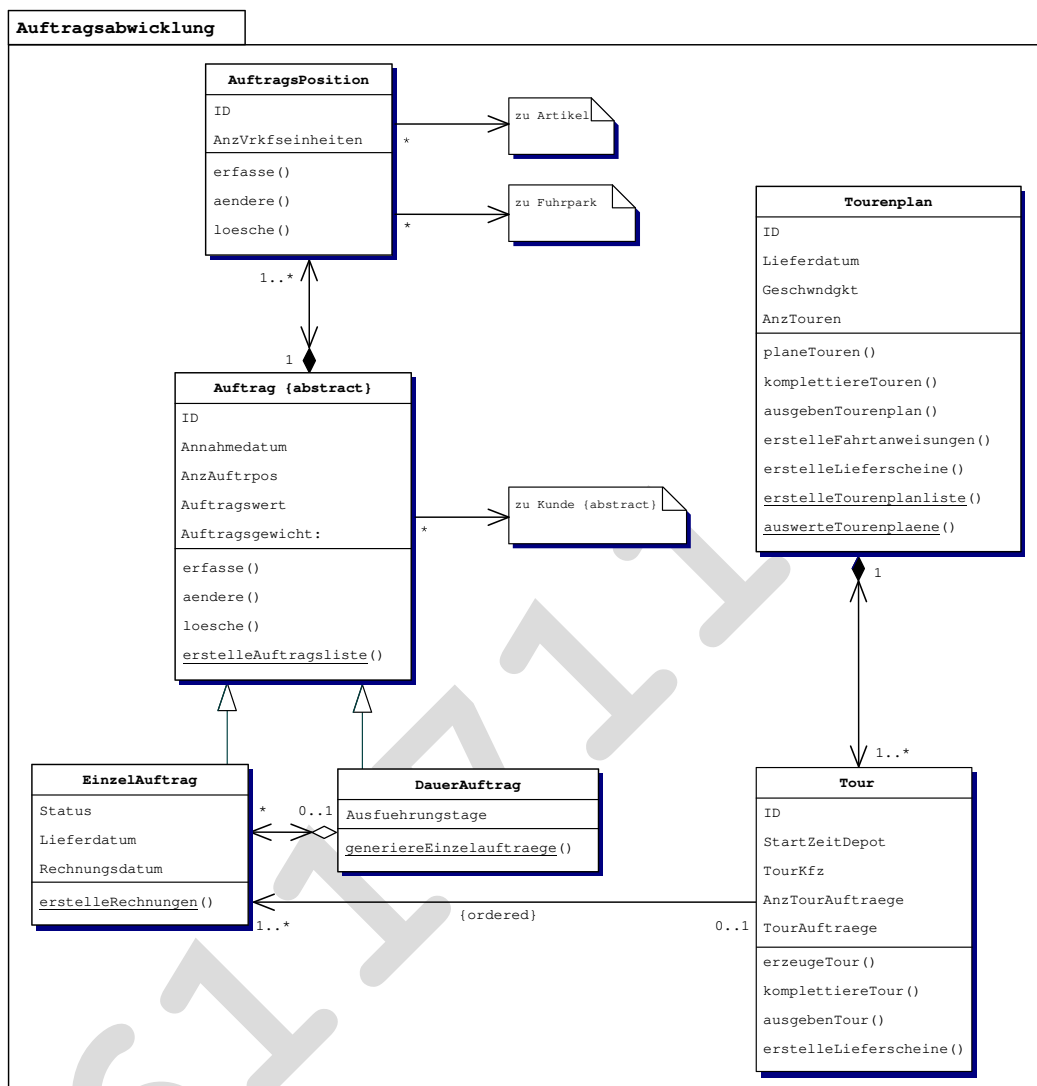


Abb. 4.29. Klassendiagramm des Pakets Auftragsabwicklung.



## 4.4.2 Klassenlexikon

## Klassenlexikon

### Paket: Stammdatenverwaltung

<b>Klasse:</b>	<b>Artikel</b>	<b>Artikel</b>
<b>Beschreibung:</b>	Ein Artikel-Objekt repräsentiert einen Artikel des Sortiments.	
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}; firmeninterne Artikelnummer ab 1 fortlaufend.</li> <li>- Bezeichnung: String; Artikelbezeichnung.</li> <li>- Verkaufseinheit: String; kleinste Abgabemenge, z.B. Stückzahl, Gewicht.</li> <li>- Preis: Float; {Euro, Preis &gt; 0}; Verkaufspreis pro Verkaufseinheit.</li> <li>- Gewichte: Float; {kg, 0 &lt; Gewicht ≤ Fuhrpark.MaxNutzlast}; Gewicht pro Verkaufseinheit.</li> <li>- Im Angebot: Bool = true; true: Artikel aktuell im Angebot, false: sonst.</li> <li>- <u>AnzArtikel</u>: UInt = 0; Anzahl aller Artikel.</li> <li>- <u>AnzArtikelImAngebot</u>: UInt = 0; Anzahl aller aktuell angebotenen Artikel.</li> </ul>	
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Artikel.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Artikels.</li> <li>- <u>erstelleArtikelliste()</u>; gibt Liste aller Artikel alphabetisch sortiert nach Artikelbezeichnung aus.</li> </ul>	
<b>Assoziationen:</b>	<ul style="list-style-type: none"> <li>- Artikel → Fuhrpark; ein Artikel ist mit genau einem Fuhrpark verbunden; Zweck: Prüfung Gewichtsrestriktion.</li> <li>- Siehe Klasse AuftragsPosition.</li> </ul>	
<b>Oberklassen:</b>	–	
<b>Kommentar:</b>	<ul style="list-style-type: none"> <li>- Keine Löschoption für Artikel vorgesehen.</li> <li>- Keine Preishistorie vorgesehen; vgl. aber Klasse AuftragsPosition.</li> <li>- Keine Lagerfunktionen vorgesehen; kurzfristige Engpässe werden über Attribut ImAngebot registriert und bei Auftragsannahme beachtet.</li> </ul>	

<b>Klasse:</b>	<b>Entfernung</b>	<b>Entfernung</b>
<b>Beschreibung:</b>	Ein Entfernungs-Objekt gibt die Straßenentfernung zweier Kunden oder die Straßenentfernung eines Kunden zum Depot an.	

<b>Attribute:</b> <ul style="list-style-type: none"> <li>- Distanz: Float; {km, Distanz <math>\geq</math> 0.01}; Straßenentfernung zwischen den Lieferadressen der beiden mit dem Entfernungs-Objekt verbundenen Kunden; falls ein assoziierter Kunde die ID 0 besitzt, Straßenentfernung zwischen der Lieferadresse des anderen Kunden und dem Depot (Kunde 0); für Depotentfernungen von Kunden gilt ferner folgende Entfernungsrestriktion:  <math display="block">\text{Distanz} \leq \text{maxDepotDistanz},</math> wobei <math>\text{maxDepotDistanz} = \text{GeschwindigkeitNormal} \cdot \text{maxLenkzeit}/2</math>, d.h. jeder Kunde muss bei normaler Geschwindigkeit innerhalb der halben max. Lenkzeit vom Depot erreichbar sein; vgl. Klasse Fuhrpark.</li> <li>- <u>AnzBenoetigteEntfernungen</u>: UInt; {<u>AnzBenoetigteEntfernungen</u> = <math>n + n \cdot (n-1)/2</math>, wobei <math>n</math> – Kundenanzahl}; Anzahl aller zu erfassenden Entfernungen abhängig von der Kundenanzahl.</li> <li>- <u>AnzEntfernungen</u>: UInt = 0; Anzahl bereits erfasster Entfernungen.</li> <li>- <u>AnzFehlendeEntfernungen</u>: UInt; = <u>AnzBenoetigteEntfernungen</u> – <u>AnzEntfernungen</u>.</li> </ul>
<b>Operationen:</b> <ul style="list-style-type: none"> <li>- erfasse(); erfasst eine zusätzliche Entfernung.</li> <li>- aendere(); ändert eine vorhandene Entfernung, insbes. bei Änderung von Lieferadressen.</li> </ul>
<b>Assoziationen:</b> <ul style="list-style-type: none"> <li>- Entfernung <math>\rightarrow</math> Fuhrpark, eine Entfernung ist mit genau einem Fuhrpark verbunden; Zweck: Prüfung Entfernungsrestriktion.</li> <li>- Entfernung <math>\leftrightarrow</math> Kunde, jede Entfernung ist mit genau zwei Kunden-Objekten verbunden; ein Kunde ist mit beliebig vielen Entfernungen verbunden; Zweck: Interpretation eines Entfernungs-Objekts, Zugriff auf Entfernungen bei gegebenen Kunden.</li> </ul>
<b>Oberklassen:</b> <p>–</p>
<b>Kommentar:</b> <ul style="list-style-type: none"> <li>- Die Gesamtheit aller erfassten Entfernungen bildet das Entfernungswerk.</li> <li>- Das Entfernungswerk ist vollständig genau dann, wenn <u>AnzFehlendeEntfernungen</u> = 0.</li> <li>- Löschoperation wird nicht benötigt; vgl. Klasse Kunde.</li> </ul>

**Fahrzeug**

<b>Klasse:</b> <b>Fahrzeug</b>
<b>Beschreibung:</b> <p>Ein Fahrzeug-Objekt repräsentiert ein Fahrzeug des Fuhrparks.</p>
<b>Attribute:</b> <ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}; firmeninterne Fahrzeugnummer, ab 1 fortlaufend.</li> <li>- KfzNr: String; {key}; Kfz-Kennzeichen.</li> <li>- Einsetzbar: Bool; true: Fahrzeug ist zur Zeit einsetzbar, false : sonst.</li> <li>- Baujahr: Date; {optional}; Baujahr des Fahrzeugs.</li> </ul>

<ul style="list-style-type: none"> <li>- JahrAnschaffung: Date; {optional}; Anschaffungsjahr des Fahrzeugs.</li> <li>- NextTuev: Date; {optional}; nächster TÜV-Termin.</li> </ul>
<b>Operationen:</b> <ul style="list-style-type: none"> <li>- erfasse (); erfasst ein neues Fahrzeug.</li> <li>- aendere (); ändert Attributwerte eines vorhandenen Fahrzeugs.</li> <li>- loesche(); entfernt ein Fahrzeug, passt ID-Werte der verbleibenden Fahrzeuge an, so dass ID-Werte keine Lücken aufweisen.</li> </ul>
<b>Assoziationen:</b> <ul style="list-style-type: none"> <li>- Fahrzeug ↔ Fuhrpark; ein Fahrzeug ist mit einem Fuhrpark-Objekt verbunden; ein Fuhrpark-Objekt ist mit beliebig vielen Fahrzeugen verbunden; Komposition mit Fuhrpark als Aggregatobjekt; Zweck: gemeinsame Verwaltung der Fuhrparkdaten und der enthaltenen Fahrzeuge, Zugriff auf Fahrzeuge via (einziges) Fuhrpark-Objekt.</li> </ul>
<b>Oberklassen:</b> <ul style="list-style-type: none"> <li>-</li> </ul>
<b>Kommentar:</b> <ul style="list-style-type: none"> <li>- Ein Fahrzeug-Objekt fasst die individuellen Merkmale eines Fahrzeugs zusammen, während ein Fuhrpark-Objekt die übereinstimmenden Fahrzeugmerkmale bündelt.</li> </ul>

<b>Klasse:</b>	<b>Fuhrpark</b>
<b>Beschreibung:</b>	Das einzige Fuhrpark-Objekt beschreibt den Fuhrpark und fasst die übereinstimmenden Merkmale aller Fahrzeuge zusammen.
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- MaxNutzlast: Float; {kg, MaxNutzlast &gt; 0}; max. Nutzlast eines Fahrzeugs.</li> <li>- Geschwngkt: list(4) of Float; die Plätze 1 bis 4 entsprechen den Wetterlagen gemäß Typdefinition WETTER = enum (NORMAL, NEBEL, SCHNEE, GLATTEIS); Geschwngkt[i] gibt die Geschwindigkeit zur Wetterlage i (i = 1, 2, 3, 4) in km/h an.</li> <li>- MaxLenkzeit: Float; {h, MaxLenkzeit &gt; 0}; max. ununterbrochene Lenkzeit, Obergrenze für Tourdauer.</li> <li>- Pausenzeit: Float, {h, Pausenzeit &gt; 0}; vorgegebene Pausenzeit zwischen zwei Touren.</li> <li>- Fixkosten: Float, {Euro, Fixkosten &gt; 0}; Fixkosten pro Fahrzeug und Monat.</li> <li>- Varkosten: Float, {Euro, Varkosten &gt; 0}; variable Kosten pro Fahrzeug und km.</li> <li>- AnzFahrzeuge: UInt = 0, Gesamtzahl der Fahrzeuge im Fuhrpark.</li> <li>- AnzEinsetzbareFahrzeuge: UInt = 0, Gesamtzahl der aktuell einsetzbaren Fahrzeuge.</li> </ul>
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); legt einziges Fuhrpark-Objekt einmalig an.</li> <li>- aendere(); ändert Attribut-Werte des Fuhrpark-Objekts.</li> <li>- erstelleFuhrparkliste(); gibt Fuhrparkdaten und Daten aller zugehörigen Fahrzeuge aus.</li> </ul>

**Fuhrpark**

<b>Assoziationen:</b>
<ul style="list-style-type: none"> <li>- Komposition mit Fahrzeug, siehe Fahrzeug.</li> <li>- Weitere Assoziationen mit folgenden Klassen: Artikel, Entfernung, AuftragsPosition, siehe dort.</li> </ul>
<b>Oberklassen:</b>
–
<b>Kommentar:</b>
–

**Kunde**

<b>Klasse:</b>	<b>Kunde</b> {abstract}
<b>Beschreibung:</b>	Ein Kunden-Objekt der abstrakten Klasse Kunde umfasst die gemeinsamen Elemente von Lauf- und Stammkunden.
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly}; firmeninterne Kundennummer; ab 0 fortlaufend, Kunde 0 bezeichnet das Depot.</li> <li>- Firmenname: String = ""; Name der Kundenfirma.</li> <li>- PLZ: String = ""; Postleitzahl der Lieferadresse.</li> <li>- Ort: String = ""; Ort der Lieferadresse.</li> <li>- Strasse: String = ""; Straße und Hausnr. der Lieferadresse.</li> <li>- Kundentyp: KD TYP = LAUFKUNDE; Typ des Kunden, entweder Lauf- oder Stammkunde; KD TYP = enum (LAUFKUNDE, STAMMKUNDE).</li> <li>- PartnerName: String = ""; {optional}; Name Ansprechpartner.</li> <li>- PartnerAnrede: String = ""; {optional}; Anrede Ansprechpartner, (Herr, Frau, ggf. Titel).</li> <li>- PartnerTel: String = ""; {optional}; Telefon-Nr. Ansprechpartner.</li> <li>- PartnerFax: String = ""; {optional}; Fax-Nr. Ansprechpartner.</li> <li>- PartnerEMail: String = ""; {optional}; EMail-Adresse Ansprechpartner.</li> <li>- AnzAuftrQuartal: UInt; Auswertungskenngröße, Anzahl Aufträge pro Quartal.</li> <li>- <u>AnzKunden</u> : UInt = 0; Anzahl aller erfassten Kunden.</li> </ul>
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Kunden.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Kunden.</li> <li>- updateKundentyp(UInt IDKunde); wechselt den Typ des Kunden IDKunde; bei Wechsel von Laufkunde nach Stammkunde wird ein min. monatlicher Auftragswert von 10.000 Euro provisorisch eingetragen; bei Wechsel von Stammkunde nach Laufkunde werden ggf. alle Daueraufträge gelöscht.</li> <li>- <u>erstelleKundenliste()</u>; erstellt Liste aller Kunden alphabetisch sortiert nach Firmennamen.</li> <li>- <u>auswerteKunden</u>(UInt Jahr, UInt Quartal); erstellt eine Kundenauswertung für ein Quartal; Kenngrößen: AnzAuftrQuartal, weitere siehe abgeleitete Klassen.</li> </ul>

<b>Assoziationen:</b>
<ul style="list-style-type: none"> <li>- Assoziation mit Entfernung, siehe Klasse Entfernung.</li> <li>- Assoziation mit Auftrag, siehe Klasse Auftrag.</li> </ul>
<b>Oberklassen:</b>
—
<b>Kommentar:</b>
<ul style="list-style-type: none"> <li>- Kein Löschen von Kunden vorgesehen.</li> <li>- Vgl. abgeleitete Klassen LaufKunde und StammKunde.</li> <li>- Klasse Kunde ist abstrakt, weil es nur Lauf- oder Stammkunden gibt.</li> </ul>

<b>Klasse:</b>	<b>LaufKunde</b>
<b>Beschreibung:</b>	Ein LaufKunde-Objekt beinhaltet die spezifischen Elemente eines Laufkunden.
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- GesamterQuartalsAuftragswert: Float; {Euro}; Auswertungskenngröße, Summe der Auftragswerte aller abgerechneten Aufträge eines Quartals.</li> <li>- MittlQuartalsAuftragswert: Float; {Euro}; Auswertungskenngröße, Mittelwert der Auftragswerte aller abgerechneten Aufträge eines Quartals.</li> <li>- MittlAnzAuftrPos: Float; Auswertungskenngröße, Mittelwert der Anzahl der Auftragspositionen aller abgerechneten Aufträge eines Quartals.</li> <li>- <u>AnzLaufkunden</u> : UInt = 0; Anzahl aller Laufkunden.</li> </ul>
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Laufkunden.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Laufkunden.</li> <li>- <u>auswerteKunden</u>(UInt Jahr, UInt Quartal); erstellt eine Auswertung für das übergebene Quartal und alle Laufkunden alphabetisch sortiert nach Firmenname. Ermittelt werden die oben angegebenen Kenngrößen einschl. der Anzahl aller Aufträge im Quartal (siehe Klasse Kunde).</li> </ul>
<b>Assoziationen:</b>	Siehe Klasse Kunde.
<b>Oberklassen:</b>	Kunde.
<b>Kommentar:</b>	<ul style="list-style-type: none"> <li>- Löschoperation für Laufkunden nicht vorgesehen.</li> <li>- Auswertungen werden nach Erstellung ausgegeben (Anzeige/Druck) und nicht gespeichert; die Kenngrößen-Attributwerte werden pro Auswertung neu erzeugt.</li> </ul>

LaufKunde

StammKunde

<b>Klasse:</b>	<b>StammKunde</b>
<b>Beschreibung:</b>	Ein StammKunde-Objekt beinhaltet die spezifischen Elemente eines Stammkunden.
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- MinMtlAuftrwert: Float; {Euro, MinMtlAuftrwert &gt; 0}; vereinbarte minimale monatliche Wertsumme aller (abgerechneten) Aufträge des Stammkunden.</li> <li>- MtlAuftrwert: Float; {Euro}; tatsächlich erreichte monatliche Wertsumme aller abgerechneten Aufträge des Stammkunden.</li> <li>- MittlMtlAuftrwert: Float; {Euro}; Auswertungskenngröße, mittlere erreichte monatliche Wertsumme aller abgerechneten Aufträge des Stammkunden in einem Quartal.</li> <li>- QDAMittlMtlAuftrwert: Float; {%}; Auswertungskenngröße, prozentualer Anteil der Daueraufträge an der mittleren erreichten monatlichen Wertsumme aller abgerechneten Aufträge des Stammkunden in einem Quartal.</li> <li>- WertUnterschreitung: Bool; Auswertungskenngröße; true genau dann, wenn in einem Quartal gilt: MittlMtlAuftrwert &lt; MinMtlAuftrwert, d.h. wenn der mittlere monatl. Auftragswert den vereinbarten monatlichen Minimalwert des Auftragsvolumens unterschreitet.</li> <li>- <u>AnzStammkunden</u>: UInt = 0; Anzahl aller Stammkunden.</li> </ul>
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Stammkunden.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Stammkunden.</li> <li>- <u>auswerteKunden</u>(UInt Jahr, UInt Quartal); erstellt eine Auswertung für das übergebene Quartal und alle Stammkunden alphabetisch sortiert nach Firmenname. Ermittelt werden die oben angegebenen Kenngrößen einschl. der Anzahl aller Aufträge im Quartal (siehe Klasse Kunde).</li> </ul>
<b>Assoziationen:</b>	Siehe Klasse Kunde.
<b>Oberklassen:</b>	Kunde.
<b>Kommentar:</b>	<ul style="list-style-type: none"> <li>- Für die Auswertungskenngrößen gelten analoge Bemerkungen wie bei LaufKunde. Auch der Attributwert MtlAuftrwert wird bei jeder Monatsabrechnung von Stammkunden jeweils neu erzeugt.</li> </ul>

### Paket Auftragsabwicklung

#### Auftrag

<b>Klasse:</b>	<b>Auftrag</b> {abstract}
<b>Beschreibung:</b>	Ein Auftrags-Objekt der abstrakten Klasse Auftrag umfasst die gemeinsamen Elemente von Einzel- und Daueraufträgen.
<b>Attribute:</b>	<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}; identifizierende Auftragsnummer, ab 1 fortlaufend.</li> <li>- AuftragsTyp: AUFTRAGSTYP; Typ des Auftrags, entweder Einzel- oder Dauerauftrag; AUFTRAGSTYP = enum (EINZELAUFTAG, DAUERAUFTRAG).</li> <li>- Annahmedatum: Date; Datum der Annahme des Auftrags.</li> <li>- AnzAuftrpos: UInt = 0; {AnzAuftrpos &gt; 0}; Anzahl der Auftragspositionen.</li> <li>- Auftragswert: Float = 0; {Euro, Auftragswert &gt; 0}; Gesamtwert des Auftrags.</li> <li>- Auftragsgewicht: Float = 0; {kg, 0 &lt; Auftragsgewicht ≤ Fuhrpark.MaxNutzlast}; Gesamtgewicht des Auftrags.</li> <li>- <u>AnzAuftraege</u>: UInt = 0; Anzahl aller Aufträge.</li> </ul>
<b>Operationen:</b>	<ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Auftrag.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Auftrags.</li> <li>- loesche(); löscht einen vorhandenen Auftrag, IDs der verbleibenden Aufträge bleiben erhalten.</li> <li>- <u>erstelleAuftragsliste()</u>; erstellt Liste aller Aufträge.</li> </ul>
<b>Assoziationen:</b>	<ul style="list-style-type: none"> <li>- Auftrag → Fuhrpark; ein Auftrag ist mit genau einem Fuhrpark verbunden; Zweck: Prüfung Gewichtsrestriktion.</li> <li>- Auftrag → Kunde; ein Auftrag ist mit genau einem Kunden verbunden; Restriktion: Ist Auftrag ein DauerAuftrag, so muss Kunde ein StammKunde sein; Zweck: Zugriff auf Daten des Kunden eines Auftrags für Auftragsanzeige, -abrechnung sowie Kundenauswertung.</li> <li>- Auftrag ↔ AuftragsPosition, ein Auftrag ist mit mindestens einer Auftragsposition verbunden, eine Auftragsposition ist mit genau einem Auftrag verbunden; Komposition mit Auftrag als Aggregatobjekt. Zweck: gemeinsame Verwaltung der Auftragsdaten und seiner Auftragspositionen, Zugriff auf Auftragspositionen via Auftrag.</li> </ul>
<b>Oberklassen:</b>	–
<b>Kommentar:</b>	<ul style="list-style-type: none"> <li>- Die Manipulation der Attribute AnzAuftrpos, Auftragswert und Auftragsgewicht wird ausschließlich durch die Klasse AuftragsPosition initiiert.</li> <li>- Vgl. abgeleitete Klassen EinzelAuftrag und DauerAuftrag sowie die Klasse AuftragsPosition, insbes. für weitere Informationen zu den Operationen.</li> <li>- Klasse Auftrag ist abstrakt, weil es nur Einzel- und Daueraufträge gibt.</li> </ul>
<b>Klasse:</b>	<b>AuftragsPosition</b>

AuftragsPosition



<b>Beschreibung:</b>	
Ein AuftragsPosition-Objekt repräsentiert eine Auftragsposition eines Auftrags.	
<b>Attribute:</b>	
<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}, Auftragspositionsnummer, ab 1 fortlaufend.</li> <li>- AnzVrkfseinheiten: UInt = 1; {AnzVrkfseinheiten &gt; 0}, Anzahl Verkaufseinheiten des assoziierten Artikels (siehe Assoziationen).</li> <li>- Auftrposwert: Float; {Euro}; Gesamtwert der Auftragsposition, gegeben durch AnzVrkfseinheiten • Artikel.Preis.</li> <li>- Auftrposgewicht: Float; {kg, 0 &lt; Auftrposgewicht ≤ Fuhrpark.MaxNutzlast}, Gesamtgewicht der Auftragsposition, gegeben durch AnzVrkfseinheiten • Artikel.Gewicht.</li> <li>- AnnArtikelpreis: Float; {Euro}; Verkaufspreis des assoziierten Artikels zum Annahmedatum des Auftrags, dient der Kontrolle der Abrechnung, da keine Preishistorie vorgesehen, vgl. Klasse Artikel.</li> </ul>	
<b>Operationen:</b>	
<ul style="list-style-type: none"> <li>- erfasse(), erfasst eine neue Auftragsposition, aktualisiert zugleich zugehörige Auftragsdaten (AnzAuftrpos, Auftragswert, Auftragsgewicht) und garantiert Einhaltung Gewichtsrestriktion für Auftragsposition und Auftrag.</li> <li>- aendere(); ändert Attributwerte einer vorhandenen Auftragsposition, geht analog vor wie Operation erfasse().</li> <li>- loesche(), löscht vorhandene Auftragsposition, aktualisiert ebenfalls zugehörige Auftragsdaten, passt ID-Werte der verbleibenden Auftragspositionen an (vgl. Klasse Fahrzeug).</li> </ul>	
<b>Assoziationen:</b>	
<ul style="list-style-type: none"> <li>- AuftragsPosition → Fuhrpark; eine Auftragsposition ist mit genau einem Fuhrpark verbunden; Zweck: Prüfung Gewichtsrestriktion.</li> <li>- Komposition mit Auftrag, siehe Klasse Auftrag.</li> <li>- AuftragsPosition → Artikel; eine Auftragsposition ist mit genau einem Artikel verbunden, Zweck: Zugriff auf Artikeldaten pro Auftragsposition.</li> </ul>	
<b>Oberklassen:</b>	
–	
<b>Kommentar:</b>	
<ul style="list-style-type: none"> <li>- Bei einem Einzelauftrag können Auftragspositionen nur bearbeitet werden, wenn der Einzelauftrag den Status „ANGENOMMEN“ besitzt, siehe Klasse EinzelAuftrag.</li> </ul>	

**DauerAuftrag**

<b>Klasse:</b>	<b>DauerAuftrag</b>
<b>Beschreibung:</b>	
Ein DauerAuftrag-Objekt beinhaltet die spezifischen Elemente eines Dauerauftrages.	

<b>Attribute:</b> <ul style="list-style-type: none"> <li>- Ausführungstage: list(5) of Bool, Plätze 1 bis 5 entsprechen den Wochentagen Montag bis Freitag; Ausführungstage[i] = true genau dann, wenn Dauerauftrag am Wochentag i auszuführen ist (i = 1,..., 5).</li> <li>- AnzEinzelauftraege: UInt = 0; Anzahl der zum Dauerauftrag erzeugten Einzelaufträge.</li> <li>- <u>AnzDauerauftraege</u>: UInt = 0; Anzahl aller Daueraufträge.</li> </ul>
<b>Operationen:</b> <ul style="list-style-type: none"> <li>- erfasse(); erfasst einen neuen Dauerauftrag; sichert, dass Dauerauftrag an mindestens einem Wochentag auszuführen ist; zugehöriger Kunde muss Stammkunde sein.</li> <li>- aendere(); ändert Attributwerte eines vorhandenen Dauerauftrags, sichert ebenfalls vorher genannte Konsistenzbedingungen.</li> <li>- loesche(); löscht einen Dauerauftrag, zugehörige Einzelaufträge bleiben erhalten, jedoch wird Verweis auf Dauerauftrag gelöscht (siehe Assoziationen).</li> <li>- generiereEinzelauftrag(Date Datum); generiert zum Dauerauftrag einen Einzelauftrag mit Status „ANGENOMMEN“ und Annahmedatum = Datum, wenn der Dauerauftrag am Wochentag des übergebenen Datums auszuführen ist.</li> <li>- <u>generiereEinzelauftraege</u>(Date Datum); erzeugt für alle Daueraufträge aller Stammkunden ggf. einen Einzelauftrag.</li> <li>- <u>erstelleAuftragsliste</u>(); erstelle Liste aller Daueraufträge aufsteigend sortiert nach Annahmedatum.</li> </ul>
<b>Assoziationen:</b> <ul style="list-style-type: none"> <li>- Siehe Klasse Auftrag.</li> <li>- DauerAuftrag ↔ EinzelAuftrag, ein Dauerauftrag ist mit beliebig vielen, nämlich den aus ihm erzeugten Einzelaufträgen verbunden; ein Einzelauftrag ist mit keinem oder einem, in diesem Fall mit dem erzeugenden Dauerauftrag verbunden; Aggregation mit DauerAuftrag als Aggregatobjekt. Zweck: Rechnungserstellung ggf. mit Hinweis auf zugehörigen Dauerauftrag, Stammkundenauswertung (s. Klasse StammKunde), Löschen von Daueraufträgen.</li> </ul>
<b>Oberklassen:</b> Auftrag.
<b>Kommentar:</b> —

<b>Klasse:</b> <b>EinzelAuftrag</b>
<b>Beschreibung:</b>  Ein EinzelAuftrag-Objekt beinhaltet die spezifischen Elemente eines Einzelauftrages.
<b>Attribute:</b> <ul style="list-style-type: none"> <li>- Status: AUFSTATUS=ANGENOMMEN; Status des Einzelauftrags, mögliche Stati siehe folgende Typdefinition, AUFSTATUS = enum (ANGENOMMEN, GELIEFERT, ABGERECHNET, BEZAHLT).</li> </ul>

EinzelAuftrag

- Lieferdatum: Date = MaxDatum, {Lieferdatum  $\geq$  Annahmedatum}, Datum der Auftragslieferung, Startwert MaxDatum ist ein weit in der Zukunft liegendes Datum.
- Rechnungsdatum: Date = MaxDatum, {Rechnungsdatum  $\geq$  Lieferdatum}, Datum der Auftragsabrechnung, MaxDatum siehe vorher.
- Zahlungsdatum: Date = MaxDatum, {Zahlungsdatum  $\geq$  Rechnungsdatum}, Datum der Auftragsbezahlung, MaxDatum siehe vorher.
- AnzEinzelauftraege: UInt = 0; Anzahl aller Einzelaufträge.
- AnzAuftraegeAngenommen: UInt = 0; Anzahl angenommener Einzelaufträge.
- AnzAuftraegeGeliefert: UInt = 0; Anzahl gelieferter Einzelaufträge.
- AnzAuftraegeAbgerechnet: UInt = 0; Anzahl abgerechneter Einzelaufträge.

#### Operationen:

- erfasse(); erfasst einen neuen Einzelauftrag mit Status „ANGENOMMEN“, sichert Existenz wenigstens einer Auftragsposition.
- aendere(); ändert Attributwerte eines vorhandenen Einzelauftrags, sichert Existenz wenigstens einer Auftragsposition, nur bei Status „ANGENOMMEN“ anwendbar.
- loesche(); löscht einen Einzelauftrag, nur bei Status „ANGENOMMEN“ anwendbar.
- setBackLieferstatus(); setzt den Status von „GELIEFERT“ auf „ANGENOMMEN“ zurück; ist von GUI aus anzuwenden, wenn Auslieferung eines disponierten Auftrags nicht erfolgreich war, d.h. wenn kein quittierter Lieferschein vorliegt.
- erstelleAuftragsliste(); erstellt Liste aller Einzelaufträge aufsteigend sortiert nach Annahmedatum.
- erstelleRechnungen(KDTYP Kundentyp, Date AktRechnungsdatum); erstellt Rechnungen für Einzelaufträge aller Kunden mit dem Status „GELIEFERT“. Alle abgerechneten Einzelaufträge erhalten den Status „ABGERECHNET“ und das aktuelle Datum „Heute“ als Rechnungsdatum.
- Für den Kundentyp LAUFKUNDE werden alle gelieferten Aufträge abgerechnet.
- Für den Kundentyp STAMMKUNDE werden alle Aufträge abgerechnet mit Lieferdatum.Monat = AktRechnungsdatum.Monat, wobei AktRechnungsdatum.Monat ein abgelaufener Monat sein muss. Darüber hinaus wird für alle Stammkunden eine Monatsrechnung erzeugt, die den Gesamtwert aller abgerechneten Aufträge und deren Auftragsnummern ausweist.
- erstelleRechnung(); erstellt Rechnung für einen Einzelauftrag; vgl. Operation erstelleRechnungen().

#### Assoziationen:

- Siehe Klasse DauerAuftrag und Klasse Tour.

#### Oberklassen:

Auftrag.

#### Kommentar:

- Die Attribute Status, Lieferdatum und Rechnungsdatum können abgesehen von der Operation setBackLieferstatus() nicht nutzerseitig via GUI gesetzt werden; vgl. Zustandsdiagramm der Klasse EinzelAuftrag.

- Einzelaufträge können sowohl nutzerseitig erfasst als auch automatisch aus Daueraufträgen generiert werden, vgl. Klasse DauerAuftrag.
- Status „BEZAHLT“ und Zahlungsdatum sind nur Teil der vorgesehenen Datenschnittstelle zur Buchhaltung und werden im TPS nicht benutzt.
- Da Rechnungen nicht abgespeichert werden, ist auch eine wiederholte Rechnungserstellung für bereits abgerechnete Aufträge vorzusehen.

<b>Klasse:</b>	<b>Tour</b>	<b>Tour</b>
<b>Beschreibung:</b>		
Ein Tour-Objekt repräsentiert eine Tour eines Tourenplans.		
<b>Attribute:</b>		
<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}; Tourennummer, fortlaufend ab 1.</li> <li>- TourKfz: String: „,,“; Kfz-Nr. des Lieferfahrzeuges der Tour.</li> <li>- Tourstrecke: Float; {km}; gesamte Tourstrecke.</li> <li>- /Tourdauer: Float; {h, Tourdauer ≤ Fuhrpark.MaxLenkzeit}; gesamte Tourdauer.</li> <li>- StartZeitDepot: Time; Startzeit am Depot zum Lieferdatum.</li> <li>- /AnZeitDepot: Time; Ankunftszeit am Depot zum Lieferdatum.</li> <li>- AnzTourKunden: UInt; Anzahl aller Tourkunden.</li> <li>- Tourgewicht: Float; {kg, Tourgewicht ≤ Fuhrpark.MaxNutzlast}; Frachtgewicht, d.h. Gewichtssumme aller Touraufträge.</li> <li>- Tourfrachtwert: Float; {Euro}; Wertsumme aller Touraufträge, ist Tourselektions-Kriterium, vgl. Klasse Tourenplan.</li> <li>- Tourterminverzug: UInt; Summe der Differenzen Lieferdatum – Annahmedatum über alle Touraufträge; ist Tourselektions-Kriterium, vgl. Klasse Tourenplan.</li> <li>- AnzTourAuftraege: UInt; {AnzTourAuftraege &gt; 0}; Anzahl aller mit der Tour ausgelieferten Aufträge.</li> <li>- TourAuftraege: list(AnzTourAuftraege) of TourAuftrag; Liste der Touraufträge, beschreibt die Route einer Tour; ein Listenelement des Strukturtyps TourAuftrag besitzt folgende Komponenten: <ul style="list-style-type: none"> <li>- Verweis auf den zu liefernden Einzelauftrag,</li> <li>- Kunden-ID des Auftrages,</li> <li>- Teilstrecke ab Depot zum Kundenort in km,</li> <li>- Ankunftszeit am Kundenort.</li> </ul> </li> </ul>		
<b>Operationen:</b>		
<ul style="list-style-type: none"> <li>- erzeugeTour(Time DefaultStartzeit); erzeugt neue Tour mit allen zugehörigen Daten bis auf Fahrzeug und (echte) Startzeit.</li> <li>- komplettiereTour(Time Startzeit, String KfzNr); komplettiert eine Tour um Startzeit und Fahrzeug. Passt Ankunftszeiten bei Kunden und Depot an.</li> <li>- ausgebenTour(); gibt eine Tour aus.</li> <li>- updateAuftraege(Bool direktion, Time Lieferdatum); setzt Status und Lieferdatum für alle Touraufträge; falls direktion = true, wird der Status von „ANGENOMMEN“ auf „GELIEFERT“ umgesetzt, andernfalls umgekehrt, wobei als Lieferdatum wieder MaxDatum eingetragen wird (vgl. Klasse EinzelAuftrag).</li> <li>- erstelleLieferschein(); erstellt pro Tourauftrag einen Lieferschein; vgl. gleichnamige Operation der Klasse Tourenplan.</li> </ul>		

**Assoziationen:**

- Komposition mit Klasse Tourenplan, siehe Klasse Tourenplan.
- Tour → EinzelAuftrag, eine Tour ist mit beliebig vielen, jedoch mindestens einem Einzelauftrag verbunden, nämlich den mit der Tour gelieferten Aufträgen; aufgrund der Anordnung der Auftragsverweise in der Liste TourAufträge wird eine geordnete Assoziation realisiert; Zweck: Ausgabe von Touren mit zugehörigen Auftragsdaten, Aktualisierung des Status geplanter Aufträge.

**Oberklassen:**

–

**Kommentar:**

–

**Tourenplan**

<b>Klasse:</b>	<b>Tourenplan</b>
<b>Beschreibung:</b>	
Ein Tourenplan-Objekt repräsentiert einen generierten Tourenplan. Die Klasse Tourenplan kapselt die zur Generierung eines Tourenplanes erforderlichen Operationen, insbesondere den Savings-Algorithmus. Sie enthält ferner Klassenoperationen zur Verwaltung und Auswertung von Tourenplänen.	
<b>Attribute:</b>	
<ul style="list-style-type: none"> <li>- ID: UInt; {key, readonly, ID &gt; 0}; Tourenplannummer, fortlaufend ab 1.</li> <li>- Lieferdatum: Date; Datum der Belieferung, zugleich Planungsdatum, da Tourenpläne am Planungstag realisiert werden.</li> <li>- DefaultStartzeit: Time; provisorische Startzeit, siehe Op. planeTouren().</li> <li>- Wetter: WETTER; aktuelle Wetterlage, Typ WETTER siehe Klasse Fuhrpark</li> <li>- /Geschwndgkt: Float; Geschwindigkeit zur aktuellen Wetterlage, vgl. Fuhrpark.</li> <li>- Tourauswahl: TOURAUSWAHL; Tourauswahlkriterium, Varianten gemäß folgender Typdefinition; TOURAUSWAHL = enum {FRACHTWERT, TERMINVERZUG}, Bedeutung siehe Pflichtenheft.</li> <li>- AnzPlanAuftraege: UInt = 0; Anzahl aller geplanten Aufträge.</li> <li>- AnzFehlAufträge: UInt = 0; Anzahl angenommener, aber nicht geplanter Aufträge.</li> <li>- AnzTouren: UInt = 0; Anzahl der Touren des Tourenplans.</li> <li>- <u>AnzTourenplaene</u>: UInt = 0; Anzahl aller erzeugten Tourenpläne.</li> </ul> <p>// Es folgen die Auswertungskenngrößen für eine Quartalsauswertung der Tourenplanung.</p> <p>// Die zugehörigen Attributwerte werden bei Bedarf erzeugt und nur temporär benötigt.</p> <p>// Alle Kenngrößen beziehen sich auf das Auswertungsquartal.</p> <ul style="list-style-type: none"> <li>- <u>AnzAuswTourenplaene</u>: UInt ; Anzahl Tourenplaene.</li> <li>- <u>AnzAuswTouren</u>: UInt; Anzahl Touren.</li> <li>- <u>AnzAuswAuftraege</u>: UInt; Anzahl gelieferte Aufträge.</li> <li>- <u>SumFrachtwert</u>: Float; {Euro}; gesamter Frachtwert.</li> </ul>	

- SumGewicht: Float; {kg}; gesamtes transportiertes Gewicht.
- SumStrecke: Float; {km}; gesamte gefahrene Strecke.
- SumFixkosten: Float; {Euro}; Summe Fixkosten.
- SumVarkosten: Float; {Euro}; Summe variable Kosten.
- SumKosten: {Euro}; Summe fixe und variable Kosten.
- MittlStrecke: Float; {km}; mittlere Strecke einer Tour.
- MittlGewichtsauslastung: { % }; mittlere Gewichtsauslastung pro Tour; Gewichtsauslastung einer einzelnen Tour ist Quotient Tourgewicht/max. Nutzlast.
- MittlFrachtwert: Float; {Euro}; mittlerer Frachtwert pro Tour.
- MittlTerminverzug: Float; {Tage}; mittlerer Terminverzug pro Auftrag; Terminverzug eines (Einzel-)Auftrags ist Differenz Lieferdatum minus Annahmedatum.
- QWertKosten: Float; Quotient SumFrachtwert / SumKosten.

**Operationen:**

- planeTouren(); erzeugt einen Tourenplan, jedoch noch ohne Fahrzeugzuweisung an Touren und mit provisorischer einheitlicher Startzeit für alle Touren. Die Operation entspricht der ersten Phase einer Tourenplanung und umfasst folgende Schritte:
  1. Erfassung Planungsdaten Lieferdatum, DefaultStartzeit, Wetter und Tourauswahl.
  2. Prüfung der Bedingungen für eine Tourenplanung und Auftragskomplettierung:
    - Prüfung Vollständigkeit des Entfernungswerks,
    - Prüfung Existenz mindestens eines einsetzbaren Fahrzeugs,
    - Generierung von Einzelaufträgen aus Daueraufträgen zum Lieferdatum,
    - Prüfung Existenz mindestens eines angenommenen Einzelauftrags.
  3. Durchführung des Savingsalgorithmus ggf. einschließlich Tourselektion, vgl. Pflichtenheft. Abspeicherung der erzeugten Touren als Tour-Objekte.
- komplettiereTouren(); vervollständigt alle Touren durch Zuweisung von Fahrzeug und gültiger Startzeit. Diese und die folgenden beiden Operationen entsprechen der zweiten Phase einer Tourenplanung. Die Operation komplettiereTouren():
  - sichert, dass nur einsetzbare Fahrzeuge zugewiesen werden,
  - sichert, dass Startzeiten am Liefertag nicht vor der frühest möglichen Startzeit unter Beachtung evtl. vorheriger Fahrzeugeinsätze und Pausenzeiten liegen und dass alle Touren am Liefertag enden,
- aktualisiert nach Komplettierung aller Touren die geplanten Aufträge; setzt Stati aller geplanten Aufträge von „ANGENOMMEN“ auf „GELIEFERT“ und trägt das Lieferdatum ein.
- entferneKompletteTour(UInt TourID); entfernt eine komplette Tour aus dem Tourenplan; wird angeboten, um eine durch planeTouren() erzeugte Tour, die nicht konsistent komplettiert werden kann (vgl. Op. komplettiereTouren()) zu beseitigen, ohne den Tourenplan als Ganzes zu verlieren.

- `ausgebenTourenplan()`; gibt einen erstellten Tourenplan einschließlich aller Touren aus.
- `erstelleFahrplananweisungen()`; erstellt pro Tour eines Tourenplans eine Fahrplananweisung; diese enthält alle für die Durchführung einer Tour relevanten Tourdaten, darunter:
  - Lieferdatum, Fahrzeug, Abfahrtszeit und Ankunftszeit, Tourdauer und -strecke
  - alle Kunden und deren Lieferadressen sortiert entsprechend der Tourroute.
- `erstelleLieferscheine()`; erstellt pro Tour und pro Tourauftrag einen Lieferschein; dieser enthält alle für die Lieferung eines Auftrags relevanten Auftragsdaten, darunter:
  - Firmenname und Lieferadresse, Annahme- und Lieferdatum, Auftragswert,
  - Anzahl und Daten aller Auftragspositionen.
- `erstelleTourenplanliste()`; erstellt eine Liste aller erzeugten Tourenpläne;
- `loescheTourenplan()`; löscht einen Tourenplan; nur anwendbar, wenn kein Tourauftrag bereits abgerechnet wurde; Auftragsstatus und Lieferdatum der Touraufträge werden zurückgesetzt; vgl. Op. `updateAuftraege()` der Klasse Tour. Darf nur eingesetzt werden, um einen noch nicht realisierten Tourenplan zu entfernen! Kontrolle durch Disponent.
- `auswerteTourenplaene(UInt Jahr, UInt Quartal)`; wertet alle Tourenpläne des übergebenen Quartals aus; Auswertungskenngrößen siehe oben.

**Assoziationen:**

- Tourenplan ↔ Tour; jeder Tourenplan ist mit beliebig vielen, jedoch mindestens einer Tour verbunden, eine Tour ist mit genau einem Tourenplan verbunden; Komposition mit Tourenplan als Aggregatobjekt; Zweck: gemeinsame Erzeugung, Ausgabe und Verwaltung von Tourenplänen und zugehörigen Touren.

**Oberklassen:**

–

**Kommentar:**

- Assoziationen zu Auftrag bzw. abgeleiteten Klassen von Auftrag, Kunde, Entfernung und Fuhrpark sind nicht vorgesehen. Bei der Erstellung eines Tourenplans werden alle Objekte der Klassen EinzelAuftrag und DauerAuftrag durchlaufen, um angenommene Aufträge zu selektieren bzw. zu erzeugen. Der Zugriff auf Kunden sowie Entfernungen erfolgt dann über die Assoziationen der Klasse Auftrag bzw. Kunde. Zu dem einzigen Fuhrpark-Objekt wird bei der Erzeugung eines Tourenplanes nur eine temporäre Verbindung zwecks Kontrolle der Bedingungen einer Tourenplanung und Zugriff auf die assoziierten Fahrzeuge hergestellt. Nach Fertigstellung eines Tourenplans werden die angesprochenen Assoziationen ebenfalls nicht benötigt.

Szenarios

**4.4.3 Szenarios**

Artikel erfassen



## 1. Szenario „Artikel erfassen“

Szenario	Artikel erfassen
Zugehöriger Anwendungsfall	Stammdaten verwalten.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Artikel verwalten,</li> <li>- Artikel erfassen,</li> <li>- erforderliche Daten für neuen Artikel vorhanden,</li> <li>- Fuhrpark bereits erfasst.</li> </ul>
Ergebnis:	Neuer Artikel erfasst.

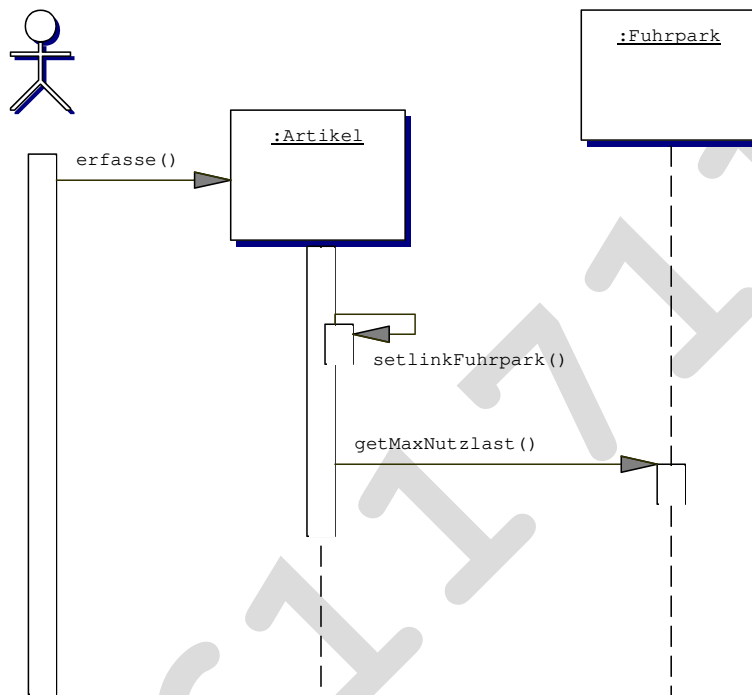


Abb. 4.30. Sequenzdiagramm zum Szenario „Artikel erfassen“.

### Erläuterung:

- Bei der Erfassung eines neuen Artikels wird ein neues Objekt der Klasse Artikel erzeugt.
- Um die Einhaltung der Gewichtsrestriktion zu gewährleisten, wird eine Verbindung der Assoziation Artikel → Fuhrpark zu dem (einzigen) Fuhrpark-Objekt aufgebaut. Anschließend wird mittels der Verbindung das Fuhrpark-Attribut MaxNutzlast gelesen und die Einhaltung der Restriktion geprüft.
- Der Aufruf einer Klassenoperation der Klasse Fuhrpark zur Bereitstellung einer Referenz auf das einzige Fuhrpark-Objekt wird nicht dargestellt. Analog wird in den folgenden Sequenzdiagrammen verfahren.

**Auftrag erfassen**

## 2. Szenario „Auftrag erfassen“

Szenario	Auftrag erfassen
Zugehöriger Anwendungsfall	Aufträge verwalten.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Auftrag erfassen,</li> <li>- erforderliche Daten eines neuen Auftrages (Einzel- oder Dauerauftrag) liegen vor,</li> <li>- Fuhrpark bereits erfasst,</li> <li>- zugehöriger Kunde bereits erfasst,</li> <li>- gewünschte Artikel bereits erfasst und aktuell im Angebot.</li> </ul>
Ergebnis:	Neuer Auftrag erfasst.

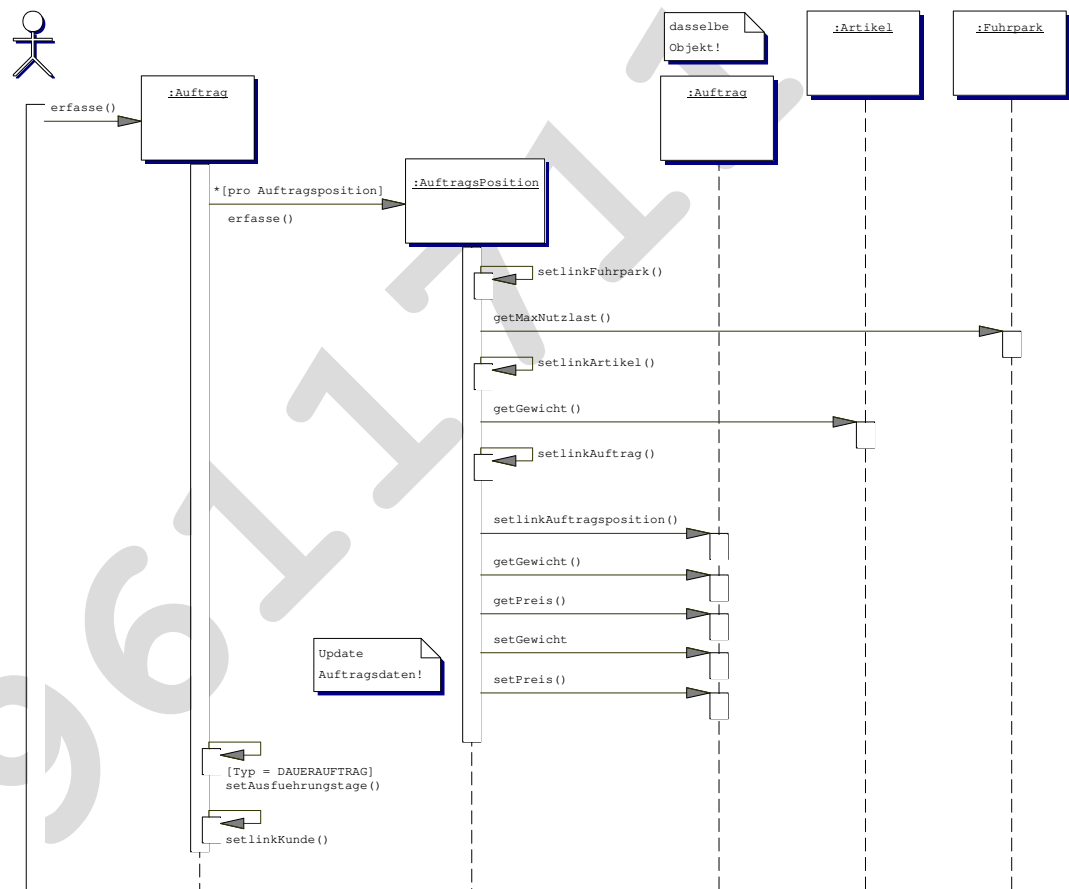


Abb. 4.31. Sequenzdiagramm zum Szenario „Auftrag erfassen“.

### Erläuterung:

- Neu erzeugt werden Verbindungen der Assoziationen Auftrag ↔ AuftragsPosition, AuftragsPosition → Fuhrpark, AuftragsPosition → Artikel und Auftrag → Kunde. Bereits bestehende Assoziationen werden nicht genutzt.
- Nicht dargestellt wurde, dass neben dem Gewicht weitere Artikel-Attribute gelesen werden.
- Das Prüfen der Gewichtsrestriktion für einen kompletten Auftrag und das Setzen der Auftragsattribute Gewicht und Preis wird nur von Objekten der Klasse AuftragsPosition vorgenommen.
- Da während der Operation erfasse() auf Auftragsattribute zugegriffen wird, erscheint dasselbe Auftrags-Objekt zweimal.

## 3. Szenario „Tourenplan erzeugen“

## Tourenplan erzeugen

Szenario	Tourenplan erzeugen
Zugehöriger Anwendungsfall	Tourenplan erzeugen.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Fuhrpark erfasst und enthält mindestens ein einsetzbares Fahrzeug,</li> <li>- Entfernungswerk vollständig,</li> <li>- Einzelaufträge mit Status „ANGENOMMEN“ vorhanden; dabei kann es sich auch um Einzelaufträge handeln, die aus Daueraufträgen erst zum Lieferdatum des Tourenplanes generiert werden.</li> </ul>
Ergebnis:	Tourenplan zum Lieferdatum vorhanden.

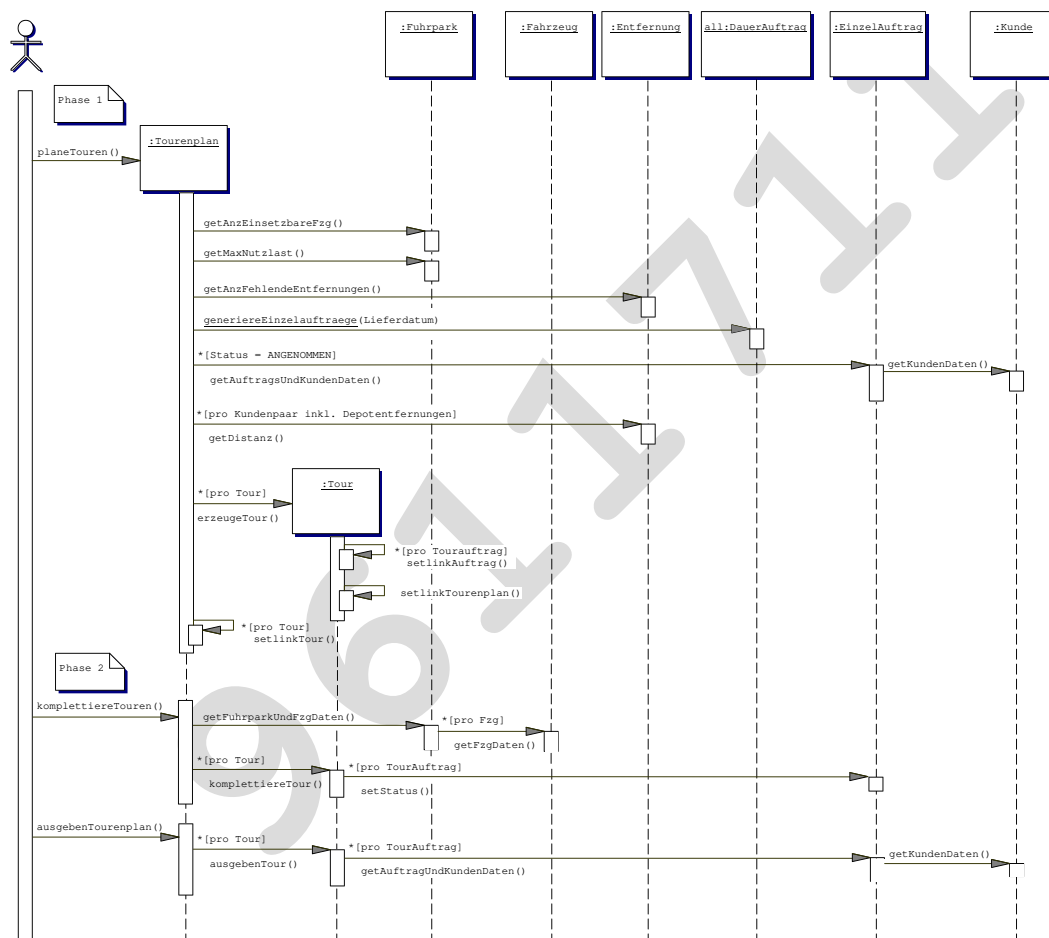


Abb. 4.32. Sequenzdiagramm zum Szenario „Tourenplan erzeugen“.

## Erläuterung:

- Neu erzeugt werden Verbindungen der Assoziationen Tourenplan ↔ Tour, Tour → Einzelauftrag sowie DauerAuftrag ↔ Einzelauftrag.
- Benutzt werden die bereits bestehenden Assoziationen Auftrag → Kunde, Kunde ↔ Entfernung, Fuhrpark ↔ Fahrzeug.
- Die Erzeugung von Einzelaufträgen aus Daueraufträgen wurde verkürzt dargestellt.
- Der Zugriff auf Einzelauftrag-Objekte in der Phase 1 der Tourenplanung durch das neu erzeugte Tourenplan-Objekt erfolgt nicht mittels Assoziation. Vielmehr werden alle vorhandenen Einzelaufträge durchlaufen. Zu dem Fuhrpark-Objekt

einschliesslich der assoziierten Fahrzeuge wird ebenso nur eine temporäre Verbindung hergestellt.

- Nicht dargestellt wurde, dass in der Phase 2 vor der Zuweisung von Fahrzeug und Startzeit an geplante Touren für jedes Fahrzeug die früheste Startzeit aus bereits vorhandenen Tourenplänen zu ermitteln ist.
- Die Aktualisierung des Auftrags-Status in Phase 2 der Tourenplanung wurde lediglich durch die Zugriffsoperation setStatus() der Klasse EinzelAuftrag dargestellt, während die Operation updateAuftraege() der Klasse Tour nicht eingetragen wurde.
- Das Lesen mehrerer Attribute eines oder mehrerer Objekte mittels get-Operationen wird in Pseudooperationen getxxxDaten() zusammengefasst.

#### Lieferscheine erstellen

#### 4. Szenario „Lieferscheine erstellen“

Szenario	Lieferscheine erstellen
Zugehöriger Anwendungsfall	Lieferscheine und Fahrtanweisungen erstellen.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Lieferscheine zu gewählttem Lieferdatum und Tourenplan erstellen,</li> <li>- mindestens ein Tourenplan zum Lieferdatum vorhanden.</li> </ul>
Ergebnis:	Pro Tour und Auftrag des Tourenplans ein Lieferschein vorhanden.

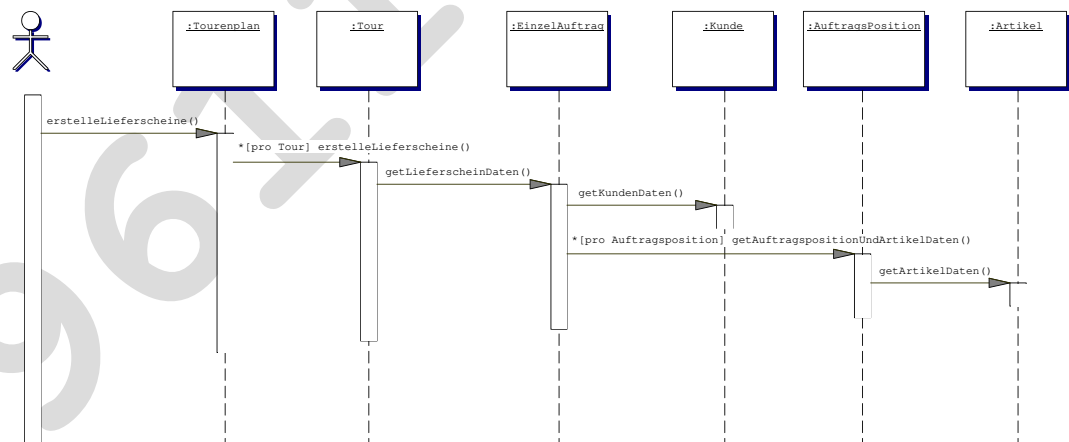


Abb. 4.33. Sequenzdiagramm zum Szenario „Lieferscheine erstellen“.

#### Erläuterung:

- Benutzt werden die bereits bestehenden Assoziationen Tourenplan ↔ Tour, Tour → EinzelAuftrag, Auftrag → Kunde, Auftrag ↔ AuftragsPosition und AuftragsPosition → Artikel. Neue Verbindungen sind nicht einzurichten.
- Pseudooperationen getxxxDaten() lesen mehrere Attribute eines oder mehrerer Objekte; vgl. 3. Szenario.

#### Rechnungen für Laufkunden erstellen

## 5. Szenario „Rechnungen für Laufkunden erstellen“

Szenario	Rechnungen für Laufkunden erstellen
Zugehöriger Anwendungsfall	Kundenrechnungen erstellen.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Gelieferte Aufträge für Laufkunden abrechnen,</li> <li>- mindestens ein Laufkunde besitzt einen Auftrag mit dem Status „GELIEFERT“.</li> </ul>
Ergebnis:	Rechnungen für alle gelieferten Aufträge aller Laufkunden, Umsetzung des Auftragstatus nach „ABGERECHNET“ und Eintrag des Rechnungsdatums „Heute“ für alle gerade abgerechneten Aufträge.

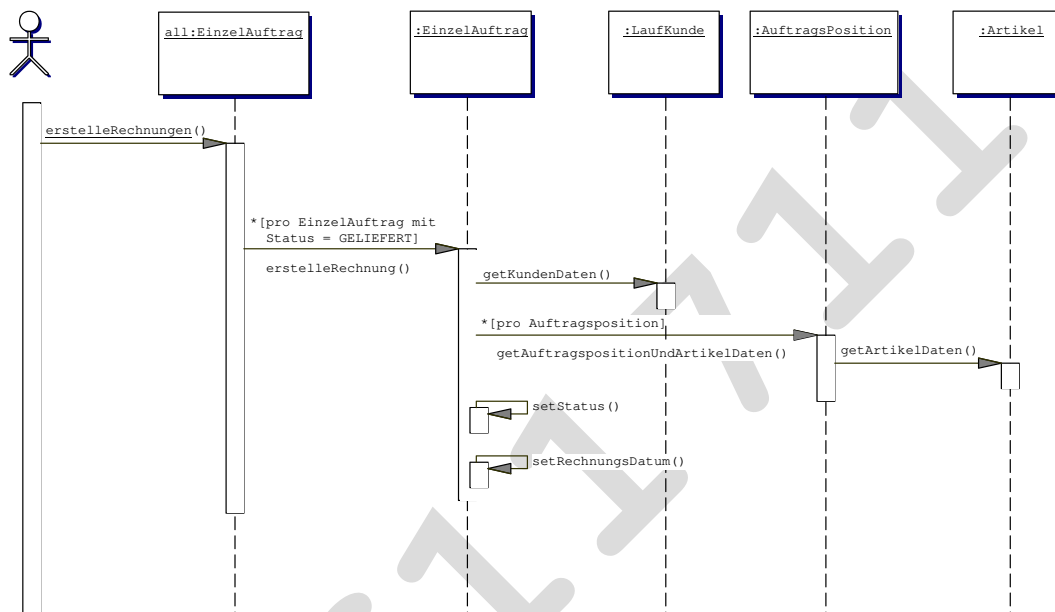


Abb. 4.34. Sequenzdiagramm zum Szenario „Rechnungen für Laufkunden erstellen“.

### Erläuterung:

- Benutzt werden die bereits bestehenden Assoziationen Auftrag → Kunde, Auftrag ↔ AuftragsPosition sowie AuftragsPosition → Artikel. Neue Verbindungen sind nicht einzurichten.
- Pseudooperationen `getxxxDaten()` lesen mehrere Attribute eines oder mehrerer Objekte; vgl. 3. Szenario.
- Die sortierte Ausgabe von Rechnungen bleibt unberücksichtigt.

## 6. Szenario „Auftragserteilung der Laufkunden auswerten“

### Auftragserteilung der Laufkunden auswerten

Szenario	Auftragserteilung der Laufkunden auswerten
Zugehöriger Anwendungsfall	Auftragsabwicklung auswerten.
Spezialisierung/ Bedingungen:	<ul style="list-style-type: none"> <li>- Auswertung der Auftragserteilung aller Laufkunden in einem gewählten Quartal,</li> <li>- mindestens ein Laufkunde besitzt im Quartal einen Auftrag mit dem Status „ABGERECHNET“.</li> </ul>
Ergebnis:	Laufkunden-Auswertung für gewähltes Quartal liegt vor.

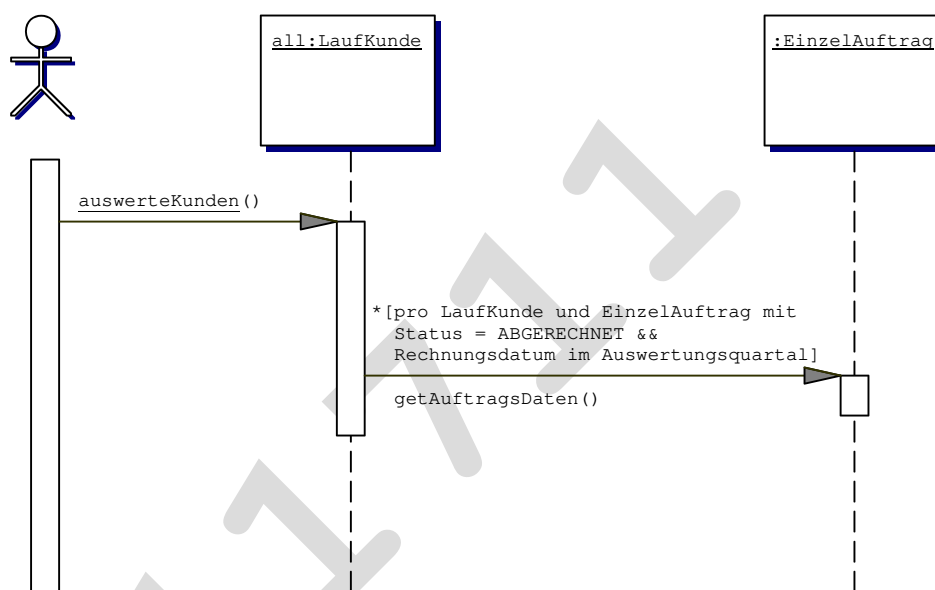


Abb. 4.35. Sequenzdiagramm zum Szenario „Auftragserteilung der Laufkunden auswerten“.

#### Erläuterung:

- Der Zugriff von Laufkunden auf ihre abgerechneten Einzelaufträge erfolgt nicht über eine Assoziation.
- Die sortierte Ausgabe nach Firmennamen bleibt unberücksichtigt.
- Die Pseudooperation `getAuftragsDaten()` stellt alle relevanten Auftragsattribute bereit; vgl. 3. Szenario.

### Zustandsdiagramm Klasse EinzelAuftrag

#### 4.4.4 Zustandsdiagramm der Klasse EinzelAuftrag

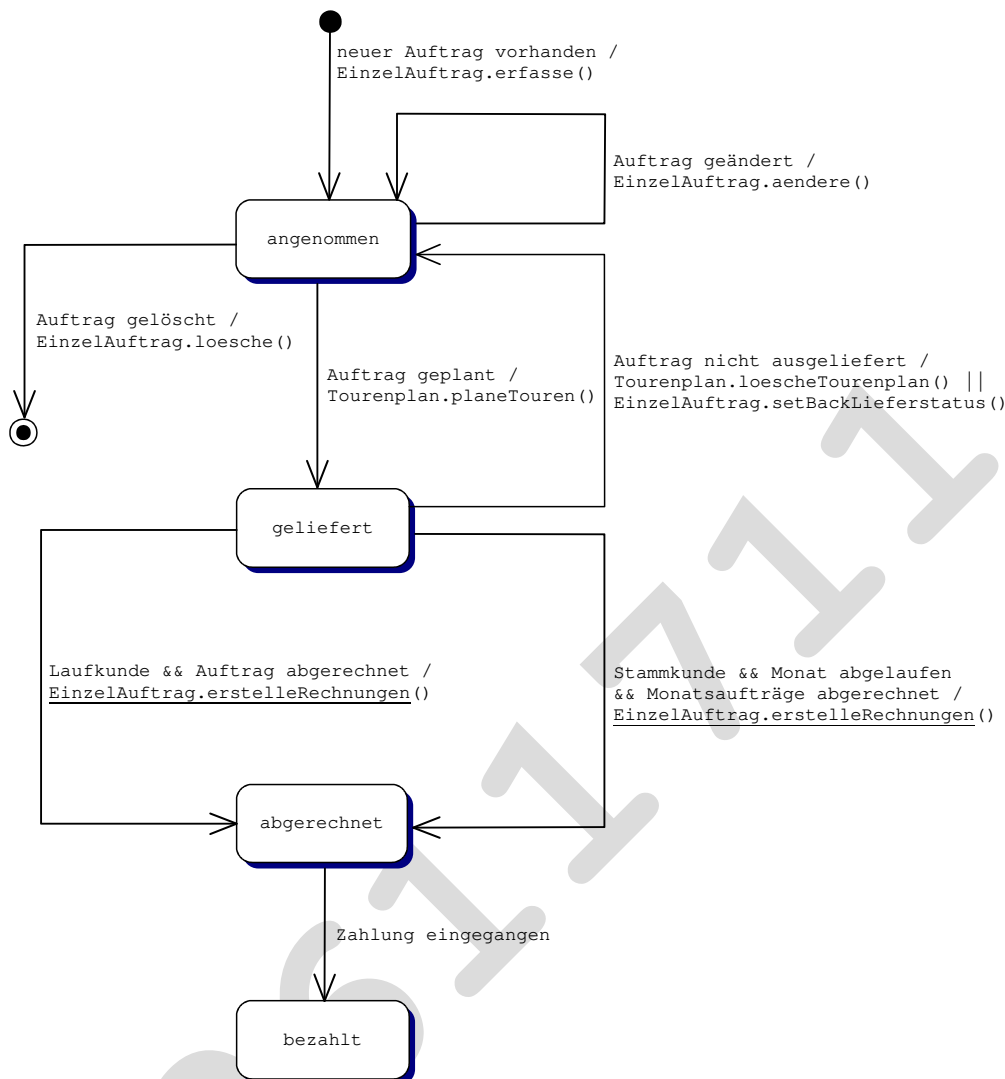


Abb. 4.36. Zustandsdiagramm der Klasse EinzelAuftrag.

##### Erläuterung:

- In Abb. 4.36 wird der Lebenszyklus eines Objekts der Klasse EinzelAuftrag dargestellt, das sich in den Zuständen „ANGENOMMEN“, „GELIEFERT“, „ABGERECHNET“ und „BEZAHLT“ befinden kann. Dabei wird der Zustand „BEZAHLT“ im Tourenplanungssystem nicht erreicht.
- Pro Zustandswechsel wird jeweils das auslösende Ereignis sowie ggf. die Aktion angegeben, die mit dem Zustandswechsel verbunden ist. Die Aktionen stellen Operationen verschiedener Klassen dar; deshalb werden die Operationsnamen durch den Klassennamen qualifiziert. Ereignisse setzen sich teils aus mehreren konjunktiv (&&) oder disjunktiv (||) verbundenen Bedingungen zusammen. Analoges gilt im Fall des Übergangs vom Zustand „GELIEFERT“ zum Zustand „ANGENOMMEN“ auch für die zugehörige Aktion, da der Übergang sowohl durch die Operation setBackLieferstatus() als auch durch die Operation loescheTourenplan() bewirkt werden kann.
- Ein Zustand „DISPONIERT“ für einen geplanten, d.h. in einen Tourenplan aufgenommen und noch nicht ausgelieferten Auftrag wurde nicht vorgesehen.



Der Grund ist, dass hierdurch ein andernfalls täglich anfallender Arbeitsschritt des Sachbearbeiters für die Auftragsabwicklung, nämlich die Eingabe einer Bestätigung der Auslieferung für alle gelieferten Aufträge, eingespart wird. Bei der vorliegenden Lösung muss der Sachbearbeiter lediglich in den relativ seltenen Fällen, dass für einen geplanten Auftrag eine Auslieferung nicht erfolgt ist, d.h. kein quittierter Lieferschein vorliegt, den Zustand des Auftrags auf „ANGENOMMEN“ zurücksetzen. Dies geschieht mittels der Operation `setBackLieferStatus()`.

## Übungsaufgaben zu Kapitel 4.4

### Übungsaufgabe 4.4.1 (Projekt Bibliotheksausleihsystem)

Erstellen Sie ein OOA-Modell des Bibliotheksausleihsystems. Das OOA-Modell soll ein statisches Modell (Klassendiagramm, Klassenlexikon) sowie ein dynamisches Modell (Szenarios mit Sequenzdiagrammen, ggf. Zustandsdiagramme) umfassen. Verwenden Sie als Grundlage die Musterlösung der Übungsaufgabe 3.6.1, d.h. das BAS-Pflichtenheft auf der CD-ROM.

## 5 Modellierung der Benutzungsoberfläche

An die fachliche Modellierung einer Anwendung schließt sich die Modellierung der Benutzungsoberfläche als letzter Schritt der objektorientierten Analyse an. Wie im Kap. 2.3 begründet, zielt die Erstellung eines Prototypen der Benutzungsoberfläche bereits am Ende der Analyse nicht nur auf eine bedarfsgerechte Oberfläche der Anwendung ab. Sie dient darüber hinaus einer Validierung des OOA-Modells gemeinsam mit dem Auftraggeber und den künftigen Benutzern.

Das vorliegende Kapitel erläutert zuerst einige Grundbegriffe grafischer Benutzungsoberflächen. Danach wird die Erstellung eines Prototyps einer Benutzungsoberfläche aus fachlich-logischer wie aus technischer Sicht behandelt. Als Basis für den theoretischen Teil des Kapitels wurden vor allem entsprechende Ausführungen in BALZERT (1996) und BALZERT (1999) verwendet. Interessierte seien auch auf HOFMANN (1998) und KRUSCHINSKI (1999) verwiesen.

### 5.1 Grundbegriffe grafischer Benutzungsoberflächen

Eine Benutzungsoberfläche, die den Bedürfnissen und Ansprüchen der Benutzer genügt und sie bei der Arbeit mit einer Dialoganwendung möglichst optimal unterstützt, besitzt heute einen hohen Stellenwert und bildet einen entscheidenden Faktor für die Akzeptanz der Anwendung. Die Software-Ergonomie als eine Teildisziplin des Software-Engineering beschäftigt sich mit der Gestaltung benutzergerechter Oberflächen und den dabei zu beachtenden Prinzipien, Kriterien und Gesetzmäßigkeiten.

**Bedeutung von Oberflächen, Software-Ergonomie**

Werden heute betriebliche Anwendungssysteme entwickelt, so werden diese so gut wie ausschließlich mit einer grafischen Benutzungsoberfläche (kurz GUI für graphical user interface) ausgestattet. Bei einer GUI erfolgt die Interaktion des Benutzers mit dem System über Fenster, die grundsätzlich an beliebiger Stelle auf dem Bildschirm erscheinen können. Als Eingabeinstrument wird überwiegend oder großenteils die Maus eingesetzt, so dass die Eingabe textueller Informationen reduziert wird. Zu manipulierende Objekte wie Dokumente und mögliche Aktionen des Benutzers werden weitgehend durch grafische Symbole repräsentiert. Kennzeichnend für grafische Benutzungsoberflächen ist daher auch die hohe Dichte der dargestellten Informationen.

**GUI**

Eine GUI besitzt eine Ein-/Ausgabekomponente und eine Dialogkomponente. Die statische Ein-/Ausgabekomponente bestimmt die Darstellung ein- und ausgegebener Daten. Die dynamische Dialogkomponente definiert die Abläufe von Interaktionen mit dem Benutzer.

**GUI-Komponenten**

Eine grafische Benutzungsoberfläche wird auf der Basis eines GUI-Systems erstellt, das auch als Fenstersystem bezeichnet wird und der Systemsoftware zugeordnet werden kann. Es bestimmt einerseits wesentlich das vom Benutzer wahrnehmbare Erscheinungsbild der GUI, indem z.B. ein spezifischer Satz von Dialogelementen (siehe unten) bereitgestellt wird.

**GUI-System**

Aus der Sicht der Entwickler ist ein GUI-System ein Framework (vgl. BOOCH 1994), das grundlegende GUI-Funktionen bereitstellt. Die bereitgestellte Grundfunktionalität wird von den Entwicklern bei der Erstellung einer konkreten GUI anwendungsspezifisch ergänzt.

**Bekannte  
GUI-Systeme**

Bekannte GUI-Systeme sind das im PC-Bereich vorherrschende Windows von Microsoft, Motif, das im UNIX-Bereich dominiert und von der Open Software Foundation vorgeschlagen wurde, sowie der Presentation Manager von IBM. Die folgenden Ausführungen beziehen sich ausschließlich auf das GUI-System Windows; Beispiele werden aus Microsoft-Anwendungen gewählt, die meist dem Office-Bereich angehören. Soll ein GUI-Prototyp im Entwurf evolutionär weiterentwickelt werden, ist bei seiner Erstellung bereits das endgültige GUI-System zu nutzen.

**Gestaltungs-  
regelwerke**

Die Hersteller von GUI-Systemen schlagen zusätzlich Gestaltungsregelwerke für die Entwicklung von GUI mit dem betreffenden GUI-System vor, das auf die einheitliche Gestaltung von Fenstern, Menüs und Dialogabläufen abzielt. Das Gestaltungsregelwerk von Windows sieht beispielsweise vor, dass im Hauptmenü einer Anwendung Menüpunkte wie Datei und Bearbeiten angeboten werden, über die jeweils gewisse Funktionen – unter dem Menüpunkt Datei z.B. das Öffnen von Dokumenten – erreichbar sind (vgl. MS 1995).

Als wesentliche Elemente von grafischen Benutzungsoberflächen seien Dialoge, Fenster, Dialogelemente und Menüs etwas näher betrachtet.

**(1) Dialoge**

**Dialog**

In einem Dialog werden zwischen einem Benutzer und einer Anwendung, die in diesem Fall auch Dialogsystem genannt wird, wechselseitig Informationen ausgetauscht, um ein bestimmtes Ziel zu erreichen.

**Primär- und  
Sekundärdialoge**

Aus inhaltlich-fachlicher Sicht lassen sich Primär- und Sekundärdialoge unterscheiden. Primärdialoge dienen der Abwicklung der hauptsächlichen Aktivitäten des Benutzers. Mit einem Primärdialog führt ein Benutzer unmittelbar einen Arbeitsschritt aus, der zu seiner eigentlichen Arbeitsaufgabe gehört. Der Primärdialog endet daher erst, wenn das gewünschte Ergebnis des Arbeitsschrittes erreicht wurde. Er korrespondiert je nach Granularität mit einem Szenario oder einem Arbeitsschritt eines Szenarios. Mit Sekundärdialogen werden zusätzliche Aktivitäten des Benutzers durchgeführt, die gelegentlich auftreten oder eine Hilfsrolle spielen und meist von kurzer Dauer sind.

**Beispiel 5.1**

Ein Kunde erscheint bei einem Autoverleih und möchte einen PKW buchen. Die Buchung wird von einem Mitarbeiter des Autoverleihs in einem Primärdialog erfasst. Vor der Auswahl des Zubehörs möchte der Kunde erst eine Liste möglicher Zubehöerteile sehen. Der Mitarbeiter unterbricht den Primärdialog, um in einem Sekundärdialog eine Liste angebotener Zubehörkomponenten anzuzeigen. Danach wird der Primärdialog fortgesetzt.

Grundsätzliche Anforderungen und Kriterien für die Dialoggestaltung wurden im Teil 10 der ISO-Norm 9241-10 (1996) fixiert. Sie werden in der Tab. 5.1 sinngemäß wiedergegeben und um einfache Beispiele ergänzt.

Grundsatz	Erläuterung	Beispiel
Aufgabenangemessenheit	Dialog unterstützt den Benutzer bei effektiver und effizienter Erledigung seiner Arbeitsaufgabe.	Zuletzt benutzte Dokumente werden im Datei-Menü von Office-Anwendungen angezeigt und müssen nicht erst gesucht werden.
Selbstbeschreibungsfähigkeit	Jeder Dialogschritt ist durch Rückmeldung des Dialogsystems unmittelbar verständlich oder wird auf Anfrage erklärt.	In der Statusleiste werden ausgelöste oder mögliche Aktionen angezeigt.
Steuerbarkeit	Benutzer kann den Dialogablauf starten und seine Richtung und Geschwindigkeit beeinflussen.	Eine Schaltfläche, z.B. „OK“, ist voreingestellt, der Benutzer kann jedoch auch die Schaltfläche „Abbrechen“ wählen.
Erwartungskonformität	Dialog ist in sich konsistent, entspricht den allgemeinen Konventionen und den Kenntnissen und Erfahrungen des Benutzers.	Einheitliche Menügestaltung bei Office-Anwendungen, z.B. enthält das Datei-Menü stets Funktionen zum Öffnen, Speichern, Drucken und Beenden der Anwendung.
Fehlertoleranz	Beabsichtigte Arbeitsergebnisse können trotz fehlerhafter Eingaben mit minimalem Korrekturaufwand des Benutzers erreicht werden.	Nach fehlerhafter Eingabe springt der Cursor unmittelbar zu einer Position, wo sich ein fehlerhaftes Datum befindet.
Individualisierbarkeit	Dialog lässt Anpassungen an die Arbeitsaufgabe sowie individuelle Bedürfnisse des Benutzers zu.	Individuelle Gestaltung von Menüleisten und Symbolleisten ist möglich.
Lernförderlichkeit	Dialog unterstützt Benutzer beim Erlernen des Dialogsystems.	Undo-Funktionen gestatten unbeabsichtigte bzw. ungewollte Aktionen rückgängig zu machen und unterstützen daher das Experimentieren.

### Anforderungen an Dialoge

**Tab. 5.1.** Grundsätze der Dialoggestaltung nach ISO 9241-10.

Aus der Perspektive der technischen Realisierung können Dialoge in verschiedener Weise differenziert werden.

So sind zum einen modale und nicht-modale Dialoge zu unterscheiden. Während ein modaler Dialog geführt wird und ein zugehöriges Fenster geöffnet ist, kann der Benutzer kein anderes geöffnetes Fenster bearbeiten, also z.B. dort keine Daten eingeben. Ein modaler Dialog muss erst abgeschlossen und sein Fenster geschlossen werden, bevor ein anderer Dialog fortgesetzt werden kann. Modale Dialoge schließen also ein abwechselndes Arbeiten in verschiedenen gleichzeitig geöffneten Fenstern aus. Jedoch kann von einem modalen Dialog aus durchaus ein weiterer Dialog gestartet bzw. ein zusätzliches Fenster geöffnet werden.

Nicht-modale Dialoge gestatten dem Benutzer, in mehreren gleichzeitig geöffneten Fenstern abwechselnd Bearbeitungsschritte durchzuführen.

### modale und nicht-modale Dialoge

**Beispiel 5.2**

Ein typisches Beispiel für einen modalen Dialog ist der Dialog Datei/Drucken, der in Office-Anwendungen unmittelbar vor dem Ausdruck eines Dokuments zu führen ist. Der Dialog Bearbeiten/Suchen ist in MS Word ein nicht-modaler Dialog. Nachdem z.B. ein bestimmtes Suchwort eingegeben und im Dokument gefunden wurde, kann die Fundstelle bearbeitet werden. Währenddessen bleibt das Fenster des Suche-Dialogs offen und das Suchwort kann nach der Bearbeitung der Fundstelle modifiziert werden, bevor eine neue Suche im Dokument gestartet wird. In MS Excel ist dagegen der Dialog Bearbeiten/Suchen als modaler Dialog ausgelegt. Bei geöffnetem Suchen-Fenster kann eine Tabelle nicht bearbeitet werden.

**Bewertung**

Nicht-modale Dialoge erlauben eine größere Flexibilität der Dialoggestaltung und sind daher gegenüber modalen Dialogen grundsätzlich zu bevorzugen. Doch können modale Dialoge auch eine Lenkungsfunction übernehmen und den Benutzer darin unterstützen, unzweckmäßige Abfolgen von Arbeitsschritten zu vermeiden. Wegen der erwähnten Möglichkeit, innerhalb eines modalen Dialogs einen weiteren aufzurufen, erscheint auch die eingeschränkte Flexibilität modaler Dialoge in vielen Fällen ausreichend. Ferner sind nicht-modale Dialoge wegen der erforderlichen Synchronisierung von Anwendungsdaten programmiertechnisch schwieriger zu realisieren. Ob und für welche Zwecke nicht-modale Dialoge benötigt werden, sollte jeweils konkret geprüft werden.

**Bedienungsvarianten von Dialogen**

Realisierungstechnisch können ferner drei Bedienungsvarianten von Dialogen unterschieden werden. Die funktionsorientierte und objektorientierte Bedienung sowie die Bedienung mittels direkter Manipulation werden in Tab. 5.2 erläutert.

Dialogbedienungs-variante	Erläuterung
funktionsorientiert	<ul style="list-style-type: none"> <li>- Der Benutzer wählt zunächst eine Funktion aus. Anschließend wählt er das Objekt aus, für das die Funktion ausgeführt werden soll.</li> <li>- Beispiel: In Office-Anwendungen kann zunächst die Funktion „Drucken“, dann das auszudruckende Dokument bestimmt werden.</li> </ul>
objektorientiert	<ul style="list-style-type: none"> <li>- Der Benutzer wählt zunächst das Objekt aus und bestimmt anschließend die auf das Objekt anzuwendende Funktion.</li> <li>- Beispiel: Im Explorer kann zuerst eine Datei markiert und anschließend über ein Kontextmenü bestimmt werden, ob die Datei z.B. kopiert oder umbenannt werden soll.</li> </ul>
direkte Manipulation	<ul style="list-style-type: none"> <li>- Die Bearbeitung von Objekten wird durch reine Mausmanipulationen ausgelöst. Zusätzlich werden Objekte oft grafisch-symbolisch repräsentiert.</li> <li>- Beispiel 1: Eine Datei wird per Doppel-Klick geöffnet.</li> <li>- Beispiel 2: Ein Dateisymbol wird mittels Maus selektiert, über ein Papierkorb-Symbol gezogen und dort losgelassen. Durch diese – drag &amp; drop genannte – Technik wird im gegebenen Fall die Datei gelöscht bzw. aus ihrem bisherigen Ordner entfernt.</li> </ul>

**Tab. 5.2.** Varianten der Dialogbedienung.

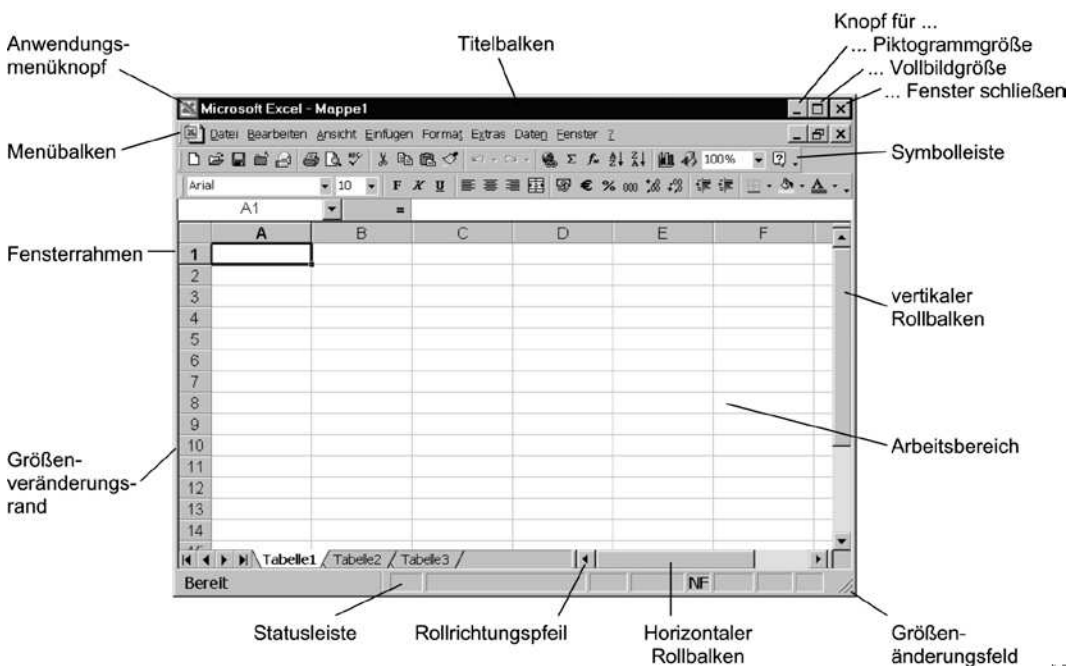
Wie die Beispiele in Tab. 5.2 bereits vermuten lassen, kommen die verschiedenen Varianten der Dialogbedienung auch kombiniert zum Einsatz. Eine objektorien-

tierte Anwendung verlangt nicht notwendig eine objektorientierte Dialogbedienung im oben definierten Sinne. Die Dialogbedienungsvariante muss jeweils anhand der besonderen Merkmale einer Anwendung und entsprechend den konkreten Bedürfnissen der Benutzer festgelegt werden.

## (2) Fenster

Ein Fenster ist ein rechteckiger Bildschirmausschnitt, in dem Daten ein- und ausgegeben und Funktionen ausgelöst werden. Die Abb. 5.1 zeigt ein Fenster einer Windows-Anwendung mit seinen Elementen, auf deren nähere Erläuterung hier verzichtet werden kann.

### Fenster und ihre Elemente



**Abb. 5.1.** Fenster einer Windows-Anwendung mit Fensterelementen (hier: Microsoft Excel 2000).

Im Zusammenhang mit Fenstertypen ist der Unterschied zwischen SDI- und MDI-Anwendungen von Bedeutung. SDI steht für single document interface, MDI für multiple document interface. Ein Dokument beinhaltet Anwendungsdaten und kann eine Datei oder eine Datenbank repräsentieren. Die Daten einer Anwendung können auf mehrere Dokumente verteilt sein. In einer SDI-Anwendung kann zu einem Zeitpunkt stets nur ein Dokument geöffnet sein, dargestellt und verarbeitet werden. In einer MDI-Anwendung können mehrere Dokumente zugleich geöffnet und in der GUI dargestellt sowie alternierend verarbeitet werden.

### SDI- und MDI-Anwendungen

#### Beispiel 5.3

Das Windows-Zubehörprogramm Editor bzw. Notepad ist eine SDI-Anwendung. Bevor eine weitere Datei geöffnet und bearbeitet werden kann, muss die zuvor geöffnete geschlossen werden. MS Excel ist eine MDI-Anwendung. Mehrere Tabellen, die verschiedenen Dateien entstammen können, lassen sich gleichzeitig in verschiedenen Fenstern darstellen und abwechselnd bearbeiten.

In der Tab. 5.3 werden verschiedene Fenstertypen erläutert, wobei die Begriffe des Primär- und des Sekundärfensters eine inhaltliche Differenzierung erlauben.



**Fenstertypen**

<b>Fenstertyp</b>	<b>Erläuterung</b>
Primärfenster	- Dient der Abwicklung der Primärdialoge, d.h. der hauptsächlichen Aktivitäten des Benutzers.
Sekundärfenster	- Dient der Abwicklung von Sekundärdialogen, d.h. der zusätzlichen Aktivitäten des Benutzers.
Anwendungsfenster	- Wichtigstes Primärfenster einer Anwendung, das stets nach ihrem Start erscheint. - Enthält immer Titelbalken, Menübalken und Arbeitsbereich und optional weitere Elemente wie Symbolleiste oder Statusleiste. - Bei SDI-Anwendungen wird der Arbeitsbereich des Anwendungsfensters für Dialoge bevorzugt genutzt. Beispiel: Editor. - Bei MDI-Anwendungen bleibt der Arbeitsbereich leer.
Unterfenster	- Primärfenster für MDI-Anwendungen. - Unterfenster können alle Fensterelemente besitzen. - Vom Anwendungsfenster aus können mehrere Unterfenster geöffnet werden, die nebeneinander oder überlappend erscheinen. - Bei mehreren geöffneten Fenstern wählt der Benutzer jeweils ein aktives Fenster für die Fortsetzung des Dialogs aus. - Verwendung meist für nicht-modale Dialoge. - Beispiel: MS Excel, pro Unterfenster kann eine Tabelle bearbeitet werden.
Dialogfenster, Dialogbox	- Sekundärfenster und Primärfenster für MDI- und SDI-Anwendungen. - Einsatz vor allem für Dateneingaben. - Dialogfenster besitzen keine Menübalken und keine veränderliche Größe. - Dialogfenster können vom Anwendungsfenster oder von Unterfenstern aus geöffnet werden. - Verwendung für modale und nicht-modale Dialoge. - Beispiel: MS Excel, mit Menüpunkt Datei/Drucken wird Dialogfenster „Drucken“ für Sekundärdialog geöffnet.
Mitteilungsfenster	- Sekundärfenster für MDI- und SDI-Anwendungen. - Nach Ausgabe einer Mitteilung kann Benutzer eine Aktion ausführen, aber keine Daten manipulieren. - Mitteilungsfenster werden von der Anwendung selbst, häufig in Ausnahmesituationen geöffnet. - Beispiel: MS Excel, bei Aktivierung der Seitenansicht für eine Tabelle ohne Daten erscheint eine entsprechende Meldung, die der Benutzer bestätigen muss.

**Tab. 5.3.** Fenstertypen.**Gestaltung von Fenstern**

Im Folgenden werden einige Hinweise und Regeln zur Gestaltung von Fenstern aufgeführt. Dabei wird teils bereits Bezug auf Dialogelemente genommen, auf die anschließend näher eingegangen wird.

**Titel**

- Für den Titelbalken eines Fensters ist eine aussagekräftige Bezeichnung zu wählen, der Zweck und Inhalt des Fensters angibt.

**Aufbau**

- Der Aufbau eines Fensters und die Anordnung von Dialogelementen sollten dem Arbeitsablauf bei der Benutzung entsprechen. Die zuerst benötigten bzw. einzugebenden Informationen sollten oben links im Fenster erscheinen. Der Benutzer sollte beim Betrachten des Fensters möglichst nicht hin und her springen müssen.

**Gruppierung**



- Werden in einem Fenster Daten in größerer Anzahl dargestellt, so erleichtert eine Gruppierung der Daten dem Benutzer die Orientierung. Jede Datengruppe darf natürlich nur inhaltlich zusammengehörende Daten umfassen. Die verschiedenen Gruppen sollten durch räumliche Abstände, Umrahmungen und evtl. Gruppenbezeichnungen optisch getrennt werden. Der Überblick des Benutzers wird gewahrt, wenn nicht mehr als ca. fünf Gruppen vorhanden sind. Einzelne Daten einer Gruppe können sofort identifiziert werden, wenn eine Gruppe nicht mehr als ca. fünf Elemente umfasst. Diese Grenzwerte lassen sich allerdings nicht immer einhalten.
- Zusätzlich können einzelne vorrangig zu beachtende Daten oder Datengruppen optisch hervorgehoben werden. Mögliche Mittel zur Hervorhebung sind u.a. besondere Schriftarten, Größe, Farben und grafische Symbole. Nur ein geringer Teil der Daten sollte hervorgehoben werden. Insbesondere sind Farben nur sparsam einzusetzen. **Hervorhebung**
- Im Interesse einer harmonischen Fenstergestaltung sollten einige weitere Aspekte berücksichtigt werden. Die Breite eines Fensters sollte nicht kleiner als seine Höhe sein. Anzustreben ist eine ausgewogene Verteilung der Fensterelemente im gesamten Fenster. Die Verlängerungen der sichtbaren Kanten von Dialogelementen bilden virtuelle Linien, die als störend empfunden und daher minimiert werden sollten. Dies kann durch eine Vergrößerung der Breite und Höhe von Dialogelementen oder ihre horizontal benachbarte Anordnung erreicht werden (vgl. Kap. 5.2.2). Zu einer Minimierung virtueller Linien trägt auch eine axialsymmetrische Anordnung von Dialogelementen bei. Empfehlenswert ist insbesondere eine Symmetrie bezüglich der (gedachten) vertikalen Mittellinie des Fensters. **harmonische Gestaltung**
- Gestaltungsmittel wie z.B. Farben oder Schriftarten sollten über verschiedene Fenster hinweg konsistent, d.h. mit gleicher Bedeutung verwendet werden, um die Erwartungskonformität der GUI zu wahren. Wird etwa die Farbe rot in einem Mitteilungsfenster benutzt, um eine Gefahr zu signalisieren, sollte dies in anderen Fenstern ebenso geschehen. **konsistente Gestaltung**


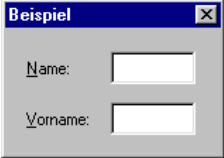
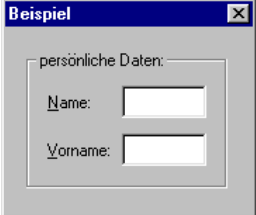

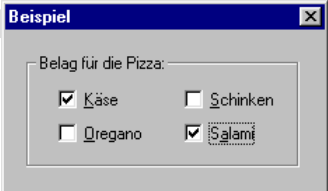

### (3) Dialogelemente

Dialogelemente, auch Interaktionselemente genannt, dienen der Ein- und Ausgabe von Daten innerhalb von Fenstern. Verschiedene GUI-Systeme bieten jeweils spezifische Typen von Dialogelementen an. In der Tab. 5.4 bzw. Tab. 5.5 werden die wichtigsten Windows-Dialogelemente vorgestellt. Teils werden gebräuchliche englische Bezeichnungen zusätzlich angegeben.


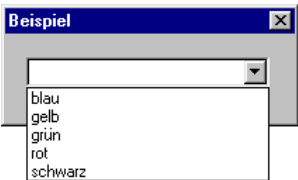
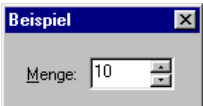

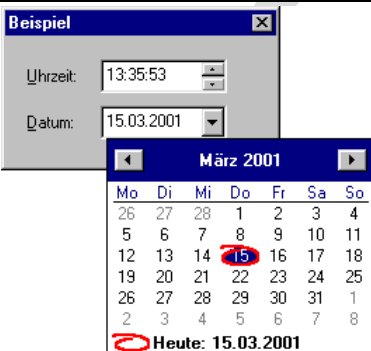
Dialogelemente werden in Dialogfenstern sowie auch in MDI-Unterfenstern benutzt. Es sei aber darauf hingewiesen, dass die Datenein- und -ausgabe in Anwendungsfenstern von SDI-Anwendungen und in MDI-Unterfenstern ohne oder überwiegend ohne Dialogelemente abgewickelt wird. Dies ist beispielsweise bei MS Excel der Fall.

**Dialogelemente,  
Begriff und  
Benutzung**

**Typen von  
Dialogelementen**

Dialogelement	Erläuterung
<b>Eingabefeld / Editfeld</b>	
	<ul style="list-style-type: none"> <li>- In einem Eingabefeld kann der Benutzer beliebigen einzeiligen Text eingeben.</li> <li>- Möglich ist auch eine Spezialisierung auf numerische Eingaben.</li> <li>- Neben dem einzeiligen Eingabefeld gibt es auch das mehrzeilige Eingabefeld zur Ein- und Ausgabe von etwas umfangreicheren Texten.</li> </ul>
<b>Statisches Textfeld / Static-Text</b>	
	<ul style="list-style-type: none"> <li>- Der Inhalt eines statischen Textfeldes ist nicht veränderlich.</li> <li>- Es wird – wie hier „Name“ und „Vorname“ – für die Beschriftung anderer Dialogelemente benutzt.</li> </ul>
<b>Gruppenfeld</b>	
	<ul style="list-style-type: none"> <li>- Ein Gruppenfeld ist ein statisches Element, das aus einem Rahmen und einer Beschriftung besteht.</li> <li>- Es ermöglicht eine optische Gruppierung mehrerer fachlich zusammenhängender Dialogelemente. Der Zusammenhang kann durch eine Beschriftung (hier: „persönliche Daten“) explizit ausgedrückt werden.</li> </ul>
<b>Schaltfläche / Button</b>	
	<ul style="list-style-type: none"> <li>- Mit einem Druck auf eine Schaltfläche wird eine Aktion ausgelöst oder bestätigt.</li> <li>- Typische Schaltflächen sind der OK-Button und der Abbrechen-Button.</li> </ul>
<b>Kontrollkästchen / Checkbox</b>	
	<ul style="list-style-type: none"> <li>- Mit Hilfe einer Checkbox können Optionen aktiviert bzw. deaktiviert werden.</li> <li>- In einer Checkbox können mehrere Optionen gleichzeitig gewählt werden.</li> </ul>
<b>Optionsfeld / Radio-Button</b>	
	<ul style="list-style-type: none"> <li>- Mit einem Radio-Button kann genau eine von mehreren möglichen Optionen ausgewählt werden.</li> <li>- Sobald eine andere Option gewählt wurde, wird die bisher eingestellte Option deaktiviert.</li> </ul>

Tab. 5.4. Dialogelemente.

Dialogelement	Erläuterung
<b>Listenfeld / Auswahlfeld / List-Box</b>	
	<ul style="list-style-type: none"> <li>- In einem Listenfeld können beliebig viele Listeneinträge vertikal dargestellt werden.</li> <li>- Sind mehr Listeneinträge vorhanden, als gleichzeitig dargestellt werden können, kann innerhalb des Listenfeldes mittels vertikalem Rollbalken geblättert (gescrollt) werden.</li> <li>- Es können entweder einer oder mehrere Einträge zugleich ausgewählt werden. Gewählte Einträge werden farblich unterlegt angezeigt.</li> </ul>
<b>Kombinationsfeld / Combo-Box</b>	
	<ul style="list-style-type: none"> <li>- Ein Kombinationsfeld umfasst ein Eingabefeld und ein Listenfeld.</li> <li>- Ein Eintrag kann je nach Variante entweder aus der Liste nur ausgewählt oder auch editiert werden.</li> <li>- Bei Dropdown-Kombinationsfeldern ist nur das Eingabefeld bzw. der aktuelle Eintrag ständig sichtbar.</li> </ul>
<b>Drehfeld / Spin-Button</b>	
	<ul style="list-style-type: none"> <li>- Der Spin-Button wird in Verbindung mit einem Eingabefeld benutzt, wenn Daten aus einem festen, geordneten Wertebereich, z.B. aus einem Intervall ganzer Zahlen, einzugeben sind.</li> <li>- Zur Ermittlung des gewünschten Wertes kann der Wertebereich durch Anklicken der Pfeile oder mittels der Pfeiltasten der Tastatur durchlaufen werden.</li> <li>- Alternativ kann der Wert direkt eingegeben werden.</li> </ul>
<b>Listenelement / Tabelle / List-View</b>	
	<ul style="list-style-type: none"> <li>- Das Listenelement wird für eine Auflistung von Tabellen mit Einträgen aus mehreren Elementen sowie die Auswahl von Einträgen genutzt.</li> <li>- Ähnlich wie im Explorer können Einträge elementweise in einer Zeile wie auch durch große und kleine Symbole angezeigt werden.</li> </ul>
<b>Datums-&amp;-Zeitauswahl / Date-Time-Picker</b>	
	<ul style="list-style-type: none"> <li>- Dieses Dialogelement ermöglicht dem Benutzer eine bequeme Eingabe einer Uhrzeit oder eines Datums. Ungültige Eingaben werden ausgeschlossen.</li> <li>- Bei der Eingabe eines Datums kann der Benutzer ein Kalenderblatt aufklappen, in dem er Monat und Jahr einfach wechseln und einen Tag durch Anklicken auswählen kann.</li> </ul>

Tab. 5.5. Dialogelemente (Fortsetzung).

Weitere Dialogelemente sind teils wie die Datums-&-Zeitauswahl für spezielle Aufgaben konzipiert. Hierzu gehören ein Element für die Eingabe einer IP-Adresse oder die Statusanzeige, die den Fortschritt einer Verarbeitung anzeigt.

weitere Dialogelemente

Regler dienen zur Anzeige und Einstellung von Werten einer Größe, wie z.B. einer Geschwindigkeit, ohne hohen Genauigkeitsanspruch.

Mittels Strukturansichten kann eine hierarchische, baumartige Struktur von Einträgen (Knoten) visualisiert werden, in der der Benutzer navigieren kann. Nachgeordnete Ebenen der Hierarchie können ein- und ausgeblendet werden. Einträge werden durch einen Text und zusätzlich evtl. durch ein Symbol dargestellt.

Register umfassen mehrere gleichgroße (Kartei-)Karten. Jede Karte kann verschiedene Dialogelemente für die Ein- und Ausgabe von Daten beinhalten. Es wird stets nur eine Karte angezeigt, wobei mittels Reitern zwischen den Karten gewechselt werden kann.

#### Anwendungshinweise für Dialogelemente

In der Tab. 5.6 finden sich einige Hinweise zur Anwendung der vorgestellten Dialogelemente.

Dialogelement	Anwendungshinweise
Eingabefeld	<ul style="list-style-type: none"> <li>- Breite groß genug wählen, so dass (einzeiliger) Text bzw. Zahl vollständig sichtbar ist.</li> <li>- Zahlen rechtsbündig, Texte linksbündig anordnen.</li> <li>- Bei rein-numerischen Eingabedaten Eingabefeld derart qualifizieren, dass nichtnumerische Eingaben abgewiesen werden (vgl. Kap. 5.2.3).</li> <li>- Reine Ausgabefelder für Eingaben sperren.</li> </ul>
Statisches Textfeld	<ul style="list-style-type: none"> <li>- Statisches Textfeld direkt vor oder über dem zu beschreibenden Dialogelement platzieren.</li> <li>- Selbsterklärende Bezeichnungen mit nur einem Wort wählen.</li> <li>- Nur übliche Abkürzungen (z.B. PLZ) benutzen.</li> </ul>
Gruppenfeld	<ul style="list-style-type: none"> <li>- Gruppenfelder verwenden, falls mehrere Gruppen inhaltlich zusammenhängender Dialogelemente vorhanden sind.</li> <li>- Ein Gruppenfeld sollte möglichst nicht mehr als etwa 5 Elemente beinhalten.</li> </ul>
Schaltfläche	<ul style="list-style-type: none"> <li>- Schaltflächen stets mit aussagekräftigem Bezeichner (ein Wort) oder Symbol versehen.</li> </ul>
Kontrollkästchen/ Optionsfeld	<ul style="list-style-type: none"> <li>- Nur einsetzen, wenn eine Auswahl aus einer begrenzten Anzahl von max. 8 feststehenden Alternativen zu treffen ist.</li> </ul>
Listenfeld	<ul style="list-style-type: none"> <li>- Zur Anzeige von und Selektion in Listen mit größerer Anzahl von Einträgen benutzen.</li> <li>- Bedingung ist, dass der Benutzer nur unter vorhandenen Einträgen selektieren muss.</li> </ul>
Kombinationsfeld	<ul style="list-style-type: none"> <li>- Siehe Listenfeld.</li> <li>- Im Vergleich zu einem Listenfeld kann Platz gespart werden.</li> <li>- Ferner können Einträge modifiziert werden.</li> </ul>
Listenelement	<ul style="list-style-type: none"> <li>- Zur Anzeige und Selektion von Objekten mit mehreren Attributen benutzen.</li> <li>- Nur die wichtigsten Attribute, darunter Schlüsselattribute, auswählen.</li> </ul>
Registerkarten	<ul style="list-style-type: none"> <li>- Registerkarten können verwendet werden, wenn eine größere Datenmenge nicht in ein Fenster passt, aber in mehrere, separat benutzbare Datengruppen zerlegt werden kann.</li> <li>- Beispiele finden sich in der Windows-Systemsteuerung.</li> </ul>

**Tab. 5.6.** Einige Hinweise zur Anwendung von Dialogelementen.

#### (4) Menüs

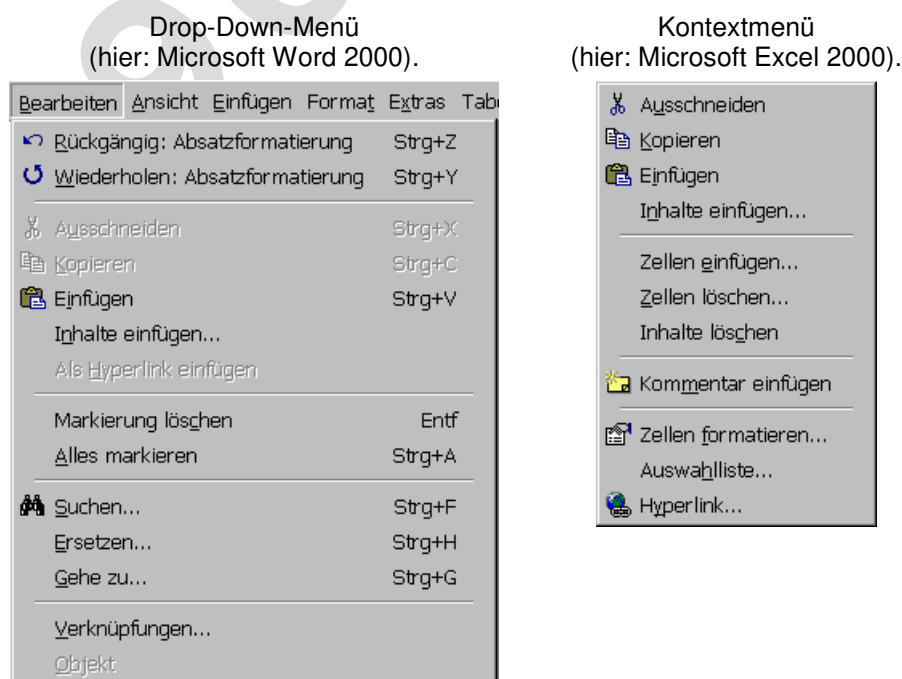
Ein Menü umfasst eine Menge von meist fest vorgegebenen Optionen, aus denen der Benutzer eine oder auch mehrere auswählen kann. In der Tab. 5.7 werden verschiedene Menütypen erläutert, während die folgende Abb. 5.2 je ein Beispiel für ein Drop-Down-Menü und ein Kontextmenü zeigt.

Menü

Menütypen

Menütyp	Erläuterung
Aktionsmenü	<ul style="list-style-type: none"> <li>- Menüoptionen lösen unmittelbar Funktionen der Anwendung aus oder führen zur Anzeige eines nachgeordneten Menüs (Untermenü).</li> <li>- Beispiel: Menü Datei in Office-Anwendungen.</li> </ul>
Eigenschaftsmenü	<ul style="list-style-type: none"> <li>- Menüoptionen gestatten die Einstellung von Verhaltensmerkmalen der Anwendung.</li> <li>- Oft können mehrere Optionen gleichzeitig gewählt werden.</li> <li>- Beispiel: Menü Extras in Office-Anwendungen.</li> </ul>
Hauptmenü, Menübalken	<ul style="list-style-type: none"> <li>- Wichtigstes Menü einer Anwendung, ständig sichtbar.</li> <li>- Bei SDI-Anwendungen existiert ein unveränderliches Hauptmenü, das zum Anwendungsfenster gehört.</li> <li>- Bei MDI-Anwendungen kann das Hauptmenü des Anwendungsfensters durch Menübalken von Unterfenstern überlagert werden und daher variieren.</li> </ul>
Drop-Down-Menü	<ul style="list-style-type: none"> <li>- Ein Drop-Down-Menü ist von einer Option des Hauptmenüs aus erreichbar.</li> <li>- Aktuell nicht wählbare Menüoptionen sind grau dargestellt.</li> <li>- Untermenüs können vorhanden sein.</li> </ul>
Kontextmenü, Pop-Up-Menü	<ul style="list-style-type: none"> <li>- Aktivierung erfolgt durch Anklicken des Objekts mittels rechter Maustaste.</li> <li>- Menüoptionen beziehen sich stets auf das Objekt, auf das der Mauszeiger aktuell verweist.</li> <li>- Untermenüs können vorhanden sein.</li> </ul>

Tab. 5.7. Menütypen.



Menü-Beispiele

Abb. 5.2. Drop-Down-Menü und Kontextmenü.

### rascher Zugriff auf Menüoptionen

Bemerkt sei, dass die Begriffe Aktions- und Eigenschaftsmenü eine Differenzierung nach dem Menüzweck gestatten, während die anderen in Tab. 5.7 vorgestellten Menüvarianten auf eine eher technische Unterscheidung abzielen.

Zur schnelleren Wahl von Anwendungsfunktionen oder anderer in Menüs angebotener Optionen existieren verschiedene Möglichkeiten, die von Office-Anwendungen her bekannt sein sollten. Genannt seien Hot-Keys, d.h. mnemonische Tastaturkürzel für Optionen eines Menüs, menüunabhängige Short-Cuts, d.h. Tastenkombinationen mit mindestens einer Steuertaste, sowie die Symbolleiste, auch Toolbar genannt, die oft unterhalb des Hauptmenüs erscheint.

### Gestaltung von Menüs

Auch zur Menügestaltung lassen sich zahlreiche Regeln und Hinweise angeben, von denen einige nachfolgend angeführt seien:

- Menüoptionen sollten klar, kurz und ggf. einheitlich über mehrere Menüs hinweg benannt werden. Abkürzungen sind zu vermeiden, ggf. können Vokale oder letzte Zeichen gestrichen werden.
- Aktuell nicht wählbare Optionen sind stets zu deaktivieren (Graudarstellung in Drop-Down-Menüs) bzw. nicht anzuzeigen (Kontextmenüs).
- Menüoptionen sollten in geeigneter Reihenfolge, z.B. alphabetisch sortiert, aufgelistet werden. Die Übersichtlichkeit kann durch eine Gliederung in Gruppen verwandter Optionen mittels horizontaler Linien verbessert werden.
- Menühierarchien sollten eher breit und flach gestaltet werden und maximal drei Stufen besitzen.
- Kontextmenüs dürfen ein Objekt nicht verdecken und sollten direkt rechts vom Objekt erscheinen.

### objektorientierte Menügestaltung

Für betriebliche Anwendungssysteme kommt neben einer funktionsorientierten auch eine objektorientierte Menügestaltung in Betracht. Bei dieser erscheinen als Optionen des Hauptmenüs sowie evtl. auch von oberem Drop-Down-Menüs die zu bearbeitenden Objekte der Anwendung. Bearbeitungsfunktionen für die Objekte werden dagegen in Drop-Down-Menüs angeboten, die entweder direkt vom Hauptmenü abzweigen oder zusätzliche Untermenüs darstellen. Die objektorientierte Menügestaltung mit der vorrangigen Gliederung nach Bearbeitungsobjekten kann bei betrieblichen Anwendungen zu einer übersichtlichen GUI erheblich beitragen, weil Dialoge in diesem Fall oft hauptsächlich durch die bearbeiteten Objekte definiert sind. Eine objektorientierte Menügestaltung wird im Kap. 5.2.2 anhand des Projekts TPS demonstriert.

## Übungsaufgaben zu Kapitel 5.1

### Übungsaufgabe 5.1.1

Entscheiden Sie jeweils, ob eine SDI-Anwendung oder eine MDI-Anwendung vorliegt:

- a) MS Word,
- b) MS WordPad,
- c) MS PowerPoint.

### Übungsaufgabe 5.1.2

Geben Sie für die folgenden Daten je ein geeignetes Dialogelement an:

- a) Farben einer Ampel,
- b) Name eines Kunden,
- c) Käufer eines Artikels,
- d) Käufer und Lagerorte eines Artikels,
- e) Solltemperaturwerte in einem Lager (13.0°C bis 21.5°C),
- f) fixe Liste von Länderbezeichnungen,
- g) Artikelkurzbeschreibung.

## 5.2 Erstellung eines Prototyps der Benutzungsoberfläche

Im Folgenden wird zunächst eine Methode umrissen, mit der die Benutzungsoberfläche einer Anwendung aus ihrem Klassendiagramm abgeleitet werden kann. Danach werden konzeptionelle Aspekte des GUI-Prototyps des Tourenplanungssystems besprochen. Schließlich wird beschrieben, wie mit der Entwicklungsumgebung MS Visual C++ ein GUI-Prototyp programmiert werden kann.

### 5.2.1 Ableitung der Dialogstruktur aus dem Klassendiagramm

BALZERT (1996) schlägt eine Methode zur systematischen Transformation des Klassendiagramms einer Anwendung in ihre Benutzungsoberfläche vor, vgl. auch BALZERT (1999). Festgelegt wird dabei vor allem die Dialogstruktur. Die Methode erlaubt eine Transformation sowohl auf eine Oberfläche mit einem Menü- und Fenstersystem als auch mit direkter Manipulation. In dem hier ausschließlich betrachteten ersten Fall lässt sich die Methode wie folgt umreißen:

- Für jede Klasse wird ein Erfassungsfenster und ein Listenfenster erstellt. In dem Erfassungsfenster werden alle Attribute auf Dialogelemente abgebildet. Im Listenfenster werden alle jeweils vorhandenen Objekte tabellarisch mit jeweils einigen wesentlichen Attributen repräsentiert.
- Das Erfassungsfenster dient dem Anlegen neuer und dem Modifizieren vorhandener Objekte, wobei auch andere Operationen möglich sind. Mit dem Listenfenster können alle Objekte angezeigt, einzelne Objekte selektiert und unmittelbar gelöscht werden. Ferner werden in Listenfenstern Klassenattribute angezeigt und Klassenoperationen angeboten. Zwischen beiden Fenstern einer Klasse kann gewechselt werden, so dass z.B. ein selektiertes Objekt anschließend im Erfassungsfenster bearbeitet werden kann.
- Jede (binäre) Assoziation wird in den Erfassungsfenstern der beiden beteiligten Klassen unter Benutzung von Schaltflächen auf- oder abgebaut. Zur Auswahl bereits vorhandener assoziierter Objekte der jeweils anderen Klasse wird deren Listenfenster benutzt. Wird ein zu assoziierendes Objekt erst neu erstellt, so wird stattdessen das Erfassungsfenster der assoziierten Klasse verwendet. Im Erfassungsfenster einer Klasse werden pro Objekt alle assoziierten Objekte anderer Klassen durch Schlüsselattribute angegeben. So erscheint etwa die Nummer eines Lieferanten im Erfassungsfenster der Klasse Artikel. Ist ein Objekt mit mehreren Objekten einer anderen Klasse verbunden, so werden diese in einem zusätzlichen Listenelement des Erfassungsfensters angezeigt. Beispiels-

**Transformation des  
Klassendiagramms auf  
die GUI**

**Erfassungs- und  
Listenfenster**

**Assoziationen**



**Vererbungs-  
beziehungen**

weise können in einem Listenfeld des Erfassungsfensters für Lieferanten alle von einem Lieferanten gelieferten Artikel erscheinen. Zwischen den Fenstern assoziierter Klassen kann jederzeit gewechselt werden.

- Auch die Abbildung von Vererbungsbeziehungen als weiterem Strukturtyp von Klassendiagrammen auf die Benutzungsoberfläche erfolgt systematisch nach gewissen Regeln, wobei zwischen abstrakten und konkreten Oberklassen differenziert wird. Erwähnt sei an dieser Stelle nur noch, dass von einer Oberklasse ererbte Attribute stets einheitlich in den Fenstern der Unterklassen dargestellt werden sollen, damit der Benutzer ihre Übereinstimmung erkennt.

**Charakterisierung  
der Methode**

Charakteristisch für die vorgeschlagene Methode ist, dass sie zu einer Benutzungsoberfläche mit einer sehr einheitlichen Dialogstruktur führt. Diese entspricht sicherlich dem Grundsatz der Erwartungskonformität in hohem Maße und ist vom Benutzer leicht zu erlernen. Die vorgegebenen Regeln sollten auch die Erstellung der Oberfläche erleichtern.

Wie in BALZERT (1999) ausgeführt, kann die Methode ohne weiteres auch auf die Erstellung eines Prototyps angewendet werden. Dieser veranschaulicht dann nicht nur den Aufbau von Fenstern, die Datendarstellung in Fenstern sowie die allgemeinen Dialogabläufe, sondern zeigt darüber hinaus bereits den Auf- und Abbau von Assoziationen. Dabei ist allerdings zu beachten, dass oft ein großer Teil der Assoziationen in der realisierten Anwendung nicht innerhalb der GUI, sondern durch interne Operationen der Fachklassen auf- bzw. abgebaut wird.

## 5.2.2 Projekt Tourenplanungssystem: GUI-Prototyp

**Ziele und Aspekte  
der Erstellung  
des GUI-Prototyps  
für das TPS**

Bei der Erstellung des GUI-Prototyps für das Tourenplanungssystem standen folgende Ziele und Aspekte im Vordergrund:

- Der Prototyp sollte in erster Linie den Besonderheiten der einzelnen Klassen mit ihren Attributen und Assoziationen bzw. fachlichen Zusammenhängen zu anderen Klassen gerecht werden. Eine weitgehend homogene Abbildung der Klassen und ihrer strukturellen Beziehungen wurde also nicht angestrebt.
- Während es pro Klasse in der Regel ein Erfassungsfenster für die Erzeugung und Modifikation von Objekten gibt, wurde bei weitem nicht für jede Klasse ein Listenfenster vorgesehen. Stattdessen wurde versucht, Listenfenster auf zwei Wegen einzusparen.
- Die erste Variante sieht vor, in einem zusätzlichen Listenelement des Erfassungsfensters einer Klasse alle vorhandenen Objekte der Klasse anzuzeigen. Bei der zweiten Variante wird ein Listenfenster für die vorhandenen Objekte der Klasse innerhalb des Erfassungsfensters einer fachlich eng verwandten Klasse platziert.
- Die allgemeinen Regeln für die Gestaltung von Fenstern, Menüs und Dialogelementen sollten natürlich beachtet werden.
- Auf die Realisierung des Auf- und Abbaus von Assoziationen innerhalb des Prototyps wird verzichtet. Der Prototyp zeigt also nur den Aufbau von und die Datenrepräsentation in Fenstern sowie die allgemeinen Dialogabläufe, d.h. den Aufruf von Dialogen ausgehend von Menüs sowie ggf. verschachtelte Dialogaufrufe.

Aus dem letzten Punkt geht bereits hervor, dass ein Prototyp mit einem Menü- und Fenstersystem anstelle einer direkten Bedienung erstellt werden sollte. Darüber hinaus wurden im Zusammenhang mit der Prototyp-Erstellung und seiner beabsichtigten evolutionären Weiterentwicklung folgende Entscheidungen bezüglich der späteren Anwendung und ihrer GUI getroffen:

#### Grundsatz- entscheidungen zum TPS

- Die Anwendung TPS soll als SDI-Anwendung realisiert werden. Maßgeblich hierfür war das Argument der einfacheren Realisierung. Doch erscheint eine SDI-Anwendung für die geplante Funktionalität der Anwendung zunächst einmal auch als ausreichend.
- Die Dateneingabe erfolgt ausschließlich über Dialogfenster bzw. -boxen. Diese werden auch für die Anzeige der meisten Anwendungsdaten benutzt.
- Dagegen werden Daten, die auch ausdrückbar sein sollen, im Anwendungsfenster ausgegeben.
- Es kommen ausschließlich modale Dialoge zum Einsatz, weil deren Flexibilität als ausreichend erscheint.

Aus methodischer Sicht ist anzumerken, dass die angeführten Entscheidungen bereits dem Entwurf zuzuordnen sind. Die Erstellung eines Prototypen befindet sich also bei seiner evolutionären Weiterentwicklung an der Nahtstelle zum Entwurf. Zu beachten ist in diesem Fall, dass alle grundlegenden Entwurfsentscheidungen aufeinander abgestimmt und daher gemeinsam zu treffen sind, worauf hier allerdings nicht näher eingegangen wird.

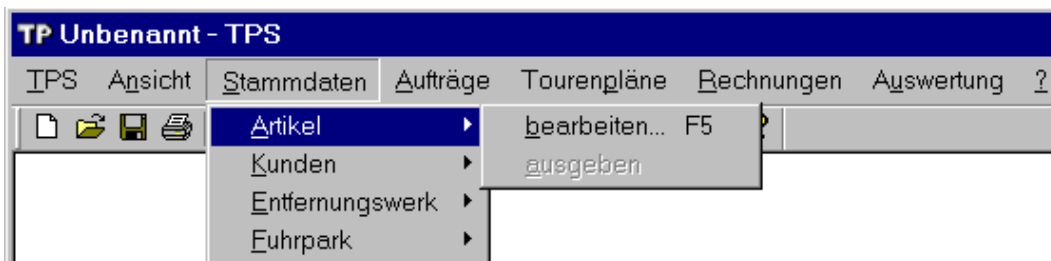
#### GUI-Prototyp und Entwurf

Der vollständige Prototyp befindet sich als lauffähiges Programm auf der CD-ROM zum Kurs. Für Details wie die Bedeutung von Schaltflächen sei auch auf die Benutzeranleitung zum TPS verwiesen. Im Folgenden werden nur einige Komponenten des Prototyps vorgestellt, um die bei seiner Erstellung berücksichtigten Aspekte exemplarisch zu verdeutlichen.

### (1) Objektorientierte Menügestaltung

Die Abb. 5.3 veranschaulicht die objektorientierte Menügestaltung des Prototyps. Im Hauptmenü und ggf. in unmittelbar nachgeordneten Drop-Down-Menüs erscheinen die wesentlichen Bearbeitungsobjekte der Anwendung. Erst nach der Wahl eines Objekttyps kann für diesen in einem (weiteren) Untermenü eine Bearbeitungsfunktion ausgewählt werden.

#### objektorientierte Menügestaltung



**Abb. 5.3.** Objektorientierte Menügestaltung.

Die primäre Menügliederung nach Bearbeitungsobjekten sollte dem Benutzer die Orientierung erleichtern. Sie erscheint in wesentlich höherem Maße selbsterklärend als eine primäre Gliederung nach Bearbeitungsfunktionen (z.B.: „bearbeite“ – was denn?). Außerdem gestattet die Gliederung nach Objekten, in gewissem Maße den grundlegenden Arbeitsablauf nachzubilden. Aufträge werden erfasst,

bevor sie in Tourenplänen disponiert werden, Rechnungen werden jeweils am Ende eines Arbeitszyklus erstellt. Kontextmenüs besitzt der Prototyp nicht.

## (2) Dialogfenster „Kunde bearbeiten“ und „Ansprechpartner“

Fenster für  
Kunden  
und  
Ansprechpartner

Beide Dialogfenster werden in Abb. 5.4 und Abb. 5.5 gezeigt. In dem Fenster „Kunde bearbeiten“ werden die Daten eines Kunden erfasst oder modifiziert. Zusätzlich zeigt ein Listenelement alle vorhandenen Kunden an und gestattet ihre Selektion. Ein separates Listfenster wird auf dem ersten eingangs genannten Weg gespart. Allerdings werden die Daten eines Ansprechpartners in das zweite hier gezeigte Fenster ausgelagert. Doch sind dies weniger wichtige, optionale Daten und der Name eines Ansprechpartners wird ggf. im Kundenfenster angezeigt.

**Kunden bearbeiten**

Allgemeine Daten:

Kunden-Nr.:  Firmenname:

Ansprechpartner:

Anschrift:

PLZ:  Ort:

Strasse:

Kunden-Typ:

☐ Laufkunde ☐ Stammkunde Min. Auftragswert (Euro/Monat):

Kunden:

Kunden...	Firmenname	Ort	Typ

Abb. 5.4. Dialogfenster „Kunde bearbeiten“.

Das einfache Fenster für einen Ansprechpartner veranschaulicht, wie vertikale virtuelle Linien durch eine geeignete Wahl der Breite von Eingabefeldern und ihre Platzierung nebeneinander reduziert werden können.

Abb. 5.5. Dialogfenster „Ansprechpartner“.

### (3) Dialogfenster „Entfernungswerk bearbeiten“

In diesem Fenster können je zwei Kunden selektiert und anschließend ihre Entfernung eingegeben werden, vgl. Abb. 5.6. Die Schaltfläche „Nächste“ unterstützt die Selektion, indem jeweils ein weiteres Kundenpaar ohne Entfernungseintrag ausgewählt wird. Zuvor wird die zuletzt eingegebene Entfernung abgespeichert, so dass die Schaltfläche „Speichern“ weniger häufig benutzt werden muss. Entfernungen können entweder neu erfasst oder modifiziert, d.h. überschrieben werden.

Fenster für  
Entfernungswerk

Abb. 5.6. Dialogfenster „Entfernungswerk bearbeiten“.

### (4) Dialogfenster „Auftrag bearbeiten“ und „Auftragsposition bearbeiten“

Beide Dialogfenster werden in Abb. 5.7 und Abb. 5.8 gezeigt. In diesem Fall wird ein Listenfenster für Auftragspositionen auf dem zweiten eingangs genannten Weg gespart. Ein Listenelement für die Auftragspositionen befindet sich im Fenster „Auftrag bearbeiten“.



Abb. 5.8. Dialogfenster „Auftragsposition bearbeiten“.

### (5) Interaktion von Dialogen

Die Abb. 5.9 veranschaulicht eine mögliche Interaktion von Dialogen bei der Auftragsbearbeitung. Ausgehend vom Listenfenster für Aufträge wird ein neuer Auftrag angelegt. Erst bei der Zuordnung des Kunden zum Auftrag stellt sich heraus, dass der Kunde noch gar nicht im System vorhanden ist. In diesem Fall kann der Kunde erfasst werden, wie das obenaufliegende Ansprechpartner-Fenster zeigt. Und zwar ohne, dass der Auftragsdialog zuvor abgebrochen werden muss.

### Interaktion von Dialogen

Abb. 5.9. Beispiel einer Interaktion von Dialogen.

Analoges gilt, wenn sich bei der Erfassung einer Auftragsposition nachträglich ergibt, dass der gewünschte Artikel noch nicht erfasst wurde.

Die Beispiele verdeutlichen, dass auch mit modalen Dialogen ein gewisses Maß an Handlungsflexibilität erreicht werden kann. Um Missverständnissen vorzubeugen, sei hinzugefügt, dass modale Dialoge in jedem Falle dann nicht mehr ausreichen, wenn eine MDI-Anwendung zur gleichzeitigen Bearbeitung mehrerer Dokumente erforderlich ist.

## **(6) Tourenpläne und Touren**

### **Tourenpläne und Touren**

Hingewiesen sei zunächst auf eine Besonderheit bei der GUI-Darstellung von Tourenplänen und Touren. Bei einem Tourenplan können nur die Planungsdaten wie z.B. die aktuelle Witterung erfasst werden. Dagegen wird der eigentliche Tourenplan intern berechnet. Erfasste und berechnete Tourenplandaten werden wiederum in einem Fenster dargestellt, da eine Aufspaltung in zwei Fenster für den Benutzer eher umständlich wäre. Das Tourenplanfenster kann also nur in einem stark eingeschränkten Sinne als Erfassungsfenster bezeichnet werden. Für Touren gibt es gar kein Erfassungsfenster, weil sie ausschließlich durch Berechnung gewonnen werden. Dies zeigt, dass eine einfache Abbildung jeder Klasse des OOA-Modells auf ein Erfassungs- bzw. Listenfenster offenbar nicht in jedem Falle möglich ist (vgl. Kap. 5.2.1).

Da ein Tourenplan und seine Touren fachlich eng zusammengehören, wird wiederum eine Liste aller Touren als Listenelement im Tourenplanfenster platziert, so dass ein besonderes Listenfenster für Touren nicht erforderlich ist. Dementsprechend gibt es analog zu Aufträgen ein separates Listenfenster für Tourenpläne. Vom Standpunkt des Benutzers ist es sinnvoll, die globalen Informationen eines Tourenplans und seiner Touren auf einen Blick zu sehen. Dagegen wäre es eher störend, zusätzlich zu einem konkreten Tourenplan die seltener benutzte Liste aller Tourenpläne vor Augen zu haben.

## **(7) Ausgabe von Rechnungen**

### **Ausgabe von Rechnungen**

Rechnungen werden im Anwendungsfenster des TPS ausgegeben und können anschließend auch ausgedruckt werden. Bereits an dieser Stelle sei bemerkt, dass die programmiertechnisch schwierigere Ausgabe im Anwendungsfenster nicht im Rahmen des Prototyps realisiert wurde, vgl. Kap. 5.2.3, Abschnitt (9). Die folgende Abb. 5.10 zeigt also eine Rechnung, die mit der fertigen GUI erzeugt wurde.



**Rechnung der Firma Schraube & Partner**

**An**  
Gärtnerei Rosenbeet  
Fettnäpfchen-Str. 73e  
03645, Unterhütten

**KundenNr.:** 00008  
**RechnungsNr:** 00002

**Betrifft:** Ihr Auftrag vom 18.01.01

*Hagen, den 03.02.01*

*Sehr geehrte Damen und Herren,*

*gemäss Ihrem Auftrag vom 18.01.01 haben wir Ihnen am 19.01.01 die gewünschten Artikel geliefert. Bitte überweisen Sie den angegebenen Rechnungsbetrag auf unser Konto mit der Nummer 123456789 bei der Banko 'd International auf den Bahamas.*

*Mit freundlichen Grüßen, Siegfried Schraube*

**Gelieferte Artikel**

Anzahl	Artikel	Verkaufseinheit	Preis Euro	Betrag Euro
4	Wellbleche "Donauwelle"	100m x 3m (gerollt)	756.78	3027.12
<b>Summe der Beträge:</b>				<b>3027.12</b>
<b>Umsatzsteuer (16%):</b>				<b>484.34</b>
<b>Rechnungsbetrag:</b>				<b>3511.46</b>

**Abb. 5.10** Beispiel einer Ausgabe im Anwendungsfenster.

### 5.2.3 Prototyperstellung mit MS Visual C++

Im Folgenden wird anhand des Tourenplanungssystems beschrieben, wie ein GUI-Prototyp mit Hilfe der Entwicklungsumgebung MS Visual C++ (MSVC) und den Microsoft Foundation Classes (MFC) erstellt werden kann. Für eine umfassende Darstellung von MSVC und der MFC seien Interessierte auf KRUGLINSKI et al. (1998) verwiesen.

Die Entwicklungsumgebung MS Visual C++ gestattet die Anwendungsprogrammierung mit Hilfe der MFC. Die MFC sind die Programmierschnittstelle des Betriebssystems Windows für die Sprache C++. Sie stellen eine Klassenbibliothek und darüber hinaus ein Framework dar. Man sagt auch, dass die MFC ein Anwendungsgerüst bilden. Wird eine Anwendung auf der Basis der MFC entwickelt, so besitzt sie eine vorgegebene Struktur. Die allgemeine Struktur von MFC-Anwendungen wird als Dokument-Ansicht-Architektur bezeichnet. Der Kern dieser Architektur besteht darin, die Anwendungsdaten von der Benutzersicht auf die Anwendungsdaten zu trennen. Dies erlaubt z.B. die Erzeugung mehrerer Ansichten derselben Daten, etwa einer tabellarischen und einer grafischen Ansicht.

Die mit der Ansicht-Komponente von Anwendungen korrespondierenden Klassen der MFC bilden die Basis für die Erstellung grafischer Benutzeroberflächen

**MFC  
als  
Anwendungsgerüst**

**GUI-Klassen  
von MFC**

entsprechend dem GUI-System Windows. Anders ausgedrückt bilden die GUI-Klassen der MFC eine Programmierschnittstelle zum GUI-System Windows. Die GUI-Klassen kapseln das Verhalten, d.h. die Steuerung von verschiedenen GUI-Elementen, darunter die in Kap. 5.1 erläuterten Typen von Fenstern, Dialogelementen und Menüs. Für jede Art von GUI-Elementen ist eine eigene Klasse zuständig und mehrere Elemente gleichen Typs werden von je einer Instanz dieser Klasse gesteuert. Indem man eine eigene Klasse von einer MFC-Klasse ableitet, kann man das Verhalten eines GUI-Elements den eigenen Wünschen anpassen.

#### MSVC-Komponenten

Als integrierte Entwicklungsumgebung (kurz IDE für integrated developing environment) umfasst MSVC alle für eine Anwendungsprogrammierung benötigten Werkzeuge, von denen einige in Tab. 5.8 kurz vorgestellt werden.

Komponente	Erläuterung
Quelltexteditor	- Erfasst und bearbeitet Quelltextdateien (*.h, *.cpp).
C/C++-Compiler	- Compiliert Quelltextdateien (*.h, *.cpp) in Objektdateien (*.obj).
Ressourceneditoren	- Ressourcen umfassen u.a. Dialogboxen, Menüs, Symbolleisten (Toolbars), Zeichenfolgen und Symbole (Icons). - Mit einem Ressourceneditor können Ressourcen des jeweiligen Typs einfach und teils grafisch erstellt werden. - Die Ressourcenscriptdatei (*.rc) eines Projekts (s. unten) beschreibt dessen Ressourcen im Textformat.
Ressourcencompiler	- Compiliert Ressourcen (*.rc) in zugehörige Binärdateien (*.res).
Anwendungsassistent	- Erstellt das Gerüst einer konkreten Anwendung entsprechend der MFC-Anwendungsarchitektur und mitzugebenden Parameterwerten.
Klassenassistent	- Erstellt den Rahmencode zusätzlicher Klassen oder Funktionen. - Stellt Verbindungen zwischen Ressourcen und Quellcode her.
Linker	- Fügt alle übersetzten Anwendungskomponenten zu einem lauffähigen Programm (*.exe) zusammen. - Komponenten sind: übersetzte Quellcodedateien (*.obj), übersetzte Ressourcen (*.res), Windows-, Laufzeit- und MFC-Bibliotheken.
Debugger	- Dient der Lokalisierung und Behebung von Fehlern. - Compilierte und gelinkte Programme können im Debugger gestartet werden.

**Tab. 5.8.** Einige Komponenten der IDE MS Visual C++.

Die Abb. 5.11 zeigt die IDE, wobei im Vordergrund der Klassenassistent, oben rechts der Ressourceneditor für Dialoge und unten rechts der Quelltexteditor zu erkennen ist. Nachfolgend werden die verschiedenen Ressourceneditoren auch zusammenfassend als *der* Ressourceneditor bezeichnet.

#### Projekt

Für jede Anwendung ist zunächst ein Projekt einzurichten. Ein Projekt fasst alle Komponenten zusammen, die für die Erstellung eines Programms erforderlich sind. Dazu gehören sämtliche Quelltextdateien des Programms, ferner die Ressourcen und Compilereinstellungen. Jedes Projekt gehört seinerseits zu einem Arbeitsbereich, der auch mehrere Projekte umfassen kann.



Die Dateiansicht listet alle im Projekt integrierten Dateien auf, getrennt nach den Implementierungs-Dateien (in der Dateiansicht Quellcodedateien genannt), den Definitions-Dateien (auch: Header-Dateien) und den Ressourcen-Dateien.

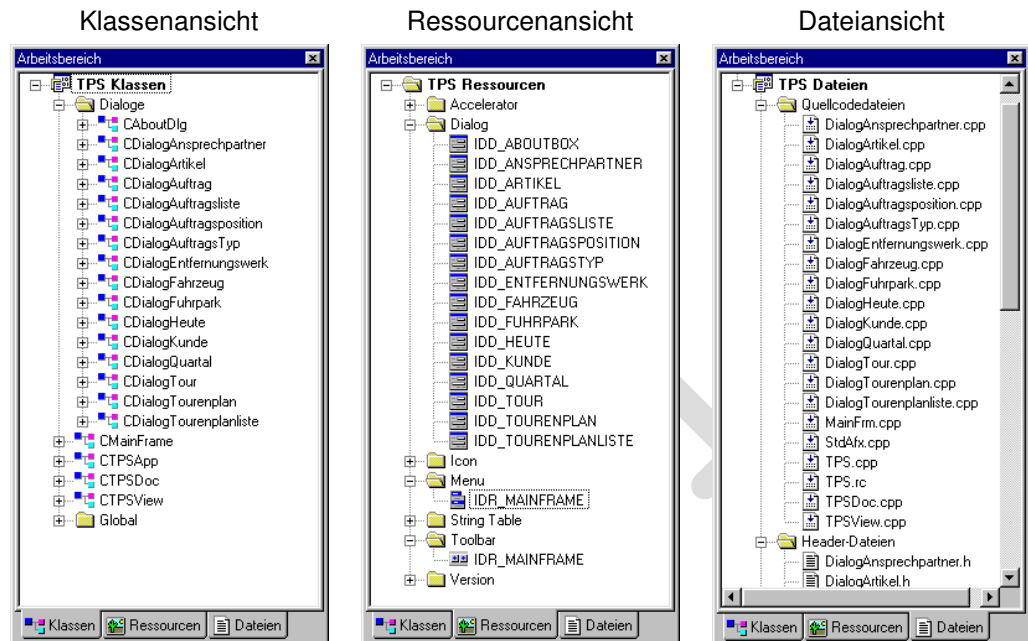


Abb. 5.12. Ansichten eines Projekts in der IDE.

#### Schritte zur Erstellung eines GUI-Prototypen

Gemäß den in Kap. 5.2.2 getroffenen Entscheidungen zum Umfang und den Merkmalen des Prototyps des Tourenplanungssystems lässt sich seine Programmierung in folgende Schritte gliedern:

1. Erstellung des Rahmencodes (Anwendungsassistent).
2. Anpassung der Menüstruktur (Ressourceneditor).
3. Anpassung der Toolbar (Ressourceneditor).
4. Erstellung der Dialogboxen (Ressourceneditor).
5. Erstellung der Behandlungsklassen für Dialoge (Klassenassistent).
6. Erstellung von Behandlungsroutinen für Menüpunkte, die Dialoge aufrufen (Klassenassistent).
7. Erstellung von Behandlungsroutinen für Buttons auf Dialogboxen, die den Dialog beenden oder weitere Dialoge aufrufen (Klassenassistent).
8. Initialisierung von Dialogelementen (Klassenassistent).
9. Entwurf von Ausgaben im Arbeitsbereich des Anwendungsfensters.

Die angegebenen Schritte sollen nun im Einzelnen behandelt werden.

#### (1) Erstellung des Rahmencodes

##### Rahmencode mit Anwendungsassistent erzeugen

Bei dem Anlegen eines Projekts für den Prototyp hilft der Anwendungsassistent. Er generiert einen Rahmencode für das Projekt, das heißt ein rudimentäres Programm, das zwar lauffähig ist, aber nur die Basis für eigene Erweiterungen bildet. Um ein neues Projekt anzulegen, benötigt der Anwendungsassistent eine Reihe von Angaben.

##### exe-Datei

Da der Assistent verschiedenartige Projekte unterstützt, muss ihm zunächst mitgeteilt werden, dass er für exe-Dateien aufgerufen werden soll, vgl. Abb. 5.13. Au-

ßerdem müssen ein Projektname und ein Verzeichnis (Pfad) für die zum Projekt gehörenden Dateien angegeben werden.

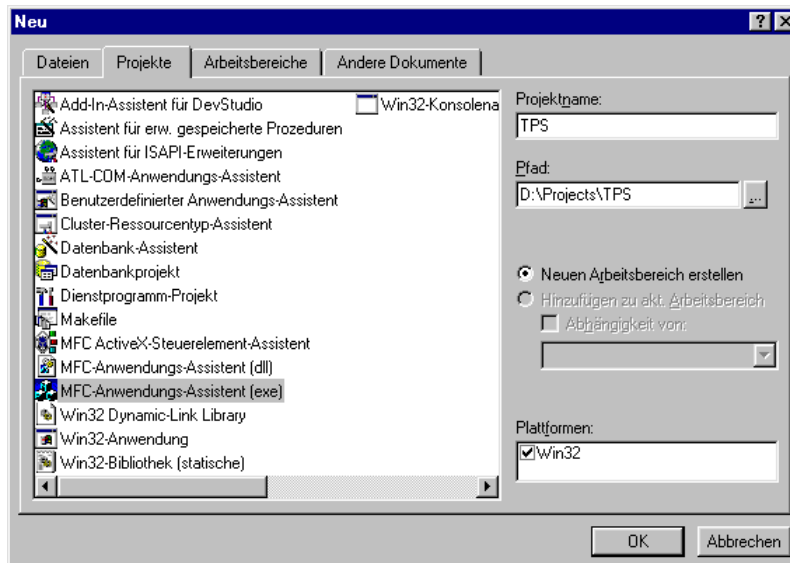


Abb. 5.13. Neues Projekt erstellen.

In den nächsten sechs Schritten können Einstellungen verändert werden, die die Funktionalität des Rahmencodes beeinflussen, vgl. Abb. 5.14.

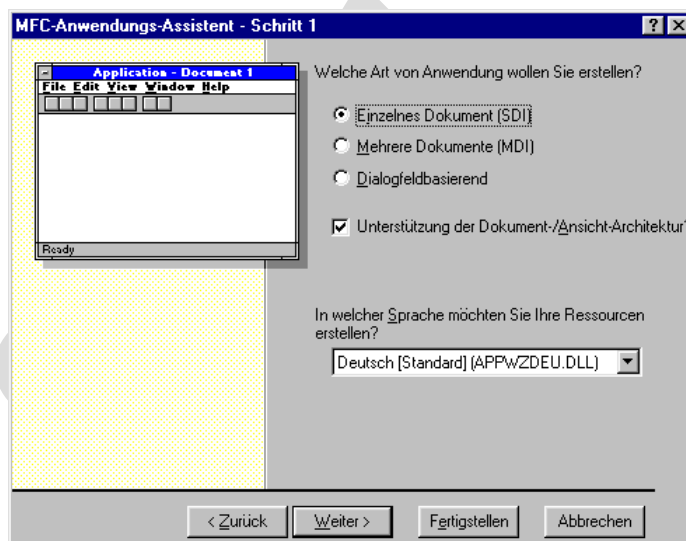


Abb. 5.14. MFC-Anwendungsassistent (Schritt 1).

Die meisten Standard-Einstellungen sind passend für das Tourenplanungssystem. Die einzige für das TPS zwingend notwendige Änderung ist die Wahl der Anwendungsart SDI, da das Programm zu einem Zeitpunkt nur einen Satz von Anwendungsdaten bearbeiten und nur ein Rahmenfenster besitzen soll. Alle anderen Einstellungen und Optionen in den nächsten fünf (hier nicht gezeigten) Schritten beeinflussen den Prototypen für das TPS nicht oder können später ergänzt werden.

Nachdem alle Schritte durchlaufen wurden, erstellt der Anwendungsassistent eine Übersicht über die gewählten Einstellungen und Optionen. Hier kann nochmals alles kontrolliert werden, bevor der Assistent den Rahmencode erstellt.

Dieser Rahmencode kann sofort kompiliert und gestartet werden, vgl. Abb. 5.15. Das erzeugte Programm besitzt allerdings nur rudimentäre Funktionen. Um einen

**Anwendungsart:  
SDI**

anwendungsspezifischen Prototyp zu erhalten, muss das Programm erweitert und modifiziert werden.

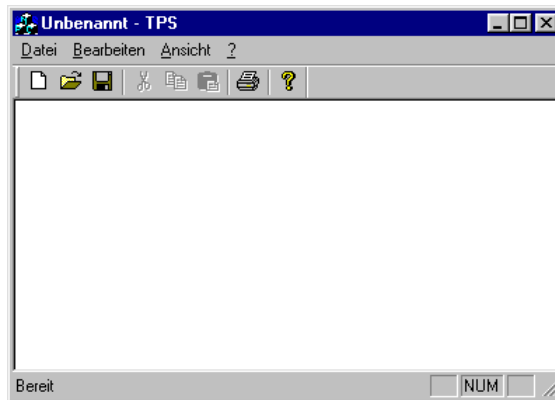


Abb. 5.15. Erster Programmstart.

## (2) Anpassung der Menüstruktur

### Menüstruktur anpassen ...

Zuerst empfiehlt sich eine Anpassung der Menüstruktur, da das Menü bei der Steuerung eines Programms eine zentrale Rolle übernimmt. Die Abb. 5.16 zeigt zur Orientierung bereits die fertige Menüstruktur.

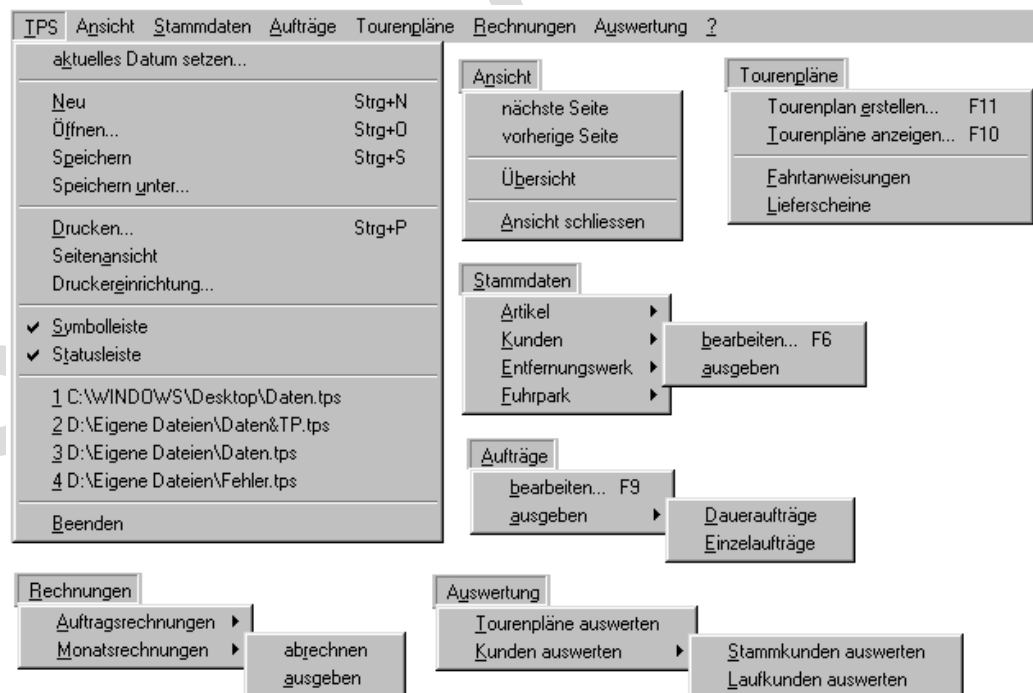


Abb. 5.16. Komplette TPS-Menüstruktur.

### ...mit Ressourceneditor

Wählt man in der Arbeitsbereich-Ansicht das Register Ressourcen, findet man dort das Hauptfenster-Menü mit der Ressourcen-ID IDR\_MAINFRAME. Ein Doppelklick darauf öffnet den Ressourceneditor für Menüs, in dem das Hauptmenü des Programms editiert werden kann, vgl. Abb. 5.17.

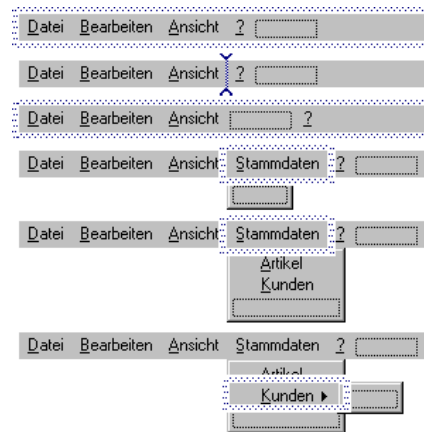


Abb. 5.17. Editieren von Menüs.

Die Menüpunkte können mit drag-&-drop an die gewünschten Positionen verschoben werden. So ist z. B. in der Hauptmenüzeile am rechten Ende eine Einfügemarke, die an die gewünschte Position verschoben werden kann. Gibt man nun die gewünschte Bezeichnung des Menüpunkts ein, so erscheint unter dem Menüpunkt eine weitere Einfügemarke, mit der das Untermenü editiert werden kann.

Menüpunkte

Sobald die Bezeichnung eines Menüpunkts eingegeben wird, öffnet sich automatisch das Eigenschaften-Fenster, in dem die Eigenschaften des aktuell markierten Menüpunkts eingegeben werden können, vgl. Abb. 5.18.

Eigenschaften-Fenster

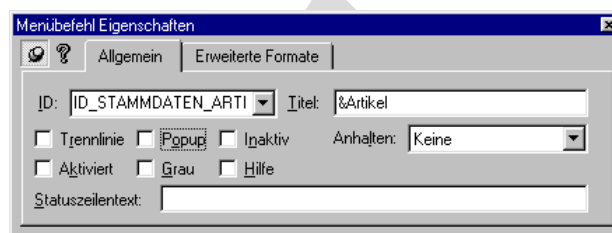


Abb. 5.18. Eigenschaften-Fenster für Menüpunkte.

Damit das Fenster nicht wieder geschlossen wird, sobald es seinen Fokus verliert, kann man das Schließen mit einem Klick auf die Reißzwecke in der linken oberen Ecke des Fensters verhindern. Ist das Eigenschaften-Fenster schon geschlossen, kann es mit der Tastenkombination Alt-Enter wieder geöffnet werden. Alternativ kann man das Fenster mit Rechtsklick -> Eigenschaften auf einen Menüpunkt öffnen.

Die Eigenschaft ID definiert eine gleichnamige #define-Konstante, mit der der Menüpunkt im Quelltext eindeutig referenziert werden kann. Der Zahlenwert der Konstanten ID wird automatisch vom Menü-Editor bestimmt, der Name der Konstanten kann aber im entsprechenden Feld geändert werden. Meist ist eine Änderung des Namens jedoch nicht nötig, da er anhand der Menüstruktur gebildet wird. So wird z. B. der Menüpunkt „bearbeiten“ im Untermenü Aufträge mit ID\_AUFTRGE\_BEARBEITEN benannt (Umlaute werden entfernt).

Wird die Eigenschaft Popup eines Menüpunkts aktiviert, so öffnet ein Klick auf diesen ein neues Untermenü. Dieses Untermenü kann wie zuvor erläutert mit Hilfe der entsprechenden Einfügemarke rechts neben dem Menüpunkt editiert werden.



Der Statuszeilentext kann ebenfalls im Eigenschaften-Fenster eingegeben werden und erscheint in der Statuszeile, wenn sich der Mauszeiger über den Menüpunkt bewegt. Der Text sollte die Funktion eines Menüpunkts kurz beschreiben.

Um einen schnellen Zugriff auf die Menüpunkte mit der Tastatur zu ermöglichen, sollte jeder einen eindeutig definierten Hot-Key besitzen. Wenn man im Titel des Menüpunkts einem Zeichen ein & voranstellt, wird es unterstrichen und ein Druck auf die entsprechende Taste (im Hauptmenü in Kombination mit der Alt-Taste) wählt diesen Menüpunkt aus. So lässt sich z. B. das Menü Stammdaten mit der Tastenkombination Alt-S öffnen, das Untermenü Kunden allein mit der Taste K.

### (3) Anpassung der Toolbar

#### Toolbar anpassen...

Häufig benutzte Menüpunkte können in der Toolbar durch kleine passende Symbole repräsentiert werden. In der Abb. 5.19 wird bereits die fertige Toolbar des Prototypen gezeigt.



Abb. 5.19. Komplette TPS-Toolbar.

#### ...mit Ressourceneditor

Um den Ressourceneditor für Toolbars zu öffnen, doppelklickt man in der Ressourcen-Ansicht des Arbeitsbereichs-Fensters auf die Ressource IDS\_MAINFRAME im Unterpunkt Toolbars.

#### Reihenfolge und Zeichnen der Symbole

Im oberen Bereich des Toolbar-Editors ist die gesamte Toolbar zu sehen, in der die Symbole per drag-&-drop in die gewünschte Reihenfolge gebracht werden können. Wird die Einfügemarke am rechten Ende angeklickt, kann ein neues Symbol gezeichnet und wieder an die gewünschte Stelle verschoben werden. Zum Zeichnen eines Symbols stehen verschiedene Werkzeuge und Farben zur Verfügung, die in der editoreigenen Toolbox ausgewählt werden können.

#### Eigenschaften-Fenster

Damit ein Symbol einen bestimmten Menüpunkt repräsentieren kann, muss die ID des Symbols mit der ID des entsprechenden Menüpunkts übereinstimmen. Die ID des Symbols kann im Eigenschaften-Fenster festgelegt werden, das durch die Tastenkombination Alt-Enter geöffnet werden kann, vgl. Abb. 5.20.

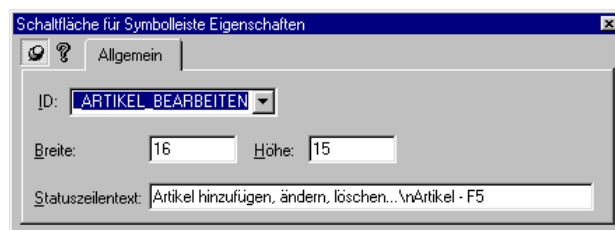


Abb. 5.20. Eigenschaften-Fenster für Symbole.

Der Statuszeilentext entspricht dem, der für den Menüpunkt festgelegt wurde und erscheint auch, wenn der Mauszeiger sich über dem Symbol in der Toolbar bewegt. Wird ein Teil des Textes durch einen Zeilenumbruch ('\n') vom Rest getrennt, erscheint dieser als Tooltip, wenn man die Maus kurz über dem Symbol ruhen lässt.

#### (4) Erstellung der Dialogboxen

Wenn man in der Ressourcen-Ansicht auf den Unterpunkt Dialog rechtsklickt, kann man in dem dann erscheinenden Kontextmenü mit dem Menüpunkt „Dialog einfügen“ eine neue Dialogbox erzeugen.

**Dialogbox  
anlegen**

Gleich darauf sollte man in der Ressourcen-Ansicht auf die eingefügte Ressource rechtsklicken und mit dem Menüpunkt Eigenschaften das Eigenschaften-Fenster öffnen. Dort sollte man den ID-Namen dieser Dialogressource von IDD\_DIALOG1 passend ändern und den Titel der Dialogbox anpassen. Weitere Änderungen an den Eigenschaften des Dialogs sind meist nicht notwendig.

Es empfiehlt sich, zunächst eine Skizze einer zu erstellenden Dialogbox auf einem Blatt Papier anzufertigen. Die folgende Abb. 5.21 zeigt eine solche Skizze für die Dialogbox „Auftrag bearbeiten“.

**Skizze**

**Abb. 5.21.** Entwurf der Dialogbox „Auftrag bearbeiten“.

Ist man sich über die Positionen und Typen der verschiedenen Dialogelemente im Klaren, können im geöffneten Dialogeditor die gewünschten Dialogelemente innerhalb der Dialogbox ausgewählt und positioniert werden. Dazu wählt man in der Toolbox rechts im Editor das gewünschte Dialogelement aus und klickt an die gewünschte Position auf der Dialogbox. Nachträglich ändert man die Position, indem man das Element an die gewünschte Position verschiebt.

**Dialogelemente  
wählen**

Bei Dialogelementen mit einer Beschriftung kann diese sofort nach dem Platzieren eingegeben werden.

Alle Dialogelemente haben Eigenschaften, die jeweils separat für ein Element im Eigenschaften-Fenster geändert werden können. Dabei zeigt das Eigenschaften-Fenster immer die Eigenschaften des aktuell markierten Elements an. Es kann wie

**Eigenschaften-  
Fenster**

gewohnt mit der Tastenkombination Alt-Enter geöffnet werden, oder indem man auf das Element rechtsklickt und im Kontextmenü die Option Eigenschaften wählt.

Jedem Element in der Dialogbox weist der Dialogeditor eine eindeutige ID zu. Im Gegensatz zu den Namen der Menüpunkte sollte im Falle von Dialogelementen der vorgeschlagene Name der Konstanten angepasst werden. Zum Beispiel sollte der Name eines Editfeldes, in dem ein Vorname einzutragen ist, von IDC\_EDIT1 in IDC\_EDIT\_Vorname geändert werden.

#### Positionieren

Unter dem Menüpunkt Layout stehen verschiedene Befehle zur Verfügung, die beim Positionieren und Arrangieren von Elementen helfen. Dazu müssen vorher ein oder mehrere Elemente markiert werden. Mehrere Elemente markiert man gleichzeitig, indem man die Strg-Taste hält und mehrere Elemente auswählt oder indem man die linke Maus-Taste drückt und hält, während man zugleich mit der Maus eine Auswahlbox aufzieht, die alle zu markierenden Elemente komplett umfasst.

#### Tabulatorreihenfolge

Die Tabulatorreihenfolge bestimmt, in welcher Reihenfolge die Elemente mit der Tabulatortaste ausgewählt werden, also den Fokus bekommen. Üblicherweise werden die Elemente von oben links nach unten rechts durchlaufen.

Den Modus für die Bestimmung der Tabulatorreihenfolge startet man mit der Tastenkombination Strg-D. Jetzt kann die Reihenfolge festgelegt werden, indem man die Dialogelemente in der gewünschten Reihenfolge anklickt, vgl. Abb. 5.22.

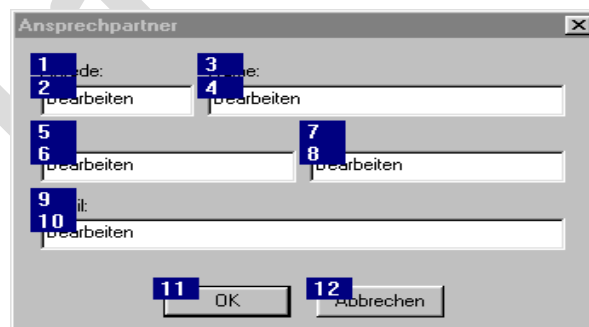


Abb. 5.22. Tabulatorreihenfolge.

Bei einigen Elementen spielt die Tabulatorreihenfolge eine Rolle bei der Interaktion mit anderen Elementen, wofür Abb. 5.23 ein Beispiel angibt.

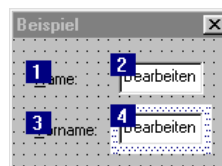


Abb. 5.23. Tabulatorreihenfolge und Interaktion von Dialogelementen.

#### Hot-Keys

Wenn man im Titel des Textfeldes einem Zeichen ein & voranstellt, wird dieses Zeichen unterstrichen und als Hot-Key bestimmt. Drückt man dann die Alt-Taste in Kombination mit der entsprechenden Taste, springt der Fokus zum in der Tabulatorreihenfolge nächsten Element. In dem Beispiel der letzten Abbildung sollte bei der Tastenkombination Alt-N der Cursor zum Editfeld neben „Name“ und bei der Kombination Alt-V zu dem Editfeld neben „Vorname“ springen.

Die folgende Tab. 5.9 erläutert einige Eigenschaften, die beim Editieren von Dialogelementen im jeweiligen Eigenschaften-Fenster angeboten werden.

#### Eigenschaften von Dialogelementen

Dialogelement	Eigenschaften
Editfeld	<ul style="list-style-type: none"> <li>- <b>Nummer.</b> Erzwingt Eingabe ganzer Zahlen.</li> <li>- <b>Schreibgeschützt.</b> Feld wird grau dargestellt und Änderung des Inhalts wird abgewiesen.</li> </ul>
Button	<ul style="list-style-type: none"> <li>- <b>Standardschaltfläche.</b> Zeichnet schwarzen Rahmen um den Button; der Button ist voreingestellt und kann durch Druck auf die Enter-Taste aktiviert werden.</li> </ul>
Listenfeld	<ul style="list-style-type: none"> <li>- <b>Auswahl.</b> Bestimmt ob nur jeweils ein Eintrag oder mehrere gleichzeitig ausgewählt werden können.</li> <li>- <b>Sortieren.</b> Bewirkt eine automatische alphabetische Sortierung der Einträge.</li> </ul>
Kombinationsfeld	<ul style="list-style-type: none"> <li>- <b>Typ.</b> Die Einstellung „Dropdown-Listenfeld“ lässt nur die Wahl eines Eintrags aus der Liste zu, ein Editieren ist nicht möglich. Ist „Dropdown“ ausgewählt, kann im Eingabefeld eine beliebige Zeichenkette editiert werden oder ein beliebiger Eintrag der Liste gewählt werden, der dann im Editfeld erscheint.</li> </ul>
Spin-Button	<ul style="list-style-type: none"> <li>- <b>Autom. Buddy.</b> Wenn die Eigenschaft aktiviert und die Anordnung „rechts“ gewählt ist, wird der Spin-Button rechts innerhalb des Eingabefeldes gezeichnet, das in der Tabulatorreihenfolge vor dem Spin-Button liegt.</li> </ul>
Listenelement	<ul style="list-style-type: none"> <li>- <b>Ansicht.</b> Wenn die Einstellung „Bericht“ gewählt ist, wird jedes Element in einer eigenen Zeile dargestellt, und zu jedem Element können in weiteren Spalten nähere Details angegeben werden.</li> <li>- <b>Sortieren.</b> Bewirkt eine Sortierung der Elemente nach den Angaben in der ersten Spalte.</li> </ul>
Date-Time-Picker	<ul style="list-style-type: none"> <li>- <b>Format.</b> Bestimmt, ob das Datum in kurzer oder in langer Form oder die Uhrzeit angezeigt wird. Im Datums-Modus kann das Datum mittels Kalenderblatt gewählt werden.</li> </ul>
Alle	<ul style="list-style-type: none"> <li>- <b>Gruppe.</b> Aktivierung der Eigenschaft bei einem Dialogelement legt den Beginn einer Gruppe fest. Zur gleichen Gruppe gehören alle in der Tabulatorreihenfolge nachfolgenden Elemente, bei denen diese Eigenschaft nicht gesetzt ist. Sobald in der Reihenfolge ein Element wieder die Eigenschaft <b>Gruppe</b> hat, beginnt mit diesem die nächste Gruppe. Durch Gruppenbildung wird bei Radio-Buttons und Checkboxes die Menge der möglichen Alternativen festgelegt. Gehören zur Gruppe der Radio-Buttons Dialogelemente, die selbst keine Radio-Buttons sind, hat das keinen Einfluss auf die Auswahlfunktion.</li> </ul>

**Tab. 5.9.** Einige Eigenschaften von Dialogelementen.

Nachdem alle gewünschten Elemente auf der Dialogbox platziert wurden, sollte man die Tabulatorreihenfolge überprüfen und kontrollieren, ob alle Elemente, auf die der Benutzer zugreifen soll, mit einem Hot-Key erreichbar sind. Um sicherzustellen, dass kein Hot-Key mehrfach belegt ist, rechtsklickt man im Dialogeditor und wählt im Kontextmenü den Menüpunkt „Zugriffstasten prüfen“.

#### abschließende Kontrolle

Bei Dialogelementen, die erst unter bestimmten Voraussetzungen ausgewählt werden dürfen, sollte die Eigenschaft „Deaktiviert“ gesetzt werden, damit diese grau dargestellt werden. Ein Beispiel dafür wäre ein Löschen-Button, der erst aktiviert wird, wenn ein Objekt zum Löschen ausgewählt wurde.

Die folgende Abb. 5.24 zeigt die fertige Dialogbox „Auftrag bearbeiten“.

Abb. 5.24. Dialogbox „Auftrag bearbeiten“.

## (5) Erstellen von Behandlungsklassen für Dialoge

Dialogklassen  
erzeugen...

Nachdem eine Dialogbox-Ressource erstellt wurde, muss eine Behandlungsklasse (bei Dialogen Dialogklasse genannt) für den Dialog erzeugt werden. Die Dialogklasse übernimmt die Steuerung des Dialogs, reagiert auf Eingaben und Knopfdrücke des Anwenders, gibt ggf. Fehlermeldungen aus, aktualisiert Dialogelemente und ruft weiterführende Dialoge auf.

Sofern man sich auf den Prototyp beschränkt, müssen nur gegenseitige Dialogaufrufe realisiert werden, um so den Arbeitsablauf beim späteren Produkt verdeutlichen zu können.

### a) Behandlungsklasse hinzufügen

...mit Klassenassistent

Der Rahmencode einer Dialogklasse wird vom MFC-Klassenassistenten erzeugt. Wenn man im Dialogeditor rechtsklickt und im Kontextmenü „Klassenassistent“ wählt, erkennt der Assistent, dass noch keine Behandlungsklasse für diese Dialogressource existiert. Er erstellt eine Klasse oder verbindet eine existierende Klasse mit diesem Dialog. Im darauffolgenden Dialog gibt man den Klassennamen ein. Alle anderen Einstellungen, wie Dateiname, Basisklasse und Dialogfeld-ID werden automatisch gewählt. Das Hinzufügen einer Behandlungsklasse wird in Abb. 5.25 veranschaulicht.

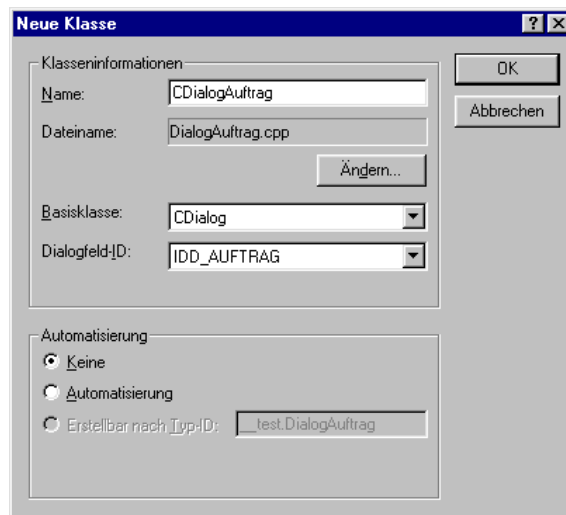


Abb. 5.25. Behandlungsklasse hinzufügen.

Die eigene Behandlungsklasse wird von einer existierenden Klasse aus der MFC-Bibliothek abgeleitet, die die generelle Funktionalität eines Dialogs implementiert. Nur die individuellen Anpassungen des eigenen Dialogs müssen ergänzt werden. Darauf sei im Folgenden näher eingegangen.

## b) Nachrichtentabelle

Dialogboxelemente senden verschiedene Nachrichten an die Dialogklasse, die dann entsprechend der Nachricht eine bestimmte Nachrichten-Behandlungsfunktion (kurz: Behandlungsfunktion, Behandlungsroutine) aufruft. Die Nachrichten der MFC stellen spezielle Botschaften dar.

Jede Dialogklasse besitzt eine Nachrichtentabelle. Mit dieser wird festgelegt, bei welcher Nachricht von welchem Dialogelement welche Behandlungsfunktion aufgerufen werden soll.

Zum Beispiel schickt ein Button die Nachricht BN\_CLICKED an die Dialogklasse, wenn er geklickt wurde. Da häufig mehrere Buttons auf einer Dialogbox sind, muss die Behandlungsklasse erst differenzieren, von welchem Button die Nachricht stammt. Danach kann die entsprechende Behandlungsfunktion aufgerufen werden. Schickt beispielsweise der Button Schließen die Nachricht BN\_CLICKED, wird die Funktion OnButtonSchliessen() aufgerufen. Schickt der Button Speichern diese Nachricht, wird hingegen OnButtonSpeichern() aufgerufen.

Darüber hinaus gibt es weitere Nachrichten, die nicht von Dialogelementen geschickt werden, sondern vom System oder vom MFC-Framework. Beispielsweise wird die Nachricht WM\_INITDIALOG geschickt und die Behandlungsfunktion OnInitDialog() aufgerufen, bevor die Dialogbox auf dem Bildschirm angezeigt wird, um so Dialogelemente initialisieren zu können, bevor diese sichtbar werden.

Wurde eine Dialogklasse vom Klassenassistenten hinzugefügt, so kann im nun geöffneten Fenster des Klassenassistenten die Nachrichtentabelle der Dialogklasse bearbeitet werden, vgl. Abb. 5.26.

**Nachrichten**

**Nachrichtentabelle**

**und ...**

**...ihre Bearbeitung**



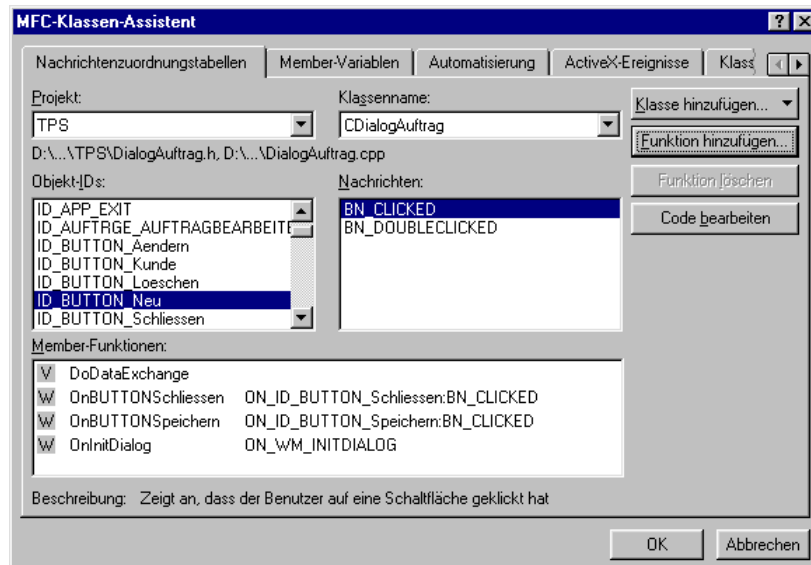


Abb. 5.26. Nachrichtentabelle einer Dialogklasse.

#### Nachrichten- Behandlungs- funktionen

Unter Objekt-IDs trägt der Klassenassistent die IDs aller Dialogelemente, aller Menüpunkte und den Klassennamen der Dialogklasse ein. Wird nun ein Objekt selektiert, erscheinen alle dazu passenden Nachrichten (und bei der Selektion des Klassennamen auch die überschreibbaren Funktionen der Basisklasse) unter Nachrichten. Wenn zusätzlich eine dieser Nachrichten selektiert wird, kann man eine entsprechende Behandlungsfunktion bzw. eine Überschreibung hinzufügen, die in der Liste der Member-Funktionen eingetragen wird. Ist die Funktion mit einem V markiert, handelt es sich um eine virtuelle Funktion (Überschreibung der Basisklassen-Funktion), ist sie mit einem W markiert, ist die Funktion eine Nachrichten-Behandlungsfunktion für eine Windows-Nachricht.

#### weitere Funktionen

Es gibt noch weitere Funktionen, die nicht von Nachrichten angestoßen werden, sondern bestimmte Aufgaben innerhalb der Dialogklasse erledigen. Überschreibt man deren Implementierungen aus der Basisklasse, können Teile des Dialogablaufs individualisiert werden. Als Beispiel sei die Funktion DoDataExchange() genannt, die jedesmal aufgerufen wird, wenn Daten zwischen Dialogelementen und Attributen der Dialogklasse ausgetauscht werden.

Beide letztgenannten Typen von Funktionen werden auch in der Nachrichtentabelle aufgelistet, um so eine Übersicht aller eigenen bzw. individualisierten Funktionen zu erhalten. Indem man seine eigene Klasse gegenüber der Basisklasse nach den eigenen Wünschen erweitert bzw. durch Überschreibungen modifiziert, kann man auf deren Funktionalität zurückgreifen, während meist nur wenige Anpassungen erforderlich sind.

### c) Behandlungsfunktion hinzufügen

#### Nachrichten- Behandlungs- funktion hinzufügen

Bei einem Prototyp sind nur zwei Nachrichtentypen von Bedeutung. Eine Nachricht des ersten Typs signalisiert, dass ein Menüpunkt angeklickt wurde, woraufhin eine Dialogbox geöffnet werden soll. Eine Nachricht des zweiten Typs übermitteln, dass ein Button gedrückt wurde, woraufhin entweder die Dialogbox wieder zu schließen oder eine weitere Dialogbox zu öffnen ist.

Um z.B. eine Behandlungsfunktion für den Button Schliessen hinzuzufügen, wählt man in der Liste Objekt-IDs die ID des Buttons. In der Liste der Nachrich-



ten erscheinen zwei Nachrichten, die der Button an die Dialogklasse senden kann. Die Nachricht BN\_CLICKED wird geschickt, wenn der Button einmal geklickt wird, die Nachricht BN\_DOUBLECLICKED, wenn der Button doppelt geklickt wird. Wählt man eine Nachricht aus, erscheint unten in der Dialogbox eine Beschreibung, unter welchen Bedingungen die Nachricht geschickt wird, und die Schaltfläche Funktion hinzufügen wird aktiviert.

Wenn man mit einem Klick auf diese Schaltfläche eine Behandlungsfunktion hinzufügt, wird nach einem Funktionsnamen gefragt und danach der Rahmencode der Funktion automatisch im Quelltext hinzugefügt, vgl. Abb. 5.27. Die Funktion und die zugehörige Nachricht werden in die Liste der Member-Funktionen aufgenommen.

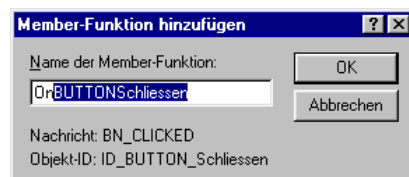


Abb. 5.27. Behandlungsfunktion hinzufügen.

Klickt man nun auf Code bearbeiten, springt der Quelltexteditor direkt an die entsprechende Stelle und man kann die Behandlungsroutine ergänzen, die nachfolgend fett dargestellt ist.

```
// Button Schliessen wurde geklickt
void CDialogAuftrag::OnBUTTONSchliessen()
{
    // TODO: Code für die Behandlungsroutine der Steuerelement-
    // Benachrichtigung hier einfügen

    // Hinzugefügte Routine:
    // Aufruf der Basisklassen-Funktion EndDialog()
    EndDialog(0);    // Dialog beenden
}
```

Wenn nun im Dialog der Button Schliessen gedrückt wird, wird die Behandlungsfunktion OnBUTTONSchliessen() aufgerufen. In dieser Funktion sorgt der Aufruf EndDialog() dafür, dass der Dialog beendet wird.

## (6) Erstellen von Behandlungsroutinen für Menüpunkte zum Dialogaufruf

Damit ein Dialog beendet werden kann, muss er zunächst einmal aufgerufen werden. Üblicherweise werden Dialoge über Menüpunkte aufgerufen, so dass eine weitere Behandlungsroutine gebraucht wird, nämlich eine, die bei einem Klick auf einen entsprechenden Menüpunkt angestoßen wird.

Im Ressourceneditor für Menüs wählt man den gewünschten Menüpunkt aus, rechtsklickt und wählt im Kontextmenü den Punkt Klassenassistent. Im nun geöffneten Fenster des Klassenassistenten steht jetzt die Klasse CMainFrame unter Klassename, vgl. Abb. 5.28.

**Dialogaufruf  
von Menüs**

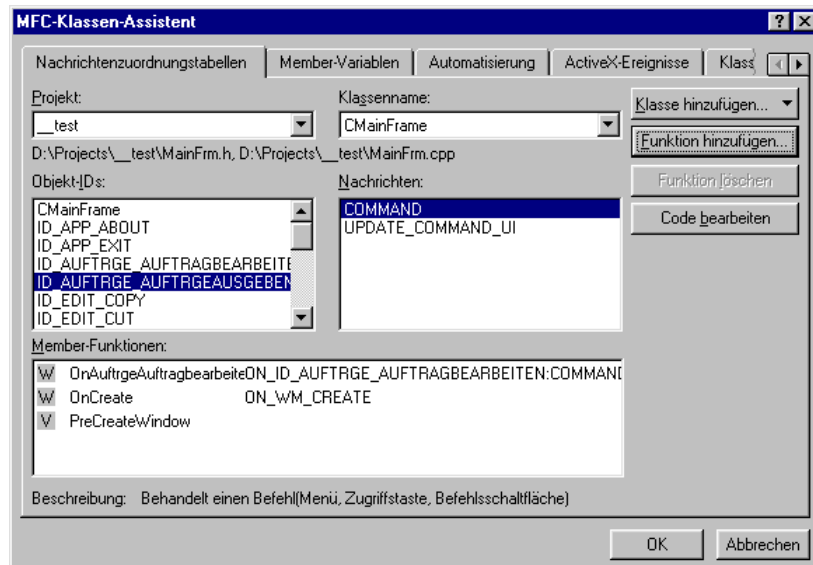


Abb. 5.28. Nachrichtentabelle der Hauptfenster-Behandlungsklasse.

#### Nachrichten- Behandlungs- funktion hinzufügen

Da das Menü zum Hauptfenster gehört, ist die Behandlungsklasse des Hauptfensters für die Nachrichten des Menüs zuständig. In der Liste Objekt-IDs sollte nun die ID des gewünschten Menüpunkts ausgewählt sein. In der nebenstehenden Liste sind zwei Nachrichten eingetragen, wobei die Nachricht COMMAND die hier relevante ist. Wird diese ausgewählt, kann man mit der Schaltfläche „Funktion hinzufügen“ eine entsprechende Behandlungsfunktion zum Quelltext hinzufügen und mit der Schaltfläche „Code bearbeiten“ an die entsprechende Stelle springen.

```
#include "DialogAuftrag.h"
...
// Menüpunkt Aufträge bearbeiten wurde ausgewählt
void CMainFrame::OnAuftrageAuftragbearbeiten()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen

    // Hinzugefügte Routine:
    CDialogAuftrag dialog;          // Dialog-Objekt erzeugen
    dialog.DoModal();              // Dialog aufrufen

    // weitere Anweisungen können folgen
}
```

#### Aufbau Behandlungsroutine

In der Behandlungsroutine wird zunächst ein Dialog-Objekt erzeugt, indem eine Instanz der Dialogklasse erzeugt wird. Damit diese Dialogklasse in der Quelldatei bekannt ist, muss die entsprechende Definitionsdatei eingefügt werden. Ruft man nun die Methode DoModal() der Dialogklasse auf, die sie von der Basisklasse CDialog geerbt hat, wird die Dialogbox angezeigt. Der Ablauf der aufrufenden Methode wird solange angehalten, bis der aufgerufene Dialog beendet wurde. Nach DoModal() können weitere Anweisungen stehen, die aber erst nach Beendigung des Dialogs ausgeführt werden.

## (7) Erstellen von Behandlungsroutinen für Buttons zum Aufruf und Beenden von Dialogen

### a) Aufruf eines weiteren Dialogs

Dialoge können verschachtelt werden. Ein Dialog kann also auch weitere Dialoge aufrufen. Eine entsprechende Behandlungsroutine sieht analog zu der eben gezeigten aus. Wenn z. B. der Button Neu in der Dialogbox für Aufträge einen weiteren Dialog zur Eingabe einer Auftragsposition aufrufen soll, fügt man eine entsprechende Behandlungsroutine hinzu. Dazu rechtsklickt man im Ressourceneditor für Dialoge auf den entsprechenden Button und wählt im Kontextmenü den Punkt Klassenassistent. Im Fenster des Klassenassistenten sollte die korrekte Klasse (CDialogAuftrag) gewählt und die gewünschte Objekt-ID (ID\_BUTTON\_NEU) markiert sein. Markiert man nun die Nachricht BN\_CLICKED und fügt mit der Schaltfläche „Funktion hinzufügen“ eine Behandlungsfunktion hinzu, kann man die Routine für den Aufruf des nachfolgenden Dialogs analog aufbauen.

**Behandlungsroutine  
für weiteren  
Dialogaufruf**

```
#include "DialogAuftragsposition.h"
...
// Button Neu wurde geklickt
void CDialogAuftrag::OnBUTTONNeu()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen

    // Hinzugefügte Routine:
    CDialogAuftragsposition dialog;    // Dialog-Objekt erzeugen
    dialog.DoModal();                 // Dialog aufrufen

    // weitere Anweisungen können folgen
}
```

### b) Beenden eines Dialogs und Datenaustausch mit einem Dialog

Häufig ist es notwendig, dass zwischen dem aufgerufenen Dialog und der aufrufenden Methode Daten ausgetauscht werden müssen. Meist beschränkt sich der Austausch auf die Information, wie der aufgerufene Dialog beendet wurde, also ob er z. B. komplett abgearbeitet und mit „OK“ beendet wurde, oder ob der Benutzer den Dialog mit „Abbrechen“ beendet hat.

**Datenrückgabe am  
Ende eines Dialogs**

Die Funktion EndDialog(), die das Beenden des Dialogs bewirkt, hat einen frei wählbaren Parameter des Typs int. Der Wert dieses Parameters wird zum Rückgabewert von DoModal(), der Funktion für den Aufruf des Dialogs. So kann die den Dialog aufrufende Methode – CDialogAuftrag::OnBUTTONNeu() im obigen Beispiel – dem Rückgabewert entsprechend reagieren. Es ist sinnvoll, als Rückgabewert die ID des beendenden Buttons zu wählen. So hat man eindeutig definierte Werte und braucht nicht neue Konstanten zu definieren.

Als Beispiel wird hier ein Dialog betrachtet, der dem Benutzer die Frage stellt, ob ein Einzel- oder Dauerauftrag zu erstellen ist. Auf der Dialogbox sind drei Buttons vorhanden, die alle den Dialog beenden, aber verschiedene Auswirkungen auf die aufrufende Methode haben, vgl. Abb. 5.29. In den entsprechenden Behandlungsroutinen für diese Buttons wird jedes Mal ein anderer Parameter für die Funktion EndDialog() bestimmt, in diesem Fall die IDs der entsprechenden Buttons.

**Beispiel**

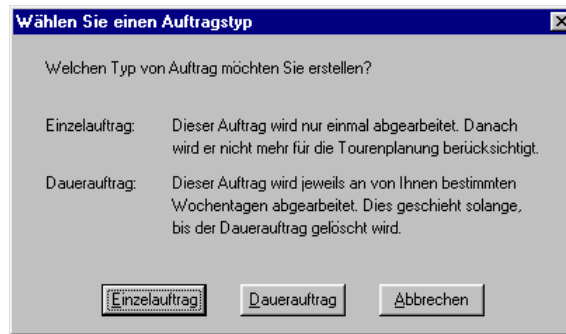


Abb. 5.29. Dialog „Auftragstyp wählen“.

#### verschiedene Rückgabewerte

Die Behandlungsroutine für den Button Einzelauftrag beendet den Dialog mit dem Parameter ID\_BUTTON\_Einzel.

```
// Button Einzelauftrag wurde geklickt:
void CDialogAuftragstyp::OnBUTTON_Einzelauftrag()
{
    // TODO: Code für die Behandlungsroutine der Steuerelement-
    // Benachrichtigung hier einfügen

    // Hinzugefügte Routine:
    EndDialog(ID_BUTTON_Einzel); // Dialog beenden, ID_BUTTON_Einzel als
                                // Rückgabewert von DoModal() bestimmen
}
```

Die Behandlungsroutine für den Button Dauerauftrag ist analog, erhält aber den Parameter ID\_BUTTON\_Dauer für die Funktion EndDialog(). Der Button Abbrechen würde als Parameter den (schon vordefinierten) Wert IDCANCEL übergeben.

```
// Button Dauerauftrag wurde geklickt
void CDialogAuftragstyp::OnBUTTON_Dauerauftrag()
{
    // TODO: Code für die Behandlungsroutine der Steuerelement-
    // Benachrichtigung hier einfügen

    // Hinzugefügte Routine
    EndDialog(ID_BUTTON_Dauer); // Dialog beenden, ID_BUTTON_Dauer als
                                // Rückgabewert von DoModal() bestimmen
}
```

#### Verarbeitung von Rückgabewerten

Nimmt man nun an, dass auf einem Dialog, der alle Aufträge auflistet, ein Button Neu existiert, muss bei einem Klick auf diesen der Benutzer gefragt werden, welcher Typ von Auftrag erstellt werden soll. Danach kann ein entsprechendes Auftrag-Objekt erzeugt werden und der Dialog für die Bearbeitung des Auftrages aufgerufen werden. Die Behandlungsroutine für den Button Neu würde in etwa wie folgt aussehen:

```
// Button Neu wurde geklickt
void CDialogAuftragsliste::OnBUTTON_Neu()
{
    // Benutzer fragen, welcher Typ von Auftrag zu erstellen ist
    CDialogAuftragstyp TypDialog; // Dialog-Objekt erzeugen
    int resultat = TypDialog.DoModal(); // Dialog aufrufen, Rückgabewert
                                      // ermitteln

    // Entsprechend der Wahl des Benutzers den richtigen Auftragstyp
    // erstellen
    Auftrag* pAuftrag = NULL;
    switch (resultat)
```

```

{
    case IDCANCEL: // Keinen Auftrag erstellen und Funktion beenden
        return;
    case ID_BUTTON_Einzel: // Einzelauftrag erstellen
        pAuftrag = new Einzelauftrag;
        break;
    case ID_BUTTON_Dauer: // Dauerauftrag erstellen
        pAuftrag = new Dauerauftrag;
        break;
}

// Initialisierung des Auftrags und Aufruf des Dialogs
// "Auftrag bearbeiten"
...
}

```

### c) Standardimplementierung der Dialoge

Schaut man sich den Quelltext für eine Dialogklasse an, so umfasst er nur wenige Behandlungsroutinen. Die Hauptarbeit, die eine Dialogklasse zu erfüllen hat, wurde in der Basisklasse CDialog implementiert. So übernimmt diese zum Beispiel das Zeichnen der Dialogbox oder das Springen durch die Elemente gemäß der Tabulatorreihenfolge. Existieren auf der Dialogbox Schaltflächen mit vordefinierten IDs, wie z. B. IDOK, IDCANCEL, IDYES oder IDNO, wird eine Standard-Behandlungsroutine aufgerufen, die die Dialogbox schließt und die entsprechende ID als Rückgabewert von DoModal() an die aufrufende Behandlungsfunktion übergibt.

### (8) Initialisierung von Dialogelementen

Manchmal steht schon bei einem Prototyp der genaue Wertebereich oder Inhalt eines Dialogelements fest. So können z. B. für einen Spin-Button Minimal- und Maximalwert festgelegt und Einträge in Listen- und Kombinationsfelder aufgenommen werden. Dialogelemente sollten daher ggf. initialisiert werden.

**wann  
initialisieren?**

#### a) Allgemeine Vorgehensweise

Für die Initialisierung der Dialogboxelemente ist die Methode OnInitDialog() der Dialogklasse zuständig, die von der Basisklasse CDialog geerbt wird. Um eigene Initialisierungen hinzufügen zu können, muss diese Methode in der eigenen Dialogklasse überschrieben werden. In dieser Methode werden dann Initialisierungsmethoden der Behandlungsklassen der Dialogelemente aufgerufen.

**Redefinition von  
OnInitDialog()**

Da in der Dialogklasse eine spezielle Methode für die Initialisierung der Elemente existiert, ist es nicht notwendig, eigene Behandlungsklassen für die Dialogelemente abzuleiten und in diesen spezielle Initialisierungen zu implementieren.

Den Rahmencode für die eigene Überschreibung erzeugt der Klassenassistent. Wenn man im Ressourceneditor die Dialogbox selbst (nicht ein Dialogelement) markiert und den Klassenassistenten aufruft (Menüpunkt Ansicht | Klassenassistent oder Strg-W) ist als Objekt-ID der Klassenname markiert. In der Liste mit den Nachrichten markiert man nun WM\_INITDIALOG und fügt eine entsprechende Funktion hinzu, vgl. Abb. 5.30.

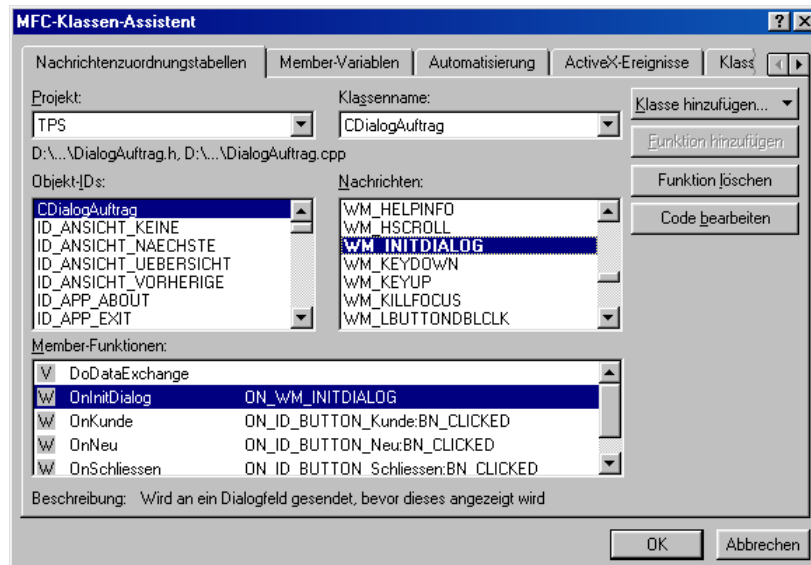


Abb. 5.30. Nachrichtentabelle einer Dialogklasse.

Diesmal wird nicht nach einem Funktionsnamen gefragt, da für eine Überschreibung der Name bereits festgelegt ist.

```

BOOL CDialogAuftrag::OnInitDialog()
{
    // Aufruf der Basisklassen-Methode
    CDialog::OnInitDialog();

    // TODO: Zusätzliche Initialisierung hier einfügen

    // Zusätzliche Initialisierung:
    ...

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX-Eigenschaftenseiten sollten FALSE
                // zurückgeben
}

```

Nach dem Aufruf der Basisklassen-Methode kann die eigene Initialisierungs-Routine eingefügt werden.

## b) Behandlungsklassen für Dialogelemente

Behandlungs-  
klassen von  
Dialogelementen ...

Ähnlich wie eine Dialogklasse für die Steuerung einer Dialogbox zuständig ist, wird jedes Dialogelement von einer Behandlungsklasse gesteuert. Dabei werden Elemente gleichen Typs jeweils von einer eigenen Instanz der Behandlungsklasse kontrolliert.

... und Initialisierung  
mit Zeigern

Um die Dialogelemente initialisieren zu können, muss zuerst ein Zeiger auf die entsprechende Behandlungsklasse ermittelt werden. Die Funktion `GetDlgItem()` erwartet als Parameter die ID des Dialogelements und liefert einen Zeiger vom Typ `CWnd*` auf die gesuchte Klasse zurück. Da alle Behandlungsklassen von `CWnd` abgeleitet sind, kann man den Rückgabewert mit dem `dynamic_cast`-Operator in einen Zeiger des passenden Typs umwandeln.

Häufig spricht man davon, einen Zeiger auf ein Dialogelement zu ermitteln, obwohl in Wirklichkeit ein Zeiger auf eine Instanz der Behandlungsklasse ermittelt wird.

```

BOOL CDialogAuftrag::OnInitDialog()
{
    // Aufruf der Basisklassen-Methode
    CDialog::OnInitDialog();

    // Zeiger auf den Button Schliessen ermitteln
    // (in Wirklichkeit einen Zeiger auf eine Instanz der
    // Behandlungsklasse ermitteln)
    CWnd* pWnd = GetDlgItem(ID_BUTTON_Schliessen);

    // Zeiger in den passenden Typ umwandeln
    CButton* pButtonSchliessen = dynamic_cast<CButton*>(pWnd);

    // Initialisierung des Buttons Schliessen
    ...

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX-Eigenschaftenseiten sollten FALSE
                // zurückgeben
}

```

Nachdem man den Zeiger ermittelt und in den passenden Typ umgewandelt hat, kann man mit den speziellen Methoden der Behandlungsklasse das Dialogelement initialisieren.

### c) Allgemeine Methoden zur Initialisierung von Dialogelementen

Da alle Behandlungsklassen von Dialogelementen von der Klasse CWnd abgeleitet sind, stehen bestimmte Methoden allen Klassen zur Verfügung.

**Initialisierung mit  
allgemeinen  
Methoden**

Wenn ein Dialogelement vom Benutzer nicht angewählt werden darf, sollte man das Element deaktivieren. Ein deaktiviertes Element wird grau dargestellt und akzeptiert keine Eingaben des Benutzers. So kann z. B. das Eingabefeld für einen Geburtstag erst aktiviert werden, wenn ein Name eingegeben wurde.

```

// CWnd::EnableWindow(BOOL enable = TRUE)
//
// enable: Wenn TRUE, wird das Element aktiviert, wenn FALSE wird es
// deaktiviert

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Schließen-Button deaktivieren
    GetDlgItem(ID_BUTTON_Schliessen)->EnableWindow(FALSE);
    ...
}

```

Man kann Dialogelemente unsichtbar machen. Wenn z. B. ein Dialogelement nur während der Testphase benötigt wird, sollte es im Prototyp versteckt werden.

```

// CWnd::ShowWindow(int nCmdShow)
//
// nCmdShow: SW_HIDE, Dialogelement verstecken
//           SW_SHOW, Dialogelement anzeigen

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Löschen-Button verstecken
    GetDlgItem(ID_BUTTON_Löschen)->ShowWindow(SW_HIDE);
    ...
}

```



Bei Dialogelementen mit einer Beschriftung wie z. B. Buttons, Checkboxes oder Radio-Buttons kann diese geändert werden. Dabei kann mit dem &-Zeichen ein Hot-Key bestimmt werden. Bei Eingabefeldern kann mit dieser Methode der Text in das Feld geschrieben werden.

```
// CWnd::SetWindowText(LPCTSTR lpszString)
//
// lpszString: Zeichenkette mit der neuen Beschriftung

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Beschriftung des Buttons von "Speichern" in "Übernehmen" ändern
    GetDlgItem(ID_BUTTON_Speichern)->SetWindowText("Über&nehmen");
    ...
}
```

#### d) Initialisierung von Radio-Buttons und Checkboxes

##### Radio-Buttons und Checkboxes

Dialogelemente dieser Typen werden wie der normale Button von der Behandlungsklasse CButton gesteuert. Bei der Initialisierung können mehrere Checkboxes und jeweils ein Radio-Button einer Gruppe ausgewählt werden.

```
// CButton::SetCheck(int nCheck)
//
// nCheck: 0, nicht auswählen
//         1, auswählen

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Radiobutton Einzelauftrag auswählen
    CButton* pButton =
        dynamic_cast<CButton*>(GetDlgItem(IDC_RADIO_Einzelauftrag));
    pButton->SetCheck(1);
    ...
}
```

#### e) Initialisierung von Listen- und Kombinationsfeldern

##### Listen- und Kombinationsfelder

Für Listenfelder ist die Klasse CListBox, für Kombinationsfelder die Klasse CComboBox zuständig. Beide Klassen besitzen die Methode AddString(), mit der eine Zeichenkette in die Liste eingetragen werden kann.

```
// CListBox::AddString(LPCTSTR lpszString)
// CComboBox::AddString(LPCTSTR lpszString)
//
// lpszString: einzufügende Zeichenkette

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Farben in die ListBox eintragen
    CListBox* pListBox
        = dynamic_cast<CListBox*>(GetDlgItem(IDC_LIST_Farben));
    pListBox->AddString("rot");
    pListBox->AddString("grün");
    pListBox->AddString("blau");
    ...
}
```

## f) Initialisierung von Spin-Buttons

Bei einem Spin-Button kann der Wertebereich, in dem gescrollt werden darf, eingestellt werden.

Spin-Buttons

```
// CSpinButtonCtrl::SetRange(int min, int max)
//
// min: untere Grenze des Wertebereichs
// max: obere Grenze des Wertebereichs

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Wertebereich für den Spinbutton Anzahl auf [0;100] setzen
    CSpinButtonCtrl *pSpinButtonCtrl
        = dynamic_cast<CSpinButtonCtrl*>(GetDlgItem(IDC_SPIN_Anzahl));
    pSpinButtonCtrl->SetRange(0, 100);
    ...
}
```

## g) Initialisierung von Listenelementen

Ist ein Listenelement mit der Ansicht „Bericht“ eingestellt, können zu jedem Eintrag in Spalten Details angezeigt werden. Für das Erstellen von Spalten ist die Methode InsertColumn() der Behandlungsklasse CListCtrl zuständig. Im folgenden Beispiel werden die Spalten eines Listenelements für Auftragspositionen geeignet beschriftet.

Listenelemente

```
// CListCtrl::InsertColumn(int nCol, LPCTSTR lpszColumn, int nFormat =
//                          LVCFMT_LEFT, int nWidth = -1)
//
// nCol:      Index der Spalte
// lpszColumn: Zeichenkette mit dem Spaltennamen
// nFormat:   LVCFMT_LEFT, LVCFMT_RIGHT oder LVCFMT_CENTER
//            Ausrichtung der Spalte
// nWidth:    Breite der Spalte in Pixel

BOOL CDialogAuftrag::OnInitDialog()
{
    ...
    // Spalten in die Positionsliste eintragen
    CListCtrl *pListCtrl
        = dynamic_cast<CListCtrl*>(GetDlgItem(IDC_LIST_AuftrPos));
    pListCtrl->InsertColumn(0, "LfdNr.", LVCFMT_LEFT, 40);
    pListCtrl->InsertColumn(1, "Artikel", LVCFMT_LEFT, 180);
    pListCtrl->InsertColumn(2, "Wert", LVCFMT_RIGHT, 80);
    ...
}
```

## h) Initialisierung weiterer Dialogelemente

Zur Initialisierung der genannten und weiterer Dialogelemente sollte man die Hilfe zur MFC-Bibliothek zu Rate ziehen, da die hier gezeigten Varianten nur einen kleinen Ausschnitt aller Möglichkeiten darstellen.

Grundsätzlich sollte man den Feinschliff des Prototyps nicht zu weit treiben, da bei der späteren Implementierung eventuell andere Wege für die Initialisierung gegangen werden müssen und dann der Aufwand für die Änderungen zu groß wird.

## (9) Entwurf von Ausgaben im Arbeitsbereich des Anwendungsfensters

### Ausgaben im Arbeitsfenster

Wenn im Arbeitsbereich des Anwendungsfensters Ausgaben stattfinden sollen, ist dies meist nicht ohne größeren Programmieraufwand zu bewerkstelligen. Häufig genügt es, im Zuge der Prototyperstellung die Fensterausgaben auf Papier zu skizzieren, um so dem Auftraggeber einen Eindruck der später realisierten Fensterausgaben zu vermitteln. Die Skizzen der Fensterausgaben werden in diesem Fall ein zusätzlicher Bestandteil des Modells der GUI.

## Übungsaufgaben zu Kapitel 5.2

### Übungsaufgabe 5.2.1 (Projekt Bibliotheksausleihsystem)

Erstellen Sie einen GUI-Prototyp des Bibliotheksausleihsystems. Verwenden Sie als Grundlage die Musterlösung der Übungsaufgabe 4.4.1, d.h. das BAS-OOA-Modell auf der CD-ROM.

Sie sollten möglichst einen vollständigen GUI-Prototyp erstellen. Wenn sich dies als zu aufwendig erweist, erstellen Sie eine reduzierte Version des Prototyps! Dabei können Sie Vormerkungen und Bestellungen vernachlässigen, während Ausleihen auf jeden Fall berücksichtigt werden sollten. Diese Einschränkung reduziert die Anforderungen an die Kontoführung eines Benutzers wesentlich. Eine Reservierungsfrist- und Ausleihfristprüfung können ebenfalls entfallen. Ferner können Sie die Gliederung der Benutzer in Studenten und Universitätsmitarbeiter vernachlässigen. Dagegen sollte die Unterscheidung zwischen Büchern und Zeitschriften erhalten bleiben. Orientieren Sie sich bei der Programmierung des BAS-Prototyps an dem Prototyp für das Tourenplanungssystem!

## Lösungen zu den Übungsaufgaben

### Lösung zu Übungsaufgabe 3.1.1

(Aufgabentext S. 7)

Informelle Gespräche mit Mitarbeitern der Auftraggeberseite, die erste Eindrücke vom Anwendungsbereich vermitteln, werden Entwickler sofort führen. Interviews und erst recht Fragebogenaktionen setzen bereits ein gewisses Maß an Kenntnissen über den Anwendungsbereich voraus. Diese können etwa durch das Studium einiger grundlegender Dokumente oder einschlägiger Fachliteratur erworben werden. Das Dokumenten- und Literaturstudium unterstützt oder ermöglicht erst eine gezielte und gehaltvolle Kommunikation mit den Mitarbeitern auf der Auftraggeberseite. Andererseits sollte sich das Dokumentenstudium anfangs eher auf allgemeine Aspekte konzentrieren, da sonst die Gefahr besteht, sich in Details zu verlieren. Dieser Gefahr kann durch Gespräche und Interviews vorgebeugt werden. Eine längere unmittelbare Beobachtung des Anwendungsbereiches sollte wiederum erst dann erfolgen, wenn bereits ein hinreichendes Maß an Vorwissen vorhanden ist, um Eindrücke und Erfahrungen richtig interpretieren und einordnen zu können. Die Pflege eines Glossars ist eine iterative Tätigkeit, mit der sofort begonnen werden kann.

### Lösung zu Übungsaufgabe 3.2.1

(Aufgabentext S. 13)

Nicht berücksichtigt wurden die Sonderfälle:

- Kunde gibt Kfz nach Unfall (einschließlich Totalschaden) nicht zurück und
- Kfz-Verlust wegen Diebstahl.

### Lösung zu Übungsaufgabe 3.2.2

(Aufgabentext S. 13)

**Anwendungsfall:** Kfz-reservieren

**Ziel:** Kfz für Kunden reservieren.

**Akteure:** Mitarbeiter Kfz-Vermietung.

**Kategorie:** Primär.

**Initiierendes Ereignis:** Kunde erscheint oder ruft an und möchte Kfz reservieren.

**Vorbedingung:** –

**Nachbedingung bei Erfolg:** Gewünschtes Kfz reserviert.

**Nachbedingung bei Misserfolg:** Keine Reservierung.

**Standardspezifikation:**

- 1 Kfz-Reservierungsdaten erfassen (Zeitraum, Fahrzeugtyp, Ausstattung).
- 2 prüfen, ob Reservierung möglich ist.
- 3 Kunde anhand Kundennummer identifizieren.
- 4 Reservierung einschließlich Reservierungsnummer abspeichern.
- 5 Kunde über Reservierung und Reservierungsnummer informieren.

**Erweiterungen der Standardspezifikation:**

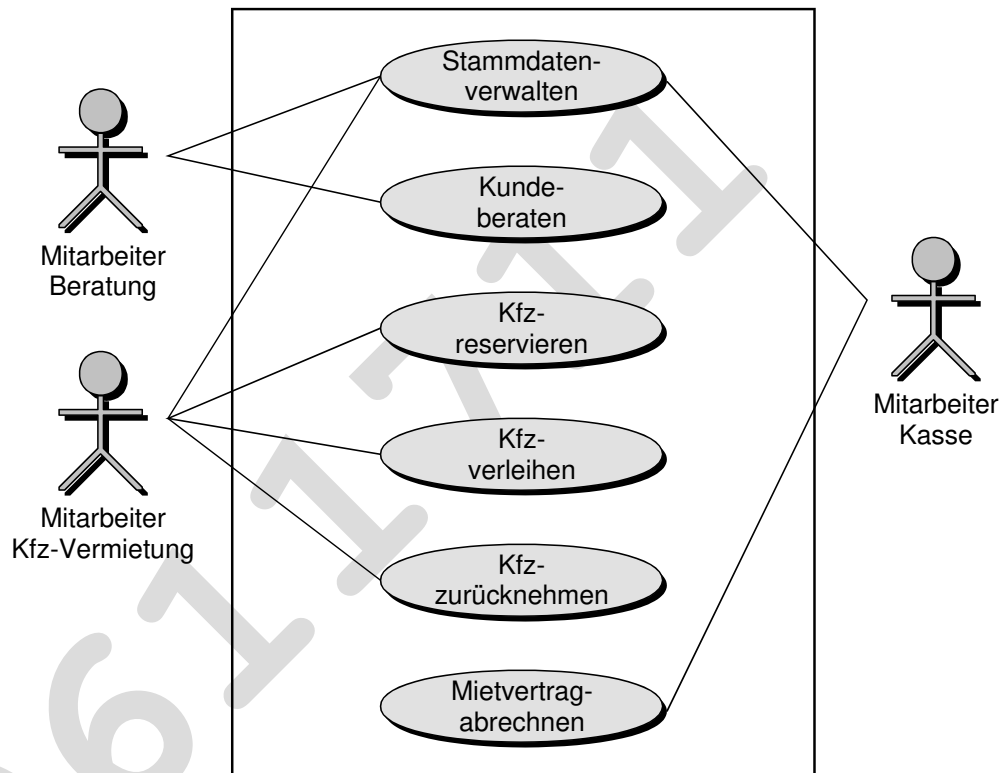
- 2a Kunde über nicht mögliche Reservierung informieren und Ersatzreservierung vorschlagen; falls diese nicht akzeptiert wird, Abbruch der Reservierung.

**Alternativen zur Standardspezifikation:**

- 3a Neuen Kunden erfassen, Kundennummer vergeben und dem Kunden mitteilen; Kundenformular einschl. Kunden-Nr. für spätere Versendung ausdrucken.

**Lösung zu Übungsaufgabe 3.2.3**

(Aufgabentext S. 14)



Da bei der Rückgabe eines Kfz zwar eine Rechnung erstellt, diese jedoch innerhalb des Anwendungsfalls „Kfz-zurücknehmen“ nicht bezahlt wird, wird ein zusätzlicher Anwendungsfall für diesen Zweck vorgesehen, der auch das Mahnwesen umfasst. Ein Zugriff auf die Stammdaten ist auch bei der Kundenberatung sinnvoll, um z.B. über die Daten eines bereits bekannten Kunden auf frühere Verträge zugreifen und seine Interessenlage einschätzen zu können.

**Lösung zu Übungsaufgabe 3.2.4**

(Aufgabentext S. 14)

Der Vertrag muss durch den Kunden persönlich unterschrieben werden, der also bei dem hier betrachteten Anwendungsfalldiagramm auch als einzige Person das Fahrzeug übernehmen kann. Soll auch eine zweite Person das Fahrzeug übernehmen können, so ist der Anwendungsfall „Kfz-verleihen“ geeignet zu zerlegen. Hierbei können etwa die Anwendungsfälle „Mietvertrag abschliessen“ und „Kfz-uebergeben“ vorgesehen werden. Das Fahrzeug kann in diesem Fall an eine beliebige Person übergeben werden, die den Vertrag vorweist. Seine Prüfung ist dann Gegenstand der Systembenutzung im Anwendungsfall „Kfz-uebergeben“.

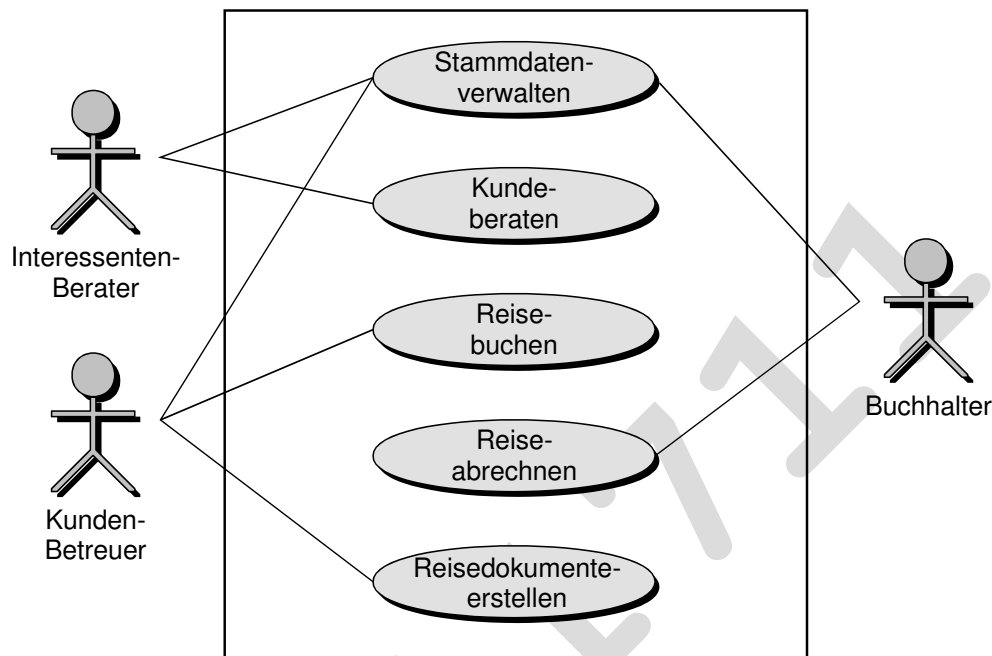
**Lösung zu Übungsaufgabe 3.2.5**

(Aufgabentext S. 14)

Typische Beispiele für Anwendungssysteme, bei denen Kunden selbst Akteure sind, stellen Auskunfts- und Bestellsysteme von Versandhäusern oder allgemeiner Internetanwendungen des E-Commerce dar.

**Lösung zu Übungsaufgabe 3.2.6**

(Aufgabentext S. 14)



Sind die Teilvorgänge des Buchens der Hin- und Rückbeförderung sowie der Unterbringung hinreichend komplex, ist auch eine entsprechende Aufspaltung des Anwendungsfalls „Reise-buchen“ in Betracht zu ziehen.

**Lösung zu Übungsaufgabe 3.2.7**

(Aufgabentext S. 14)

**Anwendungsfall:** Reisedokumente erstellen

**Ziel:** Aushändigung Reisedokumente an Kunden.

**Akteure:** Kundenbetreuer.

**Kategorie:** Primär.

**Initiierendes Ereignis:** Kunde erscheint und möchte Reisedokumente abholen.

**Vorbedingung:** Reise gebucht und abgerechnet.

**Nachbedingung bei Erfolg:** Reisedokumente erstellt und übergeben.

**Nachbedingung bei Misserfolg:** Reisedokumente nicht erstellt.

**Standardspezifikation:**

- 1 Kundendaten, Buchungsdaten und Abrechnungsquittung prüfen.
- 2 Reisedokumente Hin- und Rückbeförderung erstellen und ausdrucken.
- 3 Reisedokumente Unterbringung erstellen und ausdrucken.
- 4 Reisedokumente übergeben und Übergabe im System protokollieren.

**Erweiterungen der Standardspezifikation:**

- 1a Bei intern verursachten Datenfehlern Rücksprache mit zuständigem Mitarbeiter halten und Dateninkonsistenzen beseitigen.

**Alternativen zur Standardspezifikation:**

- 2a: Kunden informieren, dass Reise von ihm nicht gebucht bzw. abgerechnet wurde; Schritte 2 bis 4 entfallen.

**Lösung zu Übungsaufgabe 3.3.1**

(Aufgabentext S. 17)

Die Ist-Analyse eines vorhandenen Systems hilft zum einen bei der detaillierten Aufdeckung von bisherigen Schwachstellen, die in einem Nachfolgersystem abgestellt werden sollten. Ferner können Entwicklungsdokumente oder implementierte Systemkomponenten des alten Systems eventuell übernommen oder zumindest teilweise bei der Erstellung des neuen Systems genutzt werden.

**Lösung zu Übungsaufgabe 3.4.1 (BAS)**

(Aufgabentext S. 24)

Siehe kursbegleitende CD-ROM, Ordner ProjektBAS\OOA.

**Lösung zu Übungsaufgabe 3.5.1**

(Aufgabentext S. 28)

Bei der Dialogverarbeitung wechseln Eingaben eines Benutzers und Ausgaben des Systems einander ab. Spätere Eingaben hängen dabei von zuvor vom System erzeugten Ausgabedaten ab. Bei der Batchverarbeitung wird ein kompletter Auftrag mit allen zugehörigen Eingabedaten an das System übergeben. Während der angestoßenen Verarbeitung greift der Benutzer nicht mehr ein. Ausgaben werden z.B. als Drucklisten erzeugt oder auf einem Sekundärspeicher abgelegt. Die Batchverarbeitung wird meist für die Verarbeitung großer Datenbestände etwa bei der Erstellung von Auswertungen oder für komplizierte und langwierige Berechnungen eingesetzt. Dies geschieht oft nachts und wird durch eine Systemuhr gesteuert. Batchverarbeitungen lassen sich daher u.a. durch die Frage nach zeitlichen Ereignissen identifizieren, die die Ausführung von Systemfunktionen anstoßen.

**Lösung zu Übungsaufgabe 3.5.2**

(Aufgabentext S. 28)

Bei der Zusammensetzung von Datenstrukturen aus Datenelementen können neben einer einfachen additiven Aufeinanderfolge verschiedener Datenelemente vor allem folgende Varianten vorkommen:

- Wiederholung von Datenelementen, z.B. bei mehreren Adressen eines Kunden,
- optionale Datenelemente, z.B. optionaler Ansprechpartner eines Kunden,
- alternative Datenelemente, z.B. entweder Straßen- oder Großkundenadresse.

Für diese Varianten werden bei der Datenflussmodellierung (vgl. Kap. 2.4.1) geeignete Notationen eingeführt; dies geschieht hier nicht, weil eine eingehendere Datenanalyse erst innerhalb der fachlichen Modellierung vorgenommen wird.

**Lösung zu Übungsaufgabe 3.6.1 (BAS)**

(Aufgabentext S. 38)

Siehe kursbegleitende CD-ROM, Ordner ProjektBAS\OOA.



### Lösung zu Übungsaufgabe 3.7.1

(Aufgabentext S. 40)

- Operatives Dispositionssystem: Verschnittminimierung etwa beim Zuschnitt von Stangen, Papierrollen oder Blechen.
- Planungssystem: Standortoptimierung von Betrieben und Lagern, um längerfristig günstige Standorte bezogen auf bekannte Entfernungen und Materialflüsse festzulegen, die zu einem möglichst geringen Transportaufkommen führen.

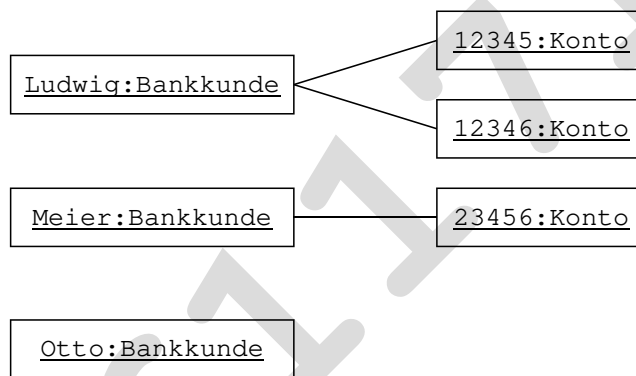
### Lösung zu Übungsaufgabe 3.8.1

(Aufgabentext S. 48)

Im Allgemeinen wird erst eine explizite Modellierung alle Details eines Entscheidungsproblems erfassen. Dies stellt eine notwendige Vorbedingung der Auswahl und Spezifizierung des Lösungsverfahrens dar. Insbesondere trägt eine Modellierung dazu bei, dass Nebenbedingungen (Restriktionen) des Problems nicht übersehen werden.

### Lösung zu Übungsaufgabe 4.1.1

(Aufgabentext S. 77)



### Lösung zu Übungsaufgabe 4.1.2

(Aufgabentext S. 77)

Sparkonto
KontoNr: UInt {key, readonly, KontoNr > 0}
Bankleitzahl: UInt {readonly}
/Bankname: String {optional}
Typ: TSparkonto = Standard {readonly};
TSparkonto = enum (Student, Standard, Exklusiv)
Kontostand: Float {Euro, Kontostand ≥ 0.0}
DatumEroeffnung: Date
BuchungsZeitpunkte: list of Date
BuchungsBetraege: list of Float
AnzahlSparkonten: UInt

### Lösung zu Übungsaufgabe 4.1.3

(Aufgabentext S. 77)

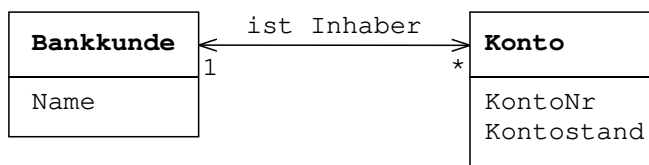
buchen(in Buchungstyp : Boolean, in Betrag : Float, out Kontostand : Float)  
: Boolean {Betrag > 0.00, Kontostand ≥ 0.00}

Der Buchungstyp besitzt für Einzahlungen den Wert true, für Auszahlungen den Wert false.

Der Rückgabewert signalisiert eine erfolgreiche (true) bzw. erfolglose (false) Buchung.

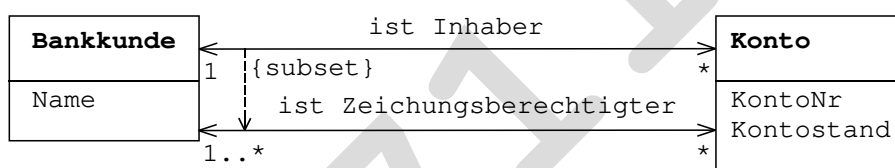
#### Lösung zu Übungsaufgabe 4.1.4

(Aufgabentext S. 77)



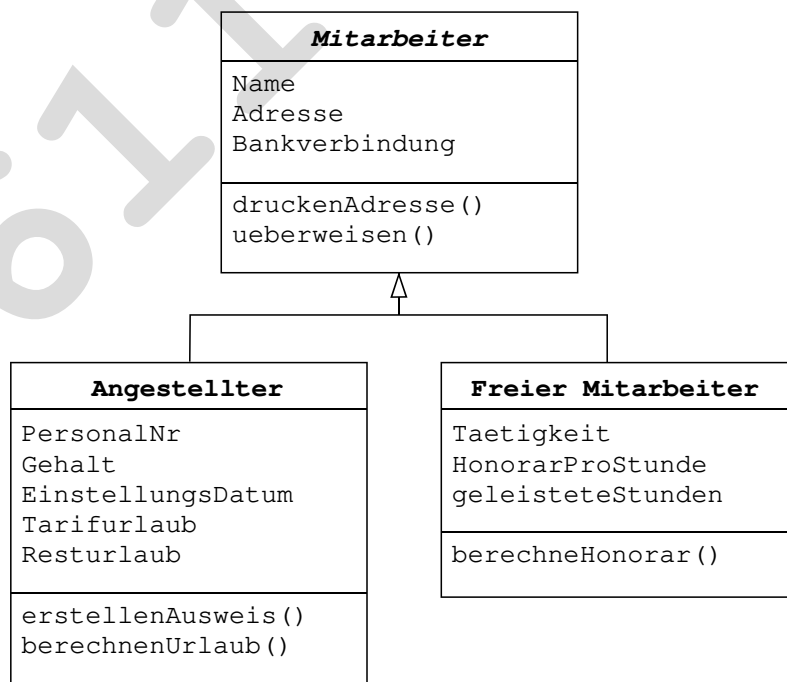
#### Lösung zu Übungsaufgabe 4.1.5

(Aufgabentext S. 77)



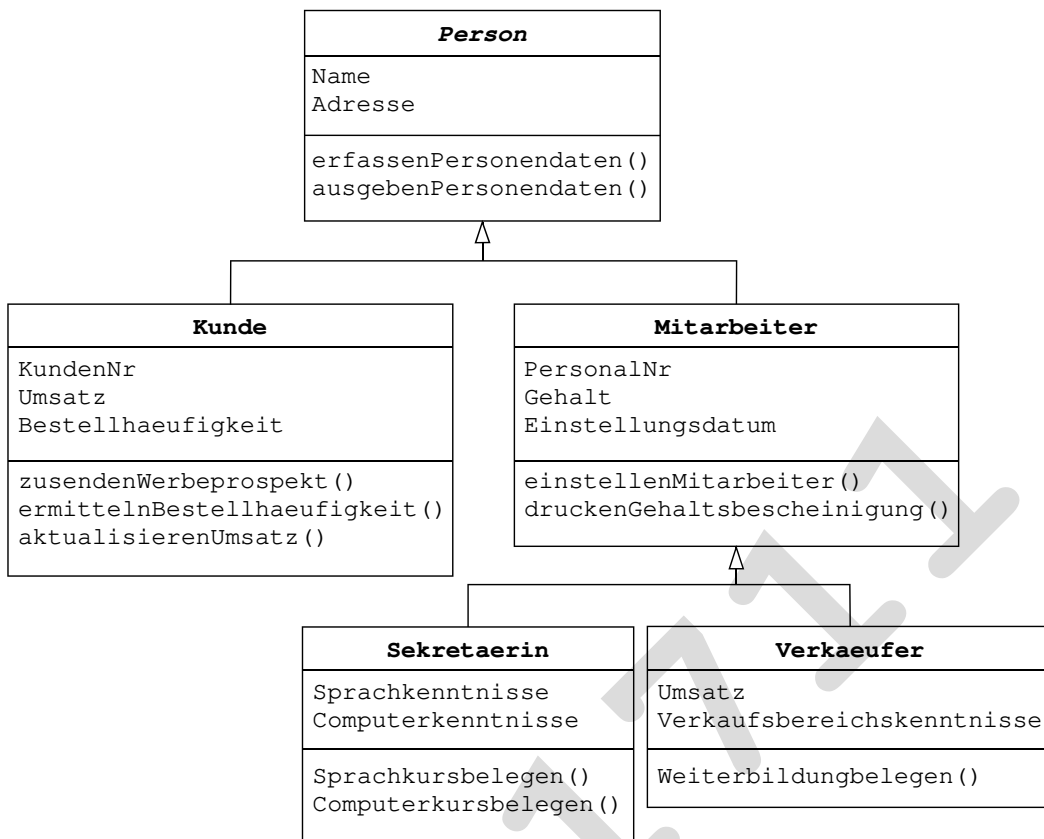
#### Lösung zu Übungsaufgabe 4.1.6

(Aufgabentext S. 78)



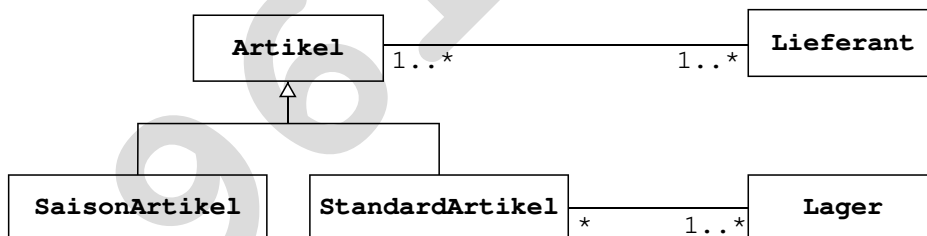
### Lösung zu Übungsaufgabe 4.1.7

(Aufgabentext S. 78)



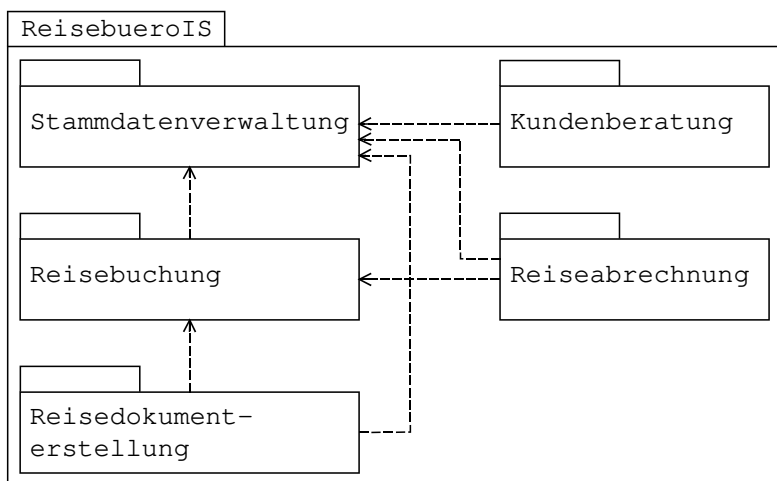
### Lösung zu Übungsaufgabe 4.1.8

(Aufgabentext S. 78)



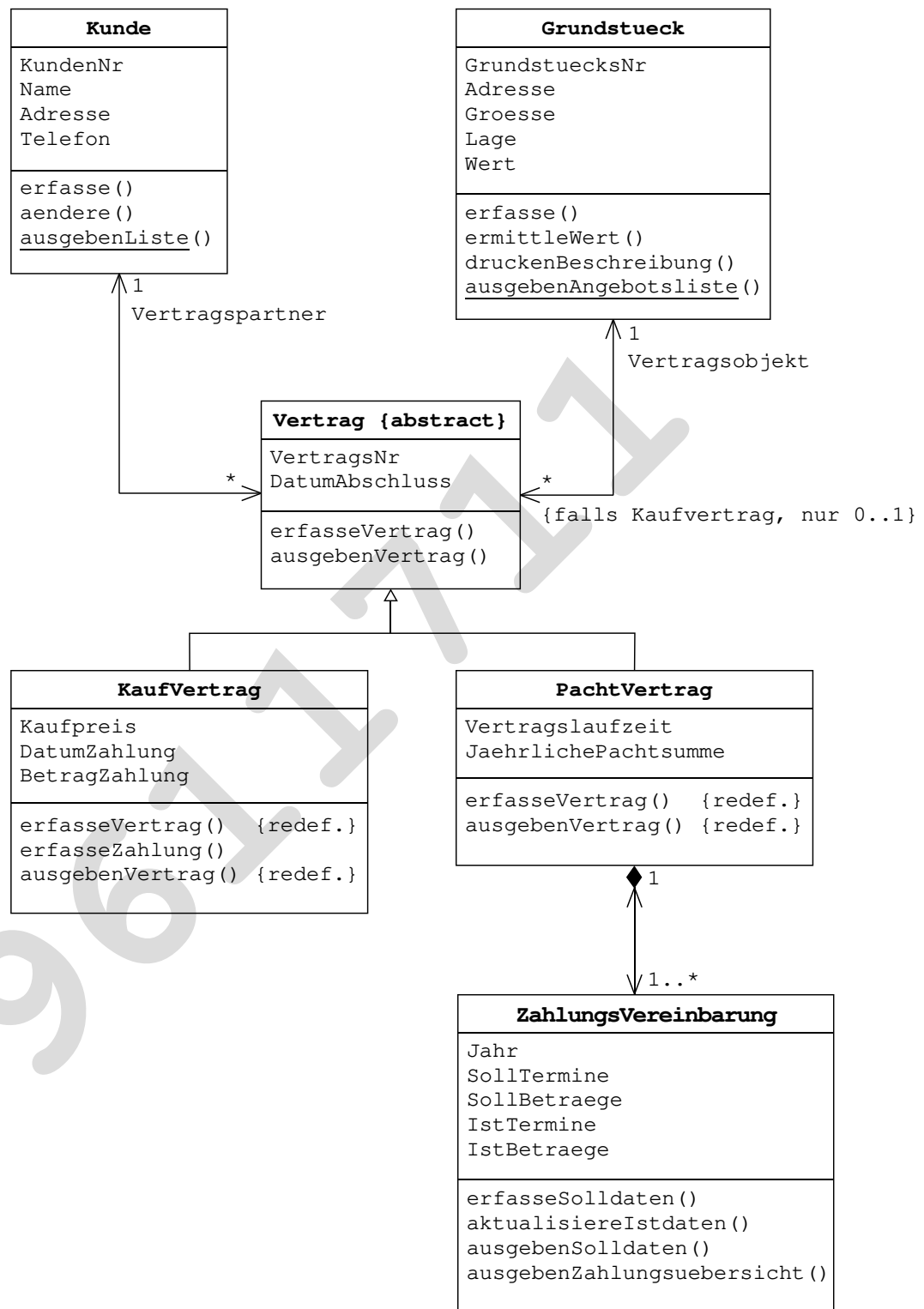
### Lösung zu Übungsaufgabe 4.1.9

(Aufgabentext S. 78)



## Lösung zu Übungsaufgabe 4.1.10

(Aufgabentext S. 79)



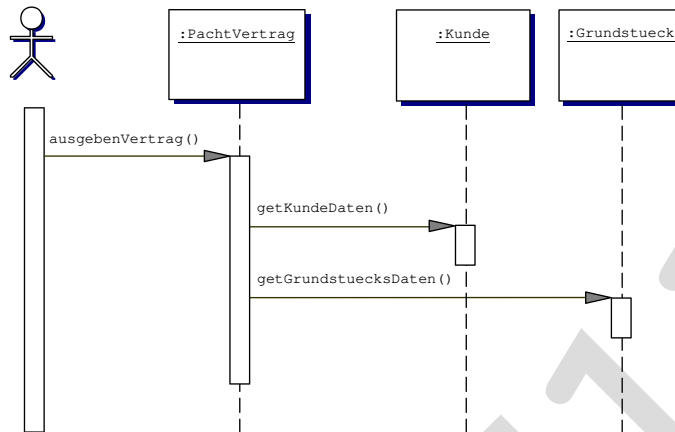
## Erläuterung:

- Die Verbindung eines Vertrags mit dem zugehörigen Kunden und Grundstück wird durch Assoziationen, nicht etwa durch Fremdschlüssel gewährleistet.
- Pro Pachtvertrag und Jahr existiert eine Zahlungsvereinbarung. Da es pro Jahr mehrere Zahlungstermine gibt, sind Attribute eines Listentyps mit jeweils mehreren Werten vorhanden; SollBetraege besitzt z.B. den Typ list of Float.

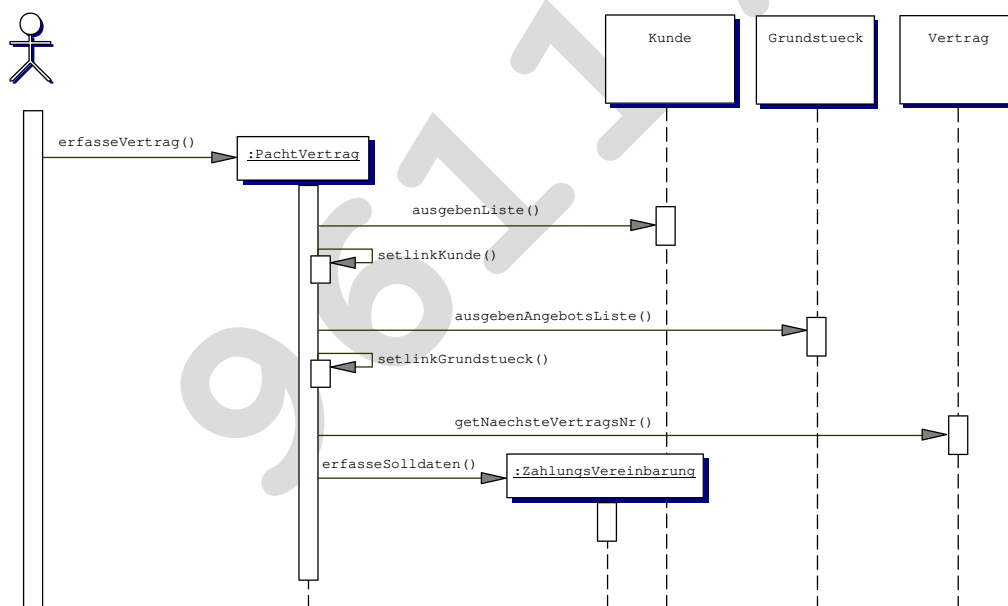
- Für die Zahlungsabwicklung bei einem Grundstückskauf wird kein Objekt der Klasse ZahlungsVereinbarung benötigt.
- Die Operation erfasseSolldaten() ist ein Konstruktor. Er übernimmt zugleich die Initialisierung der Istdaten mit Nullwerten.

**Lösung zu Übungsaufgabe 4.2.1**

(Aufgabentext S. 92)

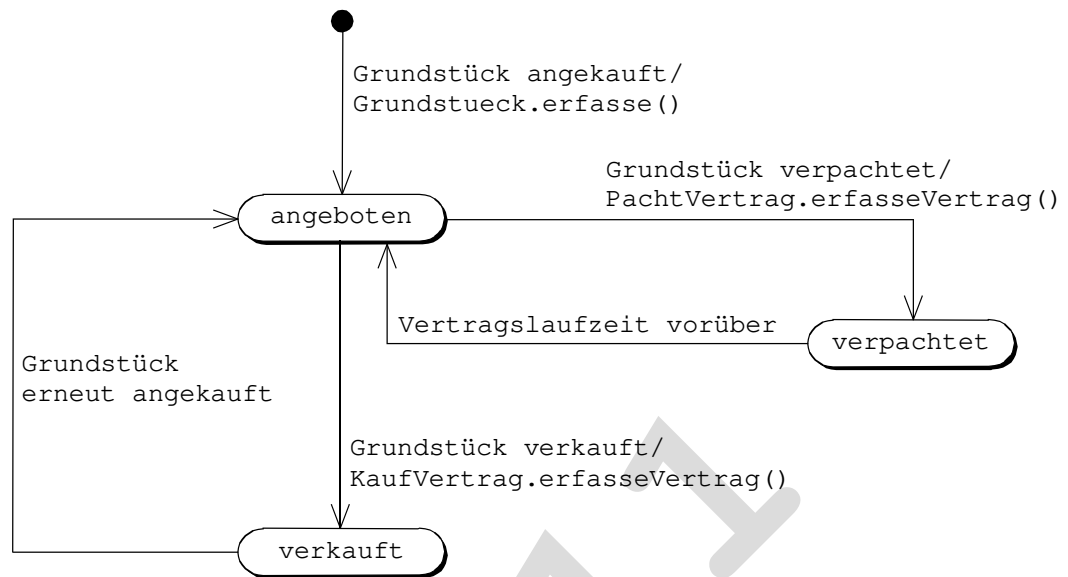
**Lösung zu Übungsaufgabe 4.2.2**

(Aufgabentext S. 92)



**Lösung zu Übungsaufgabe 4.2.3**

(Aufgabentext S. 92)

**Lösung zu Übungsaufgabe 4.3.1**

(Aufgabentext S. 102)

Die getrennte Modellierung von Bibliotheksbenutzern und Benutzerkonten ist gerechtfertigt. Beide Objekttypen besitzen hinreichende Komplexität im Anwendungskontext und sind semantisch verschieden. Eine Kurzbeschreibung eines Artikels, die nur einem Artikel zugeordnet ist, sollte dagegen als Attribut der Klasse Artikel erscheinen und nicht als separate Klasse.

**Lösung zu Übungsaufgabe 4.3.2**

(Aufgabentext S. 102)

- Ein Konto ist ein hinreichend komplexes Objekt mit eigenen Daten und eigenem Verhalten und sollte als separate Klasse modelliert werden, die alle angegebenen kontobezogenen Attribute und Operationen umfasst. Bei der angegebenen Klasse Bankkonto kann ein Bankkunde darüber hinaus nur ein Konto besitzen.
- Abgesehen von den unter a) genannten Argumenten ist an der Vererbungsbeziehung vor allem zu bemängeln, dass die vererbte Kontonummer für beide Unterklassen inhaltlich völlig verschiedene Sachverhalte wiedergibt. Im Fall eines Kunden handelt es sich um ein Konto, das die betreffende Bank für ihn führt. Im Fall eines Mitarbeiters handelt es sich um sein Gehaltskonto, das – wie die Attribute des Mitarbeiters anzeigen – im Allgemeinen bei einer anderen Bank geführt wird. Derartige inhaltlich unklare Vererbungsbeziehungen sollten vermieden werden.

**Lösung zu Übungsaufgabe 4.3.3**

(Aufgabentext S. 103)

Die Lieferantenummer stellt einen Fremdschlüssel des Lieferanten eines Artikels dar und ist nicht als Attribut zu modellieren. Die Bestellzeitpunkte charakterisieren nicht einen Artikel an sich, sondern seine Assoziation zu einem Lieferanten. Es liegt daher nahe, eine weitere assoziative Klasse Bestellung einzuführen. Diese wird insbesondere dann erforderlich, wenn es zu einem Artikel mehrere Lieferan-

ten geben kann, da andernfalls keine Zuordnung eines Bestellzeitpunktes zu einem Lieferanten mehr vorgenommen werden kann.

#### **Lösung zu Übungsaufgabe 4.3.4**

(Aufgabentext S. 103)

Die Vererbungsbeziehung entspricht nicht dem grundlegenden Kriterium für eine Vererbung, wonach jedes Objekt einer Unterklasse inhaltlich auch ein Objekt der Oberklasse sein muss. Weder ein Mitarbeiter noch ein Kunde ist eine Adresse. Eine sinnvolle Generalisierung würde auch den Namen und weitere Personenattribute in eine Oberklasse Person stellen.

#### **Lösung zu Übungsaufgabe 4.3.5**

(Aufgabentext S. 104)

Die Pakete sind offenkundig zu klein gewählt und könnten zu einem Paket vereinigt werden, das noch um weitere Klassen ergänzt werden müsste. Anders wäre die Situation, wenn etwa die Klasse Artikel an der Spitze einer umfangreicheren Vererbungshierarchie mit z.B. 8 Klassen stehen würde.

#### **Lösung zu Übungsaufgabe 4.3.6**

(Aufgabentext S. 104)

Zunächst fällt auf, dass die angegebene Klassifizierung unvollständig ist, weil Flüge mit Distanzen zwischen 600 und 2000 km nicht berücksichtigt wurden. Wesentlicher ist jedoch, dass die Unterschiede bezüglich der Distanz und der Zwischenstops keine Vererbung rechtfertigen. Die Distanz ist lediglich ein gemeinsames Merkmal beider Flugtypen mit unterschiedlichem Wertebereich pro Typ. Die Zuordnung von Zwischenstops ist problemlos zu beiden Flugtypen möglich, zumal auch Langstreckenflüge nonstop durchgeführt werden können. Unter den angegebenen Voraussetzungen sollte es also nur eine Klasse Flug mit einer Assoziation zu einer Klasse Zwischenstop geben.

#### **Lösung zu Übungsaufgabe 4.3.7**

(Aufgabentext S. 104)

Mit der angegebenen Assoziation werden die pro Werkstück eingesetzten Maschinen und die Reihenfolge ihres Einsatzes abgebildet. Doch wird hierfür nur eine unidirektionale Assoziation von der Klasse Werkstueck zur Klasse Maschine benötigt, während die entgegengesetzte Assoziation entbehrlich ist. Die bidirektionale Assoziation ist also für den angegebenen Zweck nicht angemessen.

#### **Lösung zu Übungsaufgabe 4.3.8**

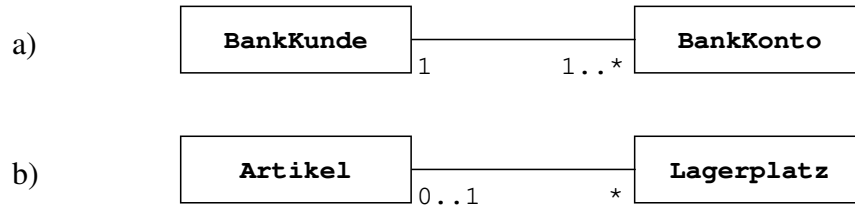
(Aufgabentext S. 105)

- a) Einfache Assoziation oder Aggregation, nicht Komposition. Die Mitarbeiter der Abteilung können auch im Unternehmen verbleiben, wenn die Abteilung aufgelöst wird.
- b) Es sind alle Bedingungen an eine Komposition erfüllt.
- c) Aggregation, nicht Komposition. Die Komponententypen eines Geräts können üblicherweise gleichzeitig auch in anderen Geräten verwendet werden.



**Lösung zu Übungsaufgabe 4.3.9**

(Aufgabentext S. 105)

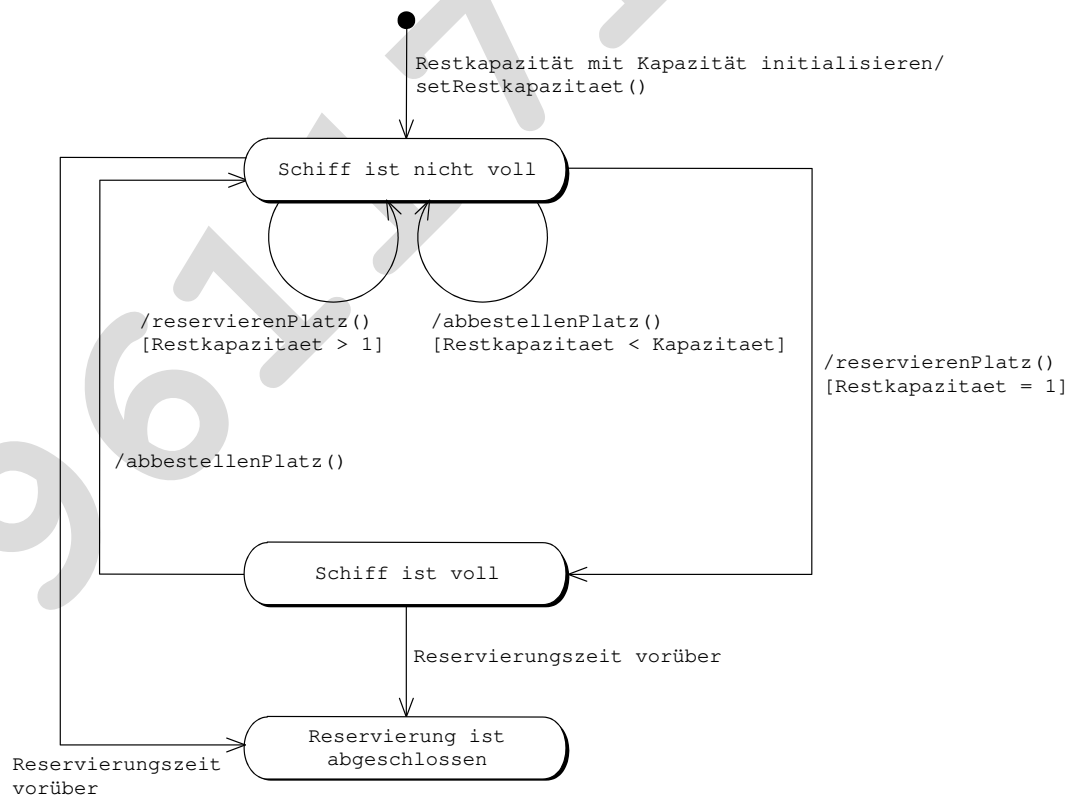
**Lösung zu Übungsaufgabe 4.3.10**

(Aufgabentext S. 106)

Es ist nicht klar, wie ausgehend von einer Auftragsposition auf das Attribut Stornosatz eines Artikels zugegriffen wird. Es bietet sich an, im Klassendiagramm eine zusätzliche Assoziation zwischen den Klassen AuftragsPosition und Artikel vorzusehen.

**Lösung zu Übungsaufgabe 4.3.11**

(Aufgabentext S. 106)

**Lösung zu Übungsaufgabe 4.4.1 (BAS)**

(Aufgabentext S. 132)

Siehe kursbegleitende CD-ROM, Ordner ProjektBAS\OOA.

**Lösung zu Übungsaufgabe 5.1.1**

(Aufgabentext S. 144)

- a) MDI-Anwendung.
- b) SDI-Anwendung.
- c) MDI-Anwendung.