

Webcode - Übungsdateien

0B8F-F147-695D

Objektorientierter Softwareentwurf mit UML

Grundlagen

(Stand 2017)

PGOS



Ricardo Hernández García

4. Ausgabe, Juni 2017

ISBN 978-3-86249-724-9

Schulversion

HERDT

Impressum

Matchcode: PGOS

Autor: Ricardo Hernández Garcia

Produziert im HERDT-Digitaldruck

4. Ausgabe, Juni 2017

HERDT-Verlag für Bildungsmedien GmbH
Am Kümmerling 21-25
55294 Bodenheim
Internet: www.herdtd.com
E-Mail: info@herdtd.com

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.

Bevor Sie beginnen...	4	7.5 Paketdiagramm	78
1 Überblick Softwareentwicklung	5	7.6 Klassendiagramm	80
1.1 Phasen der Softwareentwicklung	5	7.7 Komponentendiagramm	86
1.2 Grundlegende Überlegungen zur Aufwandseinschätzung	8	7.8 Sequenzdiagramm	87
1.3 Besonderheiten des Produkts Software	9	7.9 Kommunikationsdiagramm	89
		7.10 Zustandsdiagramm	91
		7.11 Einsatz- und Verteilungsdiagramm	94
2 Das objektorientierte Vorgehensmodell	11	8 Praxisbeispiel – Ticketsystem	96
2.1 OO-Vorgehensmodell	11	8.1 Projektvorstellung	96
2.2 Phasen des OO-Vorgehensmodells	14	8.2 Objektorientierte Analyse für das Projekt	98
2.3 UML-Notation	15	8.3 Anwendungsfälle in Pakete aufteilen	106
3 OO-Sprachelemente – Grundlagen	18	8.4 Paketdiagramme	113
3.1 Klassen und Objekte	18	8.5 Anwendungsfalldiagramme	114
3.2 OO-Prinzipien	20	8.6 Aktivitätsdiagramme	114
3.3 OO-Techniken	22	8.7 Aktivitätsdiagramme mit modellierten Objektzuständen	117
3.4 Nachrichten	26	8.8 Klassendiagramme	117
3.5 Richtlinien für die Namensvergabe von Bezeichnern	26	8.9 Sequenzdiagramme	122
4 Objektorientierte Analyse – OOA	29	8.10 Kommunikationsdiagramme	124
4.1 OOA-Konzepte – Grundlagen	29	8.11 Zustandsdiagramme	125
4.2 Analyse	29	8.12 Übung	126
4.3 Prozesssteuerung	31	9 Software für die Modellierung mit der UML	127
4.4 OOA-Musterlösungen	35	9.1 CASE-Tools	127
5 Objektorientierter Entwurf – OOD	41	9.2 Anwendungsbereich	128
5.1 OOD-Konzepte – Grundlagen	41	9.3 Anforderungen an ein CASE-Tool	128
5.2 Klassifizierung nach Klassen, Objekten und Attributen	41	9.4 Überblick CASE-Tools	130
5.3 Wiederverwendung	42	9.5 Grafische Tools	131
5.4 Klassenbibliotheken	42	9.6 Adressen im Internet	132
5.5 Framework	43	10 Einführung in die Anwendung objectiF	133
5.6 Softwarekomponenten	44	10.1 Grundfunktionen von objectiF	133
5.7 Simulation	45	10.2 Analysemodell erstellen	137
6 Einführung in die Entwurfsmustertechnik	46	10.3 Anwendungsfälle	139
6.1 Grundlagen zu Entwurfsmustern	46	10.4 Aktivitäten modellieren	145
6.2 Arten	47	10.5 Klassendiagramme	150
6.3 Ziele der Entwurfsmuster	48	10.6 Sequenzdiagramme	156
7 UML-Diagramme	67	10.7 Zustandsdiagramme	157
7.1 Einsatz und Normierung der UML	67	10.8 Speichern	158
7.2 Einsatz von Diagrammen zur Modellierung	67	Index	159
7.3 Anwendungsfalldiagramm	69		
7.4 Aktivitätsdiagramm	74		

Bevor Sie beginnen ...

HERDT BuchPlus – unser Konzept:

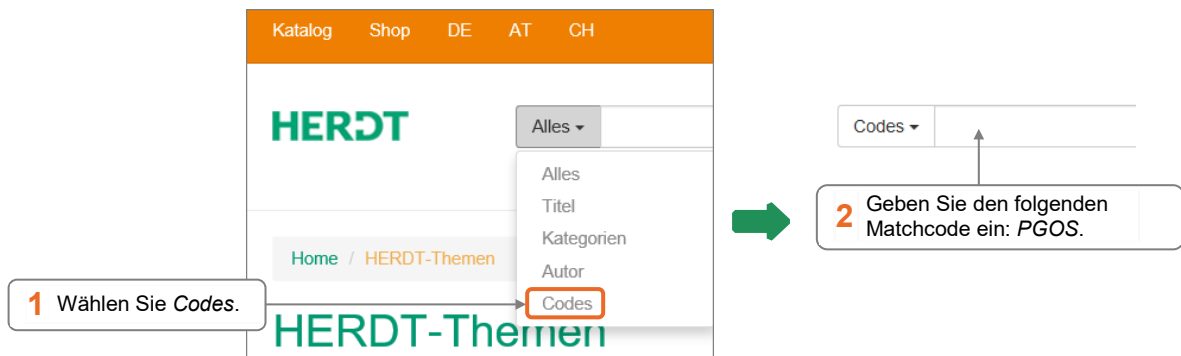
Problemlos einsteigen – Effizient lernen – Zielgerichtet nachschlagen

(weitere Infos unter www.herd.com/BuchPlus)

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:



- Rufen Sie im Browser die Internetadresse www.herd.com auf.



Für einen optimalen Lernerfolg verfügen Sie über folgende Kompetenzen:

- ✓ Sie verfügen über grundlegende Programmierkenntnisse.
- ✓ Sie besitzen erste Erfahrungen mit objektorientierten Programmiersprachen.

Hinweise zu Soft- und Hardware

- ✓ In diesem Buch wurden die Beispiele in der Programmiersprache C# und mittels der Software Microsoft Visual Studio 2017 erstellt.
- ✓ Für die UML-Modellierung wurde die Anwendung objectiF der Firma microTOOL eingesetzt.

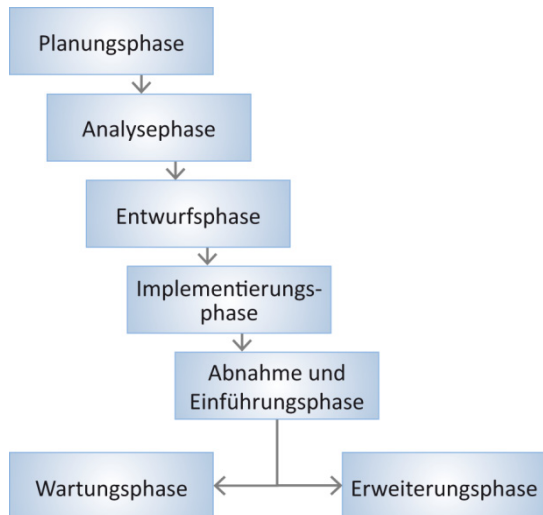
1

Überblick Softwareentwicklung

1.1 Phasen der Softwareentwicklung

Die Softwareentwicklung ist ein Prozess zur Erstellung eines Programms bzw. einer Anwendung (Vielzahl an Programmen) und kann in folgende Abschnitte unterteilt werden:

Phase	Aufgabe
Planungsphase	In ihr werden die Anforderungen an die zu erstellende Anwendung ermittelt.
Analysephase	Im Ergebnis dieser Phase liegt eine Beschreibung der Vorgänge der zu erstellenden Anwendung vor.
Entwurfsphase	Sie bildet das Ergebnis der Analysephase auf Datenstrukturen ab, die mit einer beliebigen Programmiersprache implementiert werden können.
Implementierungsphase	Das Ergebnis der Implementierungsphase ist die vollständig implementierte Anwendung.
Abnahme und Einführungsphase	Die erstellte Anwendung wird in der Zielumgebung installiert und ihre Funktionsweise nachgewiesen. Nach der Abnahme durch den Auftraggeber wird die Anwendung in den Produktionsprozess integriert.
Wartungsphase	Nach der Einführung der Software beim Auftraggeber werden fehlerhafte Funktionen nachgebessert und notwendige Anpassungen implementiert.
Erweiterungsphase	Die Funktionalität der Anwendung wird erweitert, um das Einsatzgebiet oder die Akzeptanz zu vergrößern.



Nachfolgend werden die wichtigsten der aufgezeigten Phasen näher erläutert.

Analysephase

In der Analysephase geht es vor allem darum, das Einsatzgebiet der Anwendung zu umschreiben und Handlungsweisen der Anwender zu identifizieren. Es werden Geschäftsprozesse in Szenarios modelliert, um sie im weiteren Entwicklungsprozess abstrahieren zu können.

Für jeden Vorgang wird bestimmt, welche Auslöser für den Vorgang zuständig sind, welche Vorbedingungen erfüllt sein müssen und zu welchem Ziel oder welchen Zielen die Aktion führen soll. Durch Gespräche mit dem Fachpersonal, den künftigen Anwendern und den mit der Umsetzung beauftragten Personen können Sie die angefertigten Aufzeichnungen und Modelle über die modellierten Geschäftsprozesse verfeinern.

Erweitern Sie gegebenenfalls das Modell durch die Aufteilung der bereits erfassten Tätigkeiten in mehrere kleinere Teilprozesse.

Das Ziel dieser Analyse der Geschäftsprozesse ist ein Modell der gesamten für das Softwareprodukt relevanten Umgebung. Die Entscheidung, ob das Programm diesen oder jenen Fall bearbeitet, wird hier noch nicht gefällt.

Entwurfsphase

In der Entwurfsphase werden die erstellten Modelle von der praxisorientierten Sicht auf eine Sicht, die aus Strukturen und Elementen der Programmierung besteht, übertragen und modelliert.

Gleichzeitig werden in dieser Phase der Umfang und die Schnittstellen der relevanten auszutauschenden Daten bestimmt, die für die Abarbeitung des Geschäftsprozesses notwendig sind.

Implementierungsphase

Die Implementierungsphase ist die eigentliche Umsetzung des erarbeiteten Modells in den Quellcode der gewählten Programmiersprache.

Der erste Schritt der Implementierungsphase kann auch die Erstellung eines Prototyps sein. Hiermit ist es möglich, dem späteren Anwender einen Einblick in die Bedienung des zukünftigen Systems zu geben und sehr frühzeitig Unstimmigkeiten zwischen den Vorstellungen des Auftragnehmers und des Auftraggebers aufzudecken.

Es wird unterschieden zwischen folgenden Prototypen:

- ✓ **Horizontaler Prototyp**

Stellt einen Querschnitt des künftigen Programms dar, z. B. die Benutzeroberfläche (ohne Funktionalität).

- ✓ **Vertikaler Prototyp**

Stellt eine vollständige Implementierung einer Kernfunktion des späteren Einsatzgebietes dar und deckt diese tief greifend ab. Es werden aber nicht alle möglichen Konstellationen berücksichtigt.

Der Prototyp ist nicht nur zur Demonstration bestimmt, er kann bereits ein Teil des neuen Produkts sein.

Bei einigen Methoden der Softwareentwicklung wird immer auf der Basis des Prototyps weitergearbeitet, bei anderen werden die Prototypen ständig durch neue Implementierungen ersetzt (extreme programming).

Abnahme und Einführungsphase

Jedes Softwareprodukt durchläuft bereits bei der Erstellung einen rudimentären Funktionstest. Nach der Fertigstellung erfolgen weitere tief greifende Tests, die alle Funktionen des Programms in der geplanten Interaktion ausführen. Die absolute Fehlerfreiheit eines Programms kann aber nie garantiert werden, hier gilt es, immer die Balance zwischen vertretbaren Kosten für den Test und der notwendigen Fehlerfreiheit zu finden. Ebenso ist es nicht immer realisierbar, alle möglichen Zustände eines Programms zu ermitteln, um diese zu testen.

Die Qualität einer Software wird nicht immer hundertprozentig an der Fehlerfreiheit verankert, sondern an der gesicherten Funktion der Haupteinsatzfälle bzw. deren geforderter Fehlerfreiheit. Die ordnungsgemäße Funktion dieser Hauptanwendungsfälle muss in Testserien bestätigt werden.

Nach dem Test der Software und der Abnahme durch den Auftraggeber wird die Software – im Rahmen der Einführungsphase – in das bestehende Arbeitsumfeld integriert.

Für den Auftraggeber erfolgt die Integration der neuen Software in das bestehende Arbeitsumfeld, die neue Software wird auf dem Zielsystem installiert und in Betrieb genommen. Bei dieser Umstellung ist das Vorhandensein älterer Datenbestände oder die Ankopplung an Prozesse, die mit der neuen Software kommunizieren, zu berücksichtigen (sogenannte Legacy-Anwendung).

An dieser Stelle wird eine gründliche Vorarbeit belohnt, wenn die Schnittstelle zur Anbindung der neuen Software bereits in der Analyse- und Entwurfsphase durchdacht und angepasst wurde.

Wartungsphase

Die Wartungsphase des Programms bzw. der Anwendung umfasst:

- ✓ Beheben von Fehlern, die durch den täglichen Einsatz festgestellt werden;
- ✓ Implementierung von Zusatzfunktionen, die in der Basisversion zwar erwogen, aber noch nicht eingearbeitet wurden;
- ✓ Anpassung der Software, um die Aktualität der Software zu erhalten sowie die Unterstützung neuer Hardware und neuer Betriebssysteme zu gewährleisten.

Diese Tätigkeiten müssen bei der Planung der finanziellen Mittel berücksichtigt werden.

Der Auftragnehmer stellt die zugesagte fehlerfreie Funktionalität seiner Anwendung sicher und trägt somit alle hierfür erforderlichen finanziellen Aufwendungen.

Für die Anpassungen des Programms an geänderte externe Rahmenbedingungen sind zusätzliche finanzielle Aufwendungen des Auftraggebers notwendig, die in einem neuen Auftrag vereinbart werden oder deren Bezahlung über gesonderte Verträge geregelt wird (z. B. über einen Wartungsvertrag).

Der Alterungsprozess der Anwendung tritt sofort mit der Fertigstellung der Anwendung ein und ist nicht von der Nutzung abhängig. Somit ist zu erwarten, dass jedes Programm im Laufe der Zeit durch ein neues, jüngeres ersetzt wird. Ein weiterer Grund für die Neuerstellung einer Anwendung ist die Erhöhung des Pflegeaufwandes durch jede Änderung, die zusätzlichen Änderungsbedarf verursachen kann.

Weil der steigende Wartungsaufwand hohe Kosten verursacht, ist die Pflege nach einer bestimmten Nutzungsdauer nicht mehr mit vertretbarem Aufwand möglich.

Es ist eine schwierige Aufgabe, den richtigen Zeitpunkt festzulegen, ab wann eine Neuerstellung sinnvoll oder eine Änderung der vorhandenen Software noch vernünftig ist.

1.2 Grundlegende Überlegungen zur Aufwandseinschätzung

Eine effektive Softwareentwicklung ist eine Gratwanderung zwischen dem Aufwand für die Planung, die zwar notwendig ist, aber noch kein Softwareprodukt liefert, und einer relativ schnellen Implementierung. Um den Aufwand angemessen abzuwägen, sollten Sie die folgenden Aspekte betrachten:

- ✓ Wie komplex ist die zu erstellende Anwendung?
- ✓ Wie lange wird die Anwendung eingesetzt, bevor sie durch eine neue Anwendung ersetzt oder nicht mehr benötigt wird?
- ✓ Sind bereits Erweiterungen vorgesehen, die in der ersten Version nicht implementiert werden?
- ✓ Wie viele Entwickler werden die Anwendung erstellen?
- ✓ Wie viele Nutzer werden mit der Anwendung arbeiten?

- ✓ Welche Kenntnisse und welches Vorwissen haben die zukünftigen Anwender?
- ✓ Über welche Kenntnisse verfügt das Wartungspersonal der Anwendung?
- ✓ Wie sind die Netzwerktauglichkeit und die benötigte Stabilität des Systems einzuschätzen?
- ✓ Für welche Betriebssysteme (Windows, Linux, Mac OS, UNIX) wird die Anwendung benötigt?

Ein hoher Aufwand bei der Planung und Analyse der Softwarestruktur gewährleistet im Regelfall ein Softwareprodukt mit einer niedrigen Fehlerquote, einer stringenten Implementierungsphase und einer kostengünstigen Wartung. Gleiches gilt für Anpassungen und Erweiterungen.

1.3 Besonderheiten des Produkts Software

- ✓ Software ist ein künstliches Produkt der menschlichen Abstraktion und kein materielles Produkt.
- ✓ Software kann nicht mit Ersatzteilen repariert werden.
- ✓ Software wird nicht durch physikalische Gesetze begrenzt. (Eventuell verhindert die Ausführungszeit komplizierter Algorithmen eine sinnvolle Implementierung.)
- ✓ Software unterliegt keinem materiellen Verschleiß, vielmehr ist sie von der Entwicklung der Technik abhängig und wird durch den technischen Fortschritt überholt.
- ✓ Um die Mächtigkeit einer Software zu bestimmen, benötigen Sie Verfahren zur Messung der Qualität und Quantität von Software.

Bewertungskriterien für Softwareprodukte und Entwicklungsprozesse

Die folgende Tabelle zeigt die wichtigsten Bewertungskriterien und das jeweilige Ziel der Anwendungsentwicklung bezüglich dieser Kriterien.

Bewertungskriterium	Ziel der Anwendungsentwicklung
Funktionalität	Die Produkthanforderungen müssen erfüllt werden.
Qualität	Die Qualität des Endproduktes ist mit den Qualitätsanforderungen des Auftraggebers zu vergleichen und gegebenenfalls anzugleichen.
Termin	Die gestellten Termine für die Fertigstellung der Anwendung sind einzuhalten.
Kosten	Einhaltung des geplanten Sach- und Personalaufwandes für die Produkterstellung und die Produktpflege

Qualitätskriterien

Wie jedes Produkt unterliegt auch Software besonderen Qualitätsanforderungen. Die Software muss dabei den Anforderungen zweier großer Interessengruppen entsprechen – denen der Auftraggeber und denen der Hersteller.

Der Auftraggeber achtet auf die Qualität der Software und auf die Betreuung nach dem Kauf, beispielsweise den Kundendienst. Für ihn ist es wichtig, dass die Software zum bestellten Termin, zum vereinbarten Preis und mit der vereinbarten Funktionalität ausgeliefert wird. Außerdem spielen der Einarbeitungsaufwand und der Bedienkomfort eine wichtige Rolle.

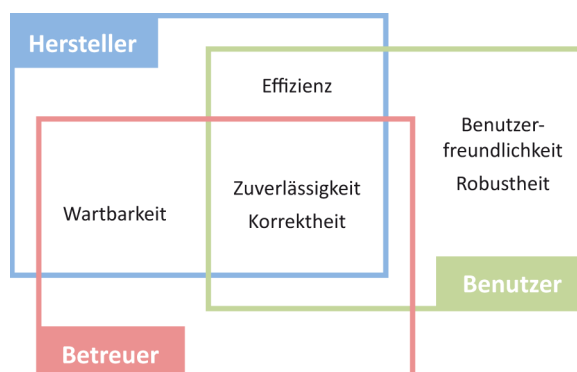
Dem Hersteller ist es wichtig, bei der Entwicklung und der nachfolgenden Wartung rentabel zu arbeiten und durch Wiederverwendung von bestehenden Modulen bzw. Komponenten den Programmieraufwand, und damit die Herstellungskosten, zu senken.

Sobald das Produkt ausgeliefert ist, geht die Verantwortung vom Hersteller auf den Betreuer der Software über, meistens auf den Hersteller selbst. Dieser übernimmt die Wartung der Software, z. B. in Form des Kundendienstes.

Die folgende Abbildung zeigt die Verbindung der einzelnen Qualitätskriterien.

Äußere Qualitätskriterien

Äußere Qualitätskriterien eines Softwareproduktes sind Korrektheit, Robustheit, Zuverlässigkeit, Effizienz und Benutzerfreundlichkeit. Diese Eigenschaften der Software haben einen großen Einfluss auf die Ausführung eines Programms und sind von den Anwendern durch die Arbeit mit der Anwendung zu beurteilen.



Inneres Qualitätskriterium

Das innere Qualitätskriterium ist die Wartungsfreundlichkeit, die für den Hersteller bei der Fehlersuche bzw. bei Änderungen an der Software von Bedeutung ist.

Software ist nach der Auslieferung weiterhin vielen Änderungen und Erweiterungen mit neuen Funktionen unterworfen. Ein weiteres sehr wichtiges Merkmal der Softwarequalität ist die Dokumentation, welche die Entwicklung und Bedienung des Programms umfasst. Sie wird ständig zur Beschreibung der internen Arbeitsweise (Dokumentation des Programmquelltextes) sowie zur Beschreibung der Arbeitsweise des Programms (Bedienungsanleitung) aktualisiert.



Wissenstest: Phasen der Softwareentwicklung und das OO-Vorgehensmodell

2

Das objektorientierte Vorgehensmodell

2.1 OO-Vorgehensmodell

Beim objektorientierten Softwareentwicklungsmodell – kurz OO-Vorgehensmodell genannt – erfolgt der Softwareentwurf mit speziellen Hilfsmitteln und wird unabhängig von einer speziellen Programmiersprache durchgeführt. Die Verteilung der Teilaufgaben wird über Klassen vorgenommen.

Um die entscheidenden Ziele bei der Entwicklung umfangreicher Softwaresysteme zu realisieren (z. B. Produktivitäts- und Qualitätssteigerung), werden bei objektorientierten Programmiersprachen die Wiederverwendbarkeit, Erweiterbarkeit und Kompatibilität korrekt arbeitender Programmelemente (Module) in den Vordergrund gestellt. Die Produktivität der Programme wird im Regelfall durch Benutzeroberflächen erhöht.

Die objektorientierte Vorgehensweise versucht durch Abstraktion, Datenkapselung, Modularität und Hierarchieprinzipien, diese Ziele zu erreichen.

Notation

Der Softwareentwurf wird oft von einer Gruppe von Entwicklern durchgeführt. Deshalb ist eine geeignete einheitliche und bereichsübergreifende Notation zur Modellierung der Abläufe notwendig. Für diese Notation ist die Unified Modeling Language (kurz UML) entwickelt und standardisiert worden. Sie bietet verschiedene Sichten und Diagrammarten zur Visualisierung der Abläufe. Mit diesem Mittel ist eine durchgängige Verständigung über die Vorgänge bis hin zur Implementierung möglich. Die Erstellung der Abläufe können Sie mit einem Notationsprogramm (CASE-Tool, vgl. Kapitel 9) unterstützen.

Vorteile des Einsatzes objektorientierter Programmiersprachen

Der große Vorteil von objektorientierten Programmiersprachen liegt in der Bereitstellung unabhängiger Module (in Form von Klassen und Objekten). Durch eine Kapselung wird die „interne“ Funktionsweise verborgen. Über Schnittstellen erfolgt die Kopplung der Module, d. h., über diese wird die Interaktion der Module untereinander realisiert. Somit ist die Austauschbarkeit der

Module durch Module mit geänderter interner Funktionsweise, aber gleicher Schnittstelle möglich.

Die Grundelemente und Grundprinzipien der objektorientierten Anwendungsentwicklung wie Klassen, Objekte, Vererbung, Kapselung und Polymorphie werden von objektorientierten Programmiersprachen direkt unterstützt, womit die programmtechnische Umsetzung erleichtert wird. Zu diesen Sprachen gehören z. B. C++, C#, Delphi, Smalltalk und Java. Sprachen, die nicht den gesamten Umfang der objektorientierten Sprachen unterstützen, die aber Objekte und Klassen kennen, werden objektbasierte Sprachen genannt, z. B. JavaScript und Visual Basic (bis Version 6/ vor .Net). Mit diesen Programmiersprachen ist eine Umsetzung der objektorientierten Anwendungsentwicklung schwerer möglich. Mit Sprachen, die Objekte und Klassen nicht unterstützen, ist die Umsetzung nur unter einem sehr hohen Aufwand möglich, da diese Elemente nachgebildet werden müssen.

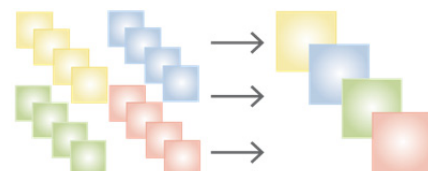
Prinzipien des objektorientierten Softwareentwurfs

Abstraktion

Beim objektorientierten Softwareentwurf werden die verschiedenen Entwurfskonzepte aufgrund ihrer Abstraktionsmöglichkeiten unterschieden. Die funktionale Abstraktion führt zu abstrakten Prozeduren und Funktionen. Bei der nächsten Stufe, der Datenabstraktion, werden abstrakte Datentypen erstellt und abstrakte Zugriffsoperationen festgelegt, die darauf angewendet werden können. Die höchste Stufe der Abstraktion wird beim objektorientierten Entwurf erreicht.

Die abstrakten Datentypen lassen sich mit den Attributen einer Klasse vergleichen, und die abstrakten Zugriffsoperationen können als deren Methoden angesehen werden.

Beim Vorgang der Abstraktion werden gemeinsame Objekteigenschaften und Zugriffsoperationen ermittelt, die in einer Klasse zusammengefasst werden. Gleiche Eigenschaften und Verhaltensweisen verschiedener Klassen können in übergeordneten Klassen zusammengefasst werden. Es entstehen Klassenhierarchien.



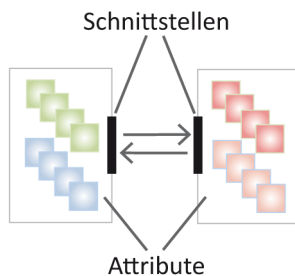
Gleiche Eigenschaften und Zugriffsoperationen finden und abstrahieren

Datenkapselung

Als Datenkapselung wird die Zusammenfassung von Daten (Attributen) einer Klasse und die auf die Objekte dieser Klasse anwendbaren Funktionen (Methoden) bezeichnet. Auf die Attribute eines Objekts darf dabei nur über die eigenen Methoden zugegriffen werden, die eine definierte externe Schnittstelle bilden. Auf diese Weise lassen sich bestimmte innere Eigenschaften eines Objekts verbergen (Information Hiding).

Gegebenenfalls notwendige Änderungen an der inneren Struktur der Klassen lassen sich so ohne Auswirkungen auf die extern zugreifenden/nutzenden Komponenten durchführen (Bedingung dafür ist eine unveränderte Schnittstelle).

Durch die Kapselung von Daten und den sie manipulierenden Methoden wird der Zugriff auf diese Daten ausschließlich über eine definierte Schnittstelle eingegrenzt, somit die problemlose Austauschbarkeit der Klassen ermöglicht und damit die Wartbarkeit der Software verbessert.



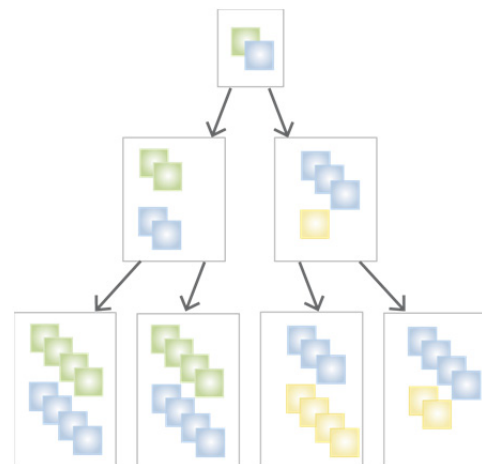
*Attribute und Methoden werden in der Klasse zusammengefasst,
Objekte können nur über definierte Schnittstellen miteinander kommunizieren*

Modularität

Das Modularitätsprinzip wird bereits bei der Strukturierung von Programmen, der schrittweisen Verfeinerung, angewendet. Wenn Sie die Vorgehensweise betrachten, Programme in einzelne Module zu unterteilen, bedeutet das in der objektorientierten Programmierung, dass Sie Klassen als einzelne Module entwickeln. Der Übersicht halber werden die Klassen, die in einem logischen Zusammenhang stehen, in einem Paket zusammengefasst.

Hierarchie

Strukturieren Sie Softwaresysteme nur nach den Prinzipien der Abstraktion, Kapselung und Modularität, erhalten Sie Klassen, die zum Teil die gleichen Attribute und Methoden enthalten können. Deshalb werden die Klassen dahin gehend untersucht, ob sich vorhandene Eigenschaften und Methoden generalisieren lassen. Ist dies der Fall, wird eine übergeordnete Klasse angelegt, welche die gemeinsamen Operationen und/oder Daten enthält. Diese übergeordnete Klasse vererbt ihre Attribute und Methoden an die untergeordneten Klassen. Vererben bedeutet, dass die Attribute und Methoden automatisch den untergeordneten Klassen zur Verfügung stehen und nicht noch einmal spezifiziert werden müssen.



Klassenhierarchie

Über Schutzattribute können Sie die Sichtbarkeit von Attributen und Methoden einer Klasse einschränken. Einer untergeordneten Klasse kann wiederum eine weitere Klasse untergeordnet sein usw., sodass sich eine Klassenhierarchie ergibt.

Typisierung

Bei stark typisierten Sprachen ist schon bei der Kompilierung bekannt, von welchem Datentyp eine bestimmte Variable oder ein bestimmtes Objekt ist. Der Datentyp ist nicht nur eine Angabe über den benötigten Speicherbedarf, sondern auch eine Festlegung, für welche Funktionen diese Variable eingesetzt werden kann. Durch die Festlegung auf einen bestimmten Typ kann schon während der Kompilierung überprüft werden, ob bestimmte Funktionen auf den entsprechenden Datentyp anwendbar sind. Bei einer objektorientierten Sprache wird zusätzlich überprüft, ob eine Methode auf ein übergebenes Objekt angewandt werden kann.

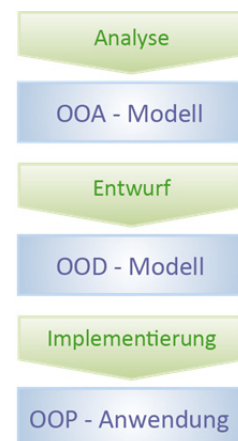
2.2 Phasen des OO-Vorgehensmodells

Der Weg vom Problem zur Softwarelösung ist beim objektorientierten Ansatz ähnlich dem der konventionellen Vorgehensweise. Es werden ebenfalls die Phasen Analyse (OOA – objektorientierte Analyse), Entwurf (OOD – objektorientiertes Design) und Implementierung (OOP – objektorientierte Programmierung) durchlaufen.

Am Ende einer jeden Phase existiert ein Modell des Prozesses, das in die nächste Phase übernommen und dort weiterentwickelt wird. Die Prozesse laufen aber nicht wie bei den konventionellen Vorgehensweisen unabhängig voneinander und streng nacheinander ab, sondern sind ineinander verzahnt (z. B. Erstellung eines Prototyps während der noch laufenden Designphase).

Die Modelle der Phasen unterscheiden sich durch verschiedene Sichtweisen:

- ✓ **OOA-Modell:**
Der zu modellierende Prozess wird aus der Sicht des Anwenders und seiner Tätigkeiten analysiert.
- ✓ **OOD-Modell:**
Das OOA-Modell wird aus der Sicht von Datenstrukturen und deren Kommunikation untereinander betrachtet.
- ✓ **OOP-Anwendung:**
Die im OOD-Modell gefundenen Datenstrukturen und Beziehungen werden in den Anweisungen einer oder mehrerer konkreter Programmiersprachen übersetzt.



2.3 UML-Notation

UML steht für „Unified Modeling Language“ und bezeichnet eine Modellierungssprache zur Beschreibung der Struktur objektorientierter Softwaresysteme.

Die UML verfolgt folgende Ziele und Aufgaben:

- ✓ Bereitstellung einer universellen Beschreibungssprache für alle Arten objektorientierter Softwaresysteme;
- ✓ Fokussierung auf die Produkte des Software-Engineering, nicht auf die Prozesse, beispielsweise auf die Planung des Projekts mit der Erstellung eines Projektplanes, das Finden eines geeigneten Prozessmodells, das Erkennen und Verfolgen von Risiken, die frühestmögliche Fehlererkennung und die klare Zuweisung von Aufträgen, Verantwortlichkeiten und Ressourcen.

Die UML ist eine Modellierungssprache, die verschiedene Diagrammart verwendet (vgl. Kapitel 7), die grafische Elemente besitzen. Die Bedeutung der Elemente ist genau festgelegt. Viele dieser Elemente können in unterschiedlichen Diagrammart verwendet werden.

Darüber hinaus enthält die UML Erweiterungsmöglichkeiten, die die Erstellung von Softwarewerkzeugen erlauben, mit denen Projekte in der UML modelliert und implementiert werden können. Diese flexiblen Erweiterungsmöglichkeiten gestatten den Einsatz in vielen Bereichen.

In der UML wird keine Methode zur Erstellung von Modellen beschrieben, d. h., Sie erhalten mit der UML keine Anleitung oder Schrittfolge, mit der Sie ein Modell erstellen können.

Notation von Objekten und Klassen

Entitäten können Sie in der UML durch Objekte und die Entitätstypen durch Klassen darstellen. Dies gestattet eine enge Bindung an die objektorientierten Programmiersprachen, weil diese Sprachen Klassen und Objekte als Datenstrukturen kennen. Somit sind der Abgleich der Begriffswelt des in der Analyse erstellten Prozessmodells und die Umsetzung in der objektorientierten Programmiersprache möglich.

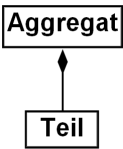
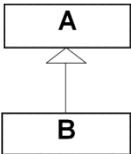
Element	Notation / verbaler Bezeichner
Objekt mit Objekt- und Klassennamen, mit Klassennamen, nur mit Objektnamen	A-Obj:KlasseX :KlasseX A-Obj
Objekt-Attribute	:KlasseX Attribut1 = WertA Attribut2 = WertB
Klasse	KlasseX
Klassenerweiterung ist eine Erweiterung der Klassenelemente, die in eine weitere Klasse verlagert wurden	extend

Element	Notation / verbaler Bezeichner
Attribut, Operation kann bei der Klasse mit angegeben werden (optional)	<pre> classDiagram class KlasseX { Attribut Operation() } </pre>
Attributtypen und Initialwerte Zusammengesetzte Attribute werden als Stereotyp <i>Struktur</i> angegeben. Stereotypen erweitern die vorhandenen Modellelemente.	<pre> classDiagram class KlasseX { "Attribut: Typ = Anfangswert" Operation() } </pre>
Klassenattribut, Klassenoperation (ohne Instanz vorhanden)	<pre> classDiagram class KlasseX { <<Klassenattribut>> <<Klassenoperation>>() } </pre>
Abstrakte Klasse, abstrakte Operation	<pre> classDiagram class abstrakteKlasse { <<abstrakt>> <<Attribut>> <<Operation>>() } class KlasseX { <<Attribut>> <<abstrakteOperation>>() } </pre>

Notation von Beziehungen

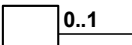
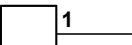
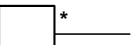
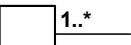
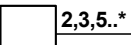
In der UML können neben den Beziehungen zwischen den Klassen auch die Beziehungen von Objekten dargestellt werden.

Element	Notation
Verbindung zwischen Objekten Können Sie über ein Attribut einer Klasse zu einem Objekt einer zweiten Klasse navigieren, wird dieser Zusammenhang als Link modelliert.	<pre> classDiagram class "KlasseX" class "KlasseY" "KlasseX" -- "KlasseY" </pre>
Assoziationen Ruft eine Klasse die Operationen einer zweiten Klasse auf, wird eine Beziehung (Assoziation) zwischen beiden Klassen modelliert.	<pre> classDiagram KlasseX -- KlasseY </pre>
Reflexive Assoziation Diese Assoziation wird modelliert, wenn eine Methode eine zweite Methode der Klasse aufruft und diese zur Schnittstelle der Klasse gehört.	<pre> classDiagram class KlasseX { "1" -- "*" KlasseX : Rolle A, Rolle B } </pre>
Assoziative Klassen Wird eine Assoziation von Attributen beschrieben, die einer dritten Klasse angehören, ist diese Klasse eine assoziative Klasse.	<pre> classDiagram KlasseX -- KlasseY KlasseZ -- KlasseX KlasseZ -- KlasseY </pre>
Aggregation ist-Teil-von-Beziehung Das Objekt einer Klasse verwaltet Objekte einer oder mehrerer anderer Klassen.	<pre> classDiagram Aggregat o-- Teil </pre>

Element	Notation
Komposition Die Komposition ist eine besondere Form der Aggregation. Im Unterschied zur Aggregation sind die verwalteten Objekte fester Bestandteil des Containerobjekts. Es existiert nur mit seinen Teilobjekten und niemals allein.	
Vererbung ist-ein-Beziehung "B ist ein A."	

Notation der Kardinalität

Die Kardinalität wird in der UML in Anlehnung der numerischen Notation mit Ziffern und dem Zeichen ***** angegeben. Das Zeichen ***** steht dabei für beliebig viele Objekte (0..n) einer Klasse. Ein Bereich wird mit zwei Punkten zwischen dem Anfangswert und dem Endwert angegeben. Kann die Anzahl mehrere einzelne Werte annehmen, werden die einzelnen Werte mit Kommata voneinander getrennt.

Kardinalität (Vorkommen von Objekten einer Klasse)				
Ein oder kein Objekt	Genau ein Objekt	Kein, ein oder beliebig viele Objekte	Ein oder beliebig viele Objekte	Zwei, drei, fünf oder mehr Objekte
				



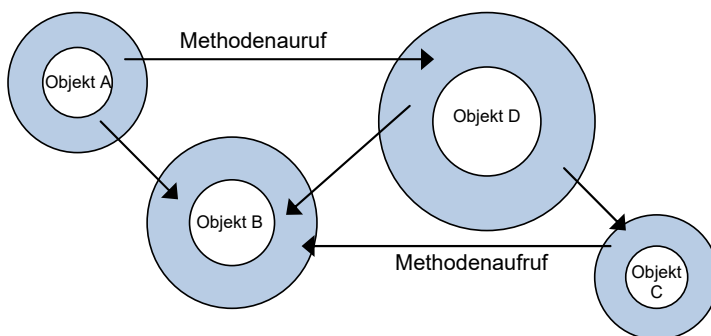
Wissenstest: *Phasen der Softwareentwicklung und das OO-Vorgehensmodell*

3

OO-Sprachelemente – Grundlagen

3.1 Klassen und Objekte

Eine Programmiersprache kann als Elemente Klassen und Objekte unterstützen und somit eine objektorientierte Softwareentwicklung gestatten. Ein Vorteil der Verteilung von Daten und Funktionalität auf Klassen und deren Objekte liegt in der einfachen Wartung. Da diese Objekte untereinander über die Methoden einer Klasse gekoppelt werden, sind sie mit beliebigen Objekten, die die gleiche Schnittstelle anbieten, kompatibel und austauschbar. Weiterhin können die erstellten Klassen in anderen Programmierteilen verwendet werden.



Methodenaufrufe unterschiedlicher Objekte

Klassen

Klassen stellen Beschreibungen für den Aufbau von Objekten dar. Alle Objekte, die Sie von einer Klasse erzeugen, werden nach dieser Vorlage erstellt.

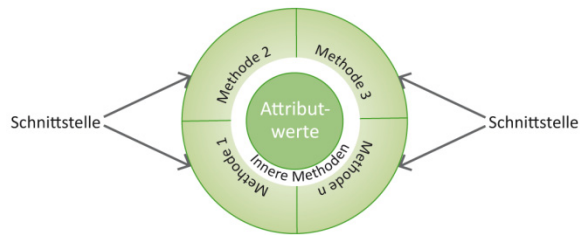
Klassen können Attribute besitzen, die die relevanten Datenwerte der Klasse darstellen.

<u>KlasseX</u>
Attribut
Operation()

*Darstellung
in der UML*

Außerdem können Klassen Methoden (Operationen) haben, die den Zugriff auf die Attribute ermöglichen und die Manipulation der Daten in den Attributen realisieren. Die Klasse hat einen privaten Kernbereich, in dem sich die Attribute und Methoden, die mit den Attributen oder mit anderen Methoden des Objekts zusammenarbeiten, befinden.

Um den Zugriff für andere Objekte zu ermöglichen, wird eine Schnittstelle definiert. Zu dieser gehören nur die Methoden, die von anderen Klassen und Objekten aufgerufen werden sollen. Dies ermöglicht die Betrachtung einer Klasse als Blackbox, die die Schnittstellenmethoden als Ein- und Ausgabekanäle anbietet. Somit setzt die Verwendung der Klasse nur die Kenntnis der jeweiligen Schnittstellenmethoden voraus.



Darstellung der Klasse als Blackbox

Objekte

Ein Objekt ist die konkrete Ausprägung (Instanz) einer Klasse. Der Bauplan für das jeweilige Objekt ist die Klasse, von der das Objekt erzeugt wird. Der Zustand eines Objekts wird durch die aktuelle Wertebelegung der Attribute gekennzeichnet. Ein Objekt ist damit etwas Einmaliges, besitzt jedoch, wie alle Objekte der entsprechenden Klasse, die gleichen Methoden.

Objekt: KlasseX
Attribut1 = WertA
Attribut2 = WertB

*Darstellung
in der UML*

Attribute und Methoden

Attribute bestimmen den Zustand eines Objekts. Im Allgemeinen sind die Attribute und deren Werte außerhalb des Objekts nicht sichtbar. Das bedeutet, dass sie weder direkt gelesen noch verändert werden können. Die Werte der Attribute werden von Methoden manipuliert, die auch zur Schnittstelle der Klasse gehören können. Über diese Methoden können die Attribute auch von anderen Objekten gelesen und verändert werden.

Schnittstellen

Schnittstellen bilden die Kanäle, über die andere Objekte mit dem Objekt kommunizieren können, d. h., die Objekte können Methoden des jeweils anderen Objekts aufrufen. Über Schnittstellen können Objekte ihre Funktionalität anderen Objekten anbieten. Schnittstellenmethoden können innerhalb der eigenen Klasse andere Schnittstellenmethoden und auch Methoden, die nicht zur Schnittstelle gehören, aufrufen. Sie können Attribute des eigenen Objekts lesen und schreiben.

Konstruktoren und Destruktoren

Ein Objekt wird zuerst erzeugt und initialisiert. Analog soll bei seiner Zerstörung die Möglichkeit gegeben sein, bestimmte abschließende Operationen auszuführen. Für diesen Zweck wurden für jede Klasse zwei spezielle Methoden eingeführt.

Konstruktoren werden automatisch bei der Erzeugung eines Objekts aufgerufen. Der Konstruktor ist wie eine Methode der Klasse aufgebaut und besitzt folgende Eigenschaften:

- ✓ Die Methode wird als Konstruktor gekennzeichnet, z. B. durch ein spezielles Schlüsselwort. In einigen Sprachen wird ein Konstruktor durch das Schlüsselwort `constructor` definiert, in anderen muss er den gleichen Namen wie die Klasse besitzen.

- ✓ Er liefert in der Regel kein Ergebnis zurück.
- ✓ Ein Konstruktor kann nicht wie eine reguläre Methode aufgerufen werden. Der Aufruf erfolgt automatisch und einmalig beim Anlegen einer Instanz der Klasse.
- ✓ Eine Klasse kann mehrere Konstruktoren besitzen, die sich in der Parameteranzahl und/oder in den Parametertypen unterscheiden müssen.

Destruktoren werden automatisch vor dem Zerstören eines Objekts aufgerufen.

Über Anweisungen, die den Destruktoren hinzugefügt werden, können Sie vor der Zerstörung eines Objekts noch erforderliche Anweisungen ausführen. Beispielsweise können Sie dynamisch angeforderten Speicher freigeben oder Daten des Objekts auf einem Datenträger speichern.

Ein Destruktor besitzt folgende Eigenschaften:

- ✓ Er hat keinen Rückgabewert und keine Parameter.
- ✓ Jede Klasse besitzt genau einen Destruktor.
- ✓ Destruktoren können im Gegensatz zu Konstruktoren nicht direkt aufgerufen werden.

Beispiel

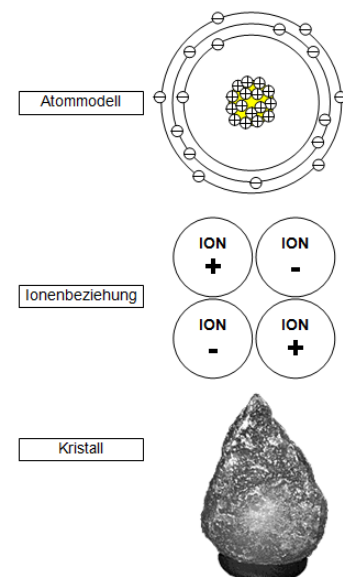
Eine Klasse wurde entwickelt, um Fahrzeuge eines Fuhrparks zu verwalten. Bei der Generierung eines Objekts dieser Klasse wird der Konstruktor aufgerufen, um Informationen eines speziellen Fahrzeugs aus einer Datenbanktabelle zu lesen und dem erzeugten Objekt zuzuweisen.

Vor dem Zerstören des Objekts wird der Destruktor aufgerufen, um die bearbeiteten Fahrzeugdaten in die Datenbanktabelle zurückzuschreiben und den benutzten Arbeitsspeicher, der die Fahrzeug-Information enthielt, freizugeben.

3.2 OO-Prinzipien

Für den Menschen ist es schwer, ein komplexes System in seiner Gesamtheit zu verstehen. Er versucht die Vielfalt durch Abstrahieren zu bewältigen, d. h. unwichtige Details zu ignorieren und mit einem verallgemeinerten, idealisierten Modell des Objekts zu arbeiten.

Eine andere Methode zur Reduzierung der Komplexität besteht in der Zerlegung des Systems in kleine Teilsysteme. Dabei tritt begünstigend eine erstaunliche Besonderheit auf. Diese besteht darin, dass die Kombination von Teillösungen nicht nur die Summe der Komplexität aller Teillösungen hervorbringt, sondern dass eine neue Qualität entsteht, die im Ergebnis ein weitaus komplexeres funktionierendes System darstellt. Anders ausgedrückt können Sie durch die Kombination von vielen kleinen funktionierenden Teillösungen ein großes komplexes Softwaresystem erstellen.



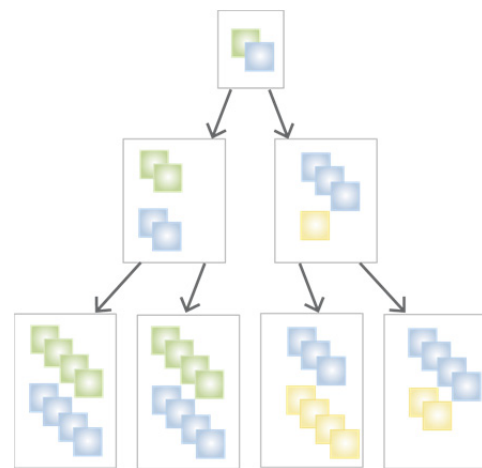
Es gibt Wissenschaftler, die behaupten, dass dies der ideale Weg ist, um komplizierteste Systemlösungen erstellen zu können. Der Aufbau des Atoms ist ein Beispiel aus der Natur für die hier angeführte Behauptung.

OO-Prinzipien erlauben die Nachbildung der Realität, die durch das zu erstellende Softwaresystem abgebildet werden soll. Sie erreichen eine robuste Software, wenn Sie aus mehreren kleinen funktionierenden Einheiten ein komplexes System zusammenstellen. Die Wartung und Weiterentwicklung dieses komplexen Systems ist relativ leicht zu bewerkstelligen, da es sich wieder zerlegen lässt.

Eine Korrektur oder Funktionsänderung wird in diesen kleinen Einheiten ausgeführt und nach erfolgreichen Tests wieder in das Gesamtsystem integriert. Da sie sich in beliebigen Stufen zu höheren Strukturen zusammenfügen lassen, ist die Wiederverwendung in unterschiedlich komplexen Konfigurationen möglich.

Prinzip der Abstraktion

Beim Vorgang der Abstraktion werden gemeinsame Objekteigenschaften und Zugriffsoperationen ermittelt und in einer Klasse zusammengefasst. Gleiche Eigenschaften und Verhaltensweisen verschiedener Klassen können in übergeordneten Klassen zusammengefasst werden. Es entstehen Klassenhierarchien.

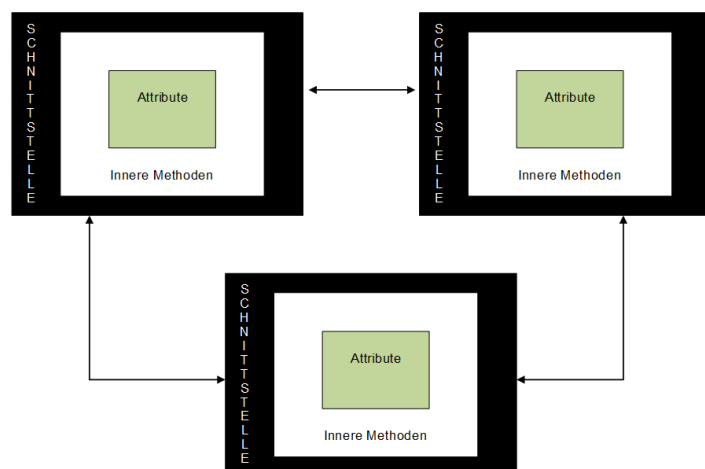


Klassenhierarchie

Prinzip der Kapselung

Die Objekte der Klassen sind die Bausteine, mit denen das OO-Prinzip praktisch umgesetzt wird. Die Objekte einer Klasse erbringen Teillösungen, die durch die Verbindung mit den Lösungen der anderen beteiligten Objekte des Systems eine Lösung für das Gesamtsystem hervorbringen.

Durch die Kapselung (Trennung) des inneren Klassenkerns von der äußeren Schnittstelle werden Klassen mit gleicher Schnittstelle austauschbar.



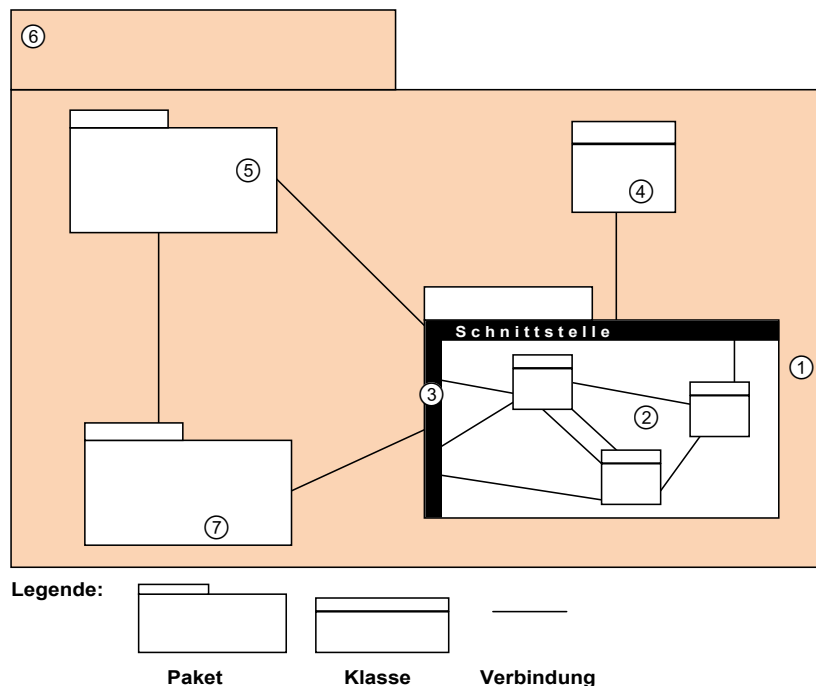
Die objektorientierten Sprachen ermöglichen eine klar abgegrenzte Implementierung der Schnittstellenmethoden und der inneren Methoden des Klassenkerns.

Prinzip der Modularität

Da Klassen nicht nur an Kindklassen vererben, sondern auch selbst Objekte anderer Klassen enthalten können, ist es möglich, mehrere verbundene Objekte in einer neuen, diese Objekte umgebenden Klasse zusammenzufassen. Diese Klasse kann ihrerseits als Baustein für ein größeres System dienen. So lassen sich aus kleineren Bausteinen Komponenten für eine bestimmte Aufgabe kombinieren und mehrere solcher Komponenten wiederum in einem übergeordneten Modul zusammenfassen.

Grafische Darstellung in der UML

Die folgende Abbildung zeigt die Zusammenfassung von Klassen in übergeordnete Einheiten, die Pakete ①, ⑤, ⑥, ⑦. Die Linien symbolisieren die möglichen Kommunikationen zwischen den Klassen eines Paketes ②. Das Paket ① besitzt eine Schnittstelle ③, über die Klassen ④, die nicht zum Paket gehören, und andere Pakete ⑤, ⑦, die die Funktionalität des Paketes ① nutzen, kommunizieren können. Die gesamte Struktur ist in ein übergeordnetes Paket ⑥ integriert.



3.3 OO-Techniken

OO-Techniken sind die Mittel, die von objektorientierten Programmiersprachen bereitgestellt und vom Software-Designer zum Erstellen eines objektorientierten Softwaresystems eingesetzt werden können. Durch den fachgerechten Einsatz dieser Werkzeuge ist es möglich, die Vorteile der objektorientierten Systemmodellierung (z. B. einfache Wartung und die Wiederverwendung von Quellcode) in der erstellten Software sicherzustellen.

Vererbung

Unter **Vererbung** wird die Fähigkeit einer Klasse verstanden, Eigenschaften und Methoden automatisch von einer anderen Klasse zu übernehmen. Die vererbende Klasse wird als Basisklasse oder Oberklasse bezeichnet, die ererbende Klasse als abgeleitete Klasse oder Unterklasse.

Beispiel

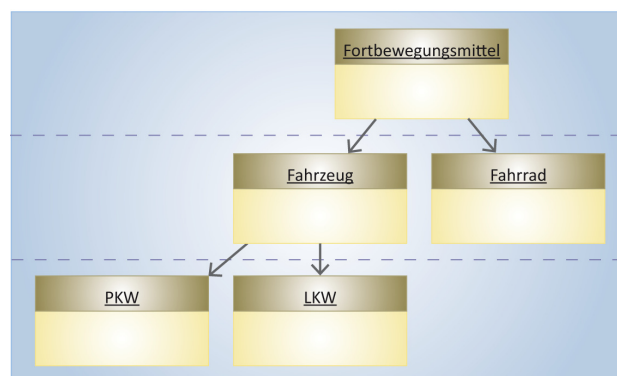


Entwurf verschiedener Klassen

Die Klassen in der vorhergehenden Abbildung haben keine direkten Beziehungen untereinander. In der realen Welt sind aber Fahrräder und Fahrzeuge Fortbewegungsmittel, so wie LKWs und PKWs Fahrzeuge sind. Deshalb besitzen auch die Klasse Fahrrad und die Klasse Fahrzeug die Attribute und Methoden der Klasse Fortbewegungsmittel. Ebenso besitzen die Klasse LKW und die Klasse PKW die Attribute und Methoden der Klasse Fahrzeug. So sind die Methoden und Attribute der Klasse Fortbewegungsmittel in allen dargestellten Klassen vorhanden. Die gemeinsamen Methoden und Attribute müssen bei jeder Klasse nicht neu definiert werden.

Die Klassen Fahrrad und Fahrzeug erben die Attribute und Methoden (Grundfunktionalität) von der Klasse Fortbewegungsmittel. Da PKW und LKW zu der Kategorie der Fahrzeuge gehören, erben sie von der Klasse Fahrzeug. Somit erben auch die Klassen PKW und LKW indirekt die Attribute und Methoden von der Klasse Fortbewegungsmittel.

Der Vorteil liegt darin, dass die Basisklasse Fortbewegungsmittel nur einmal definiert wird und alle Unterklassen die Eigenschaften von ihr erben. Sie besitzen damit automatisch die Funktionalität und Struktur der Basisklasse und können in der abgeleiteten Klasse erweitert werden. Änderungen in der Basisklasse werden somit ebenfalls in allen davon abgeleiteten Klassen wirksam.



Bei der Festlegung einer Vererbungsstruktur können Sie einerseits Gemeinsamkeiten existierender Klassen in einer Basisklasse abstrahieren (bottom-up – Generalisierung), andererseits aber auch Klassen einer höheren Abstraktionsebene in Unterklassen spezialisieren (top-down – Spezialisierung).

Auf die vererbten privaten Teile einer Basis(ober)klasse können Sie in einer abgeleiteten Klasse nicht zugreifen.

Vorteile der Vererbung

- ✓ Die Grundfunktionalität der Basisklasse steht für die erbenenden Klassen automatisch zur Verfügung.
- ✓ Änderungen der Basisklasse werden sofort und ohne Zusatzaufwand in den abgeleiteten Klassen wirksam.

Klassifizierung

Die Klassifizierung gestattet das systematische Wiedererkennen von Funktionsmustern. Es gibt Entwurfsmuster (Design Pattern, vgl. Kapitel 6), die Lösungen für oft wiederkehrende Probleme der Programmierung anbieten, wie beispielsweise Musterlösungen für Klassen, die ...

- ✓ mehrere Objekte einer anderen Klasse aufnehmen (Collection),
- ✓ den Zustand einer anderen Klasse überwachen (Observer),
- ✓ Objekte erzeugen (Factory)

können.

Auf den ersten Blick ist die Betrachtung aller möglichen Lösungen schwer darstellbar. Meist kristallisiert sich aber eine überschaubare Anzahl häufig wiederkehrender Lösungsmuster heraus. Die Problemlösung vereinfacht sich dann auf das Wiedererkennen dieser Muster.

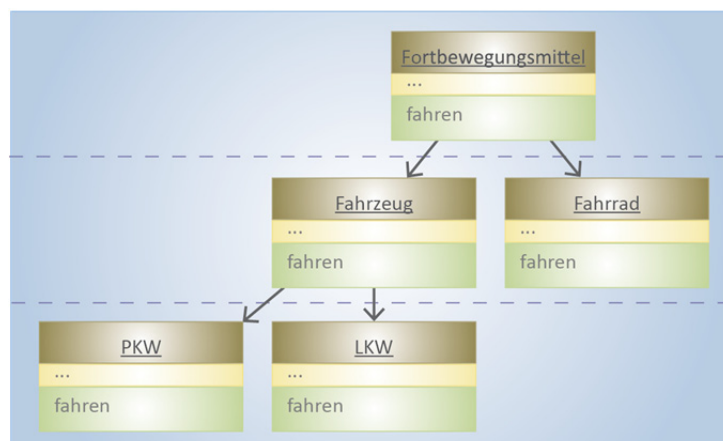
Polymorphie

Polymorphie (Vielgestaltigkeit) beschreibt die Fähigkeit, eine gleichartige Operation mit dem gleichen Namen für Objekte verschiedener Klassen aufzurufen. Beim Aufruf dieser Operation muss nur bekannt sein, dass ein Objekt diese Operation kennt, aber nicht, von welcher Klasse das Objekt erzeugt wurde.

Beispiel

Alle Fortbewegungsmittel können fahren. So fahren Fahrräder ebenso wie PKWs und LKWs, wenn auch auf unterschiedliche Art und Weise.

Von der Oberklasse Fortbewegungsmittel werden zwei verschiedene Fortbewegungsmittel abgeleitet (Fahrzeug und Fahrrad). Fahrzeug ist wiederum die Oberklasse für verschiedene motorisierte Fortbewegungsmittel, wie z. B. PKW oder LKW.



Fortbewegungsmittel können sich von einem bestimmten Ort zu einem anderen Ort bewegen. Deshalb besitzen alle Klassen die Methode `fahren`, die sie von der Klasse `Fortbewegungsmittel` erben. In den Unterklassen werden diese Methoden überschrieben (modifiziert), da jedes Fortbewegungsmittel auf eine andere Art fährt.

Wird die Methode `fahren` beispielsweise für ein Objekt der Klasse `PKW` aufgerufen, wird die Methode `fahren` der Klasse `PKW` ausgeführt. Im Programm können Sie die Methode `fahren` auch auf ein Objekt der Oberklasse `Fortbewegungsmittel` anwenden, ungeachtet dessen, welches Fortbewegungsmittel nun tatsächlich fährt. Erst zur Laufzeit wird entschieden, welche Methode nun tatsächlich aufgerufen wird. Das wird auch als späte oder dynamische Bindung bezeichnet.

Sie schreiben lediglich eine Anweisung, z. B.:

<code>Fortbewegungsmittel Fortbewegungsmittel_X;</code>	→ Objektvariable definieren
<code>Fahrrad einFahrrad = new Fahrrad();</code>	
<code>Fortbewegungsmittel_X = einFahrrad;</code>	
<code>Fortbewegungsmittel_X.fahren();</code>	→ polymorphe Methode fahren aufrufen

Bei der Übersetzung des Quellcodes kann noch nicht bestimmt werden, für welches Objekt welcher Klasse die Methode `fahren` aufgerufen werden soll. Es wird nur festgelegt, dass es ein Objekt einer Unterklasse von `Fortbewegungsmittel` ist. Erst wenn zur Laufzeit der Objektvariablen `Fortbewegungsmittel_X` beispielsweise ein Objekt der Klasse `Fahrrad` zugewiesen wird, steht fest, dass die Methode `fahren` der Klasse `Fahrrad` ausgeführt wird. Wurde der Objektvariablen `Fortbewegungsmittel_X` ein Objekt der Klasse `PKW` zugewiesen, wird die Methode `fahren` der Klasse `PKW` aufgerufen usw.

<code>Fortbewegungsmittel Fortbewegungsmittel_X;</code>	→ Objektvariable definieren
<code>PKW MeinWagen = new PKW();</code>	→ Objekt der Klasse PKW erzeugt
<code>Fahrrad MeinFahrrad = new Fahrrad();</code>	→ Objekt der Klasse Fahrrad definiert und erzeugt
<code>...</code>	
<code>Fortbewegungsmittel_X = MeinWagen;</code>	→ Der Objektvariablen wird ein Objekt der Klasse PKW zugewiesen.
<code>Fortbewegungsmittel_X.fahren();</code>	→ Die Methode fahren der Klasse PKW wird aufgerufen.
<code>...</code>	
<code>Fortbewegungsmittel_X = MeinFahrrad;</code>	→ Der Objektvariablen wird ein Objekt der Klasse Fahrrad zugewiesen.
<code>Fortbewegungsmittel_X.fahren();</code>	→ Die Methode fahren der Klasse Fahrrad wird aufgerufen.
<code>...</code>	

So lassen sich z. B. umfangreiche `case`-Anweisungen einsparen, in denen getestet wird, ob das Objekt nun ein `Fahrrad`, ein `PKW` oder ein `LKW` usw. ist, um die passende Methode aufzurufen.

3.4 Nachrichten

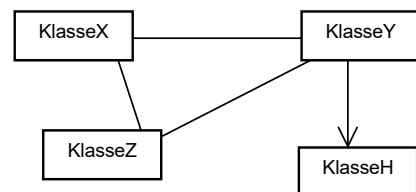
Über Nachrichten kommunizieren die Objekte in einem objektorientierten Softwaresystem. Eine Nachricht wird gesendet, wenn das Sender-Objekt eine Methode der Schnittstelle des Empfänger-Objekts aufruft. Nach der Abarbeitung kann dieses Empfänger-Objekt seinerseits zum Sender-Objekt werden und einem anderen Objekt in Abhängigkeit des Ergebnisses eine Nachricht übermitteln. Dazu ruft es von einem weiteren Objekt (Empfänger-Objekt) eine Schnittstellenmethode auf.

Der erstmalige Anstoß, also das Auslösen einer ersten Methode, erfolgt bei ereignisorientierten Systemen durch einen äußeren Einfluss. Das kann ein bestimmtes Zeitereignis, eine Tasten- oder Mausektion oder das Überschreiten eines Schwellwertes eines angeschlossenen Messgerätes sein. Im Ruhezustand wartet das System auf ein solches Ereignis. Dieser Zustand wird nach einer endlichen Zeit der Abarbeitung wieder eingenommen.

Die Nachrichten zwischen zwei Objekten können unidirektional (nur in einer Richtung) oder bidirektional (in beide Richtungen) gesendet werden.

Grafische Darstellung in der UML

Die Klassen KlasseX, KlasseY und KlasseZ haben bidirektionale Verbindungen untereinander. Die Klasse KlasseY hat eine unidirektionale Verbindung zur Klasse KlasseH und wird mit einem Pfeil gekennzeichnet. Die Klassen KlasseX und KlasseZ haben keine Verbindung zur Klasse KlasseH und somit keine Assoziation, weil keine Assoziationslinie die Klassen verbindet.



3.5 Richtlinien für die Namensvergabe von Bezeichnern

Da objektorientierte Elemente für ihre Adressierung mit Bezeichnern ausgestattet werden und da diese Bezeichner nicht mehrfach vorkommen dürfen, ist eine standardisierte Namensvergabe von Bedeutung. Weil diese Elemente mithilfe der UML in den Diagrammen der Softwareanalyse und des Designs verwendet werden und diese Modelle ohne Namensveränderung der Elemente in eine konkrete Programmiersprache überführt werden sollen, ist eine einheitliche Namensgebung zu verwenden.

Für die Bezeichner können Begriffe (Verben, Substantive, Adjektive) aus dem englischen oder deutschen Wortschatz verwendet werden. Eine Vermischung von englischem und deutschem Wortschatz sollte vermieden werden. Eine Ausnahme hierzu bilden die Lese- und Schreib-Methoden einer Eigenschaft, die in der Regel mit dem Präfix `set` bzw. `get` benannt werden.

Die jeweils eingesetzte Programmiersprache gibt standardmäßig eigene Regeln für die Namensvergabe vor. Diese können in Ausnahmefällen von den hier enthaltenen Richtlinien abweichen.

Allgemeine Regeln

- ✓ Bezeichner sind natürlichsprachliche oder problemnahe Namen oder verständliche Abkürzungen solcher Namen.
- ✓ Jeder Bezeichner beginnt mit einem Buchstaben, der Unterstrich wird nicht verwendet.
- ✓ Bezeichner enthalten keine Leerzeichen. (Verwenden Sie bei den UML-Diagrammen Bezeichner mit Leerzeichen, müssen Sie diese bei der Implementierung in eine konkrete Programmiersprache entfernen.)
- ✓ Generell ist die Groß- und Kleinschreibung zu verwenden.
- ✓ Zwei Bezeichner sollten sich nicht nur durch die Groß- und Kleinschreibung unterscheiden.
- ✓ Es wird entweder die deutsche oder die englische Namensgebung verwendet. Eine Ausnahme sind Internationalismen oder allgemein übliche Begriffe wie pop, push, get und set.
- ✓ Bei Verwendung der deutschen Namensgebung sind ä, ö, ü und ß nicht zu benutzen. (Verwenden Sie diese bei den UML-Diagrammen, müssen Sie sie bei der Implementierung durch die Entsprechungen ae, oe, ue und ss ersetzen.)
- ✓ Jeder Bezeichner, der aus mehreren Wörtern zusammengesetzt ist, wird am Anfang mit einem Großbuchstaben begonnen, z. B. AnzahlEingaben. Dabei werden keine Unterstriche zur Worttrennung verwendet (Ausnahme sind öffentliche konstante Klassenattribute).

Regeln für Klassen

Klassennamen beginnen immer mit einem Großbuchstaben und werden durch ein Subjekt im Singular benannt. Zusätzlich kann ein Adjektiv angegeben werden, z. B. Verzeichnis, SortiertesVerzeichnis.

Gehören die Klassen dem grafischen Nutzerinterface an, kann – zwecks eindeutiger Einordnung – die Vorsilbe GUI vorangestellt werden, z. B. GUIBalken.

Stellt die Klasse eine Struktur dar, die mehrere Objekte aufnimmt (Objektverwaltung), kann dem Klassennamen das Präfix Container, z. B. ContainerAuftragsbuch, vorangestellt werden.

Um Namensgleichheit in der Namensgebung mit den Attributen einer Klasse auszuschließen, können Sie beispielsweise den Klassen den Buchstaben T (wie Typ) oder C (wie Class) voransetzen, z. B. TGUIEingabefeld, CVerzeichnis.

Regeln für Schnittstellen

Die Bezeichner von Schnittstellen – auch Interfaces genannt – entsprechen weitestgehend den Bezeichnern von Klassen. Um diese von den Klassen unterscheiden zu können, wird den Bezeichnern der Buchstabe I vorangestellt, z. B. IVerzeichnis.

Regeln für Objekte

Die Bezeichner von Objekten beginnen mit einem Kleinbuchstaben und enden mit dem Klassennamen. Bei anonymen Objekten entfällt der Klassenname, z. B. ein, a, einSortiertesVerzeichnis, aAuftragsbuch.

Regeln für Eigenschaften von Klassen

Eigenschaften von Klassen (Attribute) erhalten als Bezeichner ein Substantiv oder ein zusammengesetztes Substantiv. Die Kombination aus einem Adjektiv und einem Substantiv ist ebenfalls sinnvoll.

Wenn öffentliche konstante Klassenattribute (z. B. in Interfaces) benannt werden, sollten Sie – zwecks eindeutiger Einordnung – Großbuchstaben verwenden und Interfacenamen mit einem Unterstrich vom eigentlichen Attributnamen trennen, z. B. `IClient_NAME` für das Attribut Name des Interfaces `IClient`.

Regeln für Methoden

Methodennamen bestehen aus einem Verb, das in Kleinbuchstaben geschrieben ist. Diesem Verb kann ein Substantiv folgen, z. B. `drucke`, `zeichne`, `zeichneBild`, `leseAdresse`.

Für einige oft vorkommende Operationen haben sich besondere Methodenbezeichner bewährt.

- ✓ Testet eine Methode den Zustand eines Attributes und gibt als Ergebniswert `true` oder `false` zurück, wird diese Methode mit `istAttributname` benannt, wobei der konkrete Attributname eingesetzt wird, z. B. `istLeer`, `istVerschlossen`, `istSichtbar`.
- ✓ Wenn mit einer Methode ein Attribut gelesen wird, kann für den Namen der Methode der Attributname mit der Vorsilbe `get` verwendet werden, z. B. `getName`, `getFarbe`.
- ✓ Wenn mit einer Methode ein Attribut geschrieben wird, kann für den Namen der Methode der Attributname mit der Vorsilbe `set` verwendet werden, z. B. `setName`, `setFarbe`.

Regeln für Konstanten

Die Bezeichner von Konstanten werden in Großbuchstaben geschrieben. Der Name sollte über den Zweck der Konstanten Auskunft geben.

Haben Sie sich in Ihrem Unternehmen auf eine Namensgebung geeinigt, sollten Sie diese durchgängig in allen Ihren Arbeiten verwenden. Sie ermöglichen damit das schnelle Einarbeiten im Quelltext, der von anderen Mitarbeitern erstellt worden ist.

Die oben aufgeführten Regeln sind Empfehlungen, wie eine solche Namensgebung aufgebaut sein könnte. Die konkrete Situation verlangt gegebenenfalls die Anpassung Ihrer Namensgebung an die, die in einem Programmiersystem bereits standardmäßig verwendet wird, z. B. bei der Entwicklung einer Klassenbibliothek für ein Programmiersystem, die Sie weiterverkaufen wollen.

4

Objektorientierte Analyse – OOA

4.1 OOA-Konzepte – Grundlagen

Etwa 1980 begann die Zeit der objektorientierten Programmierung. Seitdem bilden Objekte, Klassen, Attribute, Methoden, Nachrichten und die Vererbung das Konzept der modernen objektorientierten Softwareentwicklung. Die objektorientierten Grundkonzepte reichten zunächst nicht aus, um ein Fachkonzept problemnah zu modellieren. Das änderte sich 1990, als mit Edward Yourdon und Peter Coad weitere Konzepte aus dem Entity-Relationship-Modell (ER-Modell) und der semantischen Datenmodellierung zur Verfügung standen. Die Grundkonzepte Assoziationen (Beziehungstypen) und Aggregationen (Ist-Teil-von-Beziehung) ergänzten die Methode.

OOA-Konzept

Die objektorientierte Analyse (OOA) enthält die Zusammenführung der folgenden objektorientierten Grundsätze und hat zum Ziel, ein Modell des Einsatzgebietes der zukünftigen Anwendung zu erstellen.

- ✓ Umfangreiche Modelle können in Pakete unterteilt werden.
- ✓ Der Objektlebenszyklus kann durch Zustandsautomaten beschrieben werden, d. h., alle Zustände eines Objekts können mit dem Auslöser des Übergangs von einem Zustand in einen anderen in einem Diagramm eingezeichnet werden (Zustandsdiagramm).
- ✓ Die Funktionsbeschreibungen werden in Geschäftsprozessen abgebildet.
- ✓ Die zeitlichen Abläufe werden in Sequenzdiagrammen modelliert, die die Objekte und die zeitliche Abfolge von Operationsaufrufen darstellen.
- ✓ Die Rollenverteilung und die Hierarchie der beteiligten Objekte werden in Objekt- und Kollaborationsdiagrammen beschrieben (Kollaborationsdiagramme sind grafische Darstellungen von Objekten und ihrem Zusammenwirken).

4.2 Analyse

Zielbestimmung

Ziel der objektorientierten Analyse ist das Erstellen eines Modells eines neuen Softwaresystems mithilfe objektorientierter Konzepte.

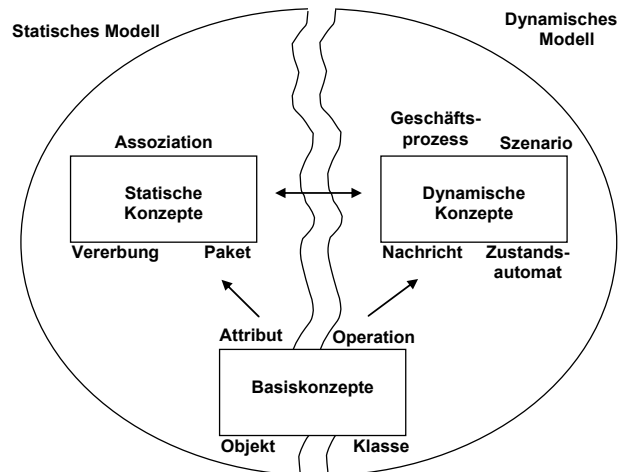
Voraussetzung

Voraussetzung für die Erstellung eines OOA-Modells ist ein vorliegendes Pflichtenheft, in dem das Fachkonzept und die Anforderungsspezifikation beschrieben sind.

Analyseprozess

Die durch das OOA-Modell entstehende Abbildung des realen Prozesses, in dem die Anwendung integriert werden soll, besteht aus einem dynamischen und einem statischen Modell.

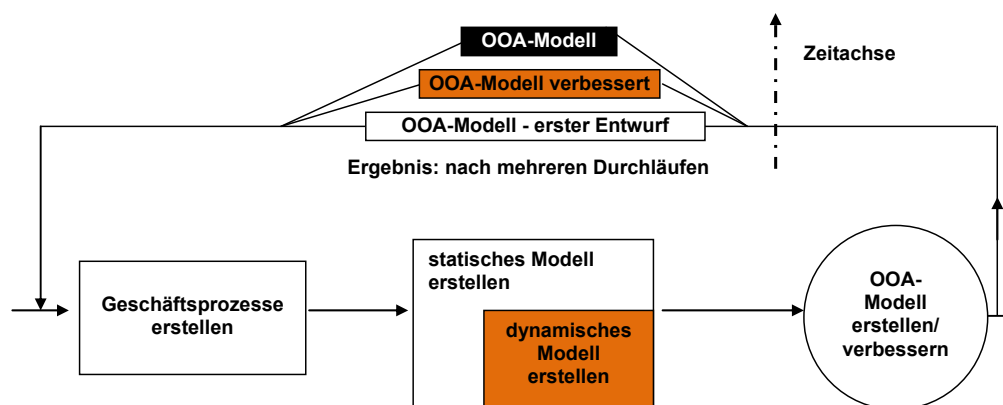
- ✓ Das statische Modell enthält die Beschreibung aller Klassen und Objekte und deren Herkunft (z. B. Basis-klassen) und Gruppierung (Pakete).
- ✓ Das dynamische Modell enthält die Zustandsbeschreibungen, deren Übergänge und Auslöser, d. h., die sich ständig ändernden Objekt-zustände werden beschrieben.



Der Umfang beider Modelle im Verhältnis zueinander hängt von der Art der abzubildenden Prozesse ab. Bei der Modellierung einer Datenbank-Anwendung wird beispielsweise die statische Modellkomponente überwiegen, und bei Systemen mit stark ausgeprägtem interaktiven Charakter (z. B. Informationssysteme) wird sich das Verhältnis umkehren.

Der evolutionäre iterative Entwicklungsprozess

Durch das mehrfache Durchlaufen der Analyseprozesse entsteht ein immer detaillierteres Modell des zu erstellenden Softwaresystems. Deshalb wird von einem evolutionären, iterativen Entwicklungsprozess gesprochen. Die Wiederholung wird beendet, wenn ein aussagekräftiges Modell entstanden ist, das für den nachfolgenden objektorientierten Softwareentwurf (OOD – Object Oriented Design) geeignet ist, und umgesetzt werden kann.



4.3 Prozesssteuerung

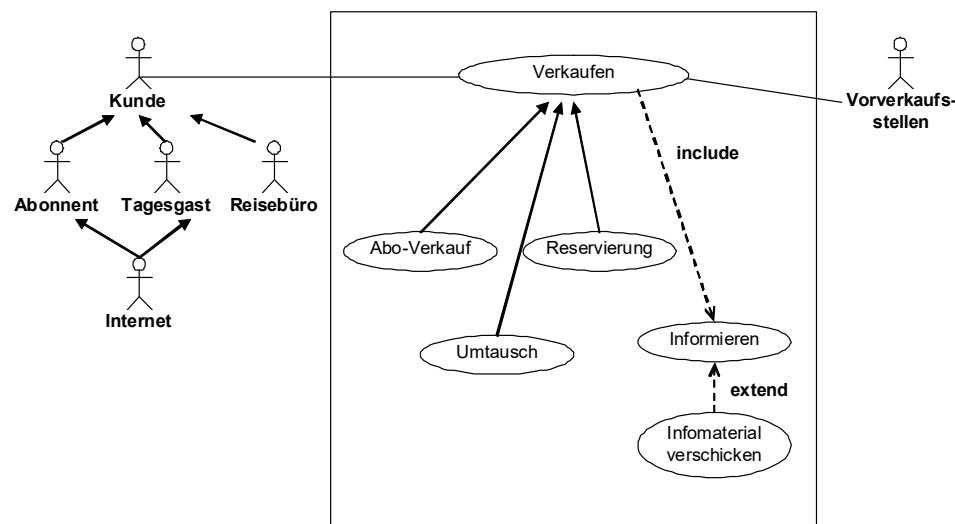
Um die Ergebnisse der Analyse zu sichten und aus dem Ergebnis ein Modell zu generieren, ist die Steuerung des Analyseprozesses notwendig. Diese Steuerung wird sowohl in der Analyse im „Großen“ (Makroprozess) als auch in den beiden „kleinen“ Analyseprozessen ausgeführt, die sich nach den Betrachtungen des zu modellierenden Prozesses in eine statische und in eine dynamische Analyse aufteilen.

Balancierter Makroprozess

Der balancierte Makroprozess übernimmt die Steuerung der Modellierung. Dabei werden vom Makroprozess mehrere Aufgaben übernommen.

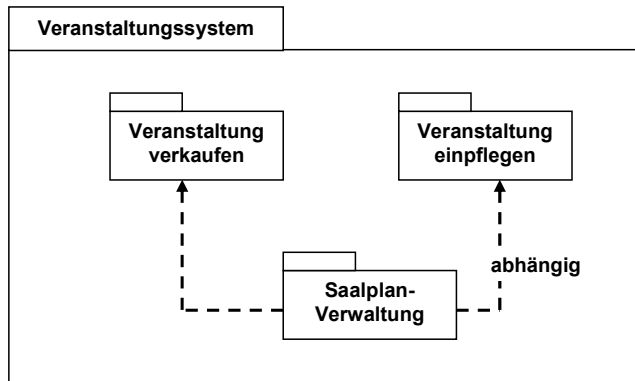
- ✓ Zuerst werden die essenziellen Geschäftsprozesse ermittelt. Essenzielle Geschäftsprozesse sind Geschäftsprozesse, bei denen die für die Umsetzung der Anwendung wichtigen Details herausgefunden und alle für die Anwendung nicht relevanten Details entfernt wurden.
- ✓ Anschließend werden verbale Beschreibungen der gefundenen Geschäftsprozesse erstellt.
- ✓ Aufbauend auf den relevanten Geschäftsprozessen wird eine grafische Beschreibung in Form eines Geschäftsprozessdiagramms erarbeitet.

Die folgende Abbildung zeigt ein vereinfachtes Geschäftsprozessdiagramm für ein Ticketsystem (Kartenverkauf in Theatern).



- ✓ In der zweiten Phase des Makroprozesses werden Teilsysteme lokalisiert und in Pakete zusammengefasst. Bei der Entwicklung großer Systeme, die von mehreren Teams bearbeitet werden, stellt die Aufteilung in Pakete die Grundlage für die Verteilung der Verantwortlichkeiten auf die einzelnen Teams dar. In diesem Fall muss die Aufteilung an den Anfang des Makroprozesses gesetzt werden. Im Ergebnis dieser Aufteilung wird ein Paketdiagramm erstellt, das die Aufteilung grafisch darstellt.

Paketdiagramme abstrahieren Details und verdeutlichen die übergeordneten Strukturen.

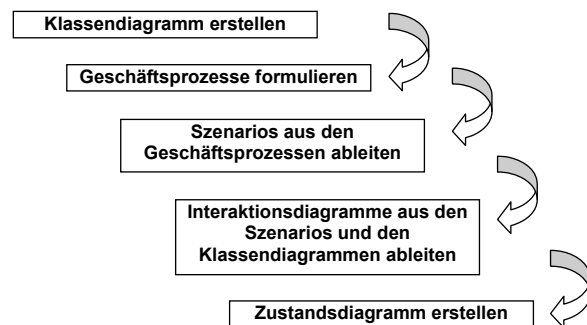


Auf die Gesamtaufgabe bezogen gewährleistet der Makroprozess die Koordinierung des Programmiererteams und deren Ergebnisse und sorgt dafür, dass die Entwicklung durch die Einarbeitung von neuen, während des Erstellungsprozesses gefundenen Erkenntnissen fortschreiten kann. Der Makroprozess erstellt auch die Anweisungen zur Wiederholung eines Schrittes, wenn das Ergebnis unbefriedigend ist oder aufgrund der Sachlage nicht mehr der aktuellen Situation entspricht. Somit kann schon frühzeitig eine Fehlentwicklung erkannt und der geplante Zeitrahmen eingehalten werden.

Daten-basierter Makroprozess

Eine Alternative zu dem gerade vorgestellten Prozess ist der Daten-basierte Makroprozess. Dieser Prozess ist dem oben genannten überlegen, wenn mindestens einer der folgenden Punkte erfüllt ist:

- ✓ Es ist ein umfangreiches Datenmodell vorhanden.
- ✓ Alte Datenbestände sollen in das neue Softwareprodukt einbezogen werden.
- ✓ Der Gesamtumfang der funktionalen Anforderungen konnte noch nicht abschließend geklärt werden.
- ✓ Das zu erstellende System ist ein Auskunftssystem, das später flexibel gestaltete Anfragen bearbeiten muss.



In diesem Makroprozess werden die dargestellten Arbeitsschritte abgearbeitet. Die Diagrammarten werden im Kapitel 7 beschrieben.

Statische Analyse

Im Rahmen der statischen Analyse sollten Sie nachfolgende Arbeitsschritte durchführen:

- ▶ Identifizieren Sie Klassen und erstellen Sie eine Kurzbeschreibung und ein Klassendiagramm für diese Klassen.
- ▶ Identifizieren Sie für jede Klasse nur so viele Operationen und Attribute, wie für das Verständnis des umzusetzenden Systems notwendig sind.

- ▶ Finden Sie Assoziationen, indem Sie zunächst nur die Verbindungen eintragen. Ergänzen Sie anschließend die Darstellung durch Hinzufügen der jeweiligen Kardinalität und der Art der Verbindung.
- ▶ Finden Sie alle Attribute, die zur fachlichen Umsetzung der Anwendung benötigt werden.
- ▶ Erstellen Sie eine Vererbungsstruktur auf der Grundlage der in den vorangegangenen Punkten zusammengetragenen Informationen. Achten Sie darauf, Klassen mit gemeinsamen Attributen und Operationen von einer gemeinsamen Basisklasse abzuleiten.
- ▶ Erweitern Sie die bereits eingetragenen Assoziationen und bestimmen Sie deren Typ (z. B. Assoziation, Aggregation, Komposition). Klären Sie die Aufgaben der Klassen und legen Sie Namen und Restriktionen fest.
- ▶ Klären Sie für alle Attribute die vollständigen Eigenschaften, wie die Schnittstellen-zugehörigkeit und den Typ.
- ▶ Suchen Sie in dem Klassendiagramm nach bereits bekannten Analysemustern (vgl. Abschnitt 4.4) und modellieren Sie die betreffenden Teile durch die Vorgaben der Muster.

Die aufgezeigten Schritte stellen eine mögliche Abfolge dar, die bedingt durch den konkreten Sachverhalt variieren kann. Da sich die Schritte nicht ausschließen, können Sie z. B. Muster bereits bei der Suche nach Klassen finden, wenn Sie die Assoziationen der Klassen bearbeiten oder wenn Sie die Vererbungshierarchie erstellen.

Klassenbeziehungen

Mit Klassenbeziehungen können Sie die Anzahl der Instanzen einer Klasse festlegen, die mit einer zweiten Klasse und deren Instanzen in Verbindung stehen sollen. Außerdem lässt sich die Art, wie die Klassen zueinander stehen, modellieren. Sie erarbeiten beispielsweise, ob durch die Verbindung die Attribute der Klasse erweitert werden oder ob eine Klasse Objekte einer zweiten Klasse verwalten soll.

Sie finden typische, oft benötigte Strukturen in den Design-Mustern wieder (vgl. Kapitel 6). Diese Beziehungen werden später in der OOD-Phase (objektorientierter Entwurf) weiter spezifiziert.

Hier wird auch festgelegt, ob eine Klasse nur die Operationen einer anderen Klasse aufruft (unidirektionale Verbindung) oder ob dies auch umgekehrt (bidirektionale Verbindung) erfolgt.

Referenz- und Nutzungsbeziehungen

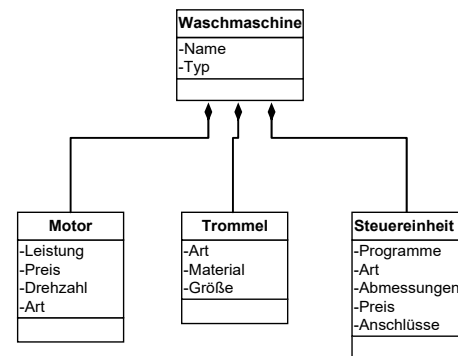
Während bei der Vererbung Klassen erweitert werden, sind die Nutzungsbeziehungen konkrete statische Beziehungen zwischen zwei beteiligten Objekten. Die Beziehungen sind als Beziehungen zwischen den Klassen der Objekte realisiert. Assoziationen werden unterschieden:

- ✓ nach der Anzahl der Objekte, die an der Assoziation beteiligt sind (Kardinalitäten);
- ✓ danach, ob die Assoziation nur existiert, wenn das zweite beteiligte Objekt bereits erzeugt wurde (existenzgebundene Beziehung);
- ✓ danach, ob die Beziehung zur Laufzeit aufgebaut werden kann (nichtexistenzielle Beziehung). Die minimale Kardinalität beträgt 0.

Assoziation, Aggregation, Komposition

Wenn Klassen miteinander kommunizieren, haben sie eine bestimmte Beziehung zueinander. Dabei gibt es mehrere Möglichkeiten, die bei der Erstellung von Software modelliert werden müssen.

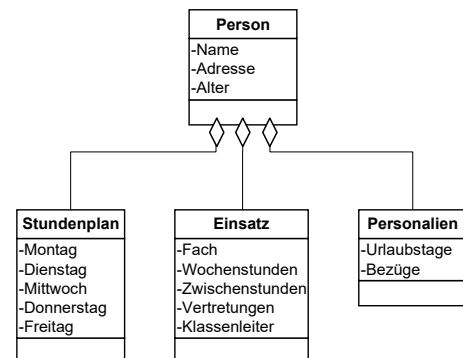
Beispielsweise können Klassen die Verwaltung von anderen Klassen übernehmen. Anders betrachtet besteht die Klasse aus Einzelteilen, die Objekte einer anderen Klasse sind. Dies ist ähnlich dem Baugruppenprinzip bei der Produktion einer Waschmaschine, die z. B. aus dem Motor, der Trommel, der Laugenpumpe und der Steuereinheit besteht. Trennen Sie einen Teil ab, ändert sich das gesamte System, es ist keine funktionierende Waschmaschine mehr.



Komposition

Eine Beziehung zwischen Klassen entsteht auch bei der Erweiterung der Eigenschaften einer Klasse mittels einer zweiten Klasse. Dabei enthält eine Klasse die Attribute, die die statischen Eigenschaften eines Objekts aufnimmt. Durch die Verbindung zu einer weiteren Klasse können in deren Attributen die Eigenschaften gespeichert werden, die sich verändern oder die in verschiedenen Einsatzbedingungen unterschiedlich genutzt werden.

Möchten Sie z. B. in einer Schule einen Lehrer einstellen, werden von ihm zunächst die persönlichen Daten erfasst. Über diese Daten wird er als Lehrer der Schule identifiziert. Diese Angaben sind die statischen Eigenschaften. Es gibt aber noch weitere Eigenschaften, die ihn bezüglich des Stundenplanes charakterisieren, z. B. die Stundenzahl oder die Klassenleitertätigkeit, die sich in jedem Schuljahr ändern. Diese Informationen werden in eine gesonderte Klasse verlagert, die über eine Beziehung mit der Klasse, die die statischen Informationen enthält, verbunden ist.



Aggregation

Für das Lehrerbeispiel kann z. B. der benötigte Speicher dadurch reduziert werden, dass ein Lehrerobjekt (mit den statischen Informationen) mit mehreren Objekten verbunden werden kann, die Informationen über die Schulklassen enthalten, die er unterrichtet.

Diese verbundenen Klassen enthalten keine Informationen zum Lehrer. Werden die statischen Informationen benötigt, können diese über die Assoziation zwischen dem Lehrer- und dem Personalienobjekt ermittelt werden.

Es gibt noch weitere Beziehungen zwischen Klassen und deren Objekten, die in der Programmierpraxis oft zu implementieren sind. Sie finden eine Auswahl dieser Beziehungen und deren Modellierung in den Entwurfsmustern (Design Pattern) wieder (vgl. Kapitel 6).

Dynamische Analyse

Szenarios

Ein Szenario ist ein konkreter Ablauf in einem Anwendungsfall. Szenarios können in einem Sequenzdiagramm, Aktivitätsdiagramm oder auch in Textform beschrieben werden. Szenarios modellieren das Verhalten eines Systems in einer bestimmten Situation. Sie können mit Szenarios feststellen, ob alle für den Anwendungsfall benötigten Operationen angelegt wurden.

Zustandsautomaten

Zustandsautomaten modellieren das System aus dynamischer Sicht. Dabei wird der Lebenszyklus eines Objekts untersucht. Im entsprechenden Zustandsdiagramm werden alle möglichen Zustände des Objekts mit den für den Wechsel zuständigen Operationen grafisch dargestellt.

Mit Zustandsautomaten können Sie kontrollieren, ob alle für den Anwendungsfall benötigten Zustände von einem Objekt eingenommen werden können und ob die Übergänge durch die vorgesehenen Operationen stattfinden.

Operationen

Operationen sind Nachrichten, über die sich die Klassen und deren Objekte mit anderen Objekten verständigen. In der Realität wird die Kommunikation durch wechselseitige Aufrufe von Schnittstellen-Methoden erreicht. Über diese Operatoren werden Objekte zu einer funktionierenden Einheit verbunden, die eine Wiederverwendbarkeit dieser Klassen durch eine flache (lose) Bindung ermöglicht.

In der dynamischen Analyse werden diese notwendigen Kopplungen analysiert, die Bedingungen für diese Kopplungen herausgefunden und die Vorbedingungen für den Operationsaufruf bestimmt. Weiterhin werden die zu übergebenden Parameter definiert. Nach dem Ausführen der Operation wird ein Ergebnis erwartet. In der Nachbereitung der Operation werden die möglichen Ergebniswerte ermittelt und auf Gültigkeit geprüft.

4.4 OOA-Musterlösungen

Ein OOA-Muster ist ein Ausschnitt aus vorhandenen OOA-Modellen (eine Idee, einen Sachverhalt zu modellieren), dessen Implementierung sich in der Praxis bereits bewährt hat und entsprechend als standardisierte Lösung angesehen werden kann.

Sie haben die Möglichkeit, sowohl allgemeine Muster als auch Muster zu verwenden, die für spezielle Szenarien vorgesehen sind und deren Einsatz nur dort sinnvoll erscheint.

Ein Muster besteht aus einer Gruppe von Klassen, die festgelegte Verantwortlichkeiten und Interaktionen besitzen.

Mit dem Einsatz von Mustern tritt der Erstellungsprozess bereits in die Phase des OOD (objektorientiertes Design) über. Der Übergang von OOA zur OOD ist fließend und die Abgrenzung beider Prozesse wird durch die Zielsetzung und nicht durch die Modellierung bestimmt.

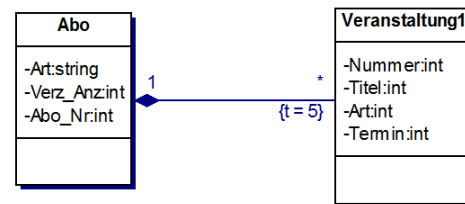
In den folgenden Abschnitten werden typische Standardmusterlösungen (allgemeine Muster) vorgestellt.

Die Liste

Eine Liste wird als Komposition modelliert. Sie liegt vor, wenn ein Ganzes aus gleichartigen Teilen (gleicher Typ) besteht. Es gibt somit nur eine Teil-Klasse, von der die Objekte des Ganzen abstammen. Teilobjekte bleiben dem Aggregat-Objekt zugeordnet, können jedoch gelöscht werden, auch wenn das Aggregatobjekt weiter besteht. Die Attributwerte des Aggregat-Objekts haben für die zugehörigen Teil-Objekte Gültigkeit. Das Aggregat-Objekt enthält mindestens ein Teilobjekt.

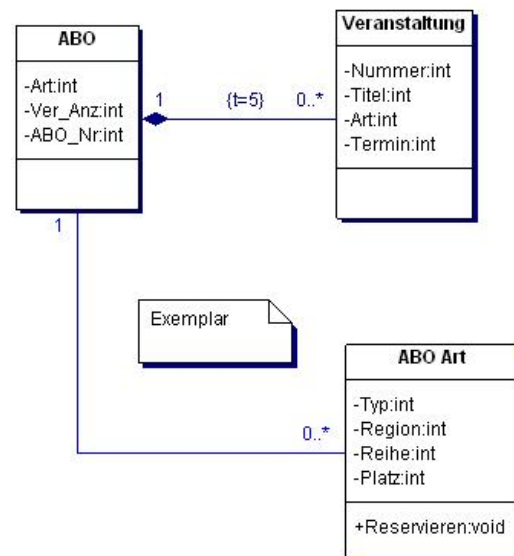
Beispiel

Über ein Kartenverkaufssystem für ein Theater werden in einem Abo fünf Veranstaltungen kombiniert und zusammen verkauft. Für die fünf Veranstaltungen werden jeweils Einzelplätze reserviert. Deshalb kann ein Abo als eine Liste von Einzelkarten oder von beliebigen Einzelplätzen betrachtet werden. Die Anzahl fünf (Zusicherung {t= 5}) ist willkürlich gewählt und wird z. B. durch den Abo-Typ näher bestimmt.



Exemplartypen

Ein Exemplartyp verhindert das mehrfache Speichern gleicher Attributwerte (überschneidender Eigenschaften) in den Objekten. Dieses wird durch die Auslagerung von Attributen in eine neue Klasse erreicht. Die beiden Klassen stehen in einer einfachen Assoziation zueinander, die keine Ist-Teil-von-Beziehung ist. Wurde eine Objektverbindung hergestellt, bleibt diese erhalten, bis das betreffende Exemplar (also beide in Beziehung stehenden Objekte) entfernt wird. Die neue Klasse enthält z. B. Typmerkmale, Gruppenmerkmale, Beschreibungen (Spezifikationen) des realen Exemplars.

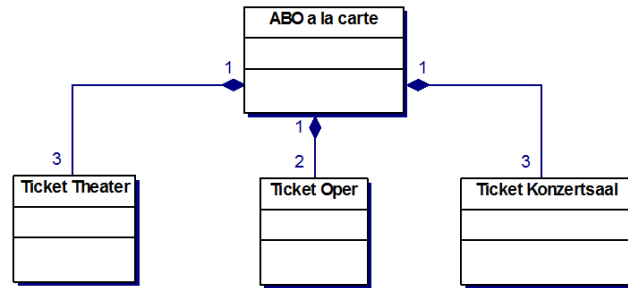


Beispiel

Bezug nehmend auf das vorangegangene Beispiel charakterisieren Abos gemeinsame Attribute, z. B. die Anzahl der Vorstellungen, den Preis und den Zeitraum. Es gibt aber auch Abo-Typen, die eine ganz andere Art von Abo darstellen. Ein solches ist das Platzmieten-Abo. Es berechtigt den Besitzer, für die im Abo verankerten Veranstaltungen immer den gleichen Sitzplatz belegen zu können. Es muss also zusätzlich ein Attribut eingefügt werden, in dem als Wert der Sitzplatz eingetragen ist. Sie haben damit zwei Exemplare der Abo-Klasse vorliegen. Ebenso sind auch mehrere Zusatzinformationen möglich.

Baugruppen

Eine Baugruppe liegt vor, wenn ein System aus physischen Teilen besteht. Sie wird dann über eine Komposition modelliert. Objektverbindungen bestehen meist über einen längeren Zeitraum. Ein Teil-Objekt kann jedoch von seinem Aggregat-Objekt getrennt werden und einem anderen Ganzen zugeordnet werden. Ein Ganzes kann aus unterschiedlichen Teilen bestehen.

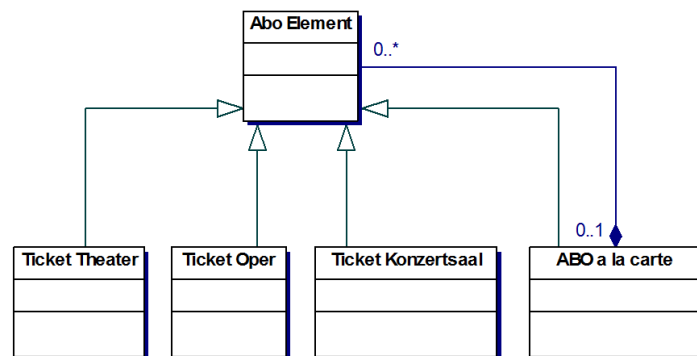


Beispiel

Ein besonderes Abo ist ein Abo, das Veranstaltungen der großen Kulturhäuser einer Stadt in einem Paket anbietet. Im Beispiel werden mit dem Abo „à la carte“ jeweils drei Veranstaltungen des Theaters, zwei Veranstaltungen der Oper und drei Veranstaltungen des Konzertsals verkauft. Programmtechnisch gesehen handelt es sich um eine Baugruppe. Mit dieser Klasse können Sie beliebig viele Veranstaltungen von beliebig vielen Häusern zu einem Abo zusammenschließen.

Stücklisten

Auch die Stückliste ist eine Komposition und wird als solche modelliert. Sowohl das Aggregat-Objekt als auch die Teil-Objekte können sowohl einzeln als auch als Einheit betrachtet werden. Dabei können die Teil-Objekte auch anderen Aggregat-Objekten zugeordnet werden. Die Kardinalität auf der Seite des Aggregat-Objekts ist 0..1. Ein Objekt der Art A kann sich aus mehreren Objekten der Arten A, B und C zusammensetzen. Ein Sonderfall liegt vor, wenn ein Stück nur aus Objekten einer einzigen Art besteht.

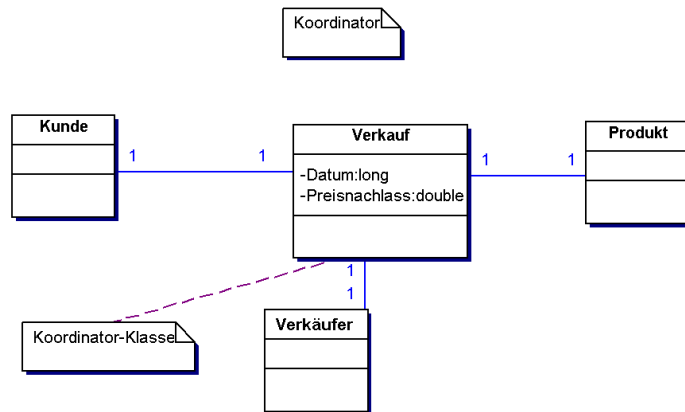


Beispiel

Im Beispiel für die Baugruppe wurde das Abo „à la carte“ modelliert. Dabei wurden Einzelveranstaltungen zu einem Ganzen (Abo) kombiniert. Möchten Sie aber in dem Abo „à la carte“ auch ganze Abos der Häuser neben Einzelveranstaltungen anbieten, ist die Modellierung mit der Stückliste besser geeignet.

Koordinatoren

Der Koordinator reduziert eine n-äre ($n \geq 2$) Assoziation auf eine binäre Assoziation mit zusätzlicher assoziativer Klasse. Diese Koordinator-Klasse besitzt kaum Attribute und Operationen. Dafür besitzt sie mehrere Assoziationen zu anderen Klassen, meist zu genau einem Objekt jeder Klasse.

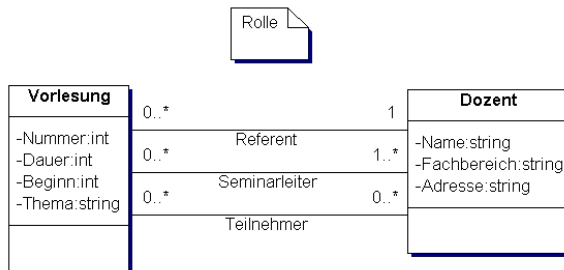


Beispiel

In der rechten Abbildung wird die Modellierung der Koordinatorklasse Verkauf gezeigt. Sie referenziert die Klassen Kunde, Produkt und Verkäufer und stellt somit das Bindeglied für die Kommunikation dieser Klassen untereinander dar.

Rollen

Dieses Analysemuster liegt vor, wenn zwischen zwei Klassen zwei oder mehrere einfache Assoziationen bestehen. Dabei kann ein Objekt zu einem Zeitpunkt in Bezug auf die Objekte der anderen Klassen verschiedene Rollen vertreten. Die Objekte besitzen, unabhängig von der Rolle, die sie einnehmen, im Wesentlichen die gleichen Eigenschaften und Operationen.

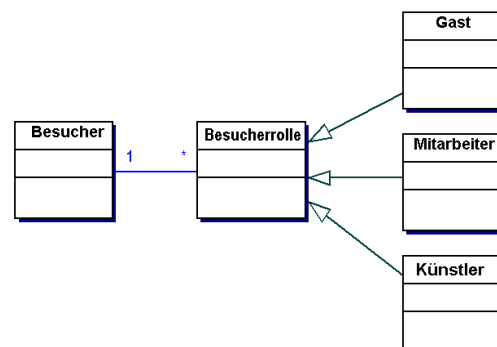


Beispiel

Im Beispiel wird anhand eines Systems zur Verwaltung von Vorlesungen verdeutlicht, dass für einen Dozenten die Buchung einer Vorlesung als Teilnehmer, Referent oder Seminarleiter (auch gleichzeitig als Referent und Seminarleiter) erfolgen kann.

Wechselnde Rollen

Das Muster der wechselnden Rollen unterscheidet sich von dem Analysemuster Rollen dadurch, dass ein Objekt verschiedene Rollen einnehmen kann, jedoch zu einem Zeitpunkt immer nur eine Rolle vertritt. Modelliert wird dieses Muster mithilfe der Vererbung. In jeder Rolle kann es verschiedene Attribute und Operationen geben. Objektverbindungen zwischen dem Objekt und seinen Rollen werden weder gelöscht noch zu anderen Objekten aufgebaut, sie werden nur erweitert.



Beispiel

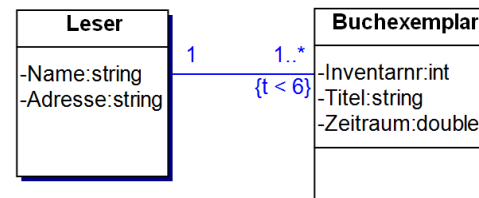
Wechselnde Rollen erhalten Sie, wenn ein Künstler als Besucher eine Veranstaltung besucht. Ist er Mitwirkender, kann er keinen Zuschauer (Gast) für das System vertreten. Ein Stück, in dem er nicht mitwirkt, kann er dennoch als Besucher (Gast) ansehen. Er kann sowohl die Rolle des Gastes (mit Eintrittskarte) als auch die Rolle des Mitarbeiters annehmen, aber in diesem Zusammenhang nicht zur gleichen Zeit.

History

Dieses Analysemuster speichert jede aufgebaute Verbindung von einem Objekt zu anderen Objekten. Dabei wird der Zeitraum für jeden Vorgang festgehalten. Die aufgebauten Verbindungen werden nur erweitert. Eine angegebene Restriktion legt eine Bedingung fest, die zu jedem Zeitpunkt erfüllt sein muss (z. B. $\{t < K\}$), wobei für das Relationszeichen auch Gleichheits-, Ungleichheits- oder weitere Relationszeichen stehen können. In K wird die Anzahl der gleichzeitig aktiven Verbindungen vorgegeben.

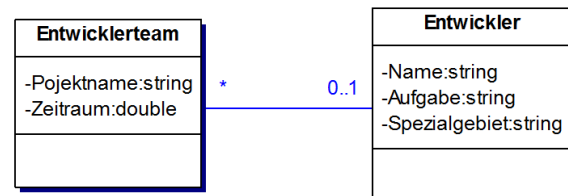
Beispiel

Im Beispiel kann ein Leser sich alle Buchexemplare einer Bibliothek ausleihen. Gleichzeitig kann er aber nur maximal 5 Bücher ausleihen.



Die Gruppe

Ein Gruppenobjekt hat zu einem bestimmten Zeitpunkt ein oder mehrere Einzelobjekte. Sie sind durch eine einfache Assoziation mit dem Gruppenobjekt verknüpft. Objektverbindungen können aufgebaut und gelöscht werden. Es ist zu überprüfen, ob ein Gruppen-Objekt ohne Einzel-Objekte existieren kann oder ob eine Mindestanzahl von verbundenen Einzelobjekten existieren muss.

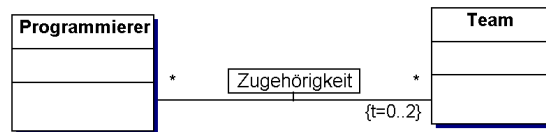


Beispiel

Einem Entwicklerteam sind Entwickler zugeordnet; dabei kann ein Entwicklerteam auch temporär ohne Entwickler existieren. Auch ein Entwickler kann ohne Zugehörigkeit zu einem Entwicklerteam sein.

Die Gruppenhistory

Dieses Muster dokumentiert die Zugehörigkeit von Einzelobjekten zu einem Gruppenobjekt nicht nur zu einem bestimmten Zeitpunkt, sondern über einen Zeitraum. Die zeitlichen Änderungen der Verbindungen werden über eine assoziative Klasse modelliert. Dadurch wird die Zuordnung zwischen Einzelobjekten und den Gruppenobjekten sichtbar. Die zeitliche Restriktion $\{t=0..2\}$ gibt an, wie viele Objekte zu einem Zeitpunkt mit dem Gruppenobjekt in Verbindung stehen. Um die Informationen der Zugehörigkeit dokumentieren zu können, werden aufgebaute Objekt-Verbindungen beibehalten und zusätzlich neue aufgebaut.



Beispiel

Im Beispiel werden die einzelnen zeitlichen Zuordnungen eines Programmierers zu einem Team in jeweils einem Zugehörigkeitsobjekt festgehalten. Mithilfe der Gruppenhistory können Sie hier die unterschiedlichen Teamzugehörigkeiten eines Programmierers nachvollziehen.

5

Objektorientierter Entwurf – OOD

5.1 OOD-Konzepte – Grundlagen

Nach der Analyse der Geschäftsprozesse wird die Transformation in geeigneten Datenstrukturen erarbeitet und in Diagrammen dokumentiert. Im Ergebnis des OOD entstehen Klassendiagramme, die eine Sammlung von Klassen und deren Beziehungen darstellen und gleichzeitig die Basis für die Implementierungsphase bilden.

Da in der OOD die in der OOA gefundenen und beschriebenen Prozesse in konkrete Datenstrukturen überführt werden, ist eine Überarbeitung der OOA-Ergebnisse jederzeit möglich, ohne die bis dahin im Design erreichte Modellierung verwerfen zu müssen. Es kommt zu keinem Bruch in der Darstellung des Modells, weil für die Darstellung die gleichen Elemente verwendet werden (Klassen, Objekte, Beziehungen – nur die Sichtweise in den Phasen ist unterschiedlich).

Mittels spezieller Werkzeuge zur Softwareentwicklung wird automatisch Quellcode auf Grundlage der in der OOA und OOD erstellten Modelle generiert. Das hat den Vorteil, dass Änderungen in der Design-Phase automatisch zu einer vollständig neuen Anwendung führen können, die entsprechend dem veränderten Modell optimiert ist und nicht in die bestehende Anwendung integriert werden muss.

Wenn Sie die gesamte Anwendung aus den Modellen, die Sie mit dem Softwarewerkzeug erstellt haben, generieren wollen, muss das Werkzeug auch den Quellcode für die Operationen verwalten.

5.2 Klassifizierung nach Klassen, Objekten und Attributen

Die objektorientierte Softwareentwicklung realisiert die gesamte Funktionalität über Klassen. Um in den Objekten der Klassen Daten speichern zu können, werden den Klassen Attribute hinzugefügt.

Die Daten können in Klassen durch Kapselung gegen unbeabsichtigte Änderungen geschützt werden. Die objektorientierten Programmiersprachen stellen dazu die Schutzattribute (`public`, `protected` und `private`) bereit. Für die kontrollierte Manipulation der Daten besitzen die Klassen Methoden. Diesen Methoden ist der Zugriff auf die zum Objekt gehörenden Daten erlaubt.

Stellt die Methode eine Funktionalität zu Verfügung, die auch von Methoden anderer Klassen benötigt wird, können Sie die Methode in die Schnittstelle der Klasse (`public`-Bereich) verlagern und somit den Aufruf von allen Objekten erlauben. Diese Schnittstellenmethoden realisieren die Kommunikation der Objekte untereinander.

Benötigen Sie Methoden auch dann, wenn noch kein Objekt der Klasse erstellt wurde, können sie Klasseneigenschaften oder Klassenmethoden verwenden. Die Umsetzung erfolgt in verschiedenen Programmiersprachen auf unterschiedliche Weise. In C++ werden die Klassenelemente mit dem Attribut `static` deklariert; andere Sprachen verwenden den Begriff Metaklasse (z. B. Smalltalk).

5.3 Wiederverwendung

Der Sinn der objektorientierten Anwendungsentwicklung liegt auch darin, einmal entwickelte und in anderen Anwendungen bereits bewährte Programmteile – unverändert – in einer neuen Anwendung einzusetzen. Der Vorteil dieser Möglichkeit besteht im schnellen Einsatz von fehlerfreiem Quellcode und in der hiermit erreichten Zeitersparnis. Allerdings verlangt die Entwicklung und Implementierung von wiederverwendbaren Komponenten die frühzeitige Beachtung der möglichen späteren Einsatzgebiete.

5.4 Klassenbibliotheken

Eine Form der Wiederverwendung stellen Klassenbibliotheken dar. Beispielsweise stehen mit der STL-Standard Template Library (für die Programmiersprache C++) und den .Net-Klassenbibliotheken (für die Programmiersprache C#) derartige Bibliotheken zur Verfügung. Klassenbibliotheken bestehen aus einer Menge wiederverwendbarer Klassen, die allgemeine Funktionalität zur Verfügung stellen, und gelten als objektorientiertes Äquivalent zu prozeduralen Unterprogrammbibliotheken.

Eine Klassenbibliothek erzwingt keine festgelegte Architektur des Anwendungsprogramms. Die Klassen haben eine lose Kopplung oder sind unabhängig voneinander.

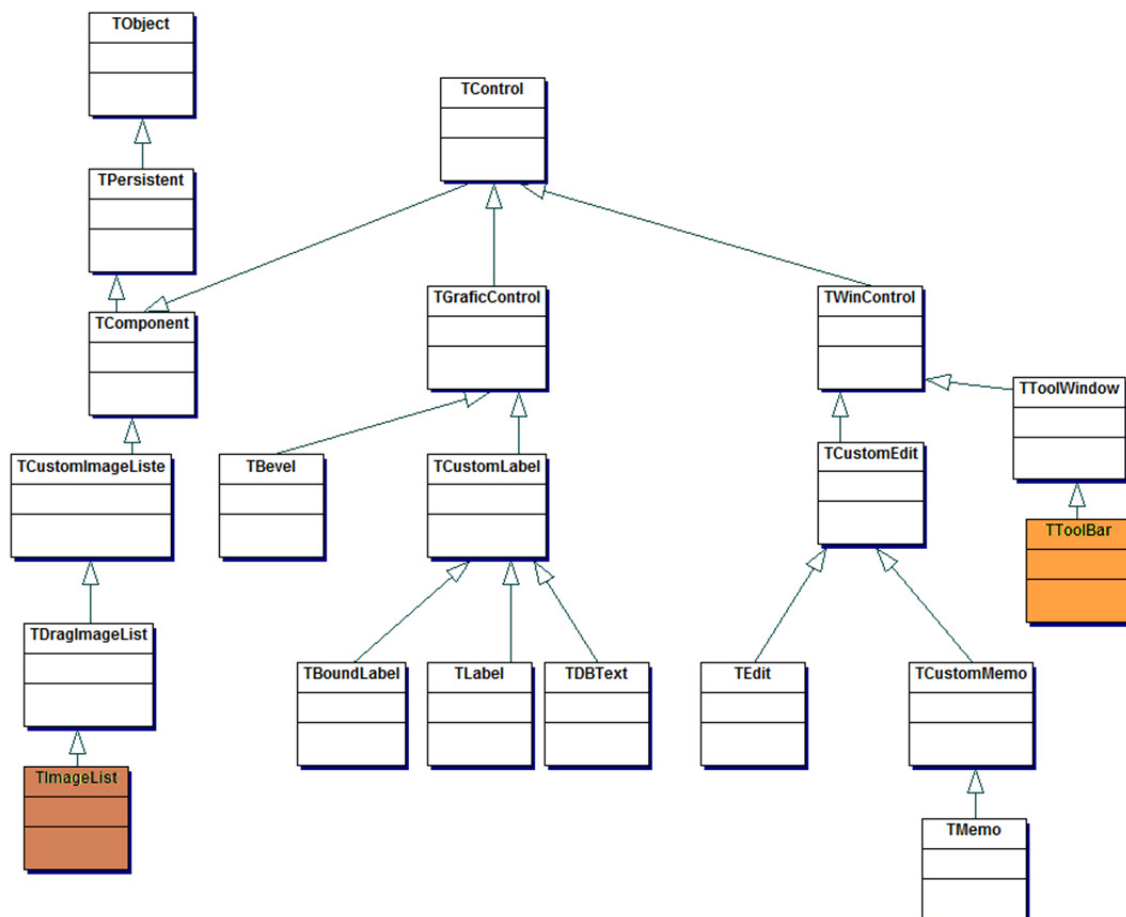
Damit die Klassenbibliotheken sich in neue Anwendungen integrieren lassen, erfordern die Schnittstellen besondere Aufmerksamkeit. Eine Schnittstellenmethode sollte deshalb von Methoden anderer Klassen aufgerufen werden können, ohne das Objekt, das die Methode aufruft, kennen zu müssen.

5.5 Framework

Ein Framework ist eine Menge von kooperierenden Klassen, die einen wiederverwendbaren Entwurf einer bestimmten Art von Software darstellen. Eine Anwendung wird durch die Ableitung von anwendungsspezifischen Unterklassen aus Klassen des Frameworks entwickelt. Dabei diktiert das Framework die Architektur der Anwendung und definiert die allgemeine Struktur, die Aufteilung in Klassen und Objekte sowie deren Hauptverantwortlichkeiten und Kommunikation.

Beispiele für bekannte Frameworks sind die VCL-Bibliothek der Fa. Embarcadero, das .Net-Framework von Microsoft oder das Java-JEE-Framework.

In der Abbildung unten wird ein Teil des Hierarchiebaums des VCL-Frameworks dargestellt. Dabei wird auf die grundlegende Klasse `TObject` aufgebaut. Alle anderen untergeordneten Klassen sind von dieser Klasse abgeleitet und wurden für den entsprechenden Einsatzzweck schrittweise erweitert.



Ausschnitt aus dem VCL-Framework

5.6 Softwarekomponenten

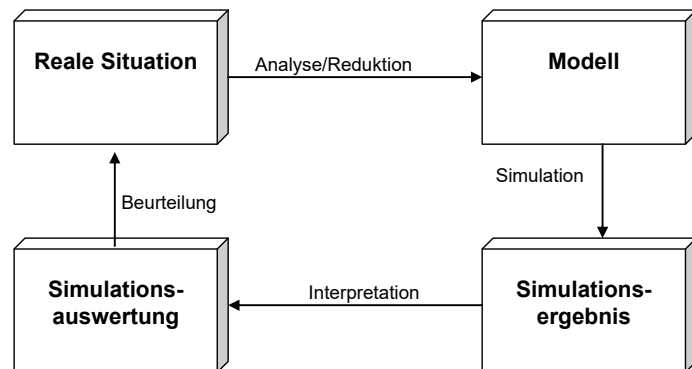
Eine Softwarekomponente ist ein lauffähiges Stück Software, das eine nützliche bzw. gewinnbringend einsetzbare Funktionalität bereitstellt. Beim Einsatz einer Komponente sollten Sie deren Funktionalität so akzeptieren, wie sie bereitgestellt wurde. Dies erfordert gelegentlich Kompromisse, bietet aber dafür eine sofortige Einsatzbereitschaft.

Eine Komponente sollte folgende Kriterien erfüllen:

- ✓ Sie sollte eine möglichst oft benötigte Funktionalität bereitstellen.
- ✓ Die interne Implementierung sollte nicht über die Komponente selbst zugänglich sein. Informationen dazu werden ausschließlich über die Dokumentation geliefert.
- ✓ Sie muss ihre Funktion selbstständig (ohne weiteren Quellcode) erfüllen.
- ✓ Sie sollte einen gewissen Mindestumfang besitzen, damit eine vollständig in sich abgeschlossene Funktionalität bereitgestellt werden kann.
- ✓ Die Funktionalität der Komponente wird ausschließlich über eine verbindliche Schnittstelle angeboten.
- ✓ Eine Komponente ist kooperativ, d. h., sie ist so ausgelegt, dass sie mit anderen Softwareteilen zusammenarbeitet.
- ✓ Sie kann sowohl als Teil ganzer Anwendungen als auch zur Konstruktion neuer größerer Komponenten eingesetzt werden.
- ✓ Sie sollte weitgehend orts- und plattformunabhängig sein, also nicht Bestandteil eines umgebenden Systems.
- ✓ Eine Nutzung der Komponente ist ohne systemspezifische Programmierkenntnisse möglich.
- ✓ Sie bietet mit ihrer Funktionalität keine Speziallösung an, sondern stellt gängige Funktionen bereit.
- ✓ Sie sollte langlebig und nicht nur im Rahmen eines Ausführungskontextes anwendbar sein.
- ✓ Sie sollte umfangreich, geschlossen und klar genug definiert sein, um sie als Einheit an Kunden für deren Weiterverwendung ausliefern zu können.

5.7 Simulation

Mit der Simulation können Sie die Qualität eines Modells bestimmen, das in der OOA und OOD erstellt wurde. Dazu wird eine virtuelle Umgebung geschaffen, in welcher Testwerte der Eigenschaften der Objekte für die Simulation bereitgestellt werden (Ausgangszustand). Nach einem Testlauf werden die Ergebnisse analysiert und anhand dieser Werte wird die Qualität des gefundenen Modells beurteilt.



Simulationsschritte

Arbeitsschritte der Simulation

Arbeitsschritt	Beschreibung
Simulation	Die gewonnenen Ausgangsdaten werden mithilfe des Modells in die Ergebnisdaten (Simulationsergebnis) überführt und der Zustand der betreffenden Objekte während der Umwandlungsphase wird beobachtet.
Interpretation	Nach Ablauf der Simulation werden die Ergebnisse der Simulation aufbereitet und für die spätere Auswertung interpretiert. Dieser Schritt wird auch Simulationsauswertung genannt.
Beurteilung	In diesem Schritt werden die Ergebnisse der Simulation mit den Erwartungswerten verglichen und die Qualität des Modells wird beurteilt.
Analyse/Reduktion	Gehören unerwartete Ergebnisse zum Simulationsergebnis, hat das Modell noch nicht die geforderte Qualität erreicht. Am Modell oder an Teilen des Modells, die die gewünschte Abarbeitung nicht erbracht haben, werden Nachbesserungen vorgenommen. Als Werkzeuge können Methoden der OOA und des OOD verwendet werden.

Um das Modell nachzubessern, sind Informationen aus dem Pflichtenheft herauszusuchen und durch Reduktion für die Einarbeitung in das Modell anzupassen. Das so verfeinerte Modell wird einer weiteren Simulation unterworfen. Dieser Prozess ist evolutionär, da er so lange wiederholt werden kann, bis das erstellte Modell das gewünschte Simulationsergebnis liefert. Konnte die Simulation erfolgreich abgeschlossen werden, geht die Anwendungsentwicklung in die Implementierungsphase über.

Die Simulation muss nicht für das gesamte Modell erstellt und durchgeführt werden, es können auch nur ausgewählte Funktionen bzw. Komponenten des Gesamtmodells betrachtet werden.

6

Einführung in die Entwurfsmustertechnik

6.1 Grundlagen zu Entwurfsmustern

Entwurfsmuster (Design Patterns) sind wiederverwendbare Vorlagen, mit denen häufig wiederkehrende Entwurfsprobleme gelöst werden können, und entstammen erprobten Entwurfserfahrungen. Mit dem Einsatz von Entwurfsmustern erhalten Sie Vorlagen, nach denen Sie für bestimmte Design-Aufgaben eine Lösung ableiten können.

In einem Muster werden folgende Merkmale beschrieben:

- ✓ der Name des Musters;
- ✓ die Problembeschreibung, die mit dem Muster gelöst werden kann;
- ✓ die Lösungsbeschreibung, wie das Muster realisiert wird;
- ✓ die Konsequenzen, die sich aus der Anwendung des Musters ergeben.

Vorzüge von Entwurfsmustern

- ✓ Entwurfsmuster beschreiben Expertenwissen auf dem Gebiet des Softwareentwurfs.
- ✓ Sie sind mehr als Algorithmen und Datenstrukturen.
- ✓ Sie erhöhen die Flexibilität von Entwürfen.
- ✓ Sie erzeugen eine wiederverwendbare Mikroarchitektur.
- ✓ Sie unterstützen die Kommunikation im Entwicklerteam durch ein gemeinsames Vokabular für einen Bereich von Design-Aufgaben.

6.2 Arten

Entwurfsmuster werden in Familien verwandter Muster zusammengefasst. Diese Familien werden nach dem Zweck und dem Geltungsbereich unterschieden.

Der Zweck gibt an, was ein Muster bewirkt.

- ✓ Ein erzeugendes Muster befasst sich mit der Erzeugung von Objekten.
- ✓ Ein strukturelles Muster beschreibt die Anordnung der Klassen und Objekte, um die Design-Aufgabe zu lösen. Sie sind mit einer Schnittstelle ausgestattet, die die Einbindung in das bestehende Klassendesign ermöglichen.
- ✓ Ein Verhaltensmuster beschreibt, wie Klassen und Objekte miteinander kommunizieren und wie die Verantwortlichkeiten verteilt sind.

Geltungsbereich

Der Geltungsbereich gibt an, ob sich das Muster primär auf Klassen oder Objekte bezieht.

Übersicht über bekannte Entwurfsmuster

In der folgenden Tabelle sind die gängigsten Entwurfsmuster mit ihrer jeweiligen Zuordnung und englischen Bezeichnung angegeben. Diese wurden bereits 1995 in dem Buch „Design Patterns – Elements of Reusable Object-Oriented Software“ von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides (genannt Gang of Four) zusammengestellt und beschrieben.

		Zweck		
		Erzeugungsmuster (Creational Patterns)	Strukturmuster (Structural Patterns)	Verhaltensmuster (Behavioral Patterns)
Geltungs- bereich	Klasse	Fabrikmethode-Muster	Adapter (klassenorientiert)	Interpreter Schablonenmethode
	Objekt	Abstrakte Fabrik (Abstract Factory) Erbauer (Builder) Prototyp (Prototype) Einzelstück (Singleton)	Adapter (Wrapper) Dekorierer (Decorator) Kompositum (Composite) Fliegengewicht (Flyweight) Fassade (Facade) Brücke (Bridge) Stellvertreter (Proxy)	Zuständigkeitsketten (Chain of responsibility) Kommando (Command) Iterator (Iterator) Memento (Memento) Beobachter (Observer) Zustand (State) Strategie (Strategy) Vermittler (Mediator) Besucher (Visitor)

6.3 Ziele der Entwurfsmuster

Mithilfe der Muster werden alle bidirektionalen Verbindungen so weit wie möglich durch unidirektionale Verbindungen ersetzt. Dadurch wird eine Wiederverwendung der Klassenstruktur ermöglicht und der Einsatzbereich erweitert; bei einer unidirektionalen Assoziation benötigt nur eine der beiden beteiligten Klassen Informationen über die andere Klasse. Klassenarchitekturen, die unidirektionale Verbindungen verwenden, lassen sich einfacher in Komponenten zerlegen.

Eine derartige Komponente kann über eine Schnittstellenmethode aufgerufen werden, ohne dass die Komponente Informationen über das aufrufende Objekt benötigt. Gleichzeitig wird über die Geheimhaltung der Zugriff auf die innere Bearbeitung verwehrt und somit ein Zielzustand für einen bestimmten Ausgangszustand garantiert.

In den folgenden Abschnitten werden einige ausgewählte Entwurfsmuster vorgestellt.

Singleton

Das Singleton-Muster ist ein sehr einfaches Muster. Dabei verwaltet eine Klasse sich so, dass genau ein Objekt von ihr erstellt wird. Der erste Zugriff auf dieses Objekt erzeugt das Objekt und gibt eine Referenz zurück. Jeder weitere Zugriff verwendet das existierende Objekt und gibt nur die Referenz auf dieses zurück.

Singleton
- Instanz : Singleton
- Singleton () + holeInstanz() : Singleton

Das Singleton-Muster eignet sich z. B. für Objekte, die Funktionalität anbieten, die nur unter bestimmten Bedingungen benötigt wird, oder als objektorientierter Ersatz für globale Variablen. Da das Objekt nur bei der ersten Verwendung erzeugt wird, ist zudem nur ein minimaler Speichereinsatz erforderlich.

Ein möglicher Einsatzbereich dieses Entwurfsmusters ist z. B. innerhalb eines Programms zur Verwaltung von Druckaufträgen auf einem Einzelplatzdrucker, um die jeweiligen Druckaufträge ausschließlich in einen einzigen Datenpuffer auszugeben.

Beispiel: Singleton.cs

Plus Beispieldatei: *Singleton.cs*

Im Beispiel wird das grundlegende Singleton für eine Klasse in einer möglichen Variante und in C# implementiert. Die Implementierung kann auf dieser Basis auch in anderen objektorientierten Programmiersprachen erfolgen. Der jeweilige Aufwand variiert abhängig von der gewählten Programmiersprache und dem entsprechenden Systemumfeld.

```
using System;
public class Singleton
{
    private static Singleton Instanz = null;
    private Singleton()
    {
    }
}
```

```

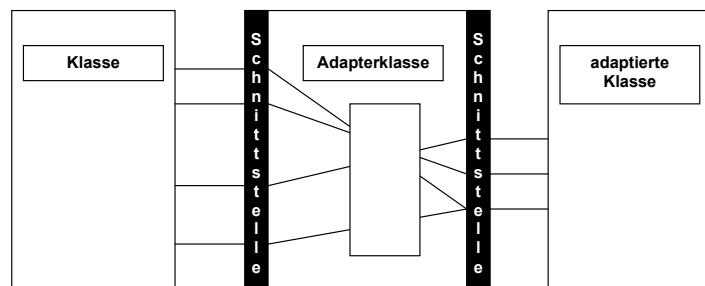
public static Singleton Instance
{
    get
    {
        if (Instanz == null)
        {
            Instanz = new Singleton();
        }
        return Instanz;
    }
}

```

Adapter (Wrapper Pattern)

Ein Adapter realisiert die Anpassung der Schnittstelle einer Klasse an die einer anderen Klasse über eine sogenannte Wrapperklasse.

Das Adapter-Muster wird verwendet, um zwei Klassen mit unterschiedlichen Schnittstellen anzupassen.

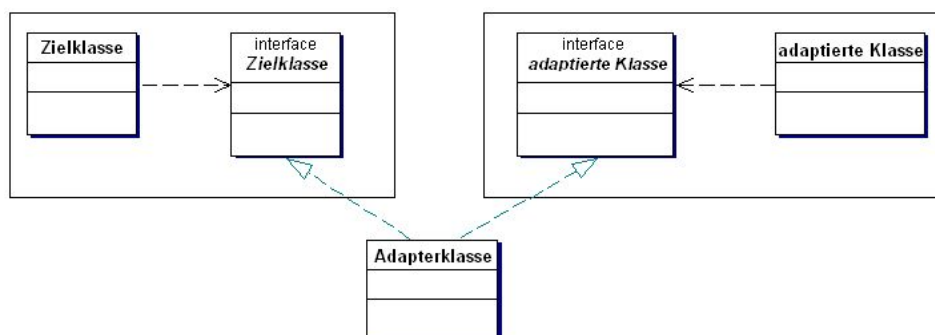


Leiten Sie von der Schnittstellenklasse und von der zu adaptierenden Klasse die Adapterklasse ab und leiten Sie die Schnittstellenaufrufe innerhalb der Adapterklasse an die entsprechenden Methoden der adaptierten Klasse weiter.

Die Adapterklasse kann somit beide Schnittstellen bedienen und verbindet den Informationsfluss intern.

Für Programmiersprachen, die eine Mehrfachvererbung nicht unterstützen, kann die Adapterklasse durch das Einfügen von Schnittstellen (Interfaces) implementiert werden.

Die Abbildung zeigt die mögliche Struktur von Klassen, die das Adapter-Muster bilden. Die Zielklasse bietet ein Interface an, das mit der Zielklasse verbunden ist. Die adaptierte Klasse kommuniziert ebenfalls über ein Interface. Die Adapterklasse erbt von beiden Schnittstellen die Methoden, diese werden in der Adapterklasse so überschrieben, dass die passende Methode des jeweils anderen Interfaces mit der entsprechenden Anpassung aufgerufen wird.



In C++ kann die Verbindung zwischen Interface und Klasse z. B. dadurch erreicht werden, dass eine Klasse von einer abstrakten Klasse abgeleitet wird, die die Schnittstelle definiert. In C# und Java werden Interfaces direkt unterstützt. Dort implementiert eine Klasse ein Interface. Die Adapterklasse implementiert beide Interfaces.

Brücke (Bridge Pattern)

Möchten Sie in einer Klasse eine Grundfunktionalität unterbringen und zusätzlich Operationen nutzen, die getrennt von der Grundfunktionalität verändert (spezialisiert) werden sollen, können Sie die Architektur des Brücken-Musters verwenden.

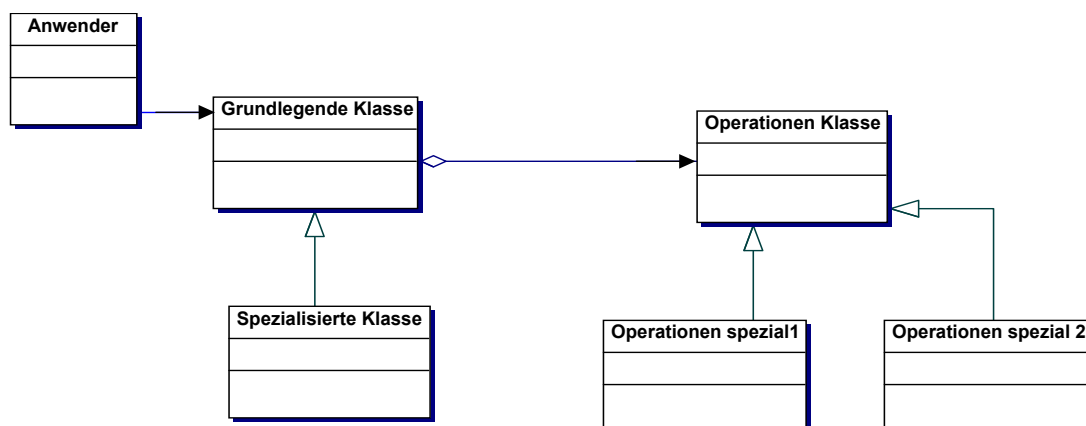
Bei diesem Muster werden die Grundfunktionalität und die Operationen in getrennten Hierarchiestrukturen verwaltet.

Eine solche Lösung benötigen Sie, wenn Sie z. B. ein grafisches Fensterelement für unterschiedliche Betriebssystem-Plattformen erstellen wollen. Eigentlich genügt die Ableitung zur Erstellung einer plattformspezifischen Klasse mit an die Plattform angepassten Methoden von der gemeinsamen Basisklasse.

Diese neue Klasse implementiert die Methoden entsprechend dem Betriebssystem. Diese Struktur wird flexibler einsetzbar, wenn sowohl die plattformspezifische Implementierung als auch die Grundfunktionalität getrennt über Vererbung spezialisiert werden können.

Sie können somit beliebige Kombinationen der für sich spezialisierten Grundfunktionalität und der spezialisierten Operationen vornehmen.

Das Prinzip beruht auf der Aufteilung der Funktionalität in zwei Basisklassen, eine für die Grundfunktionalität und eine für die angepassten Operationen. Die Klasse, die die Grundfunktionalität besitzt, bekommt als Eigenschaft eine Variable, über die ein Objekt verknüpft wird, dessen Klasse die spezifischen Operationen enthält. Diese Anordnung hat zudem den Vorteil, dass Sie zur Programmlaufzeit die Operationen durch Zuweisung eines anderen Objekts austauschen können.



Die Abbildung zeigt eine Klasse **Anwender**, die Methoden der Klasse **Grundlegende Klasse** aufrufen kann. Da die spezialisierte Klasse von dieser Klasse abgeleitet wurde, kann der **Anwender** die in der spezialisierten Klasse überschriebenen Methoden aufrufen.

Durch eine Referenz, die in der Klasse `Grundlegende Klasse` gehalten wird, kann der Anwender auch die Methoden der Klasse `Operationen Klasse` aufrufen. Die jeweilige Funktionalität hängt von der Referenz ab, die in der grundlegenden Klasse verwaltet wird. Es können auch Referenzen auf Objekte sein, die von dieser Klasse abgeleitet wurden (z. B. Klasse `Operationen spezial1`).

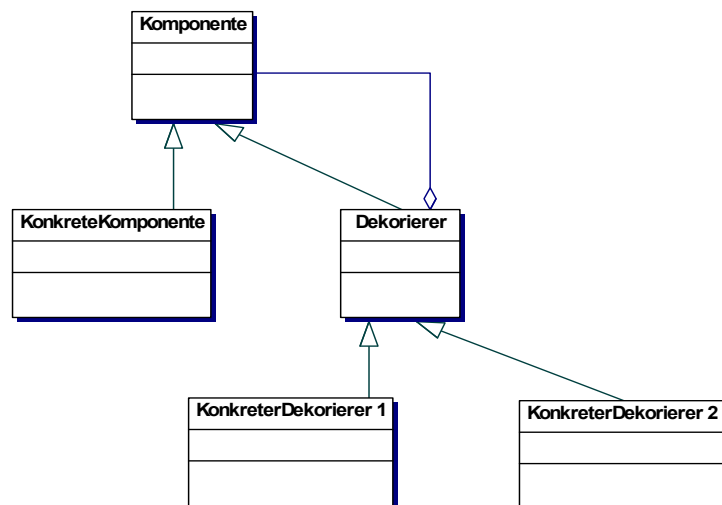
Durch Austausch des in der Referenz verwalteten Objekts kann die Funktionalität (auch zur Laufzeit) geändert werden.

Dekorierer (Decorator Pattern)

Dekorierer bieten eine flexible Alternative zur Unterklassenbildung durch Vererbung, um die Funktionalität einer Klasse zu erweitern. Dieses Muster wird auch als gebundener Umwickler bezeichnet. In diesem Muster nehmen Sie in eine neue Klasse die ursprüngliche Klasse z. B. als Referenz auf und können diese beliebig in der neuen Klasse erweitern.

Erstellen Sie eine neue Klasse, die ein Objekt der Klasse, die erweitert werden soll, verwalten (referenzieren) kann. Sie können innerhalb der neuen Klasse auf die zu erweiternde Klasse zugreifen und deren Eigenschaften über die eigenen Methoden verändern. Erweitern Sie die Methoden der Klassen, indem Sie der neuen Klasse weitere Methoden oder Attribute hinzufügen.

In der Abbildung werden die Klassen und die Beziehungen der Klassen des Dekorierer-Musters in einem Klassendiagramm dargestellt. Die Dekorierer-Klasse hält eine Referenz auf ein Objekt der Klasse `Komponente`. Durch Ableiten von Klassen von der Dekorierer-Klasse können Sie diese Klasse weiter spezialisieren. Die Klasse `Komponente`, die von der Klasse `Dekorierer` „umwickelt“ wird, kann durch Ableitung in einer von der Dekorierer-Klasse unabhängigen Hierarchie weiterentwickelt werden.



Beispiel

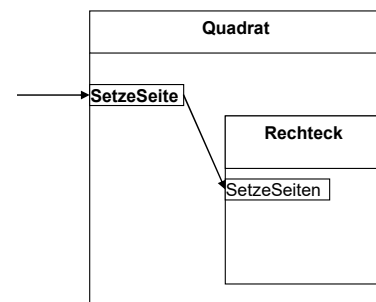
Im Beispiel, das den Einsatz dieses Musters zeigt, soll ein Grafik-Editor erstellt werden. Zur Umsetzung werden von einer Basisklasse `GeoObjekt` die grundlegenden Eigenschaften wie `Mittelpunkt` und `Sichtbarkeit` und Methoden wie `anzeigen`, `entfernen` und `verschieben` geerbt.

Von dieser Basisklasse werden die Klassen für die geometrischen Objekte `Dreieck`, `Rechteck` und `Kreis` abgeleitet. Die Methoden und Eigenschaften werden entsprechend der mathematischen Definition ergänzt. Als weiteres geometrisches Objekt wird ein `Quadrat` hinzugefügt, das eine Spezialisierung der Klasse `Rechteck` ist. Genau hier gibt es ein Problem.

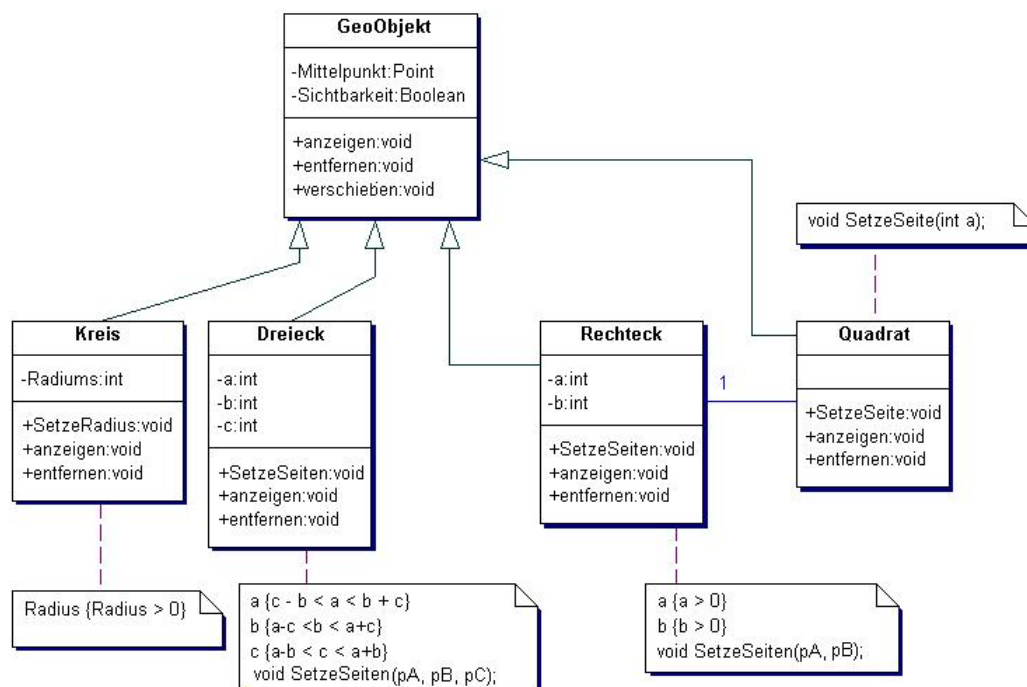
- ✓ Wenn Sie das Quadrat als Spezialisierung des Rechtecks modellieren wollen, entsteht ein Problem beim Zuweisen von zwei Seitenlängen, die ein Quadrat nicht hat. Das Zusicherungs- und Verantwortlichkeitsprinzip wird verletzt, d. h., die Zuweisung zweier Seiten mit gleicher Länge muss kontrolliert werden. Dazu wird eine Restriktion erstellt, die auf der Grundlage von Eigenschaften der Basisklasse erfolgt $\{a=b\}$. Weiterhin wird keine redundante Speicherung erreicht, weil jedes Objekt der Klasse Quadrat zwei Eigenschaften (a, b) besitzt, aber nur eine Seitenlänge hat.
- ✓ Die Ableitung der Klasse Rechteck von der Klasse Quadrat bringt eine redundante Speicherung, weil die zweite Seitenlänge in einem Attribut der Klasse Rechteck gespeichert wird und in der Klasse Quadrat nicht existiert. Diese Anordnung bringt aber nicht den gewünschten Erfolg. Sie könnten einer Variablen, die ein Quadrat referenziert, ein Objekt der Klasse Rechteck zuweisen.

Die Lösung

Einen Ausweg bietet das Dekorierer-Muster. Die Klasse Quadrat wird von der Klasse GeoObjekt abgeleitet und verwaltet ein Objekt der Klasse Rechteck. Von Methoden der Klasse Quadrat werden Methoden der Klasse Rechteck aufgerufen und die Rechteckeigenschaften entsprechend den Eigenschaften eines Quadrates gesetzt.



Zum Beispiel wird die Methode `SetzeSeite` der Klasse `Quadrat` mit einem Parameter aufgerufen. Die Methode der Klasse `Quadrat` ruft die Methode der Klasse `Rechteck` über das in sich verwaltete Rechteckobjekt auf und übergibt z. B. zweimal den gleichen Parameter. Dabei kann die Größe der Seiten kontrolliert werden, sodass das Zusicherungs- und Verantwortlichkeitsprinzip nicht mehr (wie vorher) verletzt wird.



Das Klassendiagramm zeigt die Modellierung der Klassen für die Darstellung von geometrischen Figuren. Dabei werden die Klassen `Kreis`, `Dreieck`, `Rechteck` und `Quadrat` von der Klasse `GeoObjekt` abgeleitet und erhalten damit die grundlegenden benötigten Eigenschaften.

In den abgeleiteten Klassen werden speziell auf den Figurentyp angepasste Methoden und Eigenschaften implementiert und die Methoden der Basisklasse überschrieben, um sie an die geometrische Figur anzupassen.

Die Klasse `Quadrat` referenziert auf die Klasse `Rechteck` und benutzt ein Objekt der Klasse `Rechteck`, um die Länge der Seiten zu speichern und die Methoden zur Darstellung mit den geeigneten Werten für die Parameter aufzurufen.

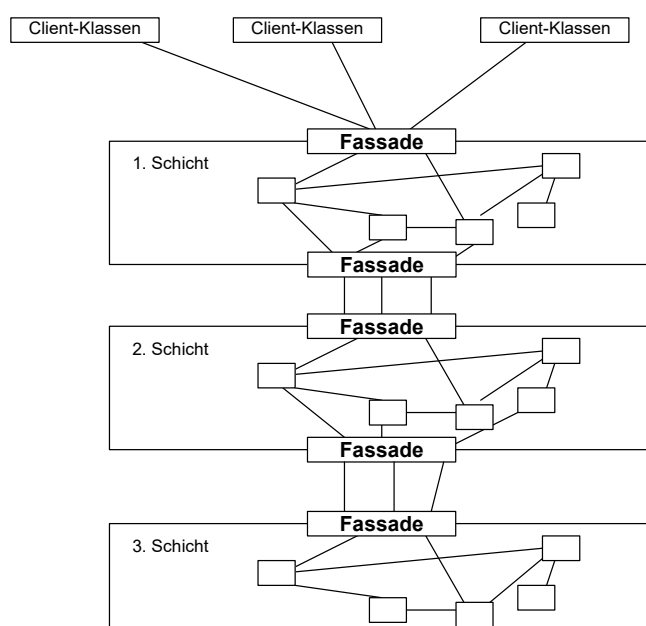
Die Ableitung von einer gemeinsamen Basisklasse `GeoObjekt` hat den Vorteil, dass alle Objekte einer Zeichnung in einem Container verwaltet werden können, weil sie typgleich sind.

Fassade (Facade Pattern)

Wenn Sie ein komplexes Teilsystem erstellt haben und der Zugriff auf dieses Teilsystem ebenfalls sehr komplex ist, können Sie die Verwendung des Teilsystems vereinfachen oder vereinheitlichen, indem Sie eine Schnittstelle für das Teilsystem anbieten, über die alle Anforderungen an das Teilsystem abgewickelt werden.

Mit dieser Vorgehensweise können Sie die Funktionalität beliebig einschränken oder zusammenfassen und gleichzeitig die Wiederverwendbarkeit des Teilsystems steigern. Eine Fassade bietet eine einfache voreingestellte Sicht auf das Teilsystem, die für die meisten Clients ausreicht. Benötigen Clients eine spezielle Anpassung, kann die Fassade mit einer entsprechenden Implementierung übergangen werden.

Die Fassade entkoppelt die implementierten Klassen des Teilsystems von dem Client-System und anderen Systemen. Dabei werden die Unabhängigkeit und die Portabilität des Teilsystems erhöht.



Beispiel

Bei der Implementierung einer Anwendung in Schichten können Sie mit einer Fassade den Eintrittspunkt in jeweils eine dieser Schichten implementieren. Eine Schichtenstruktur wird verwendet, um z. B. Nachrichten im Internet zu filtern. Die Nachrichten, die in einer Schicht als nicht vertrauenswürdig eingestuft sind, werden der nächsten Schicht nicht übergeben und somit ausgesiebt. Diese Nachrichten können dadurch keinen Schaden in anderen Schichten anrichten.

Memento

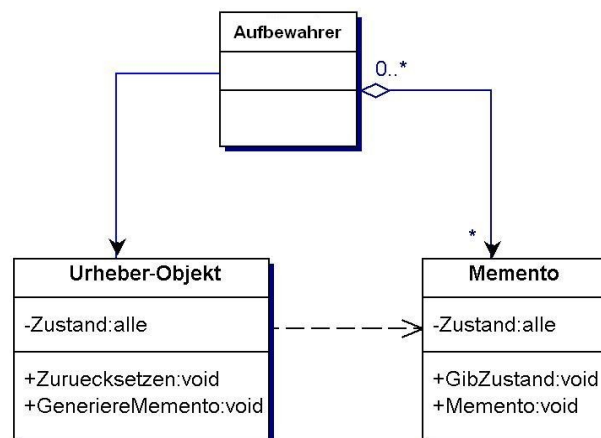
In einer Anwendung kann es notwendig sein, alle Änderungen an einem Objekt rückgängig zu machen. Für die Objekte, die dieses Objekt manipulieren und die dafür eine temporäre Kopie des Objekts erstellen müssen, sollen die konkreten Eigenschaften verborgen bleiben.

Dabei entsteht das Problem, dass Eigenschaften gespeichert und damit auch gelesen werden müssen, andererseits aber ein Zugriff auf die Eigenschaften aufgrund der Kapselung unterbunden werden soll.

Eine Vorlage für die Lösung bietet das Memento-Muster. Das Prinzip beruht darauf, dass das Aufbewahrer-Objekt das Urheber-Objekt (Original) anweist, sich selbst zu duplizieren, um den Objektzustand zu archivieren, oder die Identität eines Duplikates anzunehmen, um die ausgeführten Änderungen zu verworfen.

Der Aufbewahrer möchte den Zustand des Urheber-Objekts speichern, er hat aber keinen Zugriff auf die Eigenschaften, da er lediglich eine Referenz auf dieses Objekt besitzt. Um dem Client dennoch die Verwaltung des gespeicherten Zustandes zu ermöglichen, ohne die Kapselung aufheben zu müssen, wird die Klassenhierarchie um eine zusätzliche Klasse, das Memento, erweitert.

Die Verwendung dieser Struktur erfolgt in folgender Weise:



- ✓ Der Aufbewahrer fordert vom Urheber-Objekt ein Memento an, in dem die Operation `GeneriereMemento` des Urheber-Objekts aufgerufen wird.
- ✓ Die Operation `GeneriereMemento` ruft den Konstruktor der Memento-Klasse auf und übergibt diesem den Zustand des Urheberobjekts (Belegung der Eigenschaften).
- ✓ Der Aufbewahrer erhält Zugriff auf das Memento-Objekt in Form einer Referenz, die das Urheber-Objekt im Ergebnis des Aufrufs der Operation an den Client zurückgibt, und kann das Memento-Objekt wie eine Black-Box verwalten, erfährt also nichts über die Struktur dieses Objekts. Da die Operation `GeneriereMemento` mehrfach aufgerufen werden kann, kann der Aufbewahrer auch mehrere Zustände des Urheberobjekts verwalten, wenn dies gewünscht ist.

- ✓ Der Aufbewahrer kann das Urheber-Objekt anweisen, den Zustand eines vom Client bestimmten Memento-Objekts anzunehmen, indem es die Operation `Zuruecksetzen` des Urheber-Objekts aufruft und eine Referenz auf das Memento-Objekt übergibt, das die Daten zur Wiederherstellung enthält.
- ✓ Über die Verwaltung der Referenzen auf das oder die Memento-Objekte kann der Client ausgewählte Memento-Objekte verwerfen, ohne das Urheber-Objekt davon informieren zu müssen. Dem Urheber-Objekt sind die Memento-Objekte unbekannt, es ist nur deren Erzeuger.

Mit Einsatz dieses Entwurfsmusters kann die Kapselung beibehalten werden, da der Aufbewahrer das Memento zwar verwaltet, seine Struktur jedoch nicht kennt und so weder auf dessen Eigenschaften zugreifen noch dessen Methoden aufrufen kann.

Beispiel: *Memento.cs* – Klassendefinition der Klasse **Urheber**

 **Beispieldatei:** *Memento.cs*

Von der Klasse `Urheber` wird mit der Methode `CreateMemento` das Objekt instanziiert, dessen Zustand gespeichert und wiederhergestellt werden soll. Der Zustand wird mit dem Parameterwert initialisiert. Die Methoden `set` und `get` ermöglichen es, den aktuellen Zustand einzufrieren oder den zuvor abgelegten Zustand wiederherzustellen. Mit der Methode `Wiederherstellen` können Sie den Originalzustand wiederherstellen.

```
① class Urheber
② {
③     private string _zustand;
④     public string Zustand
⑤     {
⑥         get
⑦         {
⑧             return _zustand;
⑨         }
⑩         set
⑪         {
⑫             _zustand = value;
⑬         }
⑭     }
⑮     public Memento ErzeugeMemento()
⑯     {
⑰         return (new Memento(_zustand));
⑱     }
⑲     public void Wiederherstellen(Memento memento)
⑳     {
㉑         Zustand = memento.Zustand;
㉒     }
㉓ }
```

- ① Die Klasse `Urheber` wird definiert. Von ihr wird das `Urheber`-Objekt erzeugt, dessen Zustand gespeichert und wiederhergestellt werden soll.
- ② Mit der definierten Variablen `_zustand` wird der Zustand des `Urheber`-Objekts initialisiert.
- ③ Mit der Methode `get` kann der aktuelle Zustand ermittelt werden.
- ④ Mit der Methode `set` kann der aktuelle Zustand gesetzt werden.
- ⑤ Die Methode `ErzeugeMemento` ermöglicht die Erstellung eines `Memento`-Objekts. Diese wird mit dem im Parameter übergebenen Zustand initialisiert.
- ⑥ Mit der Methode `Wiederherstellen` können Sie den Zustand des `Urheber`-Objekts in den letzten gespeicherten Zustand zurückversetzen.
- ⑦ Die Eigenschaft `Zustand` verkörpert den Zustand des `Urheber`-Objekts.

Beispiel: *Memento.cs* – Klassendefinition der Klasse **Memento**

Die Klasse `Memento` nimmt den Zustand des `Urheber`-Objekts an. Der Konstruktor erstellt ein neues `Memento`-Objekt und initialisiert sich mit dem im Parameter übergebenen Zustand. Über die Methode `get` wird der Zustand des `Memento`-Objekts ausgelesen. Diese Methode wird beim Wiederherstellen des Zustandes des `Urheber`-Objekts aufgerufen, um den Zustand der Wiederherstellung zu ermitteln.

```
① class Memento
{
    private string _zustand;
    public Memento(string Zustand)
    {
②         this._zustand = Zustand;
    }
    public string Zustand
    {
③         get
        {
            return _zustand;
        }
    }
}
```

- ① Die Klasse `Memento` wird definiert. Sie dient zum Aufbewahren der Zustände des `Urheber`-Objekts.
- ② Der Konstruktor der Klasse initialisiert das erstellte Objekt mit dem übergebenen Zustand.
- ③ Durch den Aufruf der Methode `get` können Sie den Zustand des `Memento`-Objekts ermitteln, in der Eigenschaft `Zustand` wird der Zustand des `Urheber`-Objekts aufbewahrt.

Beispiel: *Memento.cs* – Klassendefinition der Klasse **Aufbewahrer**

Die Klasse `Aufbewahrer` dient zur Aufbewahrung der `Memento`-Objekte. Durch den Aufruf der Methode `set` wird die im Parameter übergebene Referenz auf ein `Memento`-Objekt gesetzt. Für das Wiederherstellen des Zustandes des `Urheber`-Objekts wird die Methode `get` aufgerufen.

```
① class Aufbewahrer
{
    private Memento _memento;
    public Memento Memento
    {
        ② set
        {
            _memento = value;
        }
        ③ get
        {
            return _memento;
        }
    }
}
```

- ① Die Klasse `Aufbewahrer` wird definiert. Von ihr wird ein Objekt zur Aufbewahrung der Memento-Objekte erstellt.
- ② Mit dem Aufruf der Methode `set` können Sie das im Parameter referenzierte `Memento`-Objekt für eine eventuelle Wiederherstellung ablegen.
- ③ Die Methode `get` liest das zuletzt abgelegte `Memento`-Objekt und gibt einen Zeiger darauf zurück.

Beobachter (Observer Pattern)

Möchten Sie beispielsweise die Daten und die Darstellung der Daten nicht in einer Klasse, sondern in zwei Klassen definieren und diese somit „lose“ aneinanderkoppeln, benötigen Sie eine Kommunikation zwischen dem Datenobjekt und den Objekten für die grafische Darstellung. Dabei muss das Datenobjekt nichts über die Implementierungsdetails der verschiedenen Darstellungsobjekte wissen. Infolge einer Änderung des Zustandes teilt das Datenobjekt dem Darstellungsobjekt mit, dass es eine Änderung erfahren hat. Das Darstellungsobjekt interessiert sich nicht dafür, auf welche Weise und von wem das Datenobjekt verändert wurde. Es liest nur den veränderten Datenbestand aus und stellt diesen in der entsprechenden Form dar.

Stellen Sie sich die Implementierung einer Uhr vor, deren Uhrzeit einmal analog und einmal digital angezeigt wird. Eine solche Anwendung hat ein Datenobjekt, das entsprechende Eigenschaften zur Speicherung der Uhrzeit enthält. Für die Darstellung der Uhrzeit wird jeweils ein Objekt für die Anzeige einer Analoguhr und ein Objekt für die Anzeige einer Digitaluhr erstellt.



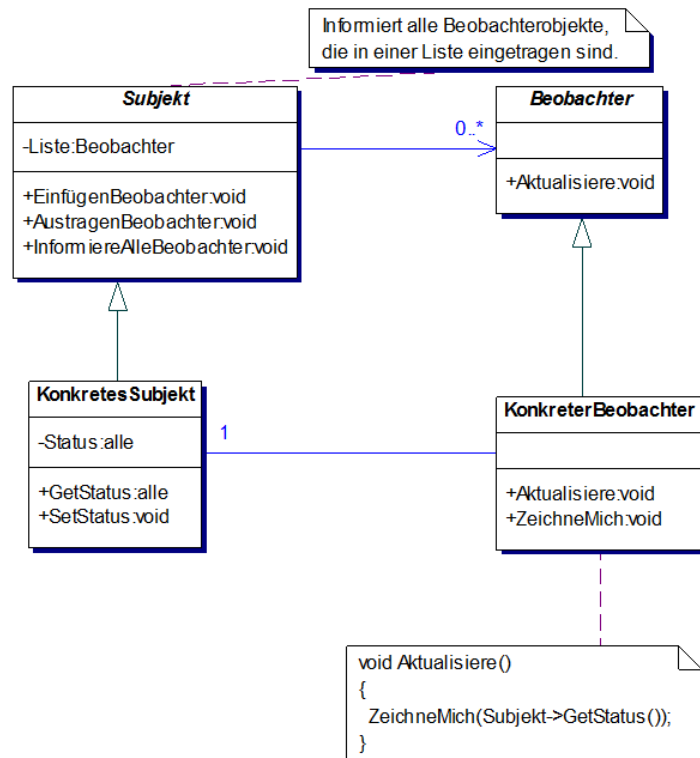
11:43:05

Nach der Veränderung der Uhrzeit im Datenobjekt informiert dieses die Darstellungsobjekte. Diese rufen den aktuellen Zustand des Datenobjekts ab und erstellen die jeweilige grafische Darstellung.

Die Subjekt-Klasse wird zur Objekterstellung des Datenobjekts und die Beobachterklasse zur Objekterstellung der Darstellungsobjekte verwendet. Beide wurden von abstrakten Basisklassen abgeleitet, die die notwendigen Verknüpfungen bereits festlegen.

Die Abbildung zeigt die Struktur der Klassen des Beobachter-Musters und deren Beziehungen. Sie besteht zunächst aus zwei Klassen (Subjekt und Beobachter), wobei die Klasse Subjekt die Beobachter-Objekte verwalten kann, die sich beim Objekt der Klasse Subjekt registriert haben.

Von der Klasse Subjekt wird die Klasse abgeleitet, deren Objekte die Datenhaltung übernehmen. Die Objekte, die die Darstellung der Daten vornehmen, werden von einer Klasse erzeugt, die von der Klasse Beobachter abgeleitet wurde.



Die Klassen **Konkreter Beobachter** und **Konkretes Subjekt** verbindet eine Beziehung, die es dem Objekt der Klasse **Konkreter Beobachter** ermöglicht, die Daten des Objekts der Klasse **Konkretes Subjekt** zu lesen.

Beispiel: *Beobachter.cs* – Basisklassen **Subjekt** und **Beobachter**



Beispieldatei: *Beobachter.cs*

Die Klasse **Subjekt** stellt die Basisklasse für alle Klassen bereit, von denen konkrete Objekte für die Uhrzeitverwaltung erstellt werden. Die Darstellung der Uhrzeit auf dem Bildschirm wird von Objekten realisiert, die von einer abgeleiteten Klasse der Klasse **Beobachter** erzeugt werden. Alle Objekte dieser Liste werden über Änderungen der Uhrzeit informiert und können auf diese Nachricht mit einer Aktualisierung der grafischen Darstellung reagieren.

```

// Basisklasse "Subjekt"
① abstract class Subjekt
{
②     public void Einfuegen(Beobachter observer)
    {
        _Beobachterliste.Add(observer);
    }
③     public void Austragen(Beobachter observer)
    {
        _Beobachterliste.Remove(observer);
    }
}
    
```

```

④ public void InformiereAlleBeobachter()
    {
        foreach (Beobachter i in _Beobachterliste)
        {
            i.Aktualisiere();
        }
    }
⑤ private List<Beobachter> _Beobachterliste = new
List<Beobachter>();
}
⑥ // Basisklasse "Beobachter"
abstract class Beobachter
{
⑦     public abstract void Aktualisiere();
}

```

- ① Die abstrakte Klasse für die Datenobjekte wird definiert.
- ② Die Methode Einfuegen wird zum Registrieren von Beobachterobjekten aufgerufen. Die zu registrierenden Objekte werden über den Parameter referenziert.
- ③ Die Methode Austragen entfernt registrierte Objekte aus der Beobachterliste des Datenobjekts. Das betreffende Beobachterobjekt wird über den Parameter referenziert.
- ④ Die Methode InformiereAlleBeobachter sorgt für den Aufruf der Methode Aktualisiere aller registrierten Beobachterobjekte.
- ⑤ Die verkettete Beobachterliste wird implementiert; der Typ der Elemente, die in der Liste gespeichert werden können, wird über den Ausdruck in den spitzen Klammern (Template) festgelegt.
- ⑥ Die abstrakte Klasse Beobachter wird definiert. Von ihr werden die konkreten Beobachterklassen abgeleitet. Anschließend können Sie Objekte der abgeleiteten Klasse erstellen.
- ⑦ Die Methode Aktualisiere wird als rein abstrakte Methode implementiert. Diese Methode wird aufgerufen, wenn die Darstellung des Datenobjekts aktualisiert werden soll. Im Beispiel wird die Methode vom Datenobjekt mit einer aktuellen Uhrzeit aufgerufen, um die unterschiedlichen grafischen Darstellungen zu aktualisieren. Die Methode wird in der konkreten Beobachterklasse mit der entsprechenden Ausgabeimplementierung überschrieben.

Beispiel: Beobachter.cs – Ermittlung aktuelle Uhrzeit / konkrete Subjektklasse / Übermittlung an Beobachterobjekte

Die aktuelle Uhrzeit wird ermittelt und von der Klasse Uhr (die von der Klasse Subjekt abgeleitet wurde) an alle registrierten Beobachterobjekte übermittelt.

```

// Zentrale Ermittlung der aktuellen Uhrzeit
...
① System.DateTime Aktuelle_Zeit = System.DateTime.Now;
    subjekt.Uhrzeit = Aktuelle_Zeit;
...
// Konkrete "Subjekt-Klasse"

```

```

② class Uhr : Subjekt
{
    private string _Uhrzeit;
    public string Uhrzeit
    {
③ get
        {
            return _Uhrzeit;
        }
④ set
        {
            _Uhrzeit = value;
        }
    }
    ...
⑤ // Übermittlung an alle Beobachterobjekte
    subjekt.InformiereAlleBeobachter();

```

- ① Die aktuelle Uhrzeit wird ermittelt (im Beispiel 11:43:05).
- ② Die konkrete Subjektklasse `Uhr` wird von der abstrakten Klasse `Subjekt` abgeleitet.
- ③-④ Die Methoden `get` und `set` ermöglichen die Abfrage und das Setzen der ermittelten und im Datenobjekt gespeicherten Uhrzeit.
- ⑤ Über den Aufruf der Methode `InformiereAlleBeobachter` wird die Uhrzeit an alle registrierten Beobachterobjekte übermittelt.

Beispiel: *Beobachter.cs* – konkrete Beobachterklasse

Beobachterobjekte werden von einer Klasse erstellt, die von der Basisklasse `Beobachter` abgeleitet wurde. Dadurch sind diese bereits entsprechend dem Beobachtermuster verbunden. Da die Existenz der Beobachterobjekte von der Existenz der die Uhrzeit verwaltenden Objekte der Klasse `Subjekt` abhängig ist, wird das Beobachterobjekt im Konstruktor beim Datenobjekt registriert.

Innerhalb der Methode `Aktualisiere` erfolgt, zwecks Veranschaulichung der Entwurfsmusterfunktion, die Ausgabe der von den jeweiligen konkreten registrierten Beobachterklassen empfangenen Uhrzeit auf dem Bildschirm.

```

// Konkrete Beobachterklasse
① class KonkreteUhr : Beobachter
{
    private string _name;
    private string _beobachterUhrzeit;
    private Uhr _subjekt;
    // Konstruktor
② public KonkreteUhr(Uhr subjekt, string name)
    {
        this._subjekt = subjekt;
        this._name = name;
    }
}

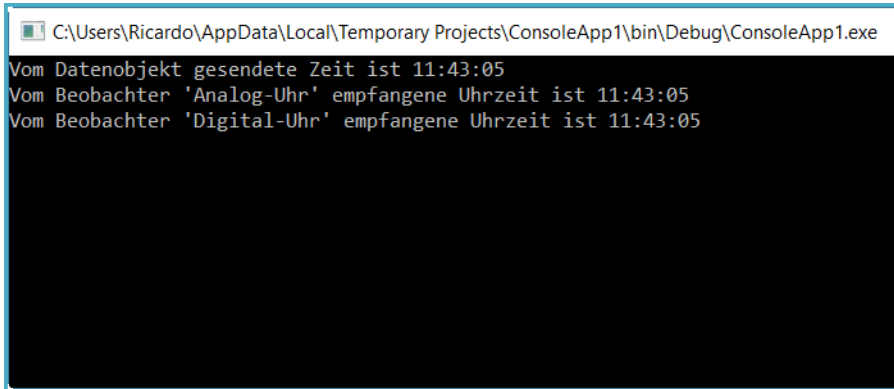
```

```
③ public override void Aktualisiere()  
{  
    ④ _beobachterUhrzeit = _Subjekt.Uhrzeit;  
    Console.WriteLine  
      ("Vom Beobachter {0} empfangene Uhrzeit ist {1}", _name,  
        _beobachterUhrzeit);  
}  
}
```

- ① Die konkrete Klasse wird von der abstrakten Klasse `Beobachter` abgeleitet.
- ② Der Konstruktor wird mit Referenz auf das Datenobjekt aufgerufen, in dem die Uhrzeit ermittelt wird.
- ③ Der Methode `Aktualisiere` wird eine Referenz auf das Datenobjekt übergeben, in dem die Ermittlung der Uhrzeit erfolgt ist.
- ④ Die von den jeweiligen konkreten Beobachterklassen empfangene Uhrzeit wird auf dem Bildschirm ausgegeben.

Zur Verdeutlichung der grundlegenden Funktionsweise dieses Entwurfsmusters werden im Hauptprogramm zwei konkrete Beobachterklassen ("Analog-Uhr" und "Digital-Uhr") registriert, die aktuelle Uhrzeit (hier beispielhaft "11:43:05") wird übermittelt und auf Bildschirm ausgegeben.

Anschließend wird die von den jeweiligen konkreten Beobachtern empfangene Uhrzeit ebenfalls auf die Bildschirmausgabe geleitet.

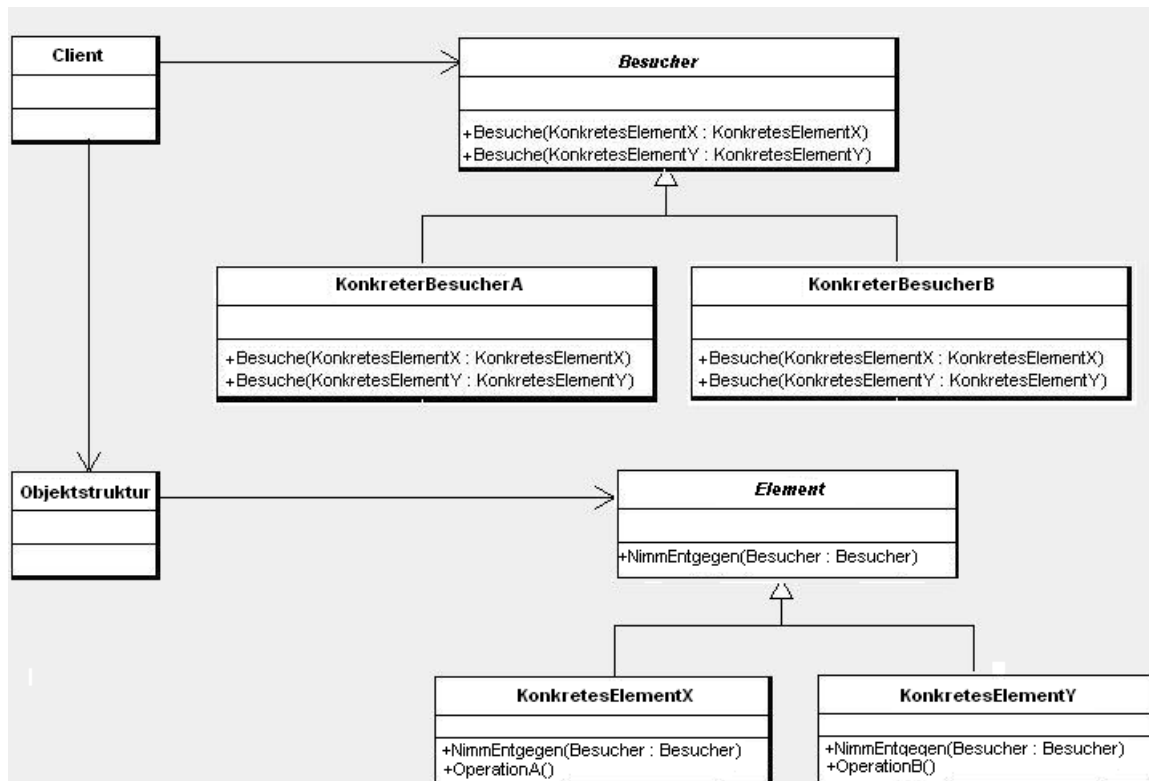


```
C:\Users\Ricardo\AppData\Local\Temporary Projects\ConsoleApp1\bin\Debug\ConsoleApp1.exe  
Vom Datenobjekt gesendete Zeit ist 11:43:05  
Vom Beobachter 'Analog-Uhr' empfangene Uhrzeit ist 11:43:05  
Vom Beobachter 'Digital-Uhr' empfangene Uhrzeit ist 11:43:05
```

Bildschirmausgabe des Programms `Beobachter.cs`

Besucher (Visitor Pattern)

Das Besuchermuster trennt die Funktionalität von der Datenstruktur mit dem Ziel, beide getrennt spezialisieren zu können. Dies ist von besonderer Bedeutung, wenn Sie in einem Strukturobjekt, z. B. in einer Liste, mehrere verschiedene Objektelemente speichern. Mit allen Elementen der Liste wird eine Methode aufgerufen. Dabei hat die aufgerufene Methode für alle Elemente die gleiche Aufgabe. Die Ausführung selbst ist aber für jede Klasse unterschiedlich. Mithilfe des Besuchermusters können Sie für alle Elemente, die in der Liste gespeichert sind, das gleiche Besucherobjekt einladen (aufrufen), um eine bestimmte Funktionalität für alle Elemente ausführen zu lassen. Dabei wird die konkrete Ausführung in einer speziell auf die Klasse des Objekts angepassten Methode implementiert.



Beispiel: *Besucher.cs*



Beispieldatei: *Besucher.cs*

Im Beispiel wird die vorstehende UML-Darstellung implementiert; zur Verdeutlichung der Funktionsweise des Entwurfsmusters werden die ausgeführten Abläufe zusätzlich mittels Bildschirmausgabe dokumentiert.

```

//Basisklasse "Besucher"
① abstract class Besucher
{
  ② public abstract void
    BesucheKonkretesElementX(KonkretesElementX
      KonkretesElementX);
    public abstract void
      BesucheKonkretesElementY(KonkretesElementY
        KonkretesElementY);
}
// Konkrete "Besucherklasse A"

③ class KonkreterBesucherA : Besucher
{

```

```

④ public override void
  BesucheKonkretesElementX(KonkretesElementX
    KonkretesElementX)
  {
    Console.WriteLine
      ("{0} wird von {1} besucht",
        KonkretesElementX.GetType().Name,
        this.GetType().Name);
  }
  public override void
  BesucheKonkretesElementY(KonkretesElementY
    KonkretesElementY)
  {
    Console.WriteLine
      ("{0} wird von {1} besucht",
        KonkretesElementY.GetType().Name,
        this.GetType().Name);
  }
}

// Konkrete "Besucherklasse B"
⑤ class KonkreterBesucherB : Besucher
{
  ⑥ public override void
    BesucheKonkretesElementX(KonkretesElementX
      KonkretesElementX)
    {
      Console.WriteLine
        ("{0} wird von {1} besucht",
          KonkretesElementX.GetType().Name,
          this.GetType().Name);
    }
    public override void
    BesucheKonkretesElementY(KonkretesElementY
      KonkretesElementY)
    {
      Console.WriteLine
        ("{0} wird von {1} besucht",
          KonkretesElementY.GetType().Name,
          this.GetType().Name);
    }
  }

  // Basisklasse "Element"
  ⑦ abstract class Element
  {
    ⑧ public abstract void NimmEntgegen(Besucher Besucher);
  }

  // Konkrete "Elementklasse" X
  ⑨ class KonkretesElementX : Element
  {
    ⑩ public override void NimmEntgegen(Besucher Besucher)
    {

```

```

⑪    Besucher.BesucheKonkretesElementX(this); public void
OperationA()
    {
    }

    // Konkrete "Elementklasse" Y
⑫    class KonkretesElementY : Element
    {
⑬        public override void NimmEntgegen(Besucher Besucher)
        {
            Besucher.BesucheKonkretesElementY(this);
        }
        public void OperationB()
        {
        }
    }
    // Klasse "Objektstruktur"
⑭    class Objektstruktur
    {
⑮        private List<Element> _elements = new List<Element>();
        public void Einfuegen(Element element)
        {
            _elements.Add(element);
        }

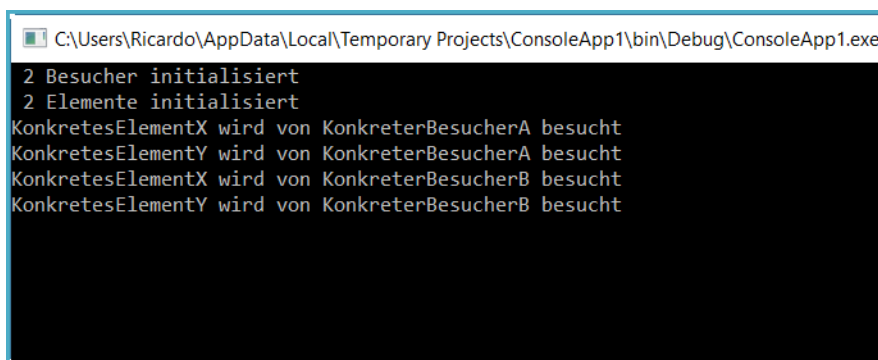
        public void Austragen(Element element)
        {
            _elements.Remove(element);
        }

        public void NimmEntgegen(Besucher Besucher)
        {
            foreach (Element element in _elements)
            {
                element.NimmEntgegen(Besucher);
            }
        }
    }

```

- ① Die Basisklasse für die Besucherobjekte wird definiert.
- ② Für jedes Element, für das die Besucherklassen eine Aktion ausführen sollen, wird eine spezielle Methode definiert. In der abstrakten Basisklasse können diese Methoden abstrakt sein, damit diese in den abgeleiteten Klassen überschrieben werden müssen.
- ③ Eine konkrete Besucherklasse `KonkreterBesucherA` wird von der Klasse `Besucher` abgeleitet.
- ④ Für den Aufruf jeder Elementklasse wird die spezielle Besuchermethode überschrieben und mittels Konsolmeldung dokumentiert.
- ⑤ Eine konkrete Besucherklasse `KonkreterBesucherB` wird von der Klasse `Besucher` abgeleitet.
- ⑥ Für den Aufruf jeder Elementklasse wird die spezielle Besuchermethode überschrieben und mittels Konsolmeldung dokumentiert.
- ⑦ Die Basisklasse für alle Elemente, die ein Besucherobjekt einladen können, wird definiert.

- ⑧ Die Methode `NimmEntgegen` der Basisklasse, der die Referenz eines Besucherobjekts übergeben wird, ermöglicht die Anwendung der speziellen Methoden auf das übergebene Besucherobjekt.
- ⑨ Eine Klasse für das konkrete Element-Objekt `KonkretesElementX` wird von der Klasse `Element` abgeleitet.
- ⑩ Innerhalb der Methode `NimmEntgegen` wird die für die jeweilige Besucherklasse implementierte Methode `BesucheKonkretesElementX` aufgerufen.
- ⑪ Hier wird beispielhaft eine spezifische Operation `A` von `KonkretesElementX` auf die Besucherobjekte angewendet.
- ⑫ Eine Klasse für das konkrete Element-Objekt `KonkretesElementY` wird von der Klasse `Element` abgeleitet.
- ⑬ Innerhalb der Methode `NimmEntgegen` wird die für die jeweilige Besucherklasse implementierte Methode `BesucheKonkretesElementY` aufgerufen.
- ⑭ Die Klasse `Objektstruktur` wird als Liste definiert; sie dient der Speicherung und Verwaltung der unterschiedlichen Elemente.
- ⑮ Mit den Methoden `Einfuegen`, `Austragen` und `NimmEntgegen` können die Listenelemente verwaltet werden.

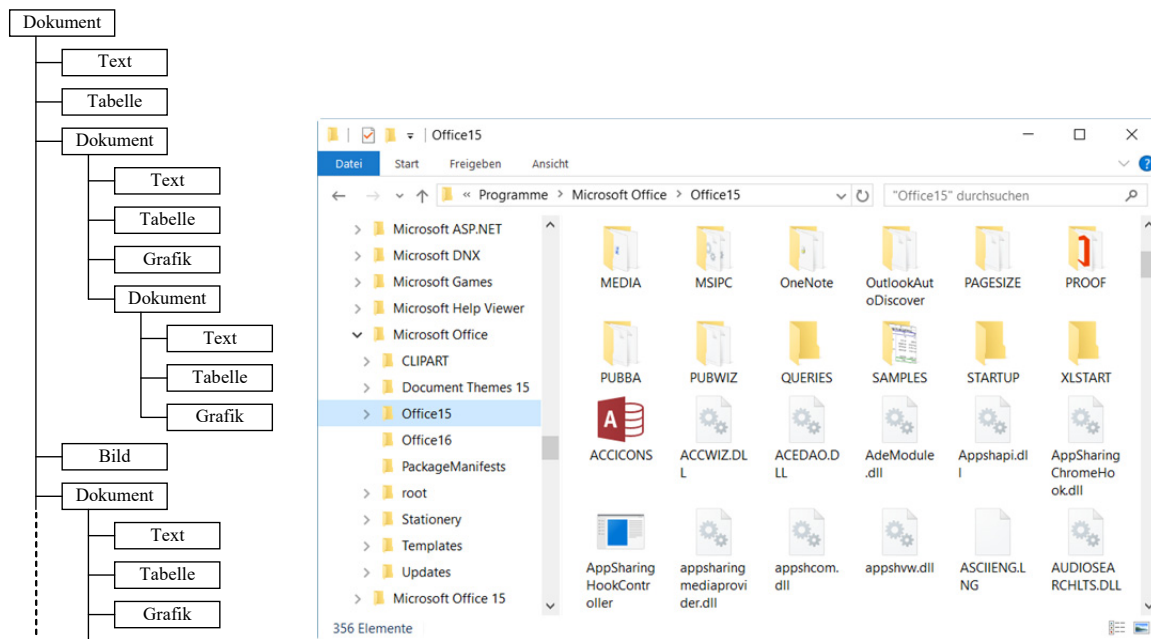


```
C:\Users\Ricardo\AppData\Local\Temporary Projects\ConsoleApp1\bin\Debug\ConsoleApp1.exe
2 Besucher initialisiert
2 Elemente initialisiert
KonkretesElementX wird von KonkreterBesucherA besucht
KonkretesElementY wird von KonkreterBesucherA besucht
KonkretesElementX wird von KonkreterBesucherB besucht
KonkretesElementY wird von KonkreterBesucherB besucht
```

Bildschirmausgabe des Programms `Besucher.cs`

Kompositum (Composite Pattern)

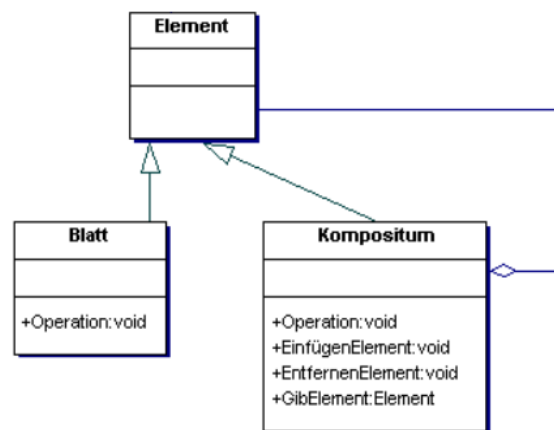
In einer Verzeichnisdarstellung, wie sie z. B. der Explorer unter Windows oder der Konquerer unter Linux erzeugen, werden Verzeichnisse als farbige Ordner angezeigt. Wird ein solches Verzeichnis geöffnet, so können Sie die enthaltenen Dateien sehen. Neben den Dateien kann das Verzeichnis aber wiederum ein oder mehrere weitere Verzeichnisse enthalten. Diese Verzeichnisse können wiederum Dateien und Verzeichnisse beinhalten. Eine solche Baumstruktur wird oft für die Abbildung komplexer Zusammenhänge benötigt. Dabei kann ein Bauelement beliebig viele Blätter-Elemente und eben wieder Teilbäume enthalten.



Die folgende Abbildung zeigt die Klassen des Kompositum-Musters mit ihren Beziehungen. Die Klasse `Kompositum` verwaltet Objekte der Klasse `Element`.

Dafür wird eine Aggregation zwischen beiden Klassen hergestellt. Die Klasse `Element` stellt die Basisklasse für alle Klassen spezieller Elemente dar. Die Klasse `Kompositum` wird auch von der Klasse `Element` abgeleitet.

Dadurch wird erreicht, dass Objekte der Klasse `Kompositum` einerseits als Verwalter von Objekten des Typs der `Element`-Klasse fungieren und andererseits selbst als `Element` verwendet werden können.



Informationsquellen

Die angeführten Muster sind ein kleiner Ausschnitt aus der Menge der bereits veröffentlichten Muster. Sie haben einen Einblick in diese Muster erhalten und können sie jetzt einordnen. Um diese Muster in Ihren Anwendungen einzusetzen, ist ein weiteres Studium von Literatur und Quellcodebeispielen nötig, in denen die Muster ausführlich beschrieben und angewendet werden.

- ✓ Grundlagenwerk von Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four) mit dem Titel: „Design Patterns: Elements of Reusable Object-Oriented Software“. Von diesem Buch existiert auch eine deutsche Übersetzung (Dirk Riehle) mit dem Titel: „Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software“.
- ✓ E-Book von Bruce Eckel mit dem Titel: „Thinking in Patterns“. Dieses Buch können Sie kostenlos im Internet z. B. unter der Internetadresse <http://www.mindviewinc.com/Books/downloads.html> downloaden.

7

UML-Diagramme

7.1 Einsatz und Normierung der UML

Die UML stellt eine Zeichensprache zur Modellierung von konkreten Abläufen mit Klassen und Objekten und deren Beziehungen dar; sie enthält keine Methode, in der eine Handlungsweise zur Erstellung von Modellen definiert ist. Das Grundprinzip besteht in der Erarbeitung von Diagrammen, die den zu modellierenden Prozess aus unterschiedlichen Sichten beleuchten.

Da sich die Modellierung in allen drei Phasen der Entwicklung mit den gleichen Elementen und Diagrammen vollzieht, ist ein gleichmäßiger Übergang ohne Änderungen von Phase zu Phase möglich. Sie können sowohl Iterationsschritte innerhalb einer Phase als auch Schritte über mehrere Phasen hinweg realisieren, die Modelle werden schrittweise verbessert und können in die jeweils nächste Phase überführt werden.

Die Normierung und Spezifikation der Sprachstruktur und Sprachelemente der UML erfolgt durch die internationale OMG (Object Management Group); im Folgenden wird Bezug auf die – bei Veröffentlichung dieses Buches aktuelle – UML-Version 2.5 genommen. Die vollständige offizielle UML-Dokumentation sowie weitere Informationen können Sie im Internet unter <http://www.omg.org/spec/UML/> einsehen.

7.2 Einsatz von Diagrammen zur Modellierung

In der objektorientierten Softwareentwicklung mit der UML wird ein Modell der Anwendung in Diagrammen modelliert. Die nachfolgend ausgewählten Diagramme werden in der dargestellten Reihenfolge erstellt, wobei ein fließender Übergang von der Analyse- in die Design-Phase stattfindet.

Name/Es wird modelliert, ...	Phase der Modellierung
Anwendungsfalldiagramm	Analyse-Phase
✓ welche Anwendungsfälle in der zu erstellenden Anwendung enthalten sind ✓ welche Akteure diese Anwendungsfälle auslösen ✓ welche Abhängigkeiten der Anwendungsfälle untereinander bestehen, z. B. ✓ ob ein Anwendungsfall in einem anderen enthalten ist ✓ ob ein Anwendungsfall eine Spezialisierung eines anderen darstellt ✓ ob ein bestehender Anwendungsfall durch einen zweiten erweitert wird	
Aktivitätsdiagramm	Analyse-Phase
✓ welche Schritte innerhalb eines Anwendungsfalles durchlaufen werden ✓ welche Zustandsübergänge die beteiligten Objekte erfahren, wenn die Abarbeitung von einer Aktivität zur nächsten wechselt	
Paketdiagramm	Analyse- und Design-Phase
✓ in welche Pakete die Anwendung zerlegt werden kann ✓ welche Pakete eine weitere Unterteilung ermöglichen ✓ welche Kommunikation zwischen den Paketen realisiert werden muss	
Klassendiagramm	Analyse- und Design-Phase
✓ welche Klassen, Komponenten und Pakete beteiligt sind ✓ über welche Kommunikation die Zusammenarbeit stattfindet ✓ welche Methoden und Eigenschaften die Klassen benötigen ✓ wie viele Objekte mindestens und höchstens in Verbindung stehen ✓ welche Klassen als Container für mehrere Objekte zuständig sind	
Komponentendiagramm	Design-Phase
✓ welche Zusammenhänge es zwischen Komponenten gibt ✓ ob bzw. welche Schnittstellen bei der Kommunikation zwischen den Komponenten genutzt werden	
Sequenzdiagramm	Design-Phase
✓ welche Methoden für die Kommunikation zwischen ausgewählten Objekten zuständig sind ✓ wie der zeitliche Ablauf von Methodenaufrufen zwischen ausgewählten Objekten stattfindet ✓ welche Objekte in einer Sequenz neu erstellt und/oder zerstört werden	
Kommunikationsdiagramm (früher Kollaborationsdiagramm – der Inhalt der Modellierung ist ähnlich dem Sequenzdiagramm)	Design-Phase
✓ wie ausgewählte Objekte miteinander kommunizieren ✓ in welcher Reihenfolge die Methodenaufrufe stattfinden ✓ welche alternativen Methodenaufrufe gegebenenfalls existieren	

Name/Es wird modelliert, ...	Phase der Modellierung
Zustandsdiagramm (für jede Klasse ein Diagramm)	Design-Phase
✓ welche Zustandsübergänge von welchen Methodenaufrufen ausgelöst werden ✓ welche Methoden in welchem Zustand aufrufbar sind ✓ welcher Zustand nach dem Erzeugen des Objekts eingenommen wird ✓ welche Methoden das Objekt zerstören	
Einsatz- und Verteilungsdiagramm	Design-Phase
✓ welche Systeme in der Anwendung zusammenarbeiten ✓ welche Module der Anwendung auf welchem System ausgeführt werden ✓ auf welchen Kommunikationsmöglichkeiten die Zusammenarbeit basiert	

Die Reihenfolge der Diagramme kann gegebenenfalls von der in der Tabelle angegebenen abweichen. Die aufgezeigte Reihenfolge stellt aber eine erprobte Möglichkeit dar, wie Sie zu einem Modell Ihrer Anwendung kommen und die Überleitung der Phasen gestalten können.

Die UML stellt ab Version 2.2 insgesamt 14 unterschiedliche Diagrammart zu Verfügung; im Folgenden werden nur die für eine effektive Anwendungsmodellierung notwendigen und mehrheitlich in der Praxis eingesetzten Diagramme beschrieben.

Für Sonderanwendungsfälle können Sie auch die restlichen UML-Diagramme einsetzen:

- ✓ Zeitverlaufsdiagramm – zur Modellierung präziser zeitlicher Spezifikationen
- ✓ Interaktionsübersichtsdiagramm – zur Modellierung komplexer Interaktionen auf Systemebene
- ✓ Kompositionsstrukturdiagramm – zur Darstellung des Inneren eines Klassifizierers und dessen Beziehungen zu anderen Systembestandteilen
- ✓ Objektdiagramm – stellt eine Art Momentaufnahme der Attributwerte und Beziehungszustands eines Objektes dar
- ✓ Profildiagramm – zur Darstellung bzw. Strukturierung von eigendefinierten Stereotypen

7.3 Anwendungsfalldiagramm

Im Anwendungsfalldiagramm wird das extern erkennbare Systemverhalten aus der Perspektive des Anwenders dargestellt; hier werden die möglichen Anwendungsfälle sowie die Interaktion der unterschiedlichen Beteiligten fixiert und beschrieben. Bei hoher Komplexität des Gesamtsystems können Sie die Darstellung auch auf mehrere Diagramme verteilen und diese anschließend verknüpfen.

Akteure

In einem Anwendungsfalldiagramm werden alle Beteiligten eines Vorganges mithilfe von Akteuren dargestellt. Diese können Personen sein, die das System bedienen, oder Fremdsysteme, die auf das System zugreifen. Es können aber auch Ereignisse sein, die ohne Beteiligte ausgelöst werden (z. B. Zeitereignisse).

Zur Kennzeichnung des Typs können Stereotypen hinzugefügt werden. Für ein Fremdsystem z. B. das Stereotyp `Secondary`. Die Akteure können auch in einer bestimmten Rolle agieren. Wenn dies der Fall ist, gibt es eventuell einen abstrakten Akteur, der durch Spezialisierung mehrere Akteure mit unterschiedlichen Rollen an die Realität anpasst.

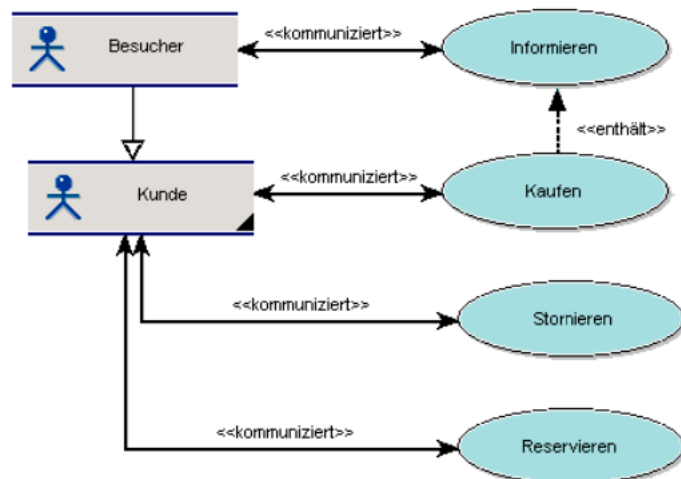
Mithilfe der Spezialisierung können Sie zusätzlich Abhängigkeiten im Zugriff auf den Anwendungsfall darstellen (z. B. Akteur A oder Akteur B; Akteur A und Akteur B).



Anwendungsfall

Anwendungsfälle sind die Aktivitäten, die Sie bei der Beschreibung eines Prozesses angeben. Für ein Ticket-system ist das z. B. das Kaufen, das Reservieren oder das Stornieren von Eintrittskarten. Ein weiterer Anwendungsfall ist das Informieren, das nicht mit einem Verkauf in Zusammenhang stehen muss.

In der Beschreibung zum Prozess könnte Folgendes stehen: „Der Anwender verkauft Eintrittskarten und kann auf Wunsch des Kunden Eintrittskarten reservieren. Möchte der Kunde die Karte zurückgeben, wird die Karte vom System storniert. Auf Anfrage erhält der Kunde eine Information über das aufgeführte Stück sowie über die verfügbaren Plätze.“



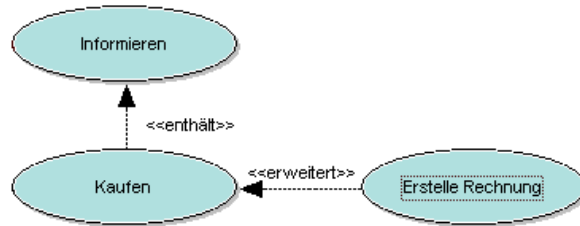
Beziehungen

Die Anwendungsfälle und die Akteure stehen in einer bestimmten Beziehung zueinander. Die Beziehungen werden mit Linien modelliert.



In einigen CASE-Werkzeugen erhalten die Linien zusätzlich auf beiden Seiten Pfeile. Eine solche Verbindung von Akteur und Anwendungsfall bedeutet, dass beide miteinander kommunizieren.

Anwendungsfälle können auch voneinander abhängig sein. So kann ein Anwendungsfall weitere Anwendungsfälle beinhalten. Diesen Sachverhalt können Sie mit einem Pfeil in Richtung des Anwendungsfalls darstellen, der enthalten sein soll. Der Pfeil erhält dann das Stereotyp enthält (include).

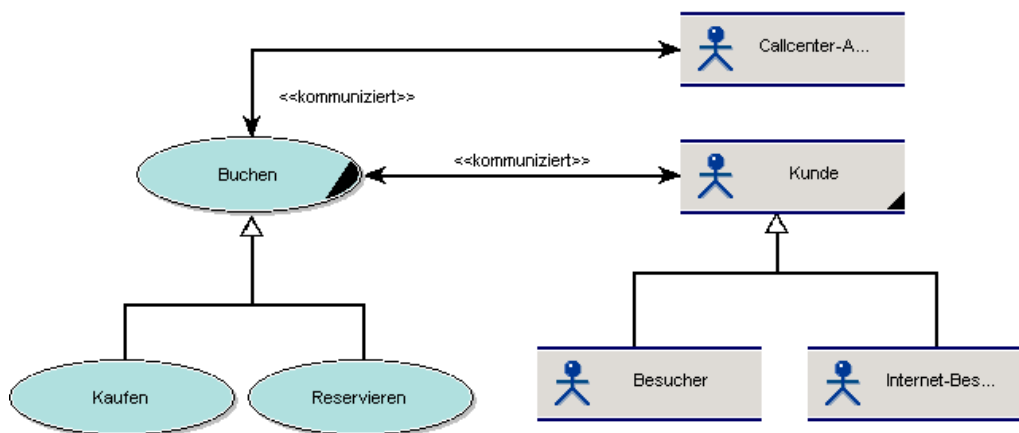


Wird ein Teil der Aufgaben von einem Sachverhalt an einen anderen übertragen (z. B. Erstellung einer Rechnung), wird dies mit einer Auslagerung modelliert. Der Pfeil erhält dann das Stereotyp erweitert (extend).

Eine weitere Form einer Abhängigkeitslinie modelliert die Spezialisierung. Ein Anwendungsfall (oder ein Akteur) stammt von einem generalisierten Anwendungsfall (oder Akteur) ab und spezialisiert diesen.

Beispielsweise ist der Verkauf an der Abendkasse mit dem Verkauf im Internet bis auf den Vertriebsweg ähnlich. Es bietet sich daher an, einen generellen Anwendungsfall Buchen zu erstellen und in der Spezialisierung dieses Anwendungsfalls die geänderten Abfertigungsschritte, die durch die verschiedenen Vertriebswege entstehen, unterzubringen.

Der Anwendungsfall erhält dadurch Vertreter für verschiedene Rollen des Anwendungsfalls oder der Akteure, für die das ebenfalls möglich ist.

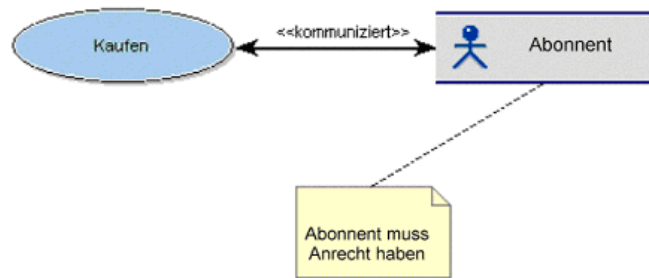


Möchten Sie modellieren, dass ein Akteur in der einen oder in einer zweiten Rolle mit dem Anwendungsfall kommuniziert, ist das über einen generalisierten Akteur möglich. In der Abbildung sind der Akteur Besucher und der Akteur Internet-Besucher ein spezialisierter Akteur Kunde. Mit der Aktivität Buchen kann ein Kunde entweder in der Rolle eines Besuchers oder eines Internetbesuchers kommunizieren.

Der Akteur Callcenter-Agent kann unabhängig von einer Kundenrolle kommunizieren, d. h., der Akteur Callcenter-Agent und der Akteur Besucher oder der Akteur Callcenter-Agent und der Akteur Internet-Besucher können mit der Aktivität Buchen kommunizieren.

Beschreibungen und Notizen

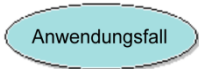
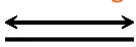
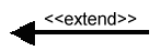
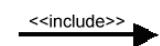
Die UML gestattet für alle Anwendungsfälle und alle Akteure, Beschreibungen in Form von verbalen Formulierungen anzufügen. Diese sind aufgrund ihres Umfangs nicht für die Anzeige in Diagrammen geeignet. Den Diagrammen können Sie Notizen hinzufügen, die auf wesentliche Gestaltungsüberlegungen hinweisen.



Notizen werden mit einem Rechteck dargestellt, dessen rechte obere Ecke eingeknickt ist. Eine gestrichelte Linie stellt die Verbindung zwischen der Notiz und dem zu erklärenden Element her.

Grafische Elemente

Die folgende Tabelle zählt die wichtigsten Symbole zur Modellierung in einem Anwendungsfall auf.

Name/Symbol	Verwendung
Anwendungsfall  	Ein Anwendungsfall wird mit einer Ellipse dargestellt, die den Namen des Anwendungsfalls enthält. Ein Geschäftsanwendungsfall wird mit einer Linie im linken Bereich der Ellipse gekennzeichnet.
Akteur 	Ein Anwendungsfall wird durch einen Akteur ausgelöst. Die Darstellung entspricht einem Strichmännchen. Sie können einen Akteur auch in einem Rechteck darstellen und das Stereotyp «Actor» über dem Namen des Akteurs angeben.
Beziehung 	Ein Akteur steht in einer Beziehung zum Anwendungsfall. Diese Beziehung wird mit einer Verbindungslinie zwischen dem Anwendungsfall und dem Akteur dargestellt. Sie können statt der Linie auch einen Doppelpfeil verwenden. Die Verbindung kann mit dem Stereotyp «kommuniziert» bezeichnet werden.
erweitert 	Wird ein Anwendungsfall durch einen zweiten erweitert, wird diese Beziehung durch die Verbindung der Anwendungsfälle mit einem Pfeil gekennzeichnet, der mit dem Stereotyp «extend» beschriftet wird. Die Pfeilspitze zeigt auf den Anwendungsfall, der erweitert wird.
enthält 	Ist ein Anwendungsfall in einem zweiten enthalten, d. h., ist er fester Bestandteil von diesem, werden beide Anwendungsfälle mit einem Pfeil verbunden, der das Stereotyp «include» als Beschriftung erhält. Die Pfeilspitze zeigt auf den enthaltenen Anwendungsfall.

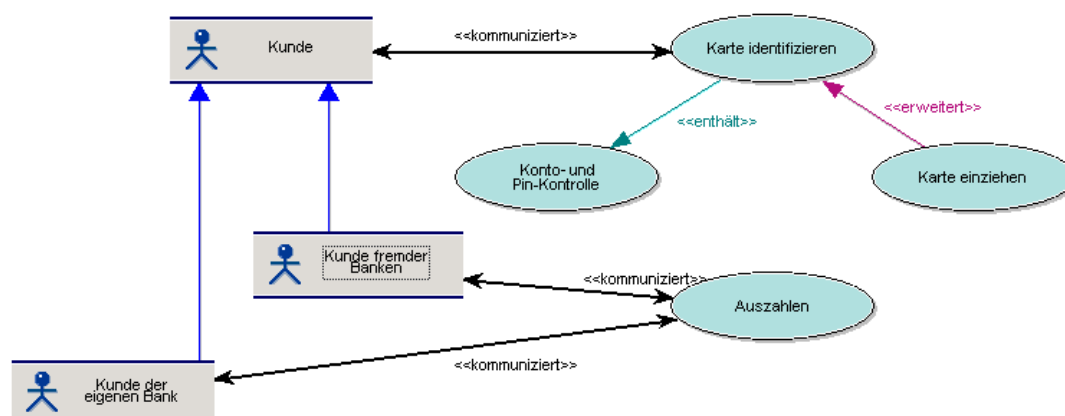
Name/Symbol	Verwendung
Generalisierung 	Diese Beziehung kann zwischen Akteuren und zwischen Anwendungsfällen modelliert werden und bedeutet, dass ein Anwendungsfall oder ein Akteur spezialisiert wird. Die Pfeilspitze zeigt auf den Akteur oder Anwendungsfall, der spezialisiert wird. Die Generalisierung von Akteuren sind meist Rollen.
Notiz  Verbindungslinie 	Notizen sind Diagrammelemente, die an anderen Modellierungselementen angebracht werden. Sie enthalten Informationen zum Verständnis des Modells und werden durch eine unterbrochene Verbindungslinie mit dem Element verbunden.

Beispiel

Ein Kunde möchte mit der Geldkarte Geld am Automaten abheben. Der Akteur Kunde charakterisiert die Rolle und ist die Generalisierung für die Akteure Kunde der eigenen Bank und Kunde fremder Banken. Die beiden spezialisierten Akteure kommunizieren über die Rolle Kunde mit dem Anwendungsfall Karte identifizieren, der für beide Kundenarten gleichermaßen abläuft.

Dieser Anwendungsfall enthält den Anwendungsfall Konto- und Pin-Kontrolle, bei dem die Berechtigung des Kunden zur Kartennutzung überprüft wird. Wurde mehrfach eine falsche PIN eingegeben, wird die Karte eingezogen. Um dies zu modellieren, wird der Anwendungsfall Karte identifizieren mit dem Anwendungsfall Karte einziehen erweitert; dieser wird nur abgearbeitet, wenn die Karte wiederholt nicht identifiziert werden konnte.

Die beiden Akteure Kunde der eigenen Bank und Kunde fremder Banken kommunizieren direkt (nicht über die Rolle Kunde) mit dem Anwendungsfall Auszahlen. Dieser Vorgang ist für beide Kunden unterschiedlich, z. B. weichen der Höchstbetrag oder die Gebühr pro Vorgang voneinander ab.

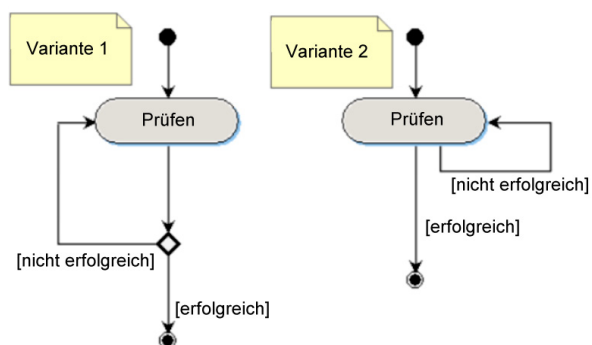


7.4 Aktivitätsdiagramm

Nachdem Sie die Anwendungsfälle identifiziert und in ihrer Abhängigkeit zueinander modelliert haben, erfolgt in den Aktivitätsdiagrammen eine schrittweise Verfeinerung des frühen Entwurfs. Dazu wird für jeden Anwendungsfall ein separates Aktivitätsdiagramm erstellt. In diesem wird der Ablauf des Anwendungsfalls modelliert.

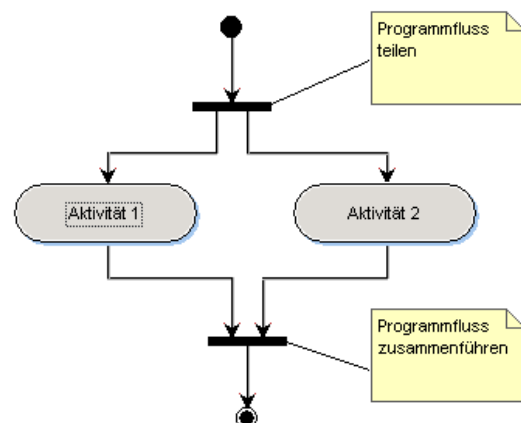
Verzweigungen

Eine Verzweigung des Programmflusses können Sie entweder direkt mit einer Aktivität oder mit einem Rautensymbol erreichen. Eine Aktivität bekommt mehrere Ausgänge, die mit der jeweiligen Bedingung beschriftet werden. In den Abbildungen wird modelliert, dass der Programmablauf nach erfolgreicher Prüfung beendet und bei erfolgloser Prüfung wiederholt wird.



Synchronisation

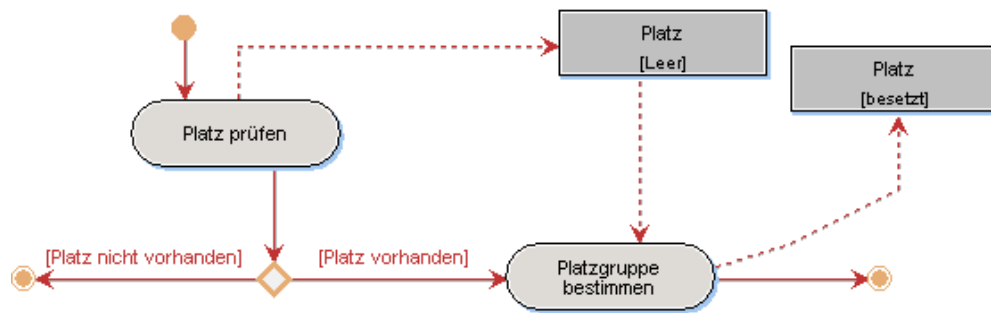
Mittels Synchronisation wird der Programmfluss zweier zeitlich parallel ablaufender Prozesse zu einem Programmfluss zusammengeführt. Solche parallel laufenden Prozesse führen zu einer Beschleunigung der Programmabarbeitung. Werden zwei getrennte Programmflüsse zusammengeführt, wird der nach der Zusammenführung stehende Programmteil erst ausgeführt, wenn beide Prozesse den Synchronisationspunkt erreicht haben.



Objektzustände

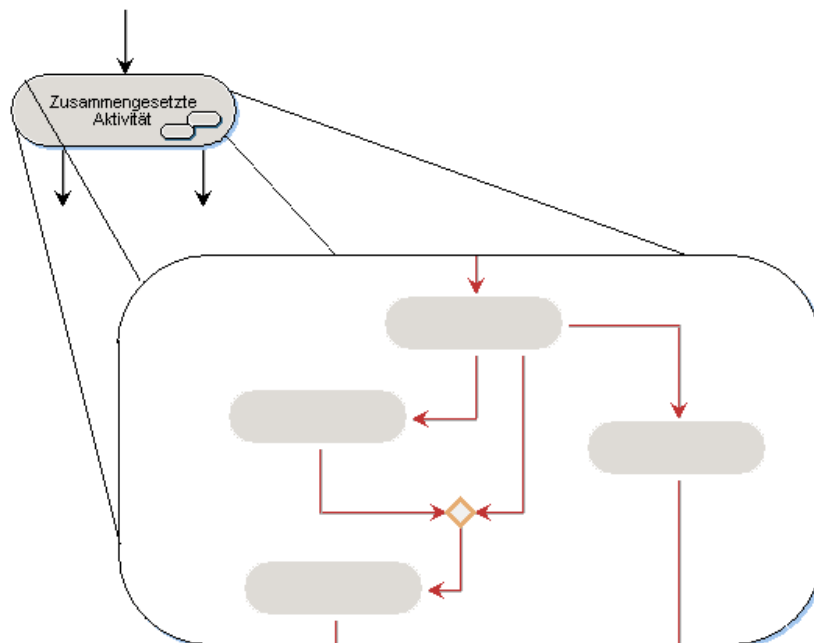
Sie können auch die Abhängigkeiten von Objektzuständen, die durch die Aktivitäten verändert werden, in das Aktivitätsdiagramm einzeichnen. Dabei unterscheiden sich die Symbole für die Aktivitäten und die Objekte.

Um den Fluss des Zustandes eines Objekts von dem Fluss der Aktivitäten unterscheiden zu können, wird für den Objektfluss eine gestrichelte Linie verwendet.



Schachteln von Aktivitätsdiagrammen

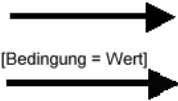
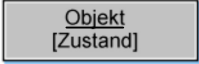
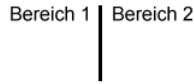
Aktivitäten lassen sich hierarchisch schachteln. Eine Aktivität kann wiederum aus einer Menge von Detailaktivitäten bestehen. Die eingehenden und ausgehenden Transitionen der zusammengesetzten Aktivität und des Detailmodells müssen übereinstimmen.



Grafische Elemente

Die folgende Tabelle enthält die wichtigsten Symbole der Aktivitätsdiagramme.

Name/Symbol	Verwendung
Aktivität 	Das Symbol der Aktivität besteht aus einem Rechteck, dessen linke und rechte Begrenzungslinie durch einen Halbkreis ersetzt wurden. In diesem Symbol wird der Name der Aktivität angegeben.
Zusammengesetzte Aktivität 	Das Symbol der zusammengesetzten Aktivität wird mit dem Symbol der Aktivität dargestellt, in das im rechten unteren Bereich zwei verkleinerte Aktivitätssymbole eingezeichnet sind.

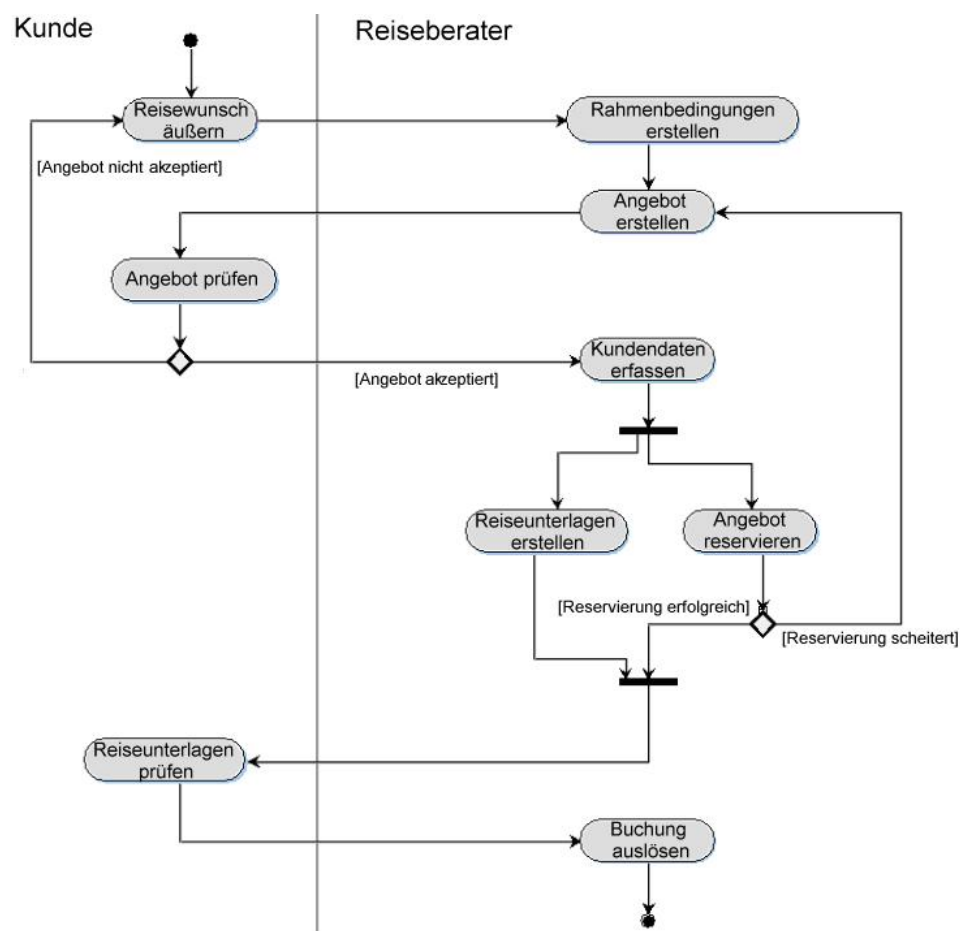
Name/Symbol	Verwendung
Programmfluss 	Zwei Aktivitäten werden mit einem Pfeil verbunden, wenn der Aktivitätsfluss von einer zur nächsten Aktivität wechselt. Die Pfeilspitze zeigt in die Richtung des Programmflusses. Der Pfeil kann z. B. eine Bedingung als Beschriftung erhalten, wenn der Programmfluss nur bei dieser Bedingung stattfindet. Dies ist der Fall, wenn mehrere Transaktionen aus einer Aktivität herausgehen oder der Fluss durch eine Raute aufgeteilt wird.
Raute 	Mit dem Rautensymbol können Sie den Programmfluss verzweigen oder wieder zusammenführen. Gehen mehr Transitionen von der Raute aus, als bei ihr eintreffen, verzweigt die Raute den Programmfluss. Im anderen Fall führt sie den Programmfluss zusammen. Seit UML-Version 2.2 ist es bei Bedarf möglich, beim Entscheidungsknoten zusätzliche Prüfparameter zu modellieren; diese werden innerhalb eines Notizsymbols  oberhalb der Raute beschrieben.
Objektzustand 	Objektzustände werden mit einem Rechteck dargestellt, in dem der Name und der Zustand eines Objekts angegeben werden.
Objektfluss 	Der Objektfluss kennzeichnet die Veränderung eines Objektzustandes durch die Abarbeitung von Aktivitäten und wird mit einem Pfeil auf einer unterbrochenen Linie dargestellt. Die Pfeilspitze zeigt auf eine Aktivität und gibt den Zustand zu Beginn dieser Aktivität an. Das andere Ende des Pfeils zeigt auf die Aktivität, die den Zustand verursacht hat.
Splitter 	Mit diesem Element können Sie den Programmfluss in mehrere Programmflüsse aufteilen, die gleichzeitig ablaufen.
Synchronisieren 	Mit diesem Element können Sie die Programmflüsse, die mit einem Splitter aufgetrennt wurden, wieder zusammenführen. Dabei findet eine Synchronisation statt, d. h., dass die Abarbeitung des weiteren Programmflusses so lange verzögert wird, bis alle Teilflüsse am Synchronisierer angekommen sind.
Startpunkt 	Der Eintrittspunkt, der nach Auslösen eines Anwendungsfalls die Abarbeitung startet, wird mit einem ausgefüllten Kreis dargestellt.
Endpunkt 	Nachdem alle Aktivitäten des Anwendungsfalls abgearbeitet wurden, endet der Programmfluss. Dieser Punkt wird mit dem Endpunkt dargestellt, dessen Symbol aus einem ausgefüllten Kreis besteht, den ein größerer Kreis umgibt.
Verantwortlichkeitsbereich 	Möchten Sie den Programmfluss mit Aktivitäten modellieren, die nicht zum gleichen Verantwortlichkeitsbereich gehören (z. B. verschiedenen Paketen angehören), können Sie die Verantwortlichkeitsbereiche mit senkrechten Linien modellieren, die einen Bereich abgrenzen. Der Name des Bereichs, der zwischen zwei Linien liegt, wird im oberen Abschnitt angegeben.

Beispiel

Für ein Reisebüro wird der Prozess der Reiseberatung in einem Aktivitätsdiagramm dargestellt. Dazu werden Aktivitäten verwendet, die zu den Verantwortlichkeitsbereichen *Kunde* und *Reiseberater* gehören. Im Diagramm wird die Trennung der Bereiche entsprechend durch eine senkrechte Linie dargestellt.

Der Anwendungsfall beginnt damit, dass ein Kunde einen Reisewunsch äußert. Der Reiseberater nennt dem Kunden daraufhin die Rahmenbedingungen und erstellt ein Angebot. Dieses Angebot wird vom Kunden geprüft. An dieser Stelle werden alternative Programmflüsse erstellt. Wenn der Kunde das Angebot nicht akzeptiert, äußert er einen neuen Reisewunsch, der wie der erste Reisewunsch bearbeitet wird. Hat der Kunde das Angebot akzeptiert, erfolgt die Erfassung der Kundendaten. Im Anschluss daran werden das Reiseangebot reserviert und die Reiseunterlagen erstellt. Es erfolgt eine Aufteilung des Programmflusses, wobei beide Zweige durchlaufen werden, eventuell sogar gleichzeitig. Scheitert die Reservierung, wird ein neues Angebot erstellt.

Dem Kunden werden die Reiseunterlagen erst dann zugestellt, wenn die Reservierung erfolgreich war und die entsprechende Erstellung der Reiseunterlagen erfolgte. Nachdem der Kunde die Reiseunterlagen geprüft hat, löst er die Buchung beim Reiseberater aus.

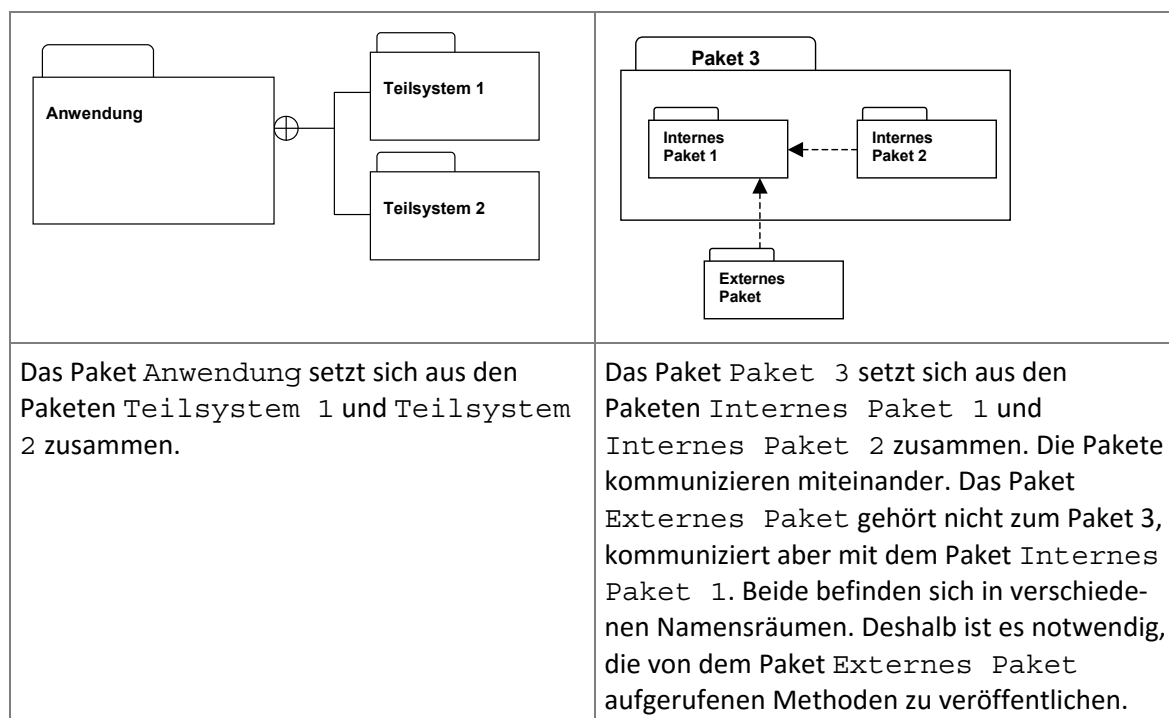


7.5 Paketdiagramm

Pakete sind Sammlungen von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere überschaubare Einheiten gegliedert wird. Ein Paket besitzt einen eigenen Namensraum. Deshalb müssen die Bezeichnernamen innerhalb eines Paketes eindeutig sein. Jedes Element kann von einem anderen Paket referenziert werden. Dennoch gehört jedes Element zu genau einem Paket. Pakete können wiederum in Pakete aufgeteilt werden.

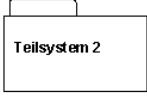
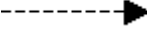
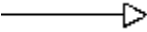
Möchten Sie z. B. eine Arbeitsteilung auf mehrere Gruppen von Entwicklern vornehmen, können Sie die Unterteilung in Pakete nutzen und jeder Gruppe ein solches Paket zur Bearbeitung übergeben. Damit die spätere Zusammenführung der Pakete problemlos möglich ist, wird eine Schnittstelle vereinbart, über die die im Paket enthaltenen Funktionen aufgerufen werden können. Hierbei ist am Anfang nur der Name für die Schnittstelle nötig, jedes Paket kann entsprechend isoliert und gekapselt entwickelt und getestet werden.

Das Zusammenspiel der Pakete des Systems wird in einem Paketdiagramm dargestellt, in dem Sie auch den internen Aufbau eines Pakets modellieren können.



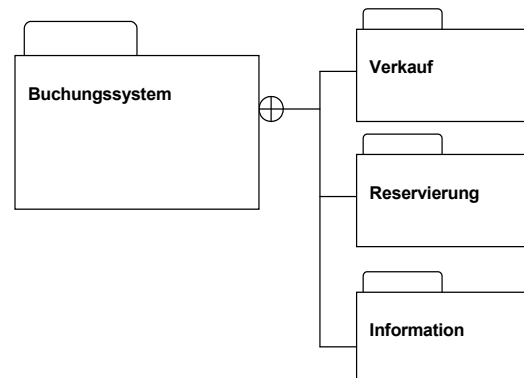
Grafische Elemente

Die folgende Tabelle enthält die wichtigsten Symbole der Paketdiagramme.

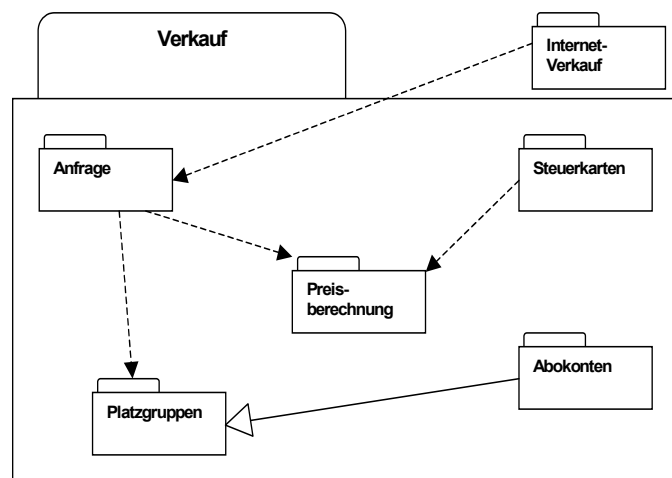
Element	Einsatz
Paket 	Das Symbol des Paketes ist einer Karteikarte nachempfunden. Innerhalb dieses Symbols wird der Paketname dargestellt. Möchten Sie innerhalb des Paketes die Kommunikationen der Unterpakete modellieren, können Sie den Paketnamen in das oben angefügte Rechteck verlagern.
Kommunikation 	Die Kommunikation der Pakete untereinander wird mit einem Pfeil dargestellt, der auf einer unterbrochenen Linie verläuft. Der Pfeil geht von dem Paket aus, von dem auch die überwiegende Kommunikation ausgeht.
Generalisierung 	Erbt ein Paket von einem anderen, wird das Symbol für die Generalisierung verwendet. Mit einem Pfeil werden die beteiligten Pakete verbunden, wobei die Pfeilspitze auf das Paket zeigt, von dem geerbt wird.
enthält 	Mit diesem Symbol können Sie die Paketgliederung vom übergeordneten Paket her modellieren. Dazu wird das Symbol enthält an das Paketsymbol des übergeordneten Paketes gezeichnet. Die untergeordneten Pakete werden durch durchgezogene Linien mit diesem Symbol verbunden.

Beispiel eines Buchungssystems

In der Abbildung wird das Paket Buchungssystem modelliert, das aus den Teilsystempaketen Verkauf, Reservierung und Information besteht.



Für das Teil-Paket Verkauf des Paketes Buchungssystem wird der interne Aufbau modelliert. Das Paket Anfrage kommuniziert mit den internen Paketen Preisberechnung und Platzgruppen. Das externe Paket Internet-Verkauf kommuniziert ebenfalls mit dem Paket. Das Paket Steuerkarten nutzt die Funktionalität des Paketes Preisberechnung und beeinflusst damit den Preis.



Das Paket Abokonten stellt eine Spezialisierung des Paketes Platzgruppen dar, d. h., das Paket Abokonten wurde von dem Paket Platzgruppen abgeleitet.

7.6 Klassendiagramm

In Klassendiagrammen werden Klassen und ihre Beziehungen untereinander modelliert. Unter Beziehungen ist z. B. zu verstehen, dass Klassen untereinander Methoden aufrufen können (Assoziation). Weitere modellierbare Beziehungen sind die Aufnahme einer Klasse in eine zweite Klasse (Aggregation bzw. Komposition) und die Vererbungshierarchie (Spezialisierung bzw. Generalisierung).

Da eine Klasse die Struktur und das Verhalten von Objekten, die von dieser Klasse erzeugt werden, modellieren muss, können die Klassen im Klassendiagramm mit Methoden und Attributen versehen werden. Weiterhin ist die Modellierung von Basisklassen und Schnittstellen über Stereotypen möglich. In einigen Programmiersprachen können Sie Template-Klassen implementieren. Die UML stellt solche Klassen im Klassendiagramm als parametrisierbare Klassen dar.

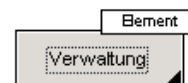
Klasse

Eine Klasse stellt das Grundelement des Klassendiagramms dar – den kleinsten Baustein. Im Diagramm wird eine Klasse als Rechteck gezeichnet. Dieses Rechteck kann in drei Teile gegliedert sein. Der oberste Abschnitt enthält den Namen der Klasse. Im folgenden Abschnitt werden die Attribute eingetragen. Der dritte Abschnitt enthält die Operationen der Klasse (Methoden).

Durch Stereotypen können z. B. abstrakte Klassen gekennzeichnet werden. Die Angabe des Stereotyps erscheint unter dem Klassennamen in französischen Anführungsstrichen « » . Sie können die Stereotypen auch durch unterschiedliche Farben oder durch die Kursivschreibweise des Namens der Klasse sichtbar machen.



Weiterhin ist es möglich, Klassenattribute und -methoden zu modellieren. Die Schutzklassen der Klassenelemente werden mit einem Zeichen vor dem Namen gekennzeichnet. Sind die Elemente öffentlich, steht vor dem Namen das Zeichen `+`. Privaten Elementen wird das Zeichen `-` vorangestellt. Das Zeichen `#` vor dem Namen bedeutet, dass das Klassenelement mit dem Zugriffsattribut `protected` gekennzeichnet ist. Sind Klassenattribute oder Klassenmethoden vorhanden, können Sie diese an dem unterstrichenen Namen erkennen.



Manche für die UML-Modellierung eingesetzte CASE-Tools (wie z. B. das in diesem Buch beschriebene und genutzte objectiF) kennzeichnen und unterscheiden abstrakte und nicht abstrakte Klassen anhand eines farbigen Dreiecks im unteren rechten Bereich der Klassendarstellung. Dieses ist zwar üblich und nützlich, aber kein offizieller UML-Notationsbestandteil.

Eine besondere Art von Klassen sind die parametrisierbaren Klassen. Bei diesen Klassen ist der Typ der enthaltenen Attribute noch nicht festgelegt. Die Festlegung erfolgt erst beim Instanzieren eines Objekts dieser Klassen. Für solche Klassen wird die grafische Darstellung abgewandelt. Das Rechteck für die Klasse bekommt im oberen Bereich ein zweites Rechteck mit einer Umrandung, in dem der variiere Typ steht.

Objekt

Objekte sind die eigentlichen konkreten agierenden Einheiten einer objektorientierten Anwendung. Sie werden nach einem Bauplan, der Klassendefinition, im Speicher erzeugt. Jedes Objekt hat eine eigene Identität. Ein Objekt besitzt ein bestimmtes Verhalten, das durch die Methoden bestimmt ist. Zwei Objekte der gleichen Klasse haben das gleiche Verhalten. Objekte besitzen weiterhin Attribute. Auch diese sind bei Objekten derselben Klasse gleich. Der Zustand eines Objekts wird durch die Werte bestimmt, die in den Attributen gespeichert sind.

<u>Carlos:Veranstaltung</u>
Name = Don Carlos Verdi
Datum = 17.06.2017
Zeit = 19.30 Uhr

Im Diagramm werden Objekte analog zu den Klassen mit einem Rechteck gezeichnet. Der Name wird unterstrichen, um sie von den Klassen unterscheiden zu können. Dem Namen des Objekts wird nach einem Doppelpunkt der Klassenname angefügt. Hat für den zu modellierenden Fall der konkrete Objektname keine Bedeutung, kann dieser auch entfallen. Es werden dann nur ein Doppelpunkt und der Klassenname dargestellt. Die Methoden sind in der Objektdarstellung irrelevant und werden entsprechend nicht angegeben.

Beziehungen

Eine Klasse muss für jede Nachricht, die sie empfängt, eine Operation besitzen. Über Nachrichten kommunizieren Objekte untereinander und erreichen so eine Koordinierung des Verhaltens. Es gibt drei wichtige Beziehungsarten:

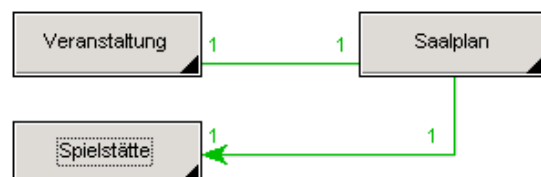
- ✓ Assoziation
- ✓ Aggregation/Komposition
- ✓ Generalisierung/Spezialisierung

Assoziation

Die Assoziation stellt die Kommunikation zwischen zwei Klassen im Diagramm dar. Die Klassen werden mit einer einfachen Linie verbunden. Mithilfe eines Pfeils können Sie eine gerichtete Assoziation modellieren. Bei dieser werden Nachrichten nur von einer Klasse zu einer zweiten Klasse versendet. Der umgekehrte Weg ist nicht notwendig. Für die Entkopplung von Klassen hat die gerichtete Assoziation eine wichtige Bedeutung. Im Diagramm können Sie weiterhin die Anzahl der Objekte bestimmen, die an der Beziehung teilnehmen. Dazu legen Sie die Kardinalität der Beziehung fest. Besteht eine Beziehung nur, wenn das Objekt eine bestimmte Rolle einnimmt, kann der Beziehung dieser Fakt über eine Beschriftung hinzugefügt werden.

Gelesen werden diese Beziehungen in folgender Weise:

- ✓ Eine Veranstaltung hat einen Saalplan.
- ✓ Ein Saalplan ist einer Spielstätte zugeordnet. Der Pfeil sagt aus, dass die Kommunikation überwiegend vom Saalplan ausgeht (bei der Implementierung erhält deshalb die Klasse eine Referenz auf die Spielstätte).

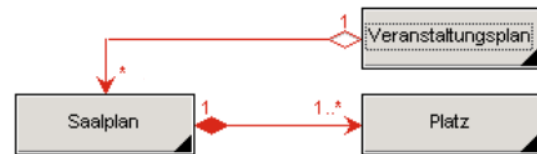


Aggregation/Komposition

Eine Aggregation stellt eine Klassenbeziehung dar, bei der ein Objekt einer Klasse Objekte weiterer Klassen in sich verwalten kann. Dabei gibt es die spezielle Form, dass die verwalteten Klassen ein notwendiger Bestandteil der Aggregatsklasse sind und diese nicht unabhängig allein existiert; in diesem Fall ist eine Komposition zu modellieren.

In der folgenden Abbildung ist zwischen der Klasse `Veranstaltungsplan` und der Klasse `Saalplan` eine Aggregation modelliert. Eine Komposition ist zwischen den Klassen `Saalplan` und `Platz` definiert. Es gibt somit kein `Saalplan`-Objekt ohne einen `Platz`. Gelesen werden die Beziehungen in folgender Weise:

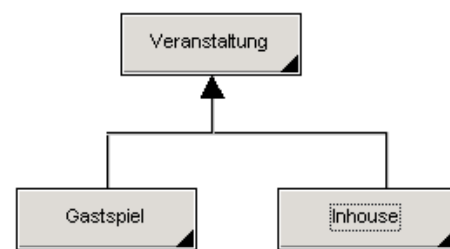
- ✓ Ein `Platz` gehört genau zu einem `Saalplan`.
- ✓ Jeder `Saalplan` gehört zu einem `Veranstaltungsplan`.
- ✓ Ein `Veranstaltungsplan` kann beliebig viele `Saalpläne` enthalten.



Generalisierung/Spezialisierung

Mit dieser Form werden Hierarchieebäume modelliert. Dabei werden weitere Klassen von einer Basisklasse abgeleitet, die deren nicht private Methoden und Attribute erben.

Aus der Sicht dieser abgeleiteten Klassen stellt die Basisklasse eine Generalisierung dar. Von der Basisklasse aus betrachtet sind die abgeleiteten Klassen eine Spezialisierung der Basisklasse.



Vererbungsbäume sind ein wichtiges Gestaltungselement bei der Modellierung von Software-Architekturen. In der Abbildung wird modelliert, dass die Veranstaltungen `Inhouse`- oder `Gastspiel`-Veranstaltungen sein können.

Stereotypen

Zur detaillierten Darstellung der Elemente werden Stereotypen verwendet. Beispielsweise kann für eine Klasse das Stereotyp `«abstract»` vergeben werden, wenn von dieser keine Objekte erstellt werden. Benötigen Sie zur Modellierung Stereotypen, die Sie noch nicht verwendet haben, können Sie weitere definieren. Der Vorteil von Stereotypen besteht darin, dass Sie alle Arten von Elementen genau modellieren können.

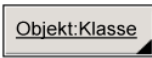
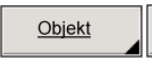
Die folgende Tabelle zeigt einige für Klassen verwendete Stereotypen. Sie werden in die Zeichen « und » eingeschlossen.

Stereotyp	Bedeutung
«utility»	Die Klasse enthält Funktionen, die sie anderen Klassen zur Unterstützung anbietet. Sie ist eine Hilfsklasse und unterstützt nur indirekt die Abarbeitung.
«type»	Die Klasse dient zur Definition einer abstrakten Spezifikation einer strukturellen Schnittstelle.
«primitive»	Primitive Klassen repräsentieren elementare Datentypen der verwendeten Programmiersprache.
«enumeration»	Die Klasse stellt eine aufzählbare und konfigurierbare Wertemenge – meist anwendungsweit – zur Verfügung (z. B. die Wochentage Montag bis Sonntag).
«structure»	Diese Klassen enthalten ausschließlich Attribute und dienen dem Datenaustausch mit anderen Systemen oder Subsystemen.
«entity»	Entitätsklassen repräsentieren einen fachlichen Sachverhalt. Sie haben wenig komplexe Operationen und enthalten eine größere Anzahl von Attributen.
«control»	Die Klassen repräsentieren einen Ablauf-, Steuerungs- oder Berechnungsvorgang und haben wenige oder keine Attribute, aber komplexe Operationen.
«boundary»	Diese Klasse stellt Schnittstellenobjekte bereit, die eine spezielle Sicht auf eine Menge anderer Objekte liefern.
«interface»	Schnittstellen sind abstrakte Definitionen rein funktionaler Schnittstellen. Sie haben keine Instanzen.
«abstract»	Von dieser Klasse können keine Objekte instanziiert werden.

Symbole

Die folgende Tabelle enthält die wichtigsten Symbole der Klassendiagramme.

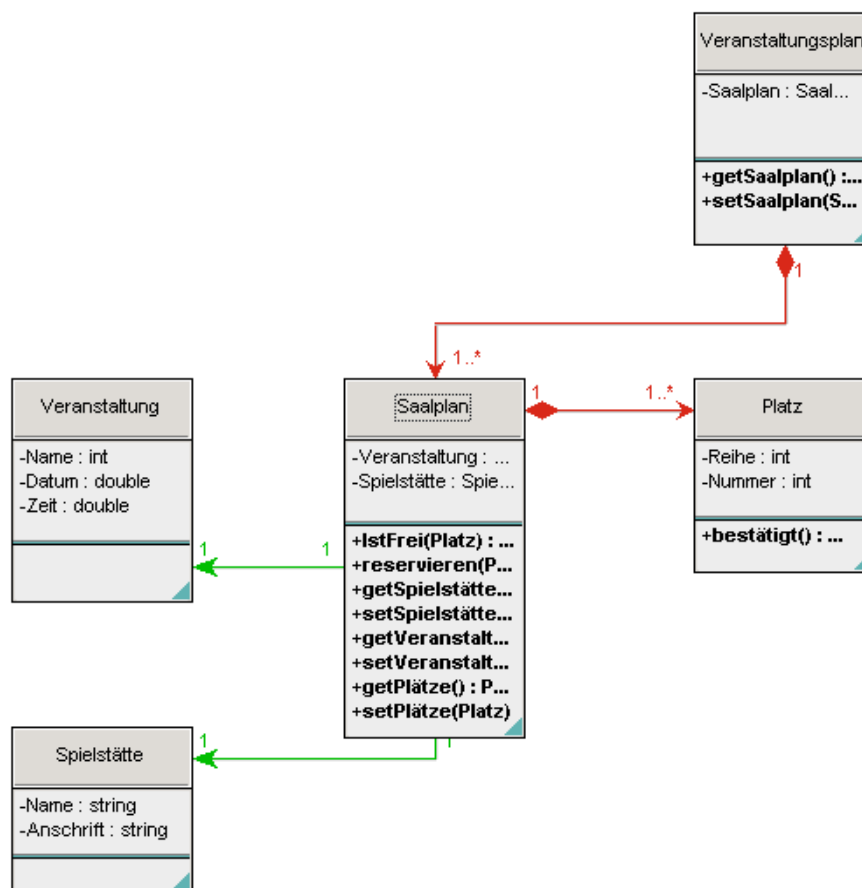
Name/Symbol	Verwendung
Klasse 	Eine Klasse wird im Diagramm mit einem Rechteck dargestellt, in dem der Name der Klasse angezeigt wird ①. Sie können das Klassensymbol mit einem Bereich zur Darstellung der Attribute und der Methoden erweitern ②. Für parametrierbare Klassen wird das Klassensymbol mit einem Rechteck erweitert, das den Namen des Parameters enthält ③.

Name/Symbol	Verwendung
Objekt ①  ②  ③ 	Objekte werden mit einem Rechteck dargestellt, in dem der Name und die Klasse des Objekts unterstrichen angezeigt werden ①. Beide werden durch einen Doppelpunkt getrennt. Sie können auch auf den Klassennamen verzichten, wenn das Objekt auch ohne Klassennamen eingeordnet werden kann ②. Ein anonymes Objekt (noch ohne Namen) wird nur mit einem Doppelpunkt und dem Klassennamen angegeben ③.
Schnittstelle Schnittstelle 	Eine Schnittstelle wird mit einem Kreis dargestellt. Dieser Kreis wird mit der Klasse durch eine durchgezogene Linie verbunden. Der Name der Schnittstelle wird neben dem Symbol angegeben.
Assoziation Klasse A Klasse B ① Klasse A 1..* Klasse B ② Person 1..* < klagt an Angeklagter Ankläger Staat ③ Klasse A > Klasse B ④	Stehen zwei Klassen in einer Beziehung, wird das durch die Verbindung der beiden Klassen mit einer durchgezogenen Linie dargestellt ①. Auf der Linie kann in der Nähe des Klassensymbols die Kardinalität angegeben werden ②. Auf der Linie können weiterhin der Name der Beziehung oder die Rolle angegeben werden ③ (Rollen: Angeklagter, Ankläger; Beziehungsname: klagt an). Sie können mit einem Pfeil die vorrangige Richtung der Kommunikation angeben ④.
Aggregation Klasse A ◊ Klasse B	Die Aggregation wird mit einer Linie, an deren einem Ende ein Rautensymbol zur Klasse des Containers zeigt, dargestellt (Klasse B kann Objekte der Klasse A verwalten). Das Rautensymbol ist nicht gefüllt. Kardinalität und vorrangige Richtung der Kommunikation werden wie bei der Assoziation angegeben.
Komposition Klasse A ◼ Klasse B	Die Komposition wird im Unterschied zur Aggregation mit einer gefüllten Raute dargestellt. Die weitere Symbolik entspricht der der Aggregation.
Generalisierung Klasse A ▷ Klasse B	Das Symbol ist ein Pfeil, der auf die Basisklasse zeigt und diese mit der abgeleiteten Klasse verbindet. Der Pfeil wird nicht gefüllt dargestellt (Klasse A erbt von der Klasse B).
Spezialisierung Klasse A ▷ <<Interface>> Interface	Importiert eine Klasse eine Schnittstelle (Interface), wird die Verbindung mit einem Pfeil und einer unterbrochenen Linie dargestellt.

Beispiel

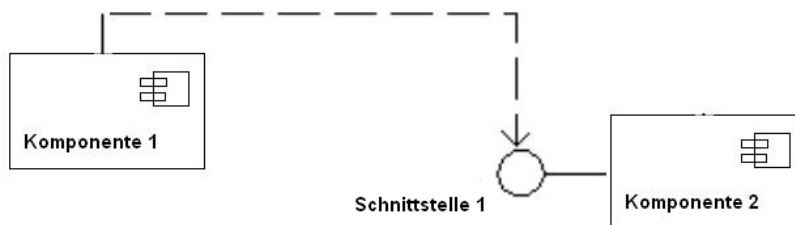
Bei einem Ticketsystem für ein Schauspielhaus werden über einen Veranstaltungsplan alle Stücke des Theaters verwaltet.

Die Klasse `Veranstaltungsplan` ist mit der Klasse `Saalplan` über eine Komposition verbunden. Somit kann die Klasse `Veranstaltungsplan` Objekte der Klasse `Saalplan` verwalten. Jeder `Saalplan` enthält die Plätze des Theaters und diese werden über den `Saalplan` verkauft. Die Klasse `Saalplan` verwaltet über eine weitere Komposition Objekte der Klasse `Platz`. Die Navigation ist in Richtung der Klasse `Platz` gerichtet. Die Klasse `Saalplan` kommuniziert mit der Klasse `Veranstaltung`, in der die Daten der Aufführung ausgelagert sind (z. B. Datum und Uhrzeit). Die Angaben zur Spielstätte (z. B. der Name und die Adresse) wurden in die Klasse `Spielstätte` ausgelagert. Die Navigation erfolgt von der Klasse `Saalplan`.



7.7 Komponentendiagramm

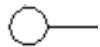
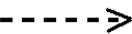
Komponentendiagramme zeigen die Beziehungen der Komponenten untereinander. Eine Komponente ist eine ausführbare und austauschbare Softwareeinheit mit definierten Schnittstellen und eigener Identität. Eine Komponente ist ähnlich einer Klasse instanzierbar und kapselt eine komplexe Funktionalität.



Die Komponente Komponente 2 stellt über ihre Schnittstelle eine Funktionalität zur Verfügung. Die Komponente Komponente 1 kommuniziert über die von der Komponente Komponente 2 angebotene Schnittstelle 1 und verwendet deren Funktionalität.

Symbole

Die folgende Tabelle enthält die wichtigsten Symbole der Komponentendiagramme.

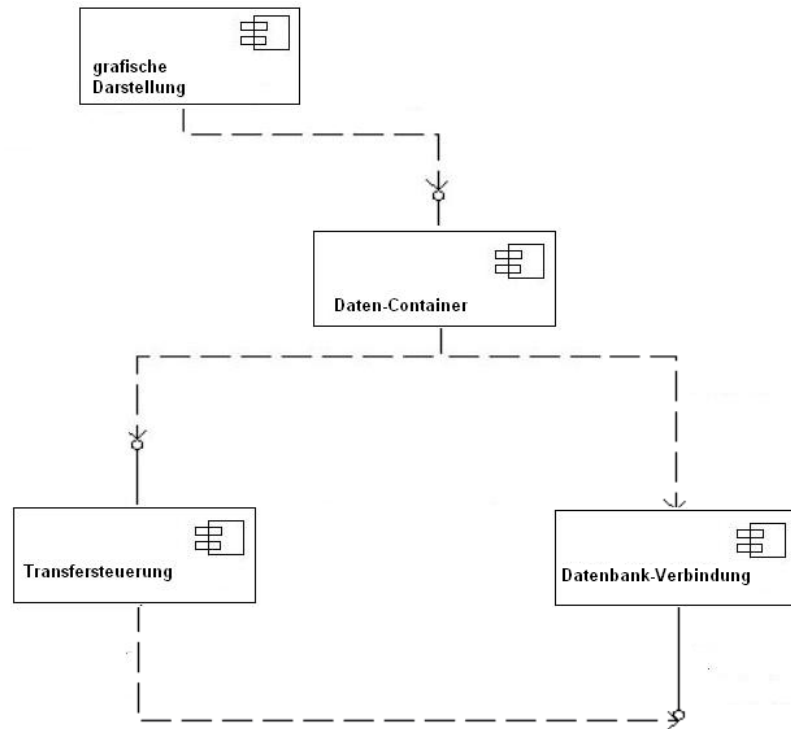
Name/Symbol	Verwendung
Komponente 	Das Symbol der Komponente besteht aus einer Anordnung von vier Rechtecken. Zwei Rechtecke werden auf der linken Seite eines dritten, größeren Rechtecks gezeichnet, diese drei werden wiederum von einem vierten Rechteck umschlossen, in diesem steht der Name der Komponente. Die Komponenten erfüllen vollständig die Aufgaben eines abgesteckten Bereiches, für die sie entwickelt wurden.
Schnittstelle 	Eine Komponente kann eine Schnittstelle anbieten. Über diese erfolgt ein verwalteter Zugriff auf die Funktionalität der Komponente.
Abhängigkeit 	Der Pfeil mit gestrichelter Linie symbolisiert den Zugriff auf die Schnittstelle. Der Pfeil zeigt auf den Kreis des Schnittstellensymbols. Er kann aber auch direkt auf eine Komponente zeigen, wenn nicht über die Schnittstelle zugegriffen wird.

Beispiel: Datenbankanbindung mit Komponenten

Im Beispiel werden vier Komponenten zu einer Anwendung verknüpft, um Daten einer Datenbank in einem Fenster anzuzeigen. Die Datenbank-Verbindungs-Komponente ist für den Aufbau, die Verwaltung und den Abbau der Verbindung zur Datenbank verantwortlich.

Die Daten-Container-Komponente verwaltet die angeforderten Datensätze. Gleichzeitig wird bei jedem Datentransfer eine Transaktion über die Schnittstelle der Transfersteuerungs-Komponente gestartet. Die Komponente für die grafische Darstellung ruft über die Schnittstelle der Daten-Container-Komponente die Datensätze ab und stellt sie auf dem Bildschirm dar.

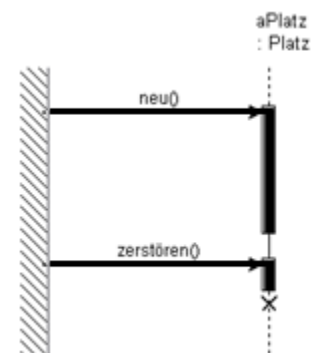
Ändert der Anwender die Daten, so übergibt die Komponente die Änderungen an die Daten-Container-Komponente. Die geänderten Daten werden von dieser über die Datenbank-Verbindungs-Komponente in die Datenbank geschrieben und die neuen Daten werden über die Transfersteuerungs-Komponente bestätigt, worauf die gestartete Transaktion beendet wird.



7.8 Sequenzdiagramm

In einem Sequenzdiagramm wird das zeitliche Verhalten der Nachrichtenübermittlung modelliert. Sie können festlegen, wann ein Objekt erstellt wird und wann Nachrichten zu welchem Objekt gesendet werden. Objekte werden auf Lebenslinien gezeichnet. Mit einem Kreuz können Sie die Objektzerstörung kennzeichnen.

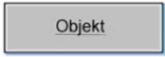
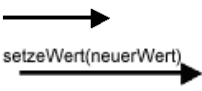
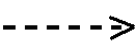
Im Sequenzdiagramm wird eine Nachricht durch einen Pfeil dargestellt, über den der Name der Nachricht geschrieben wird. Synchroner Nachrichten werden mit einer gefüllten Pfeilspitze, asynchrone Nachrichten mit einer ungefüllten Pfeilspitze gezeichnet.



Durch Voranstellen des Zeichens  modellieren Sie das wiederholte Senden der Nachricht. Wird über das Absetzen einer Botschaft ein Objekt erzeugt (z. B. über den Aufruf der Methode `neu()`), beginnt die Lebenslinie des Objekts erst an dieser Position. Die zeitliche Reihenfolge der Nachrichten entspricht ihrer horizontalen Position im Diagramm.

Symbole

Die folgende Tabelle enthält die wichtigsten Symbole der Sequenzdiagramme.

Name/Symbol	Verwendung
Systemgrenze 	Die Systemgrenze isoliert den betrachteten Programmteil vom übrigen Programm. Sie dient meist als Ausgangspunkt des auslösenden Methodenaufrufs. Nicht immer wird der Programmfluss von einem Objekt außerhalb des betrachteten Bereichs ausgelöst, sodass in diesem Fall keine Systemgrenze eingezeichnet werden muss
Objekt 	Ein Objekt wird mit einem Rechteck, das den Namen enthält, dargestellt. Das Unterstreichen des Namens kann hier entfallen, da keine Verwechslung mit dem Klassennamen auftreten kann. Klassen werden in diesem Diagramm nicht dargestellt, lediglich die Klassenzugehörigkeit kann zusätzlich und in gewohnter Weise (:Klasse) dokumentiert werden.
Lebenslinie 	Jedes Objekt liegt auf einer senkrechten Linie, der Lebenslinie. Die Lebensdauer des Objekts nimmt in Richtung untere Blattkante zu. Für Objekte, die zu Beginn des Programmteils bereits existieren, werden die Objektsymbole an der oberen Blattkante gezeichnet. Für Objekte, die innerhalb des Programmteils neu erstellt werden, wird das Symbol in Höhe des Methodenaufrufs gezeichnet, in dessen Folge das Objekt erstellt wird.
Objekt-Aktivität 	Ist ein Objekt an einem Methodenaufruf beteiligt, wird es aktiv. Die Lebenslinie verdickt sich. Ruft ein Objekt eine eigene Methode auf, verdickt sich die Lebenslinie ein weiteres Mal. Diese Aktivitäten werden nicht immer mitgezeichnet.
Methodenaufrufe 	Ruft ein Objekt eine Methode eines andern Objekts auf, wird das über einen Pfeil symbolisiert, der auf das Objekt zeigt, dessen Methode aufgerufen wird. An dieses Symbol wird der Methodenname geschrieben. Diesem Namen kann in Klammern die Parameterliste angefügt werden.
Rückkehr einer Methode 	Prinzipiell werden nur die Methodenaufrufe in das Sequenzdiagramm eingezeichnet. Möchten Sie dennoch die Rückkehr der Methoden einzeichnen, können Sie dies mit einem Pfeil und unterbrochener Linie vornehmen.
Objekt-Erstellung 	Erstellt eine Methode ein Objekt, endet der Pfeil der Methode an dem rechteckigen Symbol des Objekts. Die Lebenslinie beginnt bei diesem Symbol.
Objekt-Zerstörung 	Wird durch den Aufruf einer Methode ein Objekt zerstört, endet die Lebenslinie des Objekts mit einem Kreuz unterhalb des Symbols des Methodenaufrufs.

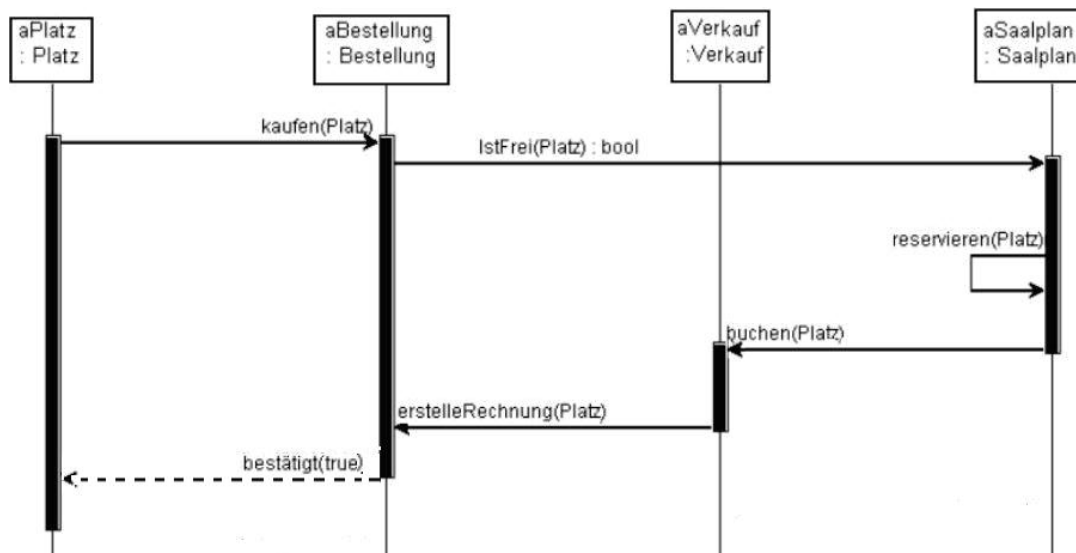
Beispiel

In einem Ticketsystem für ein Schauspielhaus werden auf einer Internetseite Eintrittskarten aus dem Saalplan heraus verkauft. Der Saalplan verwaltet die Sitzplätze einer Veranstaltung.

Wird ein Platz von einem Besucher ausgewählt, ruft das betreffende Objekt `aPlatz` die Methode `kaufen` des Objekts `aBestellung` auf und übergibt eine Referenz auf sich selbst im Parameter. Das Objekt der Klasse `Bestellung` ruft die Methode `IstFrei` der Klasse `Saalplan` auf, um zu überprüfen, ob der im Parameter übergebene Platz noch frei ist. Ist der Platz noch frei, ruft das `Saalplan`-Objekt seine eigene Methode `reservieren` auf. Damit wird der Platz zunächst reserviert.

Nachdem dies erfolgt ist, wird die Rechnung für den Platz erstellt. Dazu ruft das `Saalplan`-Objekt die Methode `buchen` mit dem ausgewählten Platz als Parameter auf. Die Methode `buchen` gehört zum Objekt `aVerkauf`. Dieses stellt eine Liste der Rechnungspositionen dar, was hier nicht modelliert ist.

Nachdem die Rechnungspositionen vom Verkaufs-Objekt zusammengestellt wurden, ruft dieses die Methode `erstelleRechnung` des Objekts `aBestellung` auf. Um dem Internet-Besucher den Erfolg seines Kaufes mitzuteilen, ruft das `aBestellung`-Objekt die Methode `bestätigt` des Objekts `aPlatz` auf und die Methode `bestätigt` kehrt mit dem Wert `true` (wahr) zurück. Die Bestellung ist abgeschlossen.



7.9 Kommunikationsdiagramm

Im Kommunikationsdiagramm wird besonderes Augenmerk auf die Zusammenarbeit der Objekte untereinander gelegt. Dazu werden ausgewählte Nachrichten verwendet, mit denen die zeitliche Abfolge der Kommunikation zwischen den Objekten erfolgt. Das Kommunikationsdiagramm stellt somit in kompakterer Form die Aussagen des Sequenzdiagramms zusammen.

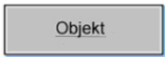
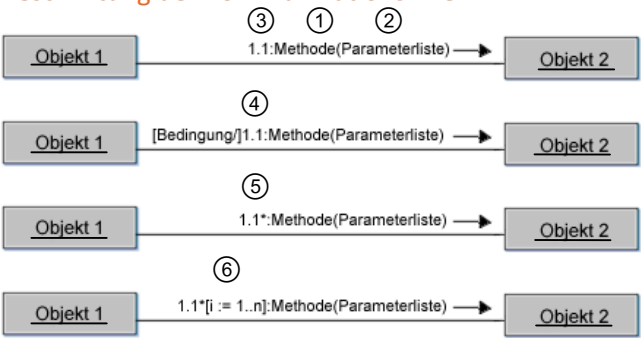
Die grafische Darstellung besteht aus einem Rechteck, das den Objektnamen und die zugehörige Klasse enthält. Ein Doppelpunkt trennt beide Namen. Die Objekte können Sie mit Assoziationslinien verbinden. Ein kleiner Pfeil zeigt jeweils die Richtung der Nachricht vom Sender zum Empfänger an. Wenn mit der Nachricht Argumente übergeben werden, sind diese aufgeführt. Mögliche Rückgabewerte können Sie ebenfalls in der Form `antwort := Nachrichtenname (Parameterliste)` angeben.

Um den zeitlichen Ablauf zu modellieren, erhalten die Nachrichten Nummern. Durch eine Nachricht können weitere Nachrichten ausgelöst werden, diese erhalten dann Unternummern der auslösenden Nachricht (z. B. 1.2). Wird eine Nachricht mehrfach ausgelöst, kann durch ein Zeichen  vor dem Nachrichtennamen diese Wiederholung modelliert werden.

Objekte, die innerhalb des dargestellten Szenarios erzeugt werden, können Sie mit dem Stereotyp `new` kennzeichnen. Für Objekte, die innerhalb des dargestellten Szenarios zerstört werden, erfolgt die Bezeichnung mit dem Stereotyp `destroy`. Objekte, die innerhalb des Szenarios erzeugt und zerstört werden, erhalten das Stereotyp `transient`.

Symbole

Die folgende Tabelle enthält die wichtigsten Symbole der Kommunikationsdiagramme.

Name/Symbol	Verwendung
Objekt 	Das Rechteck ist das Symbol eines Objekts und enthält den Objektnamen. Da keine Verwechslung mit Klassen eintreten kann, müssen Sie den Namen nicht unterstreichen. Die Klassenzugehörigkeit kann zusätzlich und in gewohnter Weise (<code>:Klasse</code>) dokumentiert werden.
Kommunikation 	Kommunizieren zwei Objekte über einen Methodenaufruf miteinander, wird diese Verbindung mit einer durchgängigen Linie dargestellt, die beide Objekte verbindet.
Kommunikationsrichtung 	Zusätzlich zur Verbindungslinie wird nach dem Methodennamen durch einen Pfeil angezeigt, zu welchem Objekt die Methode gehört. Die Pfeilspitze zeigt auf dieses Objekt.
Beschriftung der Kommunikationslinien  Die Linie wird mit dem Methodennamen ① und der Parameterliste ② beschriftet. Eine Nummerierung ③ wird zur Kennzeichnung der zeitlichen Reihenfolge der Methodenaufrufe vor dem Methodennamen angezeigt und durch einen Doppelpunkt von diesem getrennt. In einer Bedingung können Sie die Nummern der Methoden angeben ④, die als Voraussetzung bereits abgearbeitet wurden. Das Zeichen  vor dem Methodennamen kennzeichnet eine Wiederholung der Methode ⑤. Die Bedingung für die Wiederholung können Sie nach dem Zeichen  angeben ⑥.	

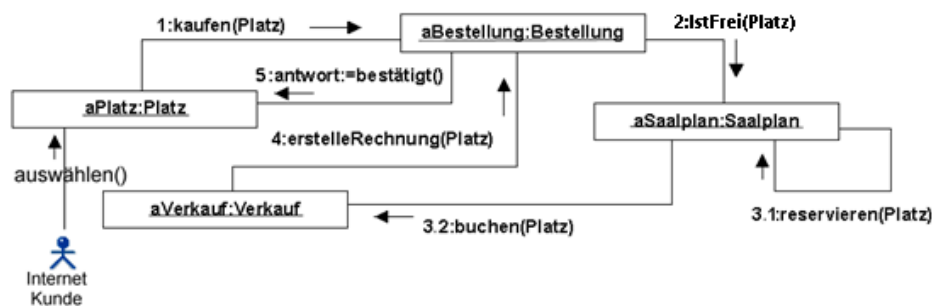
Beispiel

Das Beispiel modelliert den Kauf eines Tickets über das Internet. Der Internetkunde wählt einen Platz auf der Internetseite eines Schauspielhauses aus und ruft damit die Methode `auswählen` auf.

Das Objekt `aPlatz` ruft die Methode `kaufen` des `aBestellung`-Objekts auf und übergibt eine Referenz auf sich selbst im Parameter. Das Objekt `aBestellung` ruft die Methode `Ist-Frei` des `aSaalplan`-Objekts auf. Diese Methode ruft zum einen die Methode `reservieren` und zum anderen die Methode `buchen` auf, diese unterscheiden sich deshalb nur in der Unternummer.

Durch den Aufruf der Methode `buchen` des Objekts `aVerkauf` wird die Methode `erstelleRechnung` des `aBestellung`-Objekts aufgerufen, die Rechnungsposition des gebuchten Platzes wird als Parameter übergeben.

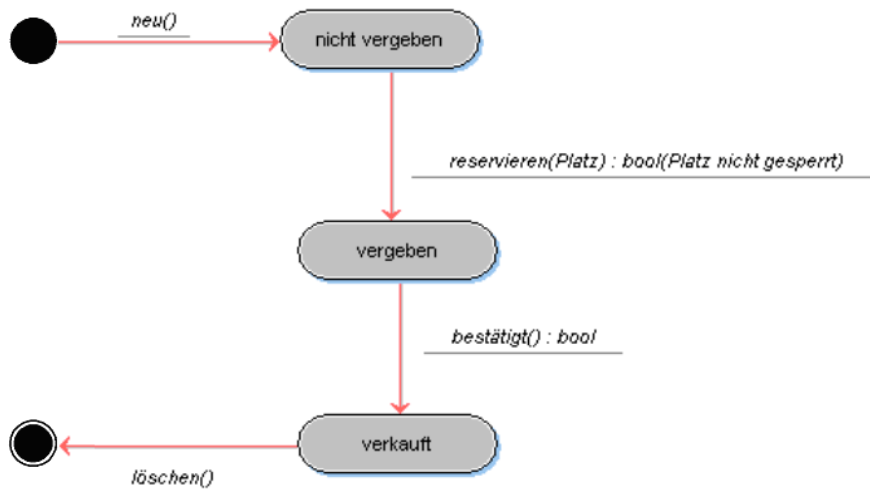
Nachdem die Rechnung vom Objekt `aBestellung` erstellt wurde, erfolgt die Benachrichtigung des Internetkunden über die erfolgreiche Buchung. Dazu wird die Methode `bestätigt` aufgerufen, die die Bestätigung des Kunden erfragt und zurückgibt.



7.10 Zustandsdiagramm

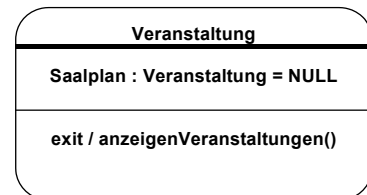
Ein Zustandsdiagramm zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann, sowie die Ursachen der Zustandsänderungen.

Sie können für ein Objekt den Zustand und die Änderung des Zustandes – von der Initialisierung bis zur Freigabe – in Abhängigkeit von ausgeführten Operationen modellieren. Weiterhin ist im Zustandsdiagramm ersichtlich, welche Belegung die Attribute des Objekts vor einem Übergang besitzen oder besitzen müssen.



Zustandsdiagramm für ein Objekt der Klasse *Platz*

Zustände werden durch abgerundete Rechtecke modelliert. Sie können einen Namen beinhalten und optional durch horizontale Linien in bis zu drei Bereiche eingeteilt werden.



Im obersten Bereich steht der Name des Zustandes. Wenn Sie den Namen nicht angeben, handelt es sich um einen anonymen Zustand. In einem weiteren Bereich können Sie existierende Zustandsvariablen mit der für diesen Zustand typischen Wertebelegung aufführen. Dies gilt auch für die Attribute der Klassen. Die Angabe hat das folgende Format:

```
variable : Klasse = Initialwert (Merkmal) {Zusicherung z. B.
Merkmal < Grenzwert}
```

Der dritte Bereich innerhalb des Zustandssymbols kann eine Liste von internen Ereignissen, Bedingungen und aus ihnen resultierende Operationen enthalten. Sie werden in dem folgenden Format notiert:

```
Ereignis / Aktionsbeschreibung
```

Ereignis steht dabei für drei mögliche Verhaltensweisen:

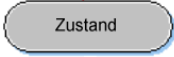
- ✓ **entry** – löst automatisch beim Eintritt in einen Zustand aus.
- ✓ **exit** – löst automatisch beim Verlassen eines Zustandes aus.
- ✓ **do** – wird immer wieder ausgelöst, solange der Zustand nicht gewechselt wird.

Sie können einen Zustand mehrfach in das Diagramm einfügen, um die Übersichtlichkeit zu erhöhen. Übergänge von einem Zustand zum nächsten werden durch Ereignisse ausgelöst. Ein Ereignis besteht aus einem Namen und einer Liste möglicher Argumente.

Ein Zustand kann Bedingungen an dieses Ereignis knüpfen, die erfüllt sein müssen, damit dieser Zustand durch dieses Ereignis eingenommen wird. Diese Bedingungen können unabhängig von einem speziellen Ereignis sein.

Symbole

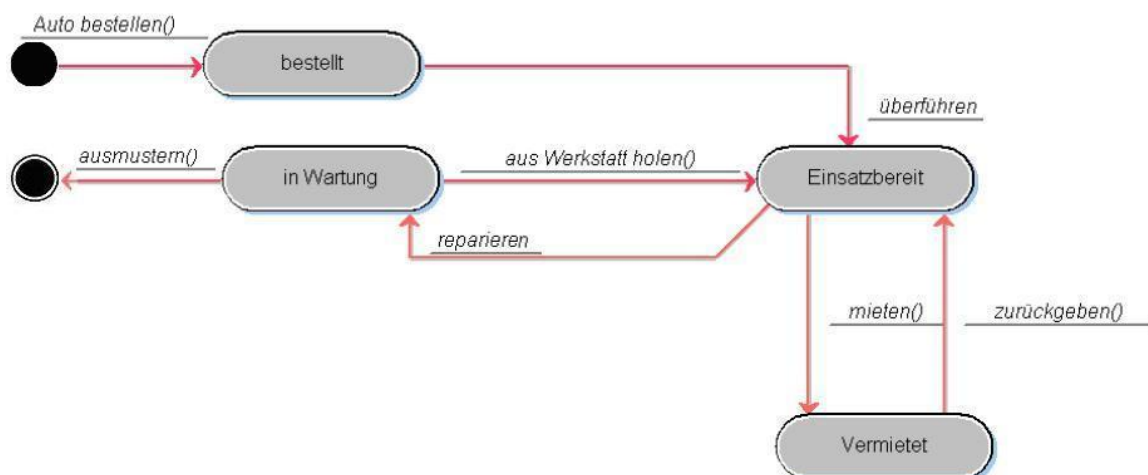
Die folgende Tabelle enthält die wichtigsten Symbole der Zustandsdiagramme.

Name/Symbol	Verwendung
Zustand 	Der Zustand eines Objekts wird durch ein Rechteck mit abgerundeten Ecken symbolisiert. Innerhalb dieses Symbols wird der Zustand benannt.
Objekterzeugung 	Der Startpunkt des Zustandsdiagramms wird mit einem gefüllten Kreis dargestellt. Er ist identisch mit der Objekterstellung.
Objektzerstörung 	Die Kette der Zustandsübergänge endet mit der Objektzerstörung. Der Endpunkt wird mit einem gefüllten Kreis dargestellt, den ein konzentrischer Kreis umgibt.
Programmfluss 	Über einen Pfeil wird der Programmfluss eingezeichnet. Der Pfeil wird mit dem Namen der Methode beschriftet, die den Objektzustand ändert. Sie können in Klammern eine Restriktion angeben, die bewirkt, dass nur bei deren Erfüllung der Objektzustand geändert wird.

Beispiel

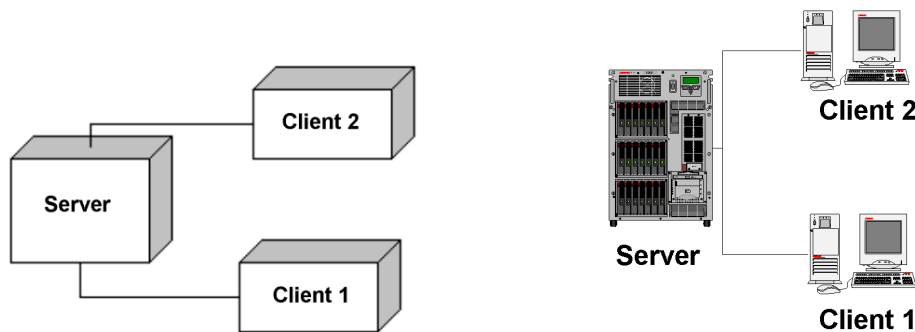
Bei einer Anwendung für eine Autovermietung wurde die Klasse `Auto` modelliert, für die ein Zustandsdiagramm erstellt wird.

Nach der Erstellung des Objekts hat es den Zustand `bestellt`. Nach der Überführung steht das Auto bei der Autovermietung. Das `Auto`-Objekt nimmt den Zustand `Einsatzbereit` an. Wenn das Auto vermietet wird, geht es in den Zustand `Vermietet` über (Ereignis `mieten`). Nach der Rückgabe hat es den Zustand `Einsatzbereit` wieder angenommen. Von Zeit zu Zeit geht das Auto in die Wartung. Das `Auto`-Objekt nimmt den Zustand `in Wartung` nach dem Ereignis `reparieren` an. Nach der Wartung wird das Objekt über das Ereignis `aus Werkstatt holen` wieder in den Zustand `Einsatzbereit` versetzt. Hat die Werkstatt feststellen müssen, dass das Auto nicht mehr zu reparieren ist, tritt das Ereignis `ausmustern` ein und das `Auto`-Objekt wird zerstört.



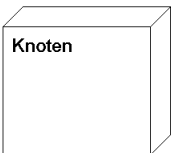
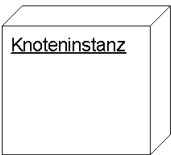
7.11 Einsatz- und Verteilungsdiagramm

Verteilungsdiagramme zeigen, welche Komponenten und Objekte auf welchen Knoten (Computer, Systeme, Prozesse) laufen, d. h., wie diese konfiguriert sind und welche Kommunikationsbeziehungen bestehen. Die Knoten werden als Quader gezeichnet. In den Quadern können Komponenten oder Laufzeitobjekte (Prozesse) eingetragen werden. Statt der Quader können Sie auch Bilder verwenden, die den Knoten grafisch darstellen.



Symbole

Die folgende Tabelle enthält die wichtigsten Symbole der Einsatz- und Verteilungsdiagramme.

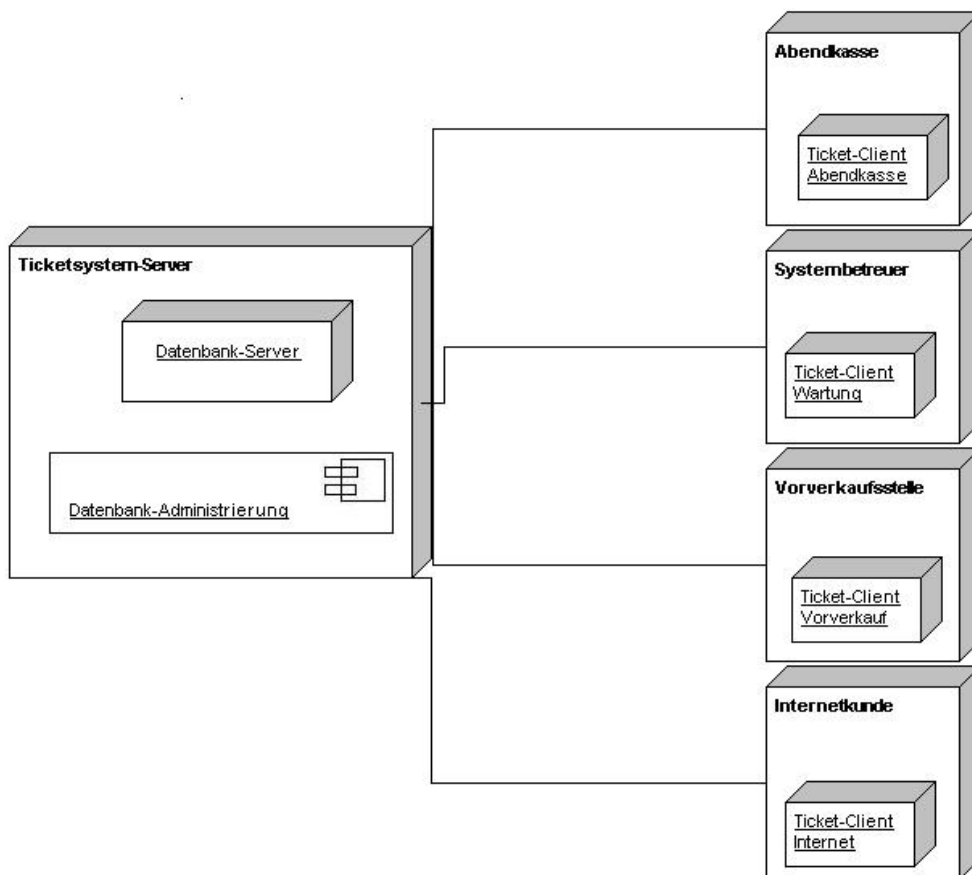
Name/Symbol	Verwendung
Knoten 	Ein Knoten wird mit einem Quader dargestellt. Die Beschriftung, die innerhalb des Quaders angegeben wird, erhält den Namen des Knotens, z. B. einen Rechnernamen, einen Client- oder einen Prozessnamen.
Teilsystem 	Wird auf einem Knoten eine Anwendung im Speicher ausgeführt, wird diese Anwendung als Knoteninstanz modelliert. Zur Unterscheidung vom Knotensymbol wird die Beschriftung im Quader unterstrichen. Dieses Symbol wird in das Knotensymbol eingefügt, auf dem der Teilprozess abläuft
Komponente 	Eine Komponente ist vom Umfang kleiner als ein Teilsystem, übernimmt aber für einen festgelegten Anwendungsbereich die Aufgaben der Anwendung. Das Komponentensymbol wird in das Knotensymbol eingefügt, auf dem die Komponente installiert wurde.

Name/Symbol	Verwendung
Komponenten- instanz 	Die Komponenteninstanz ist ein Sammelbegriff für die Objekte, deren Klassen zur Datenstruktur der Komponente gehören und von dieser instanziiert werden. Das Symbol unterscheidet sich nicht vom Symbol der Komponenten. Im Unterschied zum Komponentensymbol wird der Name der Instanz unterstrichen dargestellt. Das Symbol wird in das Komponentensymbol eingefügt, dessen Instanz es darstellt.
Beziehungslinie 	Eine Beziehung zwischen den Knoten wird mit einer durchgezogenen Linie dargestellt, die die Knoten verbindet.

Beispiel

Bei einem Ticketsystem für Lichtspieltheater kann aus verschiedenen Anwendungen heraus auf die Ticketdaten zugegriffen werden. Die Daten liegen auf einem zentralen Server, der mit weiteren PCs vernetzt und mit dem Internet verbunden ist. Auf dem Ticketsystem-Server ist ein Datenbankserver und eine Komponente zur Administration der Datenbank installiert.

Der Verkauf an der Abendkasse, der Systembetreuer, die Vorverkaufsstelle sowie der Internetkunde verfügen über eine den jeweiligen Anforderungen entsprechende/spezifische Client-Anwendung, die mit dem Ticketsystem-Server verbunden ist.



8

Praxisbeispiel – Ticketsystem

8.1 Projektvorstellung

In diesem Kapitel wird eine Anwendung zum computergestützten Ticketverkauf eines Opernhauses entwickelt. Mit objektorientierter Analyse und Design wird eine entsprechende Klassenstruktur entworfen.

Die notwendigen Diagramme können mithilfe eines CASE-Tools (vgl. Beschreibung/Übersicht in Kapitel 9) erstellt werden. In diesem Beispiel wurde das CASE-Tool objectiF der Berliner Firma microTOOL eingesetzt, das in unterschiedlichen Versionen und in einer kostenlosen 30-Tage-Testversion (z. B. unter http://www.chip.de/downloads/objectiF-Enterprise-Trial-Edition_18828757.html) zum Download im Internet angeboten wird. Eine Anleitung zur Installation und Bedienung dieser Software erhalten Sie im Kapitel 10 dieses Buchs.

Die Erarbeitung erfolgt nach der Methode des anwendungsfallgetriebenen Entwurfs, bei dem die Aktivitäten aus der Sicht der Bediener und nicht aus technischer Sicht untersucht und beschrieben werden. Für dieses Projekt wird vereinfachend angenommen, dass weder alte Daten existieren noch Anpassungen an bestehende Systeme notwendig sind.

Ticketsystem-Anwendung

Kartenverkaufssysteme (Ticketsysteme) dienen dem Verkauf von Eintrittskarten für Veranstaltungen. Für das Projekt in diesem Kapitel ist die Entwicklung eines Ticketverkaufssystems geplant, bei dem die Daten aller Vorgänge zu den Veranstaltungen im eigenen Haus gespeichert werden (Inhouse-Lösung). Die Anbindung der Vorverkaufsstellen erfolgt, mittels grafischer Programmoberfläche, über das Internet.

Bei der ersten Befragung des Auftraggebers wurden folgende Möglichkeiten des Systems gewünscht:

Verkaufsmöglichkeiten	Tickets für Veranstaltungen können an der Abendkasse, über Vorverkaufsstellen und über das Internet verkauft werden.
Abo-Verkauf	Der Kartenverkauf kann über mehrere Abo-Formen, z. B. Wahl-Abo, Platzmieten-Abo, Platzgruppen-Abo, oder als Einzelkarte erfolgen.
Veranstaltung	Jede Veranstaltung wird separat in das System eingepflegt und aus dem System heraus verkauft. Der Zugriff auf die einzelnen Veranstaltungen einer Spielzeit ist über den Spielplan möglich.
Ticketausgabe	An der Abendkasse werden Tickets gedruckt, die als Eintrittskarte und gleichzeitig als Kaufquittung dienen. Ein Ausdruck einer separaten Kaufquittung ist auf Verlangen ebenfalls möglich.
Vorverkaufsstellen	Vorverkaufsstellen erhalten Sammelrechnungen auf die von ihnen verkauften Karten. Der Rechnungsbetrag wird per Lastschrift eingezogen.
Saalplan	Für jede Veranstaltung wird mindestens ein individueller Saalplan erstellt. Um die Anfertigung zu beschleunigen, können aus einer Vorgabenliste Standardlösungen übernommen werden.
Sitzplatz	Jedem Platz ist eine eindeutige Platznummer und -reihe zugeordnet. Für jeden Platz kann eine Preisgruppe und eine Bestplatznummer vergeben werden. Ein Platz kann für eine Veranstaltung aus technischen Gründen vollständig gesperrt werden.
Rabattstaffel	Für jede Veranstaltung wird eine Rabattstaffel angelegt. In dieser Rabattstaffel werden die Schüler-, Studenten- und Rentner-Rabatte hinterlegt.
Steuerkarten	Für den Auftraggeber ist die Behandlung von Steuerkarten wichtig, die mit einem ermäßigten Preis an Mitarbeiter der Kulturstätte verkauft werden. Dabei muss die Gesamtsumme der Ermäßigungen erfasst werden, die jeder Angestellte im Laufe eines Jahres erhalten hat.
Bezahlung	Die Bezahlung von Eintrittskarten kann bar, über Rechnung mit Überweisung, per Lastschrift oder Kreditkarte erfolgen.
Technik	Für die Ablage der Daten steht im eigenen Haus ein SQL-Server bereit, der über eine feste IP-Adresse über eine Standleitung mit dem Internet verbunden ist. Vorverkaufsstellen verbinden sich per Internet mit diesem Server und können über eine spezielle Oberfläche Eintrittskarten verkaufen, die für diese Verkaufsstelle freigegeben wurden. Kunden, die über das Internet kaufen, können sich ebenfalls über das hauseigene Internetportal, das bei einem beliebigen Internetprovider gehostet ist, mit dem Datenbankserver im eigenen Haus verbinden.
Tickets	Die Tickets sind mithilfe von 2-D-Strichcode gegen Missbrauch geschützt. Dieser Code wird bei Betreten der Veranstaltung durch das Einlasspersonal mit einem Lesegerät kontrolliert. Kodiertes kostenintensives Ticketmaterial ist nicht notwendig. Für den Druck sind normale Laser- oder Thermotransferdrucker ausreichend.

8.2 Objektorientierte Analyse für das Projekt

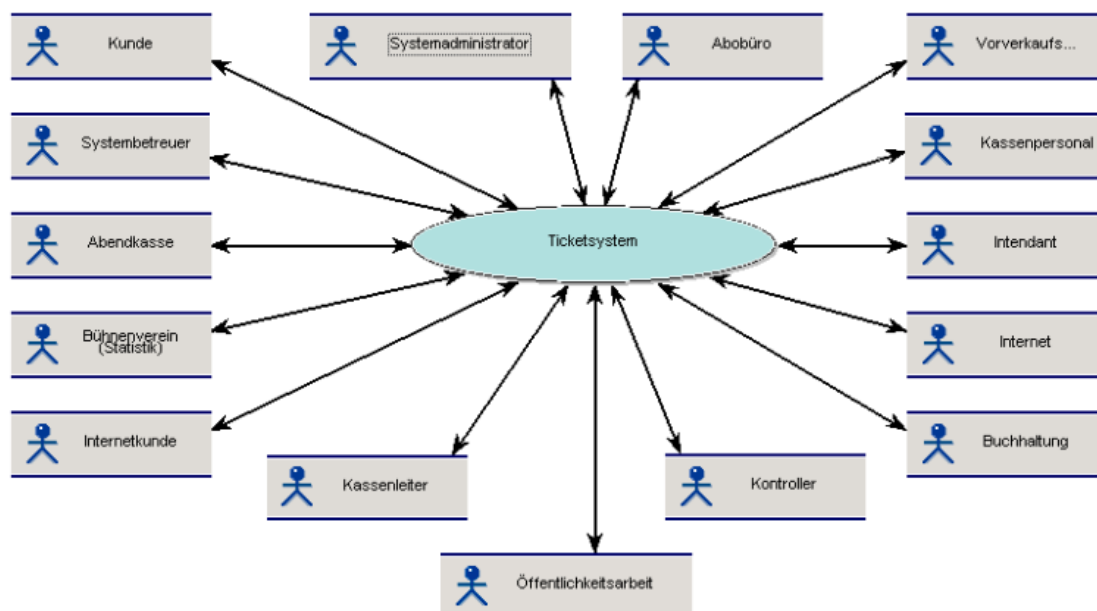
Die Analyse der Prozesse, die durch die zu entwickelnde Anwendung unterstützt werden sollen, erfolgt in mehreren Schritten. Dabei werden bereits erstellte Modelle durch detailliertere ersetzt und somit schrittweise an die Aufgaben in der Praxis angepasst.

Herausfinden der Akteure und Anwendungsfälle

Die Aufgabe für die neue Anwendung ist im vorausgegangenen Abschnitt umrissen worden. Sie können jetzt die Akteure/Projektbeteiligten herausfinden, die mit der Software arbeiten oder in Kontakt mit der Software stehen; diese werden in einer Tabelle oder einem Anwendungsfalldiagramm erfasst. Als einziger Anwendungsfall ist zunächst das System als Ganzes zu sehen.

Als Akteure kommen außer den Personen auch andere Systeme oder Zeitereignisse in Betracht (z. B. Wegfall der Vorverkaufsgebühr 30 Minuten vor Veranstaltungsbeginn).

Akteure/Projektbeteiligte		
Kunde	Internet	Systemadministrator
Systembetreuer	Intendant	Kassenleiter
Abendkasse	Kassenpersonal	Öffentlichkeitsarbeit
Internetkunde	Vorverkaufsstelle	Bühnenverein (Statistik)
Buchhaltung	Abo-Büro	Kontroller



Akteure/Projektbeteiligte des Ticketsystems

Als nächster Schritt folgt die Aufspaltung des Ticketsystems als Ganzes in die Anwendungsfälle, aus denen das Ticketsystem besteht.

Diese Anwendungsfälle werden zunächst grob aus der folgenden Sicht des Bedieners in einer Tabelle notiert.

- ✓ Welchen Vorgang möchte er bearbeiten?
- ✓ Welches Ergebnis erwartet er vom System auf seine Tätigkeit?

Da dies ein recht schwieriger Prozess ist, können Sie auch Zwischenlösungen zulassen, die Sie zu einem späteren Zeitpunkt korrigieren.

Die folgende Tabelle stellt einen möglichen ersten Vorschlag dar. Dabei wurden die angegebenen Anwendungsfälle bereits auf mehrere Bereiche verteilt. Diese Zuordnung wird auch aufgrund der unterschiedlichen Fachkompetenz der Anwender notwendig.

Anwendungsfälle			
Bereich <i>Verwaltung</i>	Bereich <i>Verkauf</i>	Bereich <i>Statistik</i>	Bereich <i>Buchhaltung</i>
Veranstaltung sperren	Platz verkaufen	Kunden verwalten	Ticket drucken
Veranstaltung anlegen	Abo verkaufen	Logo für Ticket erstellen	Rechnung drucken
Veranstaltung freigeben	Karten bestellen	Abo einlösen	Gutschein einlösen
Abo zusammenstellen	Platz reservieren	Rabattgruppen zuordnen	Gutschein verkaufen
Platz sperren	Platz stornieren	Sammelrechnung erstellen	Provision errechnen
Saalplan erstellen	Platzmieten-Abo verkaufen	Rechnung erstellen	Vorverkaufsgebühr festlegen
Saalplan einrichten		Abrechnung der Altersversorgung (für die Angestellten der Spielstätte)	Abgabe Verkehrsbetriebe (Ticketpreis)
Logo zuordnen		Auslastungsstatistik erstellen	Abrechnung der Garderobe (Ticketpreis)
Spielstätte anlegen		Archivieren abgelaufener Veranstaltungen	AVA (Ticketpreis)
Saalplan-Vorlagen erstellen		Kunden betreuen (Serienbrief, Einladung, Jahresheft)	Vorverkaufsgebühr abrechnen
Abos einrichten		statistische Auswertung erstellen	Kontingent abfragen
Rabattstaffel einrichten			Lastschrift erstellen
Kontingentverwaltung			Provision zahlen
Bestplatznummerierung einrichten			Abrechnung der Tantiemen (Ticketpreis)

Mitarbeiterbefragung

Nachdem Sie herausgefunden haben, welche Personen mit diesem System kommunizieren, ist es sinnvoll, mit diesen Personen ein Interview zu führen. Hierbei erfragen Sie die genauen Tätigkeiten dieser Personen und protokollieren die Befragungsergebnisse. Informationen über spezielle oder kritische Fälle können Sie bereits jetzt notieren. Das Hauptaugenmerk verbleibt aber auf der grundlegenden Tätigkeit. Beachten Sie auch die Haltung, die die Befragten dem neuen System gegenüber einnehmen, weil sich dadurch möglicherweise die Verwendbarkeit der Ergebnisse relativiert.

Gesprächsprotokolle

Gesprächsprotokolle, z. B. für Telefonverkaufsgespräche oder für Verkaufsgespräche an der Abendkasse und ähnlich gelagerte Geschäftsvorgänge, sind sinnvoll. Da Sie später den Entwurf für die Oberfläche der Anwendung erstellen werden, müssen Sie die benötigten Möglichkeiten des Systems ebenso wie die zur Verfügung stehende Zeit kennen. Die Brauchbarkeit des Systems könnte sonst vom Auftraggeber infrage gestellt werden.

Telefonservice	Kunde am Telefon
Guten Tag.	
	Hallo, ich möchte für Wagner für morgen Abend Karten bestellen.
Aber wir spielen morgen „Don Carlos“ von Verdi.	
	Ich heiße Wagner.
Pardon, in welcher Platzgruppe möchten Sie die Karten.	
	Welchen Preis haben denn die Karten in den Platzgruppen?
...	
	Gut, dann nehme ich die beiden Plätze in der dritten Reihe.
Möchten Sie die Karten sofort über die Kreditkarte bezahlen?	
	Ja.
Geben Sie bitte Ihre Kartennummer und das Kreditinstitut sowie das Gültigkeitsdatum an.	
	(Kunde antwortet)
Danke, Ihre Karten liegen an der Abendkasse zur Abholung auf Ihren Namen bereit.	
	Danke.

Sie können aus dieser Gesprächsnotiz wichtige Informationen erhalten.

- ✓ Ein Mitarbeiter des Telefonservice kann auf alle Veranstaltungen, auch die der nächsten Tage, zugreifen.
- ✓ Er erhält Informationen über den Titel und z. B. den Komponisten des Stückes.
- ✓ Er kann Kartenbestellungen mit Bezahlung oder Kartenreservierungen aushandeln.

Diese Notizen bieten Ihnen später eine gute Grundlage zur Erstellung der Bedieneroberfläche der Anwendung.

Faktensammlung

Bei der Mitarbeiterbefragung und bei der in diesem Zusammenhang möglicherweise durchgeführten Begehung der Arbeitsplätze können Sie wichtiges Material (z. B. Rechnungen, Quittungen, Eintrittskarten, statistische Auswertungen oder Sammelrechnungslisten) zusammentragen, das Ihnen später Hinweise und Anregungen liefern kann.

Eventuell werden Ihnen Verträge mit Künstlern oder Künstleragenturen als Kopie bereitgestellt. Sammeln Sie alle diese Materialien und archivieren Sie diese mit einer eindeutigen Nummer. Fertigen Sie eine Materialliste an, die die Nummern enthält und eine Kurzbeschreibung des gesammelten Materials, so ersparen Sie sich später das Suchen in unbeschrifteten Materialanhäufungen.

Nummer	Beschreibung
MS1	Rechnung an Großkunden
MS2	Besucherstatistik für statistisches Bundesamt
MS3	Quittung für Eintrittskarte
MS4	Ticket
MS5	Ticket für Gastveranstaltung
MS6	Internet-Ticket
...	...

Glossar

Bei der Analyse der Anwendungsfälle und der Akteure werden Begriffe zum Beschreiben der Rollen und Aktivitäten verwendet. Für die Verständigung und zur Namensgebung werden der einfacheren Handhabung wegen Kurzformen und Kürzel vereinbart und zur Beschreibung verwendet.

Um Missverständnisse über die Bedeutung dieser Kürzel zu vermeiden, wird für jeden Begriff ein Eintrag im Glossar angelegt. So kann im Laufe des Entwicklungsprozesses die Bedeutung eines Kürzels nachgelesen oder berichtigt werden.

Dieses Glossar sollte allen Beteiligten zugänglich sein und mindestens die folgenden Angaben enthalten:

Kurzform	Schreiben Sie in der Kurzform eine Wortgruppe für den Begriff auf. Gibt es mehrere Begriffe für diesen Begriff, können Sie hier alle Synonyme notieren.
Kürzel	Zur Beschriftung von Symbolen ist die Kurzform oft immer noch zu lang. Vereinbaren Sie deshalb ein eindeutiges Kürzel und beschriften Sie mit diesem Kürzel die Symbole. Das Glossar schließt eine Doppelnutzung eines Kürzels aus.
Bedeutung	In dieser Rubrik wird genau festgelegt, was unter dem Begriff, der als Kurzform angegeben ist, zu verstehen ist. Es kann auch eine Abgrenzung gegenüber anderen ähnlichen Begriffen vorgenommen werden. Sie können auch auf andere bereits bestehende Einträge des Glossars verweisen.
Angaben zur Erstellung oder Änderung	Damit die offizielle Einführung des Begriffs in den Wortschatz des bearbeiteten Projektes nachvollzogen werden kann, werden das aktuelle Datum und das Kürzel des Mitarbeiters vermerkt, der den Begriff hinzugefügt hat. Werden Projekte sehr schnell geändert, kann die Angabe der Uhrzeit ebenfalls von Bedeutung sein.

Die folgende Tabelle enthält ein Glossar für das Ticketsystem.

Kurzform	Kürzel	Bedeutung	Wer/Wann
AfA		Altersabgaben für Angestellte sind Abgaben, die je Ticket an den Fiskus zu zahlen sind. Damit wird die Altersversorgung der Theaterangestellten gewährleistet.	JP 17.06.17
Bestplatznummern		Die Bestplatznummerierung stellt eine Rangfolge der Sitzplätze in aufsteigender Reihenfolge dar, die zur automatischen Ticketvergabe verwendet wird.	JP 17.06.17
Garderobe		Der Garderobenpreis wird als Aufwendung im Ticketpreis berücksichtigt und für die Bezahlung des Garderobenpersonals verwendet.	JP 17.06.17
Kunde	K	Ein Kunde kauft oder reserviert ein Ticket oder erwirbt ein Abo.	JP 20.06.17
Platzgruppen-Abo	PGA	Das Platzgruppen-Abo ist ein Satz von Tickets für mehrere Veranstaltungen.	JP 17.06.17
Platzmieten-Abo	PMA	Ein Platzmieten-Abo ist ein Satz von Tickets für einen bestimmten Platz in mehreren Veranstaltungen.	JP 17.06.17
Preisliste		Die Preisliste wird jeder Veranstaltung zugeordnet und enthält den Ticketpreis für jeden Platz in einer Platzgruppe.	JP 17.06.17
Rabattliste		Die Rabattliste enthält die Preisabschläge oder den Festpreis für z. B. Rentner, Studenten, Schüler und Angestellte des Hauses.	JP 17.06.17

Kurzform	Kürzel	Bedeutung	Wer/Wann
Tantieme		Ein aufgeführtes Stück bleibt Eigentum des Autors bzw. Komponisten. An diese(n) Urheber wird eine Nutzungsgebühr (Tantieme) bezahlt. Diese Nutzungsgebühr wird pro Eintrittskarte berechnet. Meistens wird außerdem eine minimale und eine maximale Summe vereinbart. Die eine, um ein schlecht besuchtes Stück abzusichern, und die andere, um dem Theater bei einem gut besuchten Stück einen Gewinnzuwachs zu sichern.	JP 17.06.17
Ticketsockelpreis		Der Ticketsockelpreis ist der minimale Preis für ein Ticket, der sich aus der Summe der vertraglich vereinbarten Abgaben und Kosten ergibt.	JP 17.06.17
Verkäufer	VK	Dies ist eine Person, die mit dem Ticketsystem an der Abendkasse oder in einer Vorverkaufsstelle Tickets verkauft.	
Verkehrsbetriebe		Ein Ticket kann als Fahrkarte für die Verkehrsbetriebe oder als Parkkarte verwendet werden.	JP 17.06.17
Vorverkaufsgebühr	VVG	Der Besucher zahlt einen Preisaufschlag und erhält dafür sein Ticket bereits vor dem Abendverkauf. Geschieht der Verkauf über die Vorverkaufsstelle, kann ein Ticket auch in großer Entfernung zur Abendkasse gekauft werden.	JP 17.06.17
Vorverkaufsstelle	VVK	Eine Vorverkaufsstelle ist eine Ticketkasse, die alle Verkaufsmöglichkeiten hat und örtlich vom Betreiber getrennt ist. Die Datenkommunikation erfolgt über das Internet.	JP 17.06.17
Wahl-Abo	WA	Das Abo enthält Gutscheine für Tickets mehrerer Veranstaltungen.	JP 17.06.17
zugelassene Verkaufsstelle		Eine zugelassene Verkaufsstelle wurde vom Systembetreuer für den Verkauf von Eintrittskarten freigeschaltet. Die Freischaltung kann separat für jede Veranstaltung erfolgen.	JP 02.11.17

Aufgrund der bereits vorgenommenen Gruppenzuordnung der Anwendungsfälle kann das Ticketsystem schon jetzt in vier Teilbereiche unterteilt werden.

Das Anwendungsfalldiagramm veranschaulicht diese Zerlegung. Weniger komplexe Detaillierungen sind in dieser frühen Phase noch nicht notwendig, da der Entwurf durch Verfeinerung in einer späteren Version diese Details einarbeitet.

Anwendungsfalldiagramm

In einem ersten Schritt trennen Sie die Geschäftsanwendungsfälle von den Systemanwendungsfällen. Ein Systemanwendungsfall liegt vor, wenn der Anwendungsfall von der entwickelten Anwendung bearbeitet wird.

Trifft dieses für einen Anwendungsfall nicht zu und wird dieser dennoch für die Bewältigung der Gesamtaufgabe (z. B. zur Integration der neuen Software in das vorhandene System) benötigt, handelt es sich um einen Geschäftsanwendungsfall. Dieser wird im Geschäftsmodell modelliert und nicht in einem Anwendungsfalldiagramm des Softwaresystems.

Eine schrittweise Verfeinerung dieses Diagramms ist ebenfalls möglich, sodass Sie die notwendigen Details nach und nach einarbeiten können.

Die zusammengestellten Anwendungsfälle und Akteure können Sie in einem Anwendungsfall darstellen. Dazu sollten Sie die Anwendungsfälle zuvor auf die Einhaltung der nachfolgenden Kriterien kontrollieren.

Kriterium

Ein Anwendungsfall beschreibt in natürlicher Sprache eine konsistente und zielgerichtete Interaktion eines Benutzers mit einem System mit folgenden Eigenschaften:

- ✓ Auslöser des Anwendungsfalls ist ein geschäftlicher Anlass.
- ✓ Am Ende des Anwendungsfalls liegt ein definiertes Ergebnis von geschäftlichem Wert vor.
- ✓ Ein Anwendungsfall beschreibt das gewünschte externe Systemverhalten aus der Sicht des Anwenders.



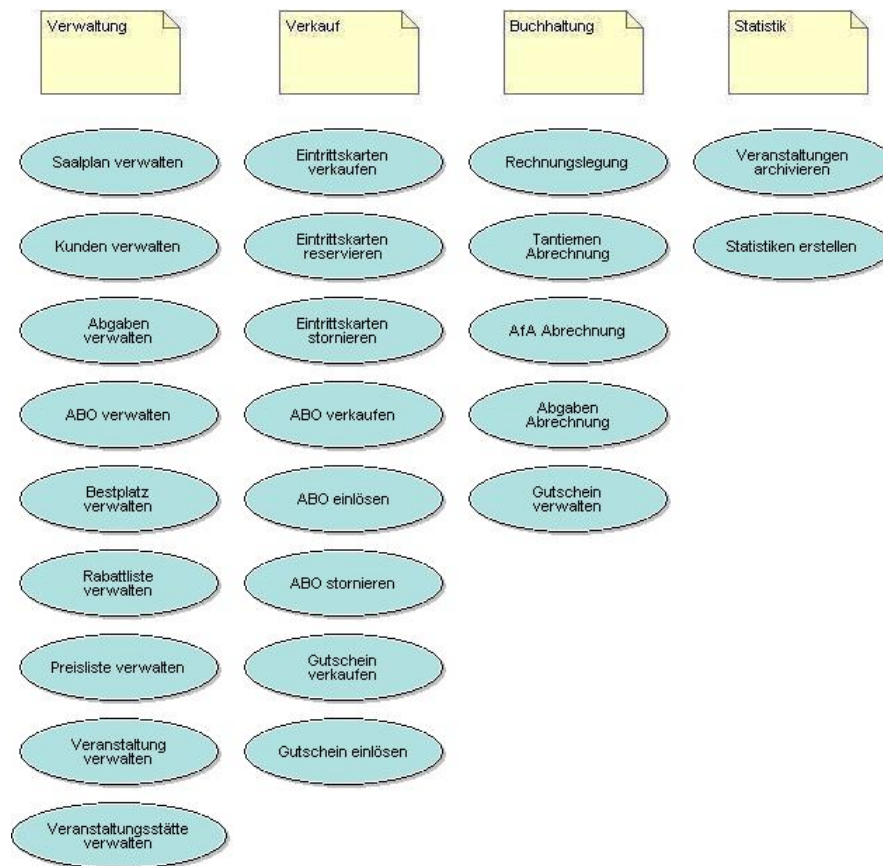
Der Anwendungsfall beschreibt die Leistung des Systems, nicht aber den Weg, wie dieses Ergebnis erreicht wird. Anwendungsfälle stellen auch keine funktionale Zerlegung dar. Ein Anwendungsfall beschreibt keine einzelnen Schritte zur Erlangung des Ergebnisses.

Natürlichsprachliche Notation

In einer Anwendungsfallbeschreibung können Sie die Merkmale eines Anwendungsfalls notieren. Geben Sie für jeden Anwendungsfall den Namen und eine Beschreibung an, die den genauen Auslöser und das zu erwartende Ergebnis des Anwendungsfalls enthält.

Erste Version des Diagramms

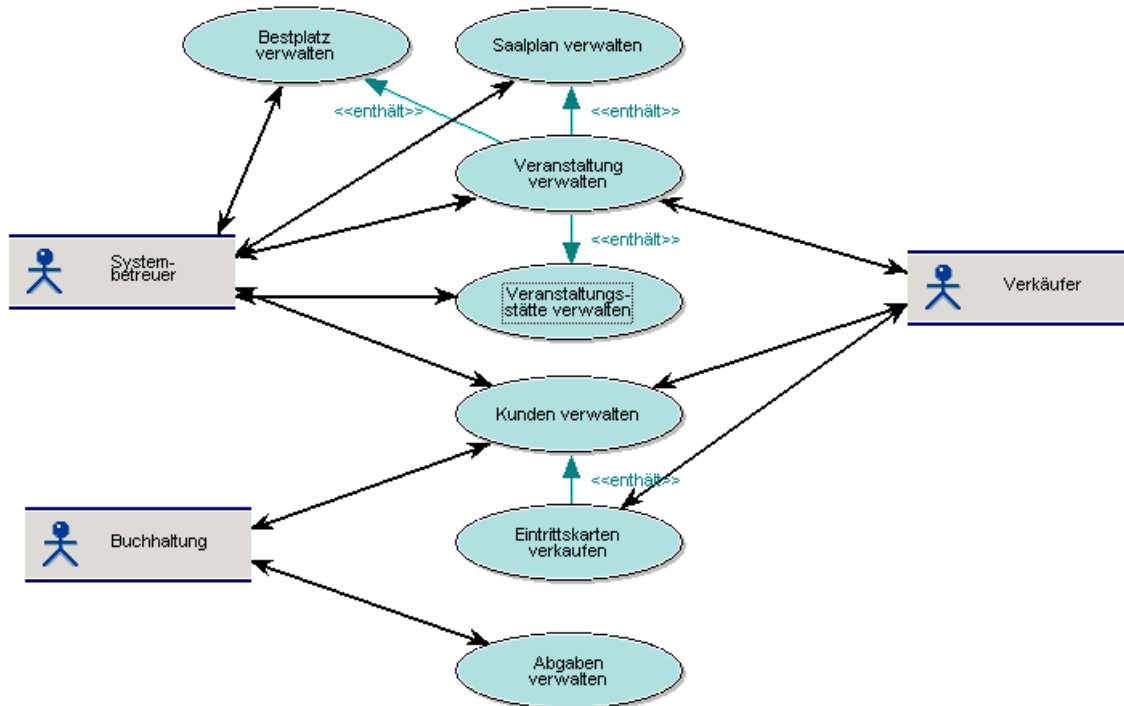
Nach der Beschreibung der Anwendungsfälle können Sie diese in einem Anwendungsfalldiagramm grafisch darstellen. Dazu werden in Ellipsen die Anwendungsfälle gezeichnet. Über Kommunikationslinien werden sie mit den zugehörigen Akteuren oder untereinander verbunden.



Erste grobe Form eines Anwendungsfalldiagramms

Zur Beurteilung der Qualität der gefundenen Anwendungsfälle und deren Bezeichnung können Sie die folgende Checkliste anwenden:

- ✓ Trägt der Anwendungsfall eine sinnvolle Bezeichnung?
- ✓ Besteht die Bezeichnung aus Substantiv und Verb, z. B. Ticket verkaufen, oder aus einem Prozesswort, z. B. Ticketverkauf? Wenn nicht, ist zu untersuchen, ob wirklich ein Anwendungsfall vorliegt.
- ✓ Handelt es sich bei dem Anwendungsfall wirklich um einen selbstständigen Anwendungsfall oder kann dieser von einem zweiten bereits erstellten abgeleitet werden?
- ✓ Sind Auslöser und Ergebnis eindeutig zu benennen?
- ✓ Ist der Anwendungsfall für den zu unterstützenden Prozess relevant – liegt also ein Systemanwendungsfall vor?
- ✓ Wird der Anwendungsfall eventuell für das Geschäft benötigt, aber vom zu erstellenden System nicht bearbeitet (Geschäftsanwendungsfall)?
- ✓ Existiert der Anwendungsfall eventuell mehrfach unter verschiedenen Bezeichnungen?
- ✓ Sind die Bezeichnungen aller Anwendungsfälle sprachlich normiert?



Verfeinerung des Anwendungsfalldiagramms (Auszug)

Ein Anwendungsfalldiagramm ist keine Ablaufbeschreibung. Es wird keine Ablaufreihenfolge dargestellt. Für die Ablaufbeschreibung werden z. B. Aktivitätsdiagramme (vgl. Abschnitt 8.6) eingesetzt.

8.3 Anwendungsfälle in Pakete aufteilen

Nachdem die gefundenen Anwendungsfälle in einem ersten Anwendungsfalldiagramm dargestellt wurden, kann die Komplexität abgeschätzt und eine Separierung vorgenommen werden. Für das Ticketsystem wurden die gefundenen Anwendungsfälle in vier Gruppen angeordnet. Da die Schrittfolge der weiteren Bearbeitung für alle vier Gruppen in der gleichen Weise erfolgt, wird die weitere Erarbeitung am Beispiel der Gruppe Verwaltung durchgeführt. Dieser Gruppe wurden neun Anwendungsfälle zugeordnet, die in weiteren Schritten untersucht und gegebenenfalls weiter unterteilt werden.

Einteilung in Pakete

Da die neun gefundenen Anwendungsfälle aus weiteren Anwendungsfällen zusammengesetzt sind, werden sie zur Grundlage der Zerlegung in Pakete herangezogen. Für die Gruppe *Verwaltung* erhalten Sie so neun Pakete.

Für jedes dieser einzelnen Pakete werden im weiteren Entwicklungsablauf die enthaltenen Anwendungsfälle identifiziert und verbal beschrieben; diese enthaltenen Anwendungsfälle dürfen dann aber keine weiteren Anwendungsfälle enthalten, weil sie sonst ihrerseits in einem weiteren Schritt zerlegt werden müssten. Alle in diesen Anwendungsfällen enthaltenen Aktivitäten sind Schritte, die für den Ablauf allein keinen selbstständigen Anwendungsfall ergeben.

Paket: Saalplan verwalten

Dieses Paket enthält folgende Anwendungsfälle:

- ✓ Saalplan anlegen
- ✓ Sitzplatz bearbeiten
- ✓ Saalplan löschen

Dazu wird für jeden Anwendungsfall eine Anwendungsfallbeschreibung erstellt. Die Anwendungsfallbeschreibung enthält die folgenden Aussagen:

- ✓ Name des Anwendungsfalls
- ✓ Auslöser des Anwendungsfalls
- ✓ Ergebnis des Anwendungsfalls

Name	Saalplan anlegen
Auslöser	Ein Systembetreuer möchte für eine Veranstaltung den Saalplan anlegen.
Ergebnis	Einer Veranstaltung wurde ein Saalplan zugeordnet, aus dem heraus verkauft werden kann. Jeder Platz hat eine Bestplatznummer und eine Platzgruppennummer.
Beschreibung	Der Systembetreuer sucht aus einer Liste der Saalpläne den passenden Saalplan heraus oder erstellt einen neuen. Er weist diesen Saalplan einer Veranstaltung zu. Er legt die Rabattstaffel fest. Er sperrt die Plätze, die nicht verkauft werden dürfen. Er weist die Preistabelle zu.
Name	Sitzplatz bearbeiten
Auslöser	Der Systembetreuer oder Verkäufer möchte Attribute eines Sitzplatzes ändern.
Ergebnis	Ein Attribut eines Sitzplatzes im Saalplan ist verändert.
Beschreibung	Die Sperrung, die Reservierung und der Verkauf eines Platzes des Saalplanes werden durch Ändern von dafür vorgesehenen Attributen gekennzeichnet.
Name	Saalplan löschen
Auslöser	Der Systembetreuer möchte den Saalplan für eine Veranstaltung löschen.
Ergebnis	Der mit einer Veranstaltung verbundene Saalplan wird gelöscht.
Beschreibung	Der Systembetreuer trennt die Verbindung zwischen einer Veranstaltung und dem Saalplan. Anschließend wird der Saalplan gelöscht.

Paket: *Kunden verwalten*

Dieses Paket enthält die folgenden Anwendungsfälle:

- ✓ Kunde einrichten ✓ Kundendaten ändern
- ✓ Kunde löschen ✓ Kunden-Historie erfassen

Name	Kunde einrichten
Auslöser	Ein Verkäufer möchte einen Kunden in den Kundenstamm aufnehmen.
Ergebnis	Der Kunde wurde dem Kundenstamm hinzugefügt.
Beschreibung	Der Verkäufer nimmt die Kundendaten entgegen und pflegt den neuen Datensatz in die Kundendatei ein.
Name	Kunde löschen
Auslöser	Der Systembetreuer möchte die Daten eines Kunden löschen.
Ergebnis	Die Kundendaten wurden aus dem System gelöscht.
Beschreibung	Der Systembetreuer sucht den Kunden in der Kundendatei und löscht den Datensatz.
Name	Kundendaten ändern
Auslöser	Der Verkäufer möchte die Daten eines Kunden ändern.
Ergebnis	Die Kundendaten wurden aktualisiert.
Beschreibung	Der Verkäufer nimmt die neuen Kundendaten entgegen. Er sucht den Datensatz im Kundenstamm. Er ändert die abweichenden Daten und aktualisiert die Kundendatei.
Name	Kunden-Historie erfassen
Auslöser	Ein Kunde, der im Kundenstamm erfasst ist, tätigt einen Ticketkauf.
Ergebnis	Der Kaufvorgang wird mit Datum, Ort und Art des Geschäftsvorganges erfasst und in einer Kunden-Historie gespeichert. Es wird der Preis ohne und mit Rabatt gespeichert.
Beschreibung	Ein Kunde kauft an der Kasse, im Abo-Büro oder in einer Vorverkaufsstelle ein Ticket. Der Kunde wird aufgrund seiner Daten im Kundenstamm gesucht. Wenn er gefunden wurde, wird ein Eintrag in der Historie angelegt. Er enthält die Kundennummer, den Vorgang, den Ort und das Datum des Geschäftsvorganges. Erhält der Kaufvorgang einen Rabatt der Rabattstaffel, wird der Preis ohne und mit Rabatt gespeichert.

Paket: Abgaben verwalten

- ✓ Abgaben einrichten
- ✓ Abgaben ändern
- ✓ Ticketsockelpreis bestimmen

Name	Abgaben einrichten
Auslöser	Der Systembetreuer möchte eine Abgabenliste für eine Veranstaltung anlegen.
Ergebnis	Eine Abgabenliste wurde für eine Veranstaltung angelegt, die alle Abgaben, die vom Ticketpreis abgedeckt werden müssen, enthält.
Name	Abgaben einrichten
Beschreibung	Der Systembetreuer ermittelt alle notwendigen Abgaben (z. B. AfA, Garderobe, Tantieme, Verkehrsbetriebe, Vorverkaufsgebühr). Er fügt jede Position in eine Tabelle ein. Die Summe der Abgaben dient zur Berechnung eines absoluten Teils des Ticketpreises.
Name	Abgaben ändern
Auslöser	Der Systembetreuer möchte, z. B. aufgrund von Vertragsänderungen, vor der Freigabe der Veranstaltung die Abgabentabelle den neuen Bedingungen anpassen.
Ergebnis	Die Abgabentabelle wurde den veränderten Vertragsbedingungen angepasst.
Beschreibung	Der Systembetreuer erstellt eine neue Abgabentabelle.
Name	Ticketsockelpreis bestimmen
Auslöser	Der Ticketpreis wird vom Verkäufer abgefragt.
Ergebnis	Ein Ticketpreis wurde berechnet.
Beschreibung	Der Verkäufer möchte den Ticketpreis erfahren. Das System berechnet den Ticketpreis aufgrund eines festgelegten Betrages, der Abgabenliste und der Rabattstaffel. Dabei wird ein anvisierter Vorgabepreis verwendet, wenn der errechnete Preis unter diesem Vorgabepreis liegt. Zusätzlich wird die Rabattstaffel berücksichtigt.

Paket: Abo verwalten

- ✓ Abo erstellen
- ✓ Abo im Saalplan buchen
- ✓ Abo-Buchung stornieren

Name	Abo erstellen
Auslöser	Der Systembetreuer möchte ein Abo anlegen.
Ergebnis	Einem neu angelegten Abo sind die Veranstaltungen und die Anzahl der Tickets zugeordnet, die mit diesem Abo verkauft werden können. Für dieses Abo wurden die Art (Platzmieten-Abo, Platzgruppen-Abo) und der Preis festgelegt.

Name	Abo erstellen
Beschreibung	Ein neues Abo wird angelegt. Es erhält einen Namen. Dem Abo werden die Veranstaltungen zugeordnet, aus denen die Tickets für das Abo reserviert werden.
Anmerkung	Es gibt mehrere Abo-Arten. Das Wahl-Abo nimmt eine Sonderstellung ein, da es eigentlich ein Gutschein (Geldwert) ist und somit nicht wie ein Abo verwaltet werden muss.
Name	Abo im Saalplan buchen
Auslöser	Der Verkäufer möchte ein Abo verkaufen.
Ergebnis	Ein Abo wird im Saalplan gebucht oder in den Saalplänen bei Platzmieten-Abo.
Beschreibung	Der Verkäufer erfragt die Art des Abos. Bei einem Platzmieten-Abo wird in allen Veranstaltungen der gleiche Platz gebucht. Bei einem Platzgruppen-Abo wird für jede Veranstaltung des Abos ein Platz in der gleichen Platzgruppe gebucht.
Name	Abo-Buchung stornieren
Auslöser	Ein Kunde möchte das Abo stornieren.
Ergebnis	Die gebuchten Plätze werden in den Veranstaltungen des Abos wieder freigegeben.
Beschreibung	Wenn der Verkäufer das Abo storniert, werden die Veranstaltungen ermittelt, aus denen Plätze im Rahmen des zu stornierenden Abos gebucht wurden. Diese Plätze werden wieder freigegeben.

Paket: *Bestplatz verwalten*

- ✓ Bestplatznummern zuweisen (Algorithmus festlegen)
- ✓ Bestplatznummern löschen

Name	Bestplatznummern zuweisen
Auslöser	Der Systembetreuer möchte für einen Saalplan eine Bestplatzbelegung zuweisen.
Ergebnis	Die Plätze eines Saalplanes einer Veranstaltung sind mit Nummern versehen, die der Reihenfolge für eine Bestplatzauswahl genügen.
Beschreibung	Über vorgegebene Algorithmen (z. B. Zufallsprinzip) oder über Vorgaben von bereits bewährten Bestplatzkonstellationen wird jedem Platz einer Veranstaltung eine Bestplatznummer zugewiesen.
Name	Bestplatznummern löschen
Auslöser	Der Systembetreuer möchte für einen Saalplan eine Bestplatzbelegung löschen.
Ergebnis	Allen Plätzen eines Saalplanes einer Veranstaltung wurde die Bestplatznummer 0 zugewiesen.

Name	Bestplatznummern löschen
Beschreibung	Jedem nicht gesperrten Platz der Veranstaltung wird die Bestplatznummer 0 zugewiesen. Damit wird die Bestplatzsuche über das Attribut <code>Bestplatzsuche</code> des Saalplanes ausgeschaltet, indem diesem der Wert <code>FALSE</code> zugewiesen wird.

Paket: *Preisliste verwalten*

- ✓ Anlegen einer Preisliste
- ✓ Ändern einer Preisliste

Name	Anlegen einer Preisliste
Auslöser	Der Systembetreuer möchte eine Preisliste auf der Grundlage der Platzgruppen für eine Veranstaltung anlegen.
Ergebnis	Einer Veranstaltung wurde eine Preisliste zugewiesen.
Beschreibung	Eine Preisliste wird angelegt. Sie enthält für jede Platzgruppennummer den Verkaufspreis.
Name	Ändern einer Preisliste
Auslöser	Der Systembetreuer möchte, z. B. aufgrund von Vertragsänderungen, vor der Freigabe der Veranstaltung die Preisliste den neuen Bedingungen anpassen.
Ergebnis	Den Platzgruppennummern wurden die geänderten Ticketpreise zugeordnet.
Beschreibung	Der Systembetreuer sucht die Veranstaltung und ändert den Ticketpreis für die Platzgruppe in der zugewiesenen Preisliste.

Paket: *Rabattliste verwalten*

- ✓ Anlegen einer Rabattliste
- ✓ Ändern einer Rabattliste

Name	Anlegen einer Rabattliste
Auslöser	Der Systembetreuer möchte eine Rabattliste für eine Veranstaltung anlegen.
Ergebnis	Einer Veranstaltung wurde eine Rabattliste zugewiesen.
Beschreibung	Eine Rabattliste wird angelegt. Ihr werden eine Nummer für eine Rabattklasse und entweder ein Prozentsatz oder/und ein Festpreis zugeordnet.
Name	Ändern einer Rabattliste
Auslöser	Der Systembetreuer möchte einen Eintrag in der Rabattliste ändern.
Ergebnis	Der Prozentsatz oder der Festpreis eines Eintrages der Rabattliste wurde geändert.
Beschreibung	Der Systembetreuer sucht aus der Rabattliste den Eintrag heraus, der geändert werden soll. Er überschreibt den zugewiesenen Festpreis oder den Prozentsatz mit dem neuen Wert.

Paket: *Veranstaltung verwalten*

- ✓ Veranstaltung anlegen
- ✓ Veranstaltung ändern
- ✓ Veranstaltung stornieren

Name	Veranstaltung anlegen
Auslöser	Der Systembetreuer möchte eine Veranstaltung anlegen.
Ergebnis	Eine neue Veranstaltung wurde angelegt.
Beschreibung	Es werden der Tag und die Uhrzeit der Aufführung zugewiesen. Der Veranstaltung wird eine Veranstaltungsstätte und ein Saalplan zugeordnet. Weiterhin wird eine Preisliste und eine Rabattliste für die Veranstaltung erstellt und zugewiesen.
Name	Veranstaltung ändern
Auslöser	Der Systembetreuer möchte, z. B. aufgrund von Vertragsänderungen, vor der Freigabe der Veranstaltung den Termin oder den Veranstaltungsort für die Aufführung ändern.
Ergebnis	Für die Veranstaltung wurde der Termin oder die Veranstaltungsstätte geändert.
Beschreibung	Der Systembetreuer sucht die Veranstaltung und ändert die Uhrzeit und das Datum oder weist eine neue Spielstätte zu.
Name	Veranstaltung stornieren
Auslöser	Der Veranstalter sagt eine Veranstaltung ab. Der Systembetreuer storniert die Veranstaltung.
Ergebnis	Eine Veranstaltung wurde gelöscht.
Beschreibung	Der Systembetreuer sucht die Veranstaltung im System. Das System ermittelt die Abhängigkeiten aufgrund von verkauften Abos zu anderen Veranstaltungen. Die betroffenen Abos werden in einer Konfliktliste gespeichert.

Paket: *Veranstaltungsstätte verwalten*

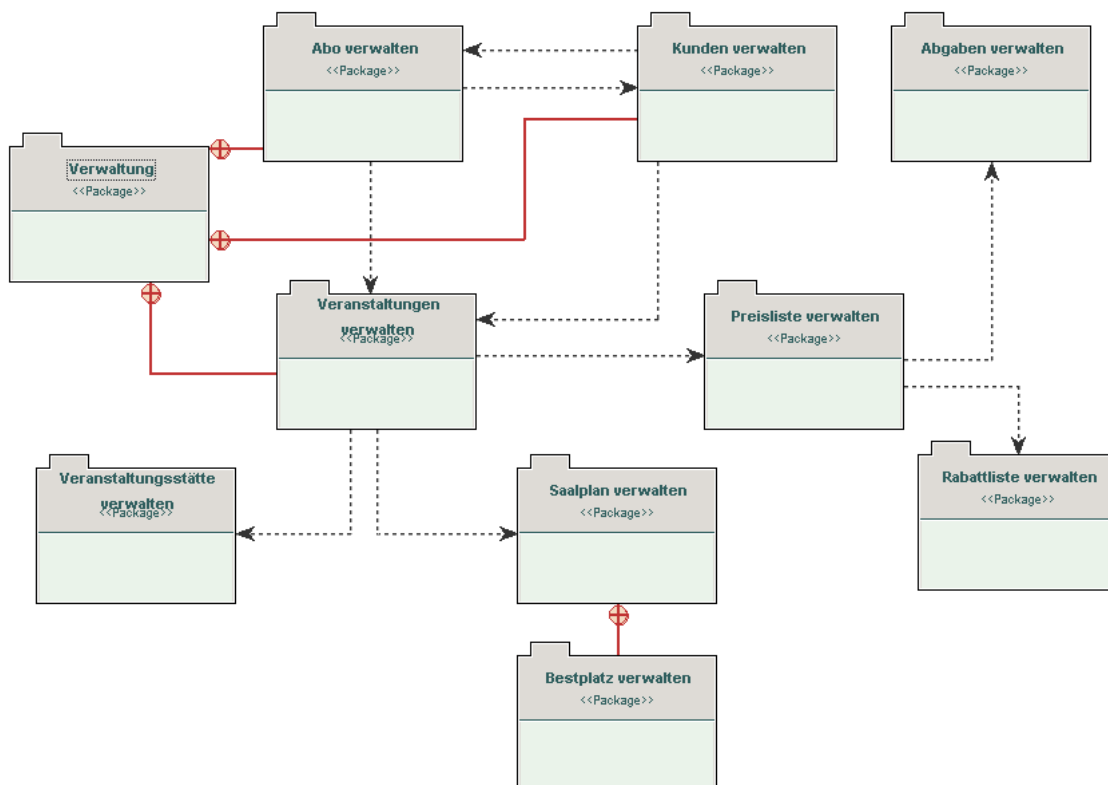
- ✓ Veranstaltungsstätte anlegen
- ✓ Veranstaltungsstätte ändern

Name	Veranstaltungsstätte anlegen
Auslöser	Der Systembetreuer möchte eine Veranstaltungsstätte anlegen.
Ergebnis	Eine neue Spielstätte wurde aufgenommen.
Beschreibung	Der Systembetreuer nimmt die Daten einer neuen Spielstätte auf und legt diese im System an.

Name	Veranstaltungsstätte ändern
Auslöser	Der Systembetreuer möchte die Daten einer Veranstaltungsstätte ändern.
Ergebnis	Die Daten einer Veranstaltungsstätte wurden geändert.
Beschreibung	Die neuen Daten einer Veranstaltungsstätte werden entgegengenommen und die betroffene Veranstaltungsstätte wird im System gesucht. Anschließend werden die Daten der Spielstätte geändert und die Änderungen gespeichert.

8.4 Paketdiagramme

Paketdiagramme veranschaulichen das Zusammenspiel der Pakete. Dabei wird sichtbar, welches Paket Ergebnisse eines anderen benötigt oder welches Paket unterstützende Aktionen ausführt.



Zusammenspiel unterschiedlicher Pakete

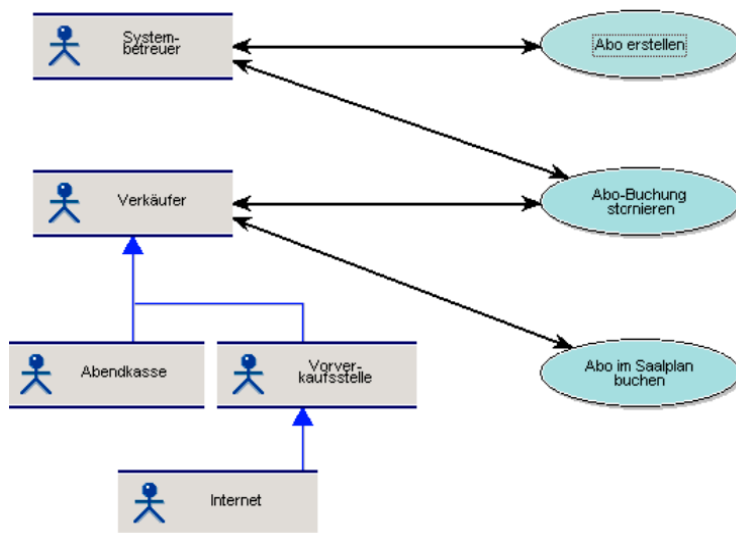
Im Diagramm wird mit der Symbolik z. B. angezeigt, dass das Paket *Verwaltung* die Pakete *Kunden verwalten*, *Abo verwalten* und *Veranstaltungen verwalten* enthält. Das Paket *Abo verwalten* kommuniziert mit dem Paket *Veranstaltungen verwalten*, in dem die Veranstaltungen für die Tickets des Abos enthalten sind.

8.5 Anwendungsfalldiagramme

Erstellen Sie für jedes Paket ein Anwendungsfalldiagramm, aus dem ersichtlich ist, welche Akteure mit den Anwendungsfällen kommunizieren und in welchen Rollen sie das tun. Sie können auch die Fremdsysteme als Akteure einbeziehen. Fremdsysteme können u. a. Anwendungsfälle aus anderen Paketen sein.

Auf diese Weise lässt sich das Modell, das im Paketdiagramm dargestellt wurde, verfeinern. In der Abbildung ist das Anwendungsfalldiagramm für das Paket *Abo verwalten* dargestellt. Da in diesem Paket die Anwendungsfälle nicht voneinander abhängig sind, werden keine Abhängigkeiten modelliert.

Über die Generalisierungsbeziehung wird die Vererbungshierarchie dargestellt. Für die Rolle *Verkäufer* wird ein Hierarchiebaum erstellt. Von dieser Rolle werden die konkreten Verkäufer, z. B. die Mitarbeiter der Abendkasse oder der Vorverkaufsstellen, abgeleitet.



Hierarchiebaum für die Rolle „Verkäufer“

8.6 Aktivitätsdiagramme

Für jeden Anwendungsfall können Sie in einem Aktivitätsdiagramm die internen Abläufe modellieren. Für die Erstellung eines Aktivitätsdiagramms benötigen Sie die Aktivitäten, die bei der Abarbeitung des Anwendungsfalls durchlaufen werden. Bei der Suche nach diesen Aktivitäten kann Ihnen das Erarbeiten der essenziellen Anwendungsfallbeschreibungen dienen, bei denen aus den Anwendungsfallbeschreibungen stichpunktartig die abzuarbeitenden Schritte herausgearbeitet wurden.

Essenzielle Anwendungsfallbeschreibung

Es werden keine neuen Anwendungsfälle gesucht, sondern die bereits in den Anwendungsfallbeschreibungen aufgeführten weiter spezifiziert. Diese werden durch eine stichwortartige Formulierung der Aktivitäten für die Modellierung in der Design-Phase vorbereitet.

Eine essenzielle Beschreibung des Anwendungsfalls enthält folgende Angaben:

Name des Anwendungsfalls	Unter diesem Punkt wird der Name des Anwendungsfalls notiert. Diesem Namen wird das Stereotyp <code>essential</code> nachgesetzt. Diese Ergänzung kennzeichnet die Anwendungsfallbeschreibung als essenzielle Beschreibung.
Kurzbeschreibung	Die Aufgabe des Anwendungsfalls für das System wird in einer kurz gefassten Beschreibung notiert.
Akteure	Alle Personen, Systeme oder Zeitereignisse, die mit dem Anwendungsfall kommunizieren, werden aufgezählt.
Auslöser	Jeder Anwendungsfall wird durch ein Ereignis ausgelöst. Dieses Ereignis wird für den Anwendungsfall unter dem Punkt „Auslöser“ notiert.
Vorbedingung	Ist für die Ausführung des Anwendungsfalls eine bestimmte Konstellation notwendig, wird diese unter den Vorbedingungen notiert.
Eingehende Informationen	Diese Informationen werden notwendigerweise dem Anwendungsfall übergeben.
Ergebnisse	Die geschäftlich relevanten Ergebnisse werden nach der Ausführung des Anwendungsfalls genannt. Sie geben Aufschluss über die Bedeutung des Anwendungsfalls.
Nachbedingungen	In den Nachbedingungen werden Aktionen notiert, die nach der Abarbeitung des Anwendungsfalls ausgeführt werden, oder Zustände, die nach der Abarbeitung erreicht werden.
Ablauf	In diesem Punkt werden die einzelnen Aktivitäten des Anwendungsfalls ermittelt und in der Reihenfolge der Ausführung notiert. Zur Kennzeichnung der Reihenfolge werden die Aktivitäten nummeriert.

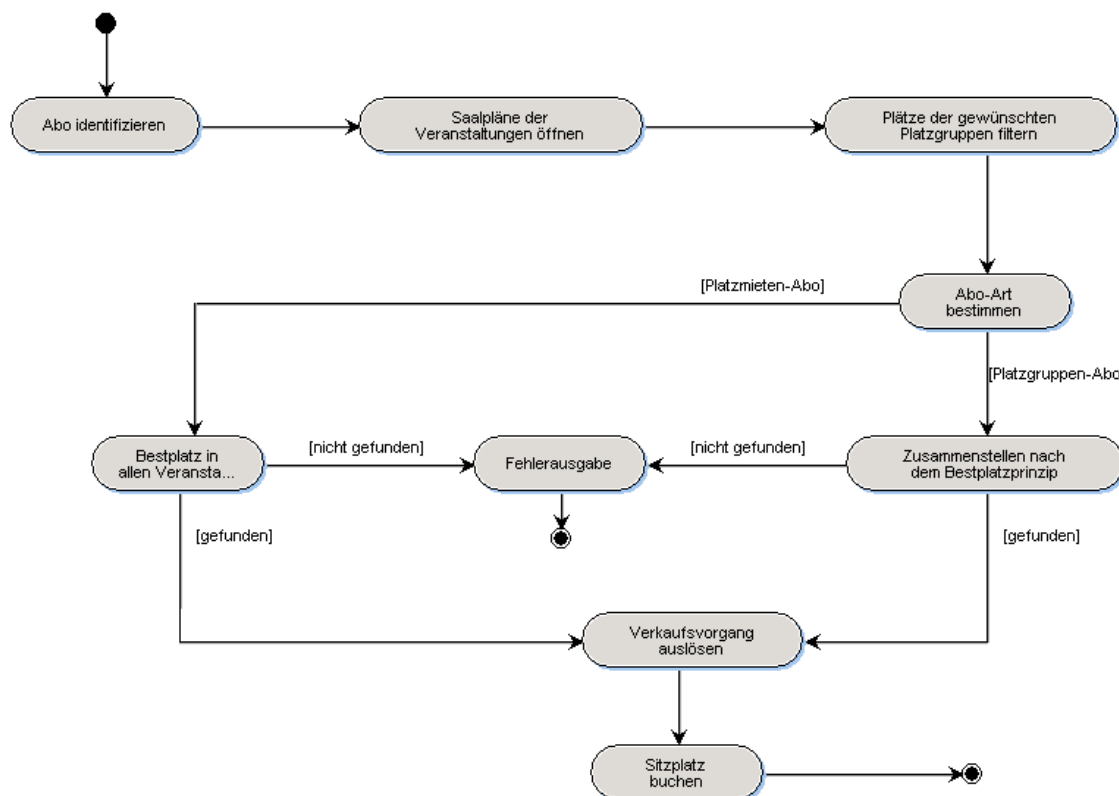
Beispiel

Die folgende Tabelle enthält die essenzielle Beschreibung für den Anwendungsfall *Abo im Saalplan buchen*. Der Ablauf wird in 8 Schritten dargestellt, in denen die Aktivitäten in der Reihenfolge des Ablaufs notiert sind.

Name	Abo im Saalplan buchen «essential»
Kurzbeschreibung	Ein Abo wird gebucht.
Akteure	Verkäufer
Auslöser	Kunde kauft ein Abo
Vorbedingung	Der Abo-Typ wurde vom Systembetreuer angelegt.
Eingehende Informationen	Abo-Typ: Platzmieten-Abo, Platzgruppen-Abo
Ergebnisse	Das Abo wurde in den Saalplänen der Veranstaltungen gebucht, d. h., für jede Veranstaltung des Abos wurde ein Sitzplatz gebucht.

Nachbedingungen	Dem Kunden wurde ein Abo verkauft. Die gebuchten Plätze können nicht mehr verkauft werden. Die gebuchten Plätze sind mit dem Kundennamen verbunden. Der Verkaufsvorgang muss registriert werden.
Ablauf	1. Abo identifizieren 2. Saalpläne der Veranstaltungen öffnen 3. Plätze der gewünschten Platzgruppe filtern 4. Abo-Art bestimmen 5. Zusammenstellen nach dem Bestplatzprinzip 6. Bestplatz suchen, der in allen Veranstaltungen verfügbar ist 7. Verkaufsvorgang auslösen 8. Sitzplatz buchen

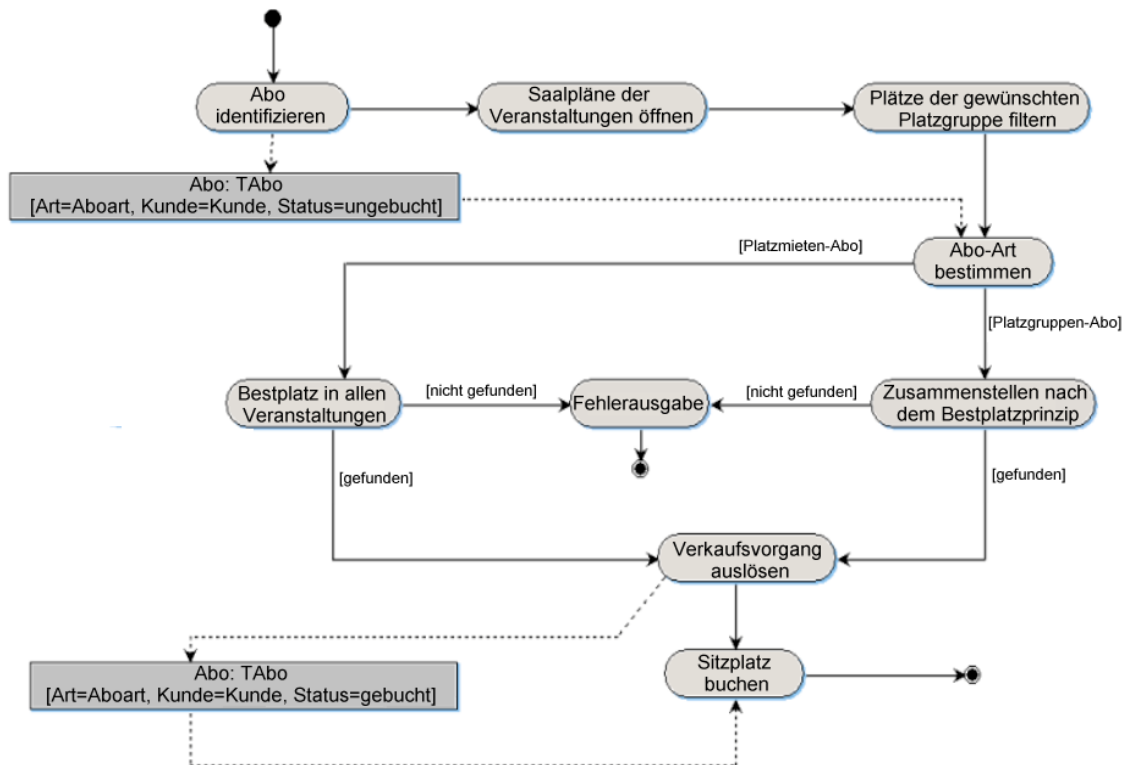
Unter Einbeziehung dieser Anwendungsfallbeschreibung können Sie ein Aktivitätsdiagramm für den Anwendungsfall erstellen. In diesem Diagramm werden die Aktivitäten mit den Bedingungen und Ergebnissen modelliert. Sie erreichen eine weitere Verfeinerung des Modells.



Aktivitätsdiagramm für den Anwendungsfall „Abo im Saalplan buchen“

8.7 Aktivitätsdiagramme mit modellierten Objektzuständen

Nachdem Sie ein Aktivitätsdiagramm erstellt haben, ist es von Bedeutung, welche Objekte während des Anwendungsfalls verändert werden. Sie können die Übergänge von einem Zustand in einen weiteren Zustand modellieren und dadurch den Objektfluss darstellen. Diese Ergänzung des Aktivitätsdiagramms bewirkt eine weitere Vervollständigung des Modells des Anwendungsfalls, von dem das Aktivitätsdiagramm erstellt wurde.



Ergänzt Aktivitätsdiagramm des Anwendungsfalls „Abo im Saalplan buchen“

8.8 Klassendiagramme

In Klassendiagrammen können Sie die Klassenstruktur für Ihre Anwendung modellieren. Dabei gehen Sie bereits den Schritt von der Analyse zum Entwurf. Betrachten Sie die erstellten Anwendungsfälle jetzt aus der Sicht der Abbildung auf Klassen, Objekte und deren Beziehungen untereinander.

Klassen identifizieren

Im ersten Schritt suchen Sie Klassen, mit denen Sie die Daten der Anwendung abbilden können. Jede Klasse ist genau für eine Aufgabe zuständig.

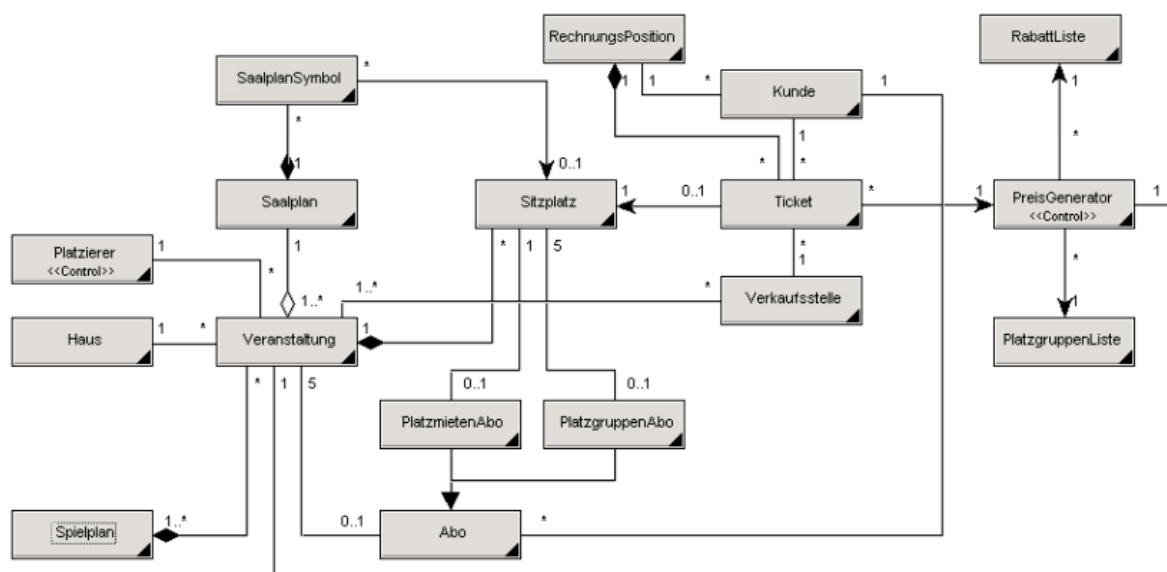
Beispielsweise können Sie für den Saalplan eine Klasse vorsehen. Eine weitere Klasse können Sie für Abos bereitstellen.

Bei der grafischen Darstellung der Klassen im Klassendiagramm werden Sie feststellen, dass diese Klassen noch nicht ausreichen. Erstellen Sie weitere Klassen und modellieren Sie die Beziehungen der Klassen untereinander. Haben Sie eine Klasse erstellt, die mehrere Aufgaben übernehmen muss, so verteilen Sie diese auf mehrere Klassen.

Arbeiten Sie nach folgenden Schritten, um ein Klassendiagramm zu erstellen und dieses schrittweise zu vervollständigen:

- ▶ zeichnen Sie die Klassen, die Daten der Anwendung speichern (Entity-Klassen), in ein Klassendiagramm ein.
- ▶ Zeichnen Sie die Beziehungen zwischen den Klassen in das Klassendiagramm ein.
- ▶ Erweitern Sie das Klassendiagramm mit Klassen, die nur Funktionalität bereitstellen (Controllerklassen).
- ▶ Erstellen Sie für jede Klasse jeweils eine Notiz zur Definition der Verantwortlichkeiten und der Anforderungen.
- ▶ Überarbeiten Sie das Klassendiagramm so lange, bis eine geeignete Struktur gefunden ist.
- ▶ Untersuchen Sie die Assoziationen und bestimmen Sie, ob Aggregationen oder Kompositionen unter ihnen zu finden sind.
- ▶ Zeichnen Sie gerichtete Assoziationen ein, wo immer das möglich ist. Sie entflechten damit die Struktur.
- ▶ Überprüfen Sie, ob alle eingezeichneten Assoziationen zur Umsetzung der Aufgabe notwendig sind.

Verwerfen Sie gegebenenfalls Teile des Diagramms oder bereits modellierte Beziehungen, wenn Sie diese für die Aufgabe geeigneter strukturieren können. Mit jeder Verfeinerung vervollständigen Sie das Modell und können so die erstellte Datenstruktur besser an die zu verarbeitenden Daten anpassen.



Klassendiagramm des Paketes „Verwaltung“

Verantwortlichkeit festlegen

Bevor die Klassen im Detail modelliert werden, können Sie die Verantwortlichkeiten der Klassen festlegen.

Es handelt sich dabei um eine verhältnismäßig unscharfe und abstrakte Beschreibung, die in der natürlichen Sprache verfasst wird. Alle in objektorientierten Modellen repräsentierten fachlichen Sachverhalte werden letztendlich genau einer Klasse zugeordnet. Operationen, Attribute, Geschäftsregeln, Zusicherungen und Assoziationen sind immer genau einer Klasse zuzuordnen, die dafür verantwortlich ist.

Die UML gestattet, die Verantwortlichkeiten im Klassendiagramm den Symbolen der Klassen in einem Abschnitt unter den Attributen und Operationen anzugeben. Einige CASE-Tools unterstützen dies nicht. Sie können dort aber die Verantwortlichkeiten als Notiz für jede Klasse angeben oder eine komplette Beschreibung der Klasse erstellen, in der Sie auch die Verantwortlichkeiten definieren können.

Anforderungen festlegen

Sie können für jede Klasse die Anforderungen, die von dieser Klasse erfüllt werden müssen, in natürlicher Sprache notieren.

Diese Anforderungen haben die folgenden Merkmale:

- ✓ Sie sind eindeutig.
- ✓ Sie sind vollständig und widerspruchsfrei.
- ✓ Sie sind umsetzbar.
- ✓ Sie können zur Überprüfung der Funktionssicherheit getestet werden.
- ✓ Sie sind für alle Projektbeteiligten verständlich formuliert.

Nachfolgend, auszugsweise, die für ausgewählte Klassen festgelegten Verantwortlichkeiten und Anforderungen.

Klasse: Ticket	
Verantwortlichkeiten	Objekte dieser Klasse repräsentieren jeweils eine Eintrittskarte. Die Klasse speichert folgende Angaben: den Platz, den Preis, den Kunden, die Veranstaltung und die Verkaufsstelle. Die Objekte stellen eine Rechnungsposition dar.
Anforderungen	Ein Ticket-Objekt darf nur von Verkaufsstellen erstellt werden, die vom Systembetreuer für den Verkauf der Tickets der Veranstaltung zugelassen wurden.

Klasse: Rabattliste

Verantwortlichkeiten	Die Rabattliste ist ein Container für auf den Ticketpreis anzuwendende Preisabschläge oder Pauschalpreise (Festpreise). Sie dienen zur Ticketpreisberechnung aufgrund besonderer Kundeneigenschaften (z. B. Erwachsener, Kind, Rentner, Student, Gruppenkarte, Steuerkarte).
Anforderungen	Die Rabattliste muss vollständig sein, d. h., sie enthält alle zu vergebenden Rabatte. Die einzelnen Rabatte müssen für die Preisbildung vom Systembetreiber für die Verkaufsstelle freigeschaltet sein.

Klasse: Veranstaltung

Verantwortlichkeiten	Objekte dieser Klasse repräsentieren eine einzelne Veranstaltung. In der Klasse werden die zu verkaufenden Plätze gespeichert. Die Klasse unterhält Referenzen zu den für den Verkauf freigeschalteten Verkaufsstellen, den eingerichteten Abo-Arten und den für die grafische Ansicht angelegten Saalplänen. Für die Ermittlung des Bestplatzes wird eine Referenz auf ein Objekt der Klasse <code>Platzierer</code> gehalten. Eine weitere Referenz bindet die gastgebende Kulturstätte an die Veranstaltung. Für die Ticketpreisberechnung referenziert die Klasse auf ein Objekt der Klasse <code>Preisgenerator</code>. Die Klasse <code>Veranstaltung</code> stellt somit das zentrale Verwaltungsobjekt für eine Veranstaltung bereit.
----------------------	--

Klasse: Spielplan

Verantwortlichkeiten	Objekte dieser Klasse repräsentieren den Jahresspielplan. Die Klasse verwaltet in sich alle Veranstaltungen einer Spielzeit. Somit können alle freigeschalteten Verkaufsstellen Tickets für alle Veranstaltungen eines Veranstalters verkaufen. Der Spielplan ist an den Mandanten der Software gebunden.
----------------------	---

Klasse: Haus

Verantwortlichkeiten	Objekte dieser Klasse repräsentieren eine Spielstätte.
----------------------	--

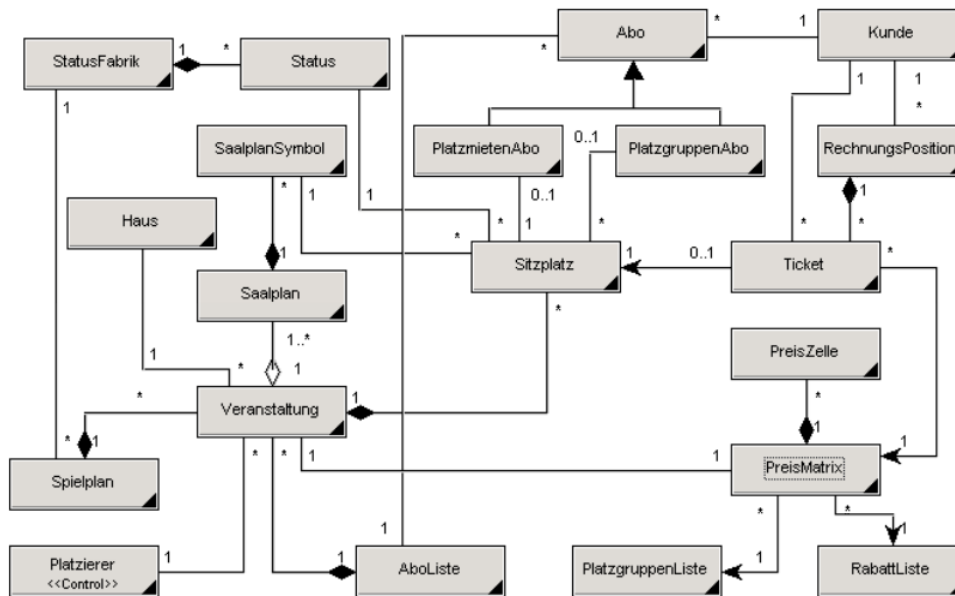
Klasse: Platzierer

Verantwortlichkeiten	Objekte dieser Klasse haben die Aufgabe, nach bestimmten Prinzipien Plätze des Saalplanes zum Verkauf anzubieten. Solche Prinzipien sind das Bestplatzprinzip, das Platzmieten-Abo oder ein Vergabeprinzip nach optischen Auslastungsüberlegungen des Saals.
----------------------	--

Klasse: Sitzplatz	
Verantwortlichkeiten	Objekte dieser Klasse repräsentieren jeweils einen Sitzplatz. Die Klasse speichert Angaben zum Platz (Reihe, Platznummer, Status-Objekt).
Klasse: Aboliste	
Verantwortlichkeiten	Objekte dieser Klasse repräsentieren eine Sammlung von Veranstaltungsnummern, die für den Verkauf eines Abos zur Auswahl stehen.
Klasse: Saalplan	
Verantwortlichkeiten	Objekte dieser Klasse repräsentieren eine grafische Darstellung des Saalplanes einer Veranstaltung. Die Klasse hält die darzustellenden Symbole im Saalplan.
Klasse: SaalplanSymbol	
Verantwortlichkeiten	Die Objekte der Klasse repräsentieren die grafischen Symbole im Saalplan. Sie können einen Sitzplatz referenzieren, dessen Status mit einer entsprechenden Grafik angezeigt wird.
Klasse: Preismatrix	
Verantwortlichkeiten	Objekte dieser Klasse repräsentieren eine Matrix, deren Spalten die Preise entsprechend den Preisgruppen und deren Zeilen die Rabattgruppen enthalten.
Anforderungen	Werden die Preise in der Platzgruppenliste oder in der Rabattliste geändert, werden die Preise automatisch angepasst. Eine Ausnahme bilden die Zellen, deren Inhalt ein absoluter Preis zugeordnet wurde.

Nach der Erarbeitung der Anforderungen und des Klassendiagramms können Sie Veränderungen sowohl in der Anordnung als auch in der Anzahl der Klassen feststellen.

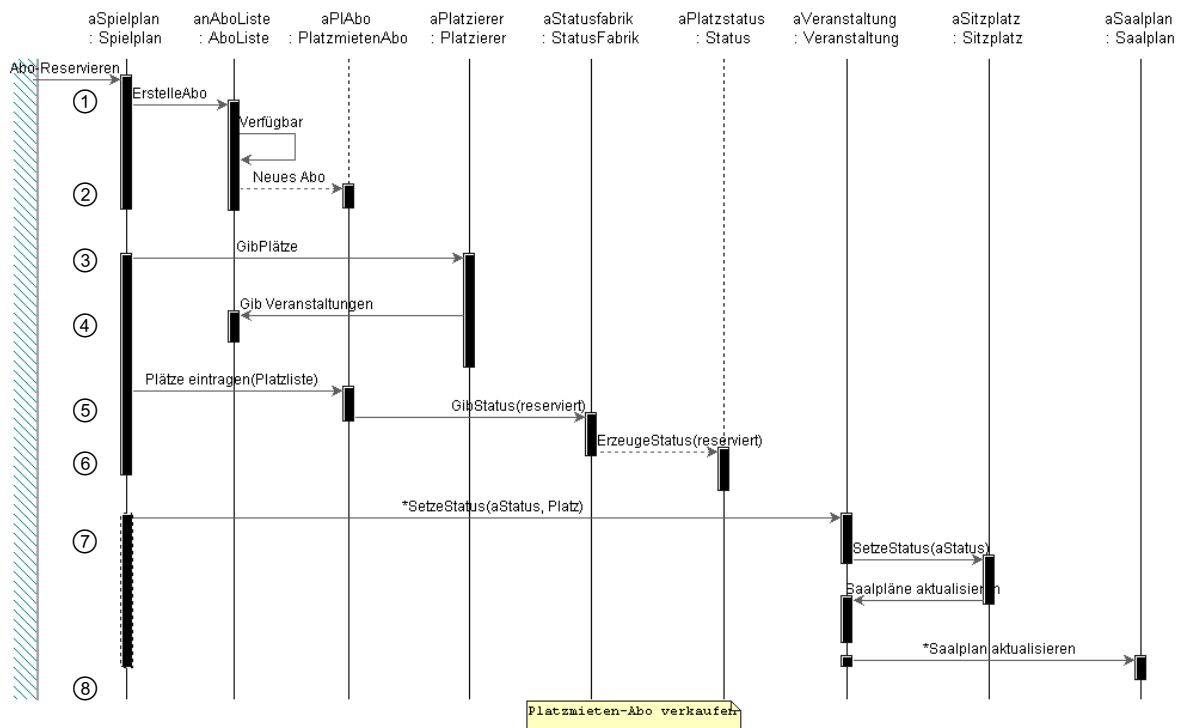
Das folgende neue Diagramm berücksichtigt bereits einige Aspekte der späteren Implementierung. So wurden z. B. leichtgewichtige Objekte für den Status eines Platzes hinzugefügt, die über eine Klasse verwaltet und bei Bedarf erzeugt werden. Mit einem solchen Status-Objekt werden alle Plätze mit gleichem Status verknüpft. Dadurch verringert sich die Anzahl der Objekte, da nicht für jeden Platz ein eigenes Objekt notwendig ist.



Verbessertes Klassendiagramm zum Paket „Verwaltung“

8.9 Sequenzdiagramme

Nachdem Sie ein Klassenmodell erstellt haben, können Sie das Zusammenspiel der Objekte aus zeitlicher Sicht modellieren und prüfen, ob alle benötigten Methoden vorhanden sind. Dazu stehen Ihnen Sequenzdiagramme zur Verfügung.



Sequenzdiagramm zum Vorgang „Platzmieten-Abo buchen“

- ✓ Erzeugen eines neuen Platzmieten-Abo-Objekts ①-②

Die Methode `AboReservieren` des `Spielplan`-Objekts löst den Vorgang `Platzmieten-Abo` buchen aus. Diese ruft die Methode `ErstelleAbo` des `AboListe`-Objekts auf. Innerhalb dieses Objekts wird die Verfügbarkeit des gewünschten Abos mit dem Aufruf einer weiteren Methode `Verfügbar` des gleichen Objekts fortgesetzt.

Nachdem über den Rückgabewert dieser Methode die Verfügbarkeit festgestellt wurde, wird die Methode `NeuesAbo` des `PlatzmietenAbo`-Objekts aufgerufen, worauf ein neues `PlatzmietenAbo`-Objekt erstellt wird und eine Referenz an das Objekt der Klasse `Spielplan` über das Objekt `aAboListe` zurückgegeben wird.

- ✓ Anfordern der Plätze zum Initialisieren des Abo-Objekts ③-④

Die Methode `GibPlätze` des `Platzierer`-Objekts wird vom `Spielplan`-Objekt aufgerufen. Dieses Objekt muss zunächst die Veranstaltungen ermitteln, aus denen die Plätze des Abos reserviert werden sollen. Dazu ruft es die Methode `GibVeranstaltungen` des `AboListe`-Objekts auf. Diese Methode gibt Referenzen der Veranstaltungen an das `Platzierer`-Objekt zurück, welches innerhalb der Methode `GibPlätze` die angeforderten Plätze bestimmen kann und diese anschließend an das `Spielplan`-Objekt zurückgibt.

- ✓ Zuweisen der Plätze und Erzeugen eines Status-Objekts mit dem Zustand `reserviert` ⑤-⑥

Das `Spielplan`-Objekt ruft die Methode `Plätze eintragen` des neu erstellten `PlatzmietenAbo`-Objekts auf und übergibt die Liste der ermittelten Plätze für das Abo. Die Plätze werden dem Abo-Objekt zugewiesen. Damit die Reservierung der Plätze auch in der Veranstaltung entsprechend markiert werden kann, wird über die Methode `GetStatus` des `StatusFabrik`-Objekts eine Referenz auf ein Status-Objekt angefordert. Wenn noch kein Status-Objekt mit dem Zustand `reserviert` vorhanden ist, wird eines über den Aufruf der Methode `ErzeugeStatus` des Status-Objekts erzeugt. Im Parameter wird der gewünschte Status übergeben.

- ✓ Setzen der Status für die Plätze und Aktualisieren der grafischen Darstellung des Saalplanes ⑦-⑧

Die Methode `SetzeStatus` des `Veranstaltung`-Objekts, welches die Sitzplätze der Veranstaltung verwaltet, wird aufgerufen. In der Methode wird die gleichnamige Methode des `Sitzplatz`-Objekts aufgerufen, durch die der Status des Sitzplatzes auf „reserviert“ gesetzt wird. Weil dieses Objekt eine Veränderung seines Zustandes erfahren hat, ruft es die Methode `Saalplan aktualisieren` jedes `Saalplan`-Objekts auf, in dem der Platz enthalten ist. Nachdem die grafischen Darstellungen der Saalpläne aktualisiert wurden, kehren die Methoden bis zum `Spielplan`-Objekt zurück, das die Aufrufe ausgelöst hat. Das Zeichen  vor der Methode `SetzeStatus` des `Spielplan`-Objekts modelliert die Wiederholung des Aufrufes der Methode für jeden Sitzplatz, der im Abo vergeben wurde.

Beachten Sie beim Lesen des Sequenzdiagramms, dass die Rückkehr der Botschaften nicht mitgezeichnet wird. Beispielsweise werden von der Botschaft `GibPlätze` die Veranstaltungsplätze zurückgegeben. Diese werden zuvor mit dem Aufruf der Botschaft `GibVeranstaltungen` zusammengestellt und nach der Analyse der Platzsituation vom Objekt `aPlatzierer` an das `Spielplan`-Objekt weitergegeben.

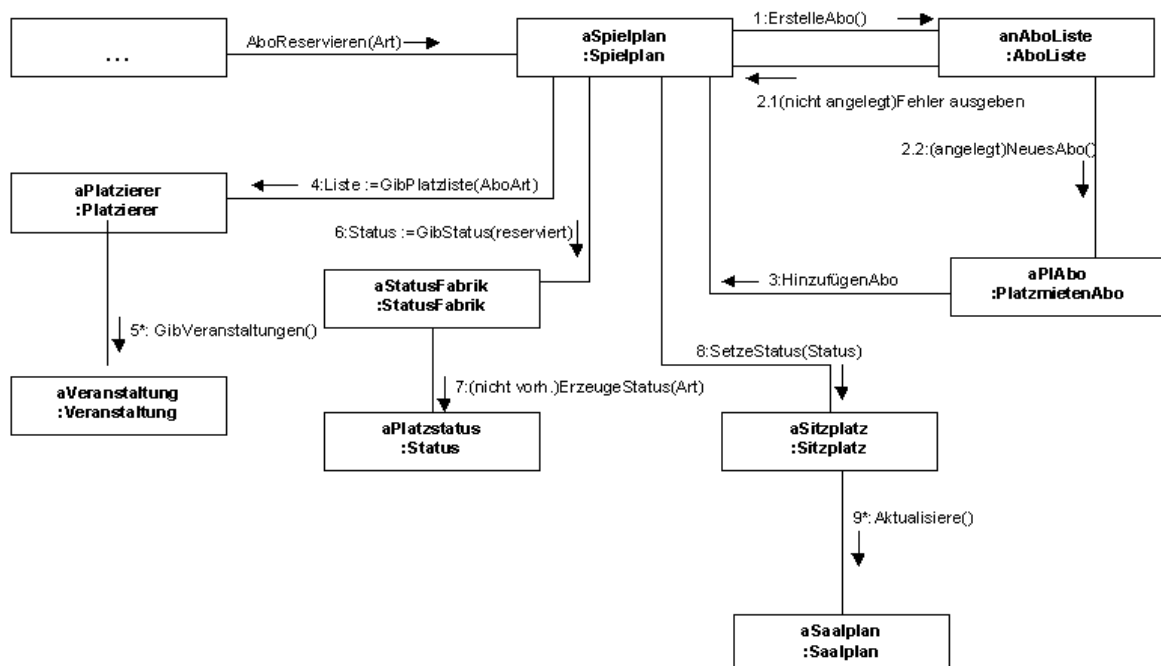


Die UML schreibt für das Erzeugen der Instanzen vor, dass der Name des Objekts horizontal erst in der Position dargestellt wird, in der es zeitlich gesehen erzeugt wird. `objectIF` zeichnet den Objektnamen immer an oberster Position des Diagramms. Die Lebenslinie wird bis zum Zeitpunkt der Erzeugung des Objekts mit einer unterbrochenen Linie dargestellt. Sobald das Objekt erstellt ist, erscheint eine durchgezogene Linie.

8.10 Kommunikationsdiagramme

Im Kommunikationsdiagramm können Sie die Zusammenarbeit von Klassen ähnlich dem Sequenzdiagramm modellieren. Die zeitliche Abfolge wird durch eine Nummerierung gekennzeichnet, wobei Verzweigungen durch eine zusätzliche Unternummerierung unterschieden werden. Diese Möglichkeit bietet Ihnen das Sequenzdiagramm nicht, da dort je Diagramm immer nur eine Wegvariante modelliert wird.

Da beide Diagramme den gleichen Zusammenhang darstellen, kann für die Modellierung auf eines der Diagramme verzichtet werden. Sie können beide Diagramme gegeneinander abgleichen, um die Richtigkeit und Vollständigkeit des Modells zu kontrollieren. objectiF unterstützt diese Diagrammart nicht.



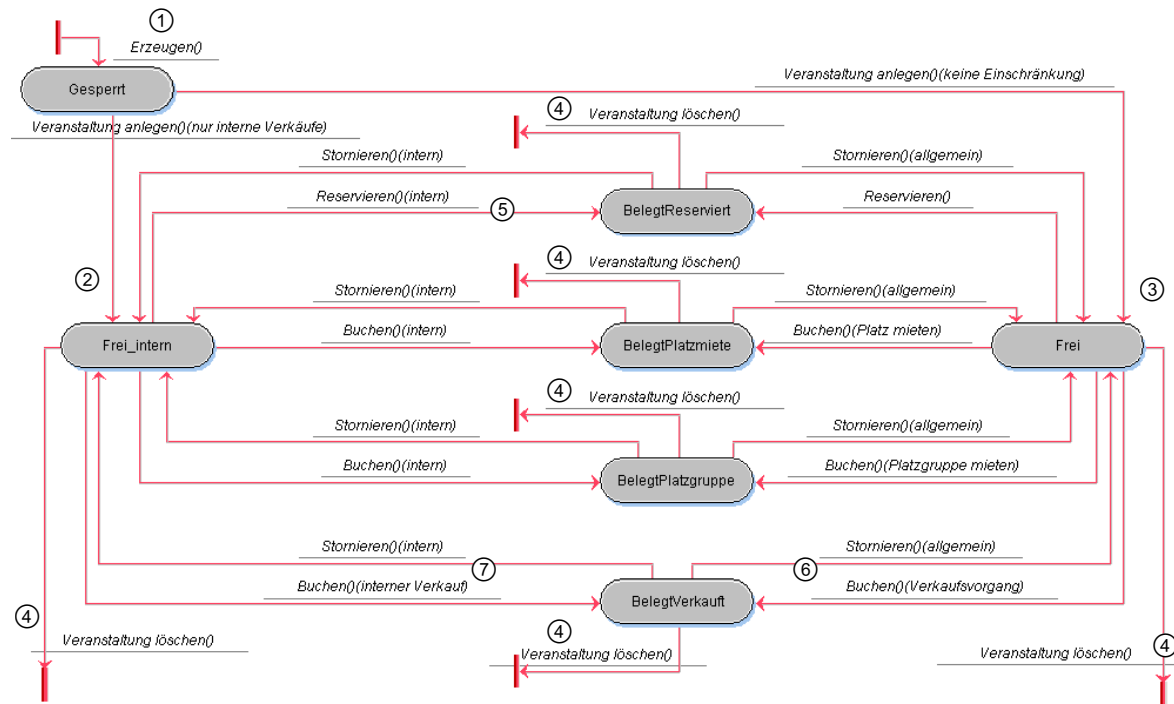
Kommunikationsdiagramm zum Vorgang „Platzmieten-Abo buchen“

Die Methode `AboReservieren` erhält keine Nummer, weil sie die erste Methode im zeitlichen Ablauf ist. Im zweiten Schritt wird die Methode `ErstelleAbo` aufgerufen, der die Nummer 1 vergeben wurde. Da alle anderen Methodenaufrufe, die von dem `Spielplan`-Objekt ausgeführt werden, höhere Nummern besitzen, wird diese Methode im zweiten Schritt aufgerufen. Im dritten Schritt gibt es zwei Methoden, die je nach der Bedingung (angelegt oder nicht angelegt) alternativ aufgerufen werden. In einem solchen Fall wird für die Methoden die gleiche Grundnummerierung mit einer nachgesetzten und unterschiedlichen Unternummerierung verwendet. Die Abfolge verläuft entsprechend den Hauptnummern in ansteigender Reihenfolge.

Auch bei diesem Diagramm werden die Rücksprünge der aufgerufenen Methoden nicht eingezeichnet.

8.11 Zustandsdiagramme

Die Daten in einer Anwendung werden durch die Zustände der Objekte charakterisiert. In einem Zustandsdiagramm können Sie die Zustände und die Zustandsübergänge für jeweils ein Objekt einer Klasse beschreiben. Die Zustandsbeschreibung einer Klasse wird, im Unterschied zur Modellierung der Zustandsübergänge im Aktivitätsdiagramm, für eine Klasse vollständig angegeben.



Zustandsdiagramm der Klasse Sitzplatz

Im Diagramm wird gezeigt, wie sich der Zustand eines Sitzplatz-Objekts durch den Aufruf von Methoden ändert. Der Aufruf der Methode Erzeugen ① legt ein Objekt der Klasse Sitzplan an. Der Zustand dieses Objekts ist durch die Attributsbelegung Gesperrt gekennzeichnet. Über den Methodenaufruf Veranstaltung anlegen wechselt der Zustand des Objekts nach Frei_intern ② oder Frei ③. Die Entscheidung wird dabei über den Parameter getroffen. Wurde ein Sitzplatz-Objekt einer Veranstaltung zugewiesen, wird durch den Aufruf Veranstaltung löschen das Sitzplatz-Objekt wieder zerstört ④. Die Objektzustände BelegtReserviert, BelegtPlatzmiete, BelegtPlatzgruppe und BelegtVerkauft werden von einem Sitzplatz-Objekt, das vor der Zustandsänderung den Zustand Frei oder Frei_intern besaß, angenommen. Beispielsweise wird durch den Aufruf der Methode Reservieren ⑤ der Zustand BelegtReserviert angenommen. Ein Objekt mit dem Zustand Frei_intern kann nur über die Methode Reservieren den neuen Zustand annehmen, wenn im Parameter intern übergeben wird. Hat ein Sitzplatz-Objekt den Zustand BelegtVerkauft, kann mit dem Aufruf der Methode Stornieren der Zustand Frei ⑥ oder Frei_intern ⑦ erreicht werden. Die Entscheidung wird mit dem Parameter intern oder allgemein getroffen.

8.12 Übung

Erstellen von UML-Diagrammen für das Ticketsystem

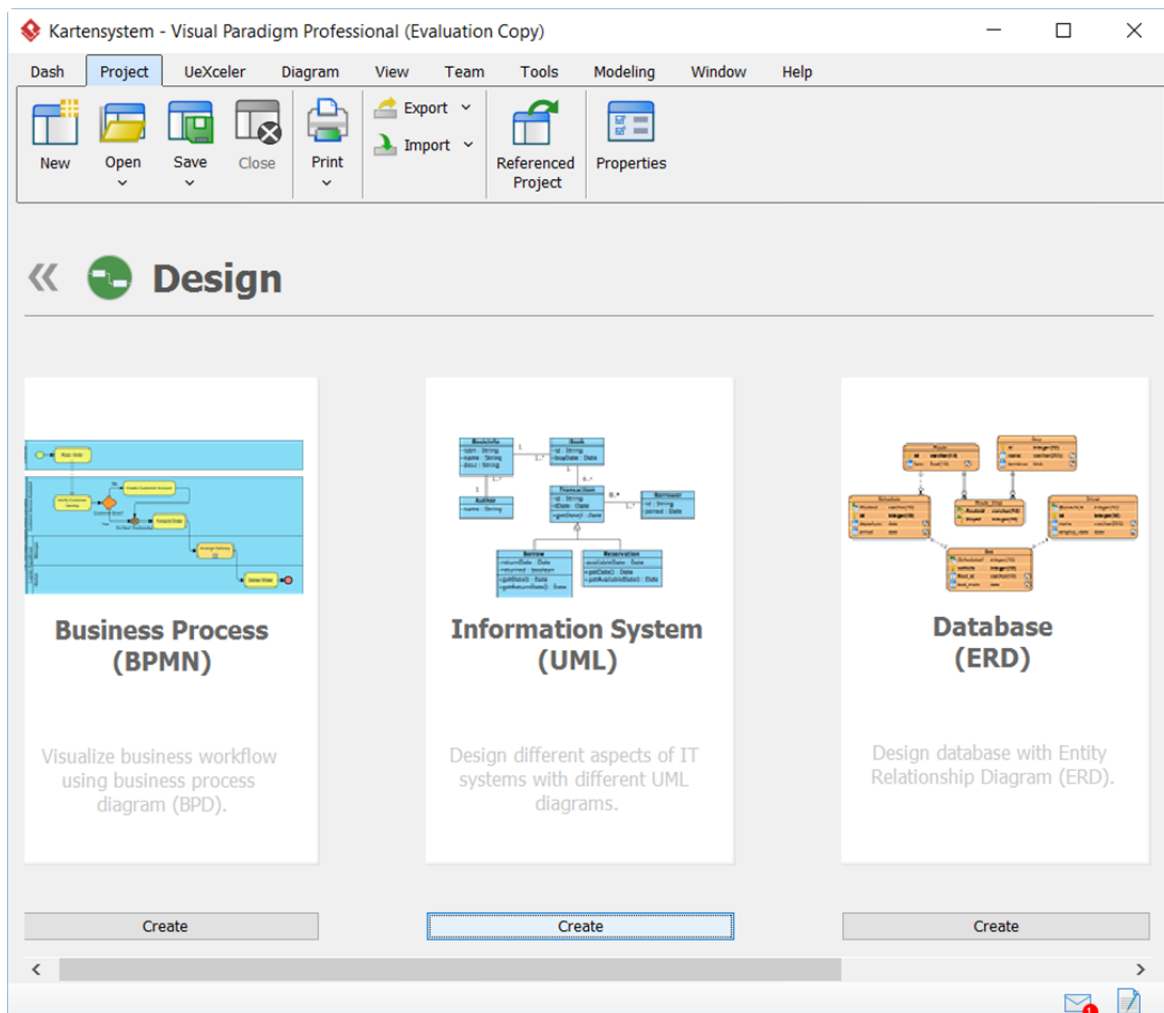
Level		Zeit	ca. 40 min
Übungsinhalte	✓ Anwendungsfalldiagramm erstellen ✓ Essenzielle Anwendungsfallbeschreibung erstellen ✓ Aktivitäts- und Klassendiagramm entwerfen ✓ Zeitlichen Ablauf eines Anwendungsfalls in einem Sequenzdiagramm modellieren		
Übungsdatei	--		
Ergebnisdatei	Kapitel_08_Loesungen_Uebungen.docx		

1. Erstellen Sie ein Anwendungsfalldiagramm für das Paket *Veranstaltung verwalten*. Berücksichtigen Sie dabei, dass Sie die Plätze aus einer Vorgabe übernehmen können oder mit dem Saalplan-Editor neue Saalpläne erstellen können.
2. Erstellen Sie für die in der Übung ① modellierten Anwendungsfälle eine essenzielle Beschreibung und zeichnen Sie für einen Anwendungsfall ein Aktivitätsdiagramm.
3. Entwerfen Sie ein Klassendiagramm für den Anwendungsfall *Veranstaltung anlegen*.
4. Modellieren Sie den zeitlichen Ablauf für den Anwendungsfall der Übung ③ in einem Sequenzdiagramm.

9

Software für die Modellierung mit der UML

9.1 CASE-Tools



Nach der Einschätzung von Wirtschaftsexperten kann die Wirtschaft trotz neuer Software-technologien nicht die jährlich wachsende Nachfrage nach hochwertigen Softwaresystemen befriedigen, die schnell und preiswert genug entwickelt werden.

Um zukünftig diese Herausforderung anzugehen, werden seit Ende der 80er-Jahre verstärkt CASE(Computer Aided Software Engineering)-Tools bei der Softwareentwicklung eingesetzt. Diese Hilfsmittel sind entweder einzelne CASE-Werkzeuge oder CASE-Werkzeug-Umgebungen. Bei letzteren handelt es sich um eine Gruppe integrierter CASE-Werkzeuge, die alle Phasen des Software-Lebenszyklus im Zusammenhang sehen und jede einzelne Phase direkt unterstützen.

Objektorientierte CASE-Tools stellen objektorientierte Modellierungsnotationen und -methoden zur Verfügung und generieren automatische Teile der objektorientierten Anwendungen (nicht alle CASE-Tools). Viele dieser Tools unterstützen mehrere Programmiersprachen (z. B. JAVA, C# und C++) und objektorientierte Datenbanken (z. B. PostgreSQL) direkt.

9.2 Anwendungsbereich

CASE-Tools bieten dem Entwickler vor allem dann Vorteile, wenn es sich um mittlere bis größere Projekte handelt. Immer weiter steigende Ansprüche der Anwender in Bezug auf Komfort und Funktionsumfang führen zu bisher ungeahnten Dimensionen der Komplexität der Anwendung. Je größer das Projekt ist, desto wichtiger ist die lückenlose Benutzung von CASE-Tools über alle Ebenen der Entwicklung hinweg.

CASE-Tools bieten für die frühen Phasen der Entwicklung eine transparente und visuelle Vorgehensweise an, die es den Entwicklern ermöglicht, das zu realisierende System als Ganzes zu betrachten; dadurch verlieren sie sich nicht in frühen Phasen der Entwicklung in Implementierungsdetails, sondern konzentrieren sich auf die Architektur ihrer Anwendung.

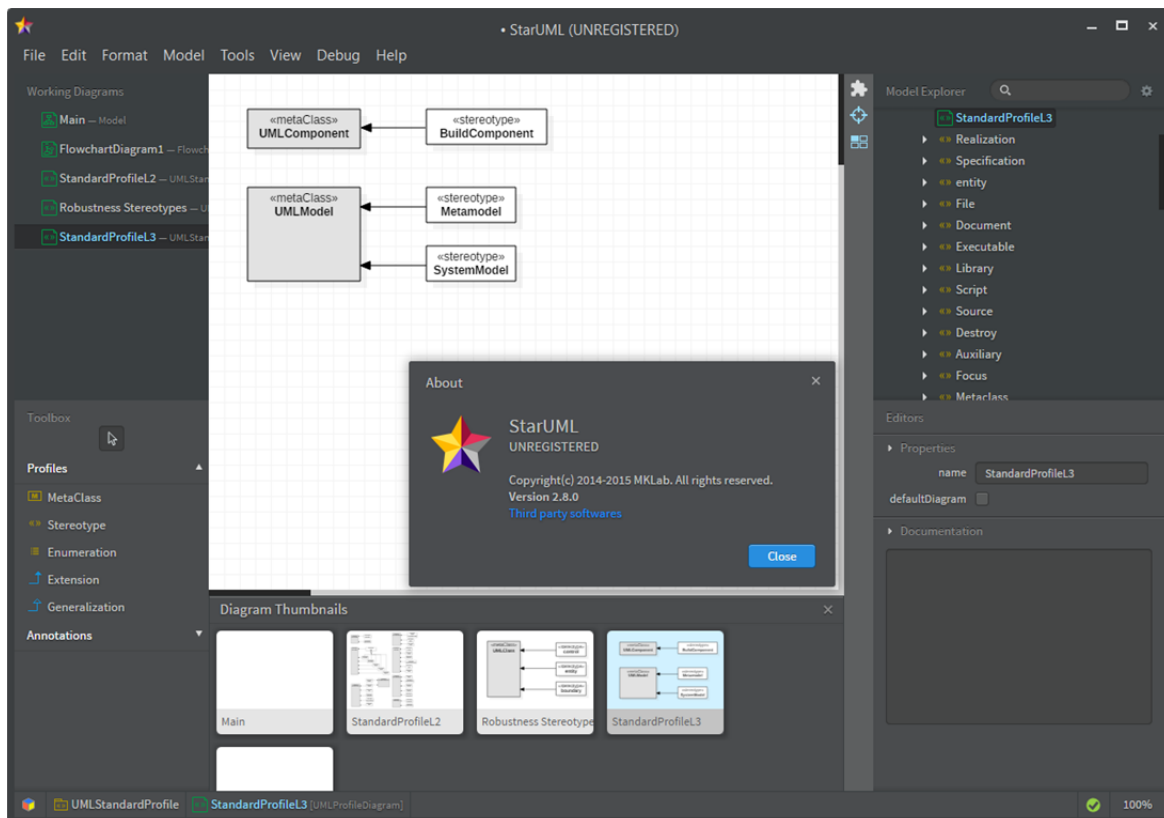
9.3 Anforderungen an ein CASE-Tool



CASE-Tools können den gesamten Lebensprozess eines Softwareproduktes begleiten. Diese Anwendungen sind oft sehr komplex. Aus diesem Grund werden auch CASE-Tools angeboten, die nicht alle Phasen der Entwicklung unterstützen und zu einem Preis angeboten werden, der auch für kleinere Projekte interessant ist; zumal auch die Bedienung einfacher ist und der Einarbeitungsaufwand entsprechend geringer ausfällt.

Die folgende Zusammenstellung zählt wichtige Anforderungen an ein CASE-Tool auf, wobei nicht alle Punkte in jedem System erfüllt sein müssen.

- ✓ Unterstützung der UML-Diagrammtypen
- ✓ Einfache, intuitive Benutzerführung
- ✓ Leicht bedienbarer Diagrammeditor
- ✓ Reverse Engineering (Rückgewinnung des Modells einer fertigen Software aus dessen Quellcode)
- ✓ Roundtrip Engineering (Parsen von Quellcode fertiger Anwendungen und automatische Modellgenerierung)
- ✓ Flexible Dokumenterstellung in mehreren Formaten (z. B. HTML, Word, RTF, XML)
- ✓ Code-Generierung für mehrere objektorientierte Sprachen
- ✓ Datenbankunterstützung
- ✓ Unterstützung von Teamarbeit in großen Projekten
- ✓ Offene Architektur für beliebige Erweiterungen
- ✓ In heterogenen Umfeldern Unterstützung von möglichst vielen Betriebssystem-Plattformen



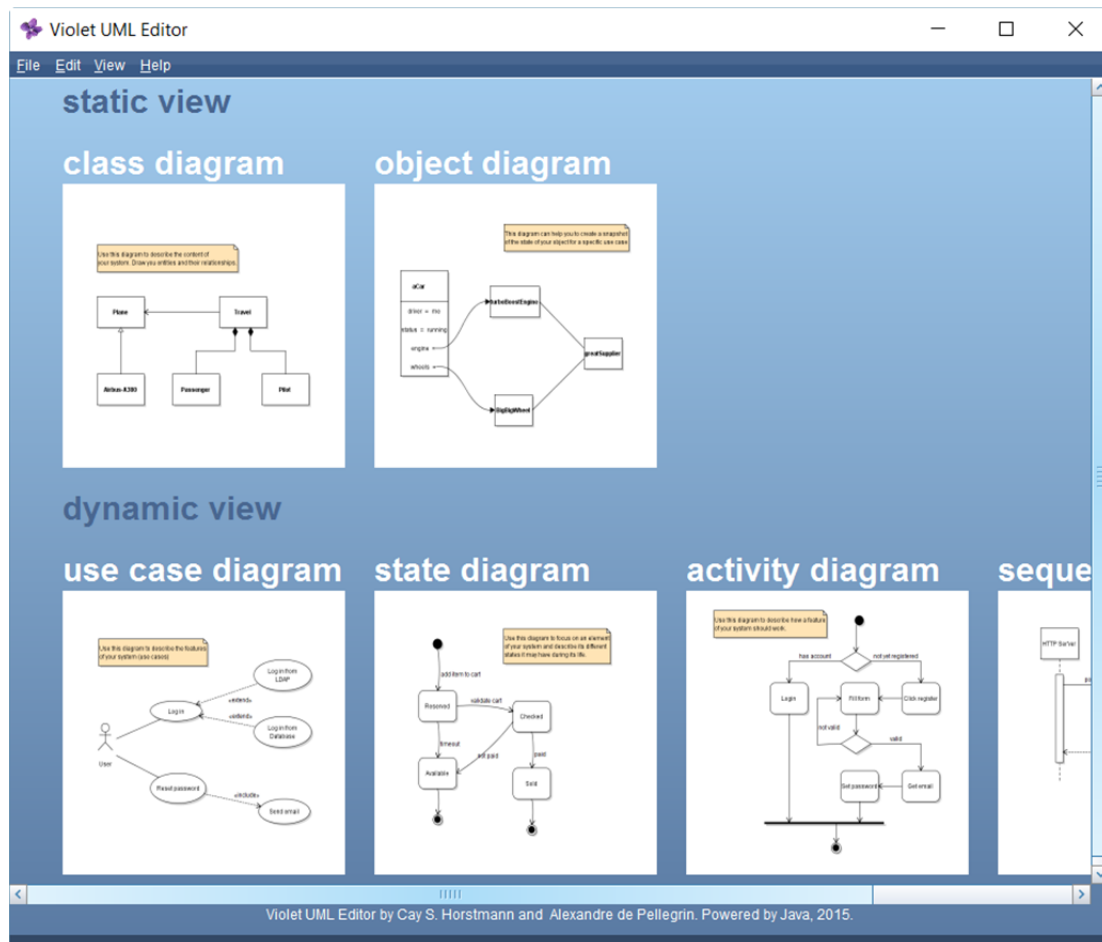
Diese Anforderungen stellen eine Sammlung dar und haben unterschiedliche Bedeutung für den Prozess der Erstellung. In großen Projekten ermöglichen CASE-Tools erst eine effiziente Entwicklung, wenn sie alle Anforderungen erfüllen. Dabei übersteigt der Nutzen die Kosten solcher Systeme. Sie sollten bei der Auswahl des CASE-Tools auch die Software-Wartung und -Pflege berücksichtigen, deren Aufwand mit dem Einsatz dieser Software-Werkzeuge erheblich reduziert werden kann.

9.4 Überblick CASE-Tools

Es gibt eine Vielzahl verwendbarer CASE-Tools, nachfolgend werden einige der meistbenutzten und deren Eigenschaften im Überblick aufgezählt.

Eigenschaften/Tool	Modelio	Visual Paradigm for UML	ArgoUML	objectiF	StarUML
Hersteller	Modeliosoft / Open Source	Visual Paradigm	University of California / Open Source	microTool GmbH Berlin	MKLab, Co.
Use-Case-Diagramm	ja	ja	ja	ja	ja
Class-Diagramm	ja	ja	ja	ja	ja
State-Diagramm	ja	ja	ja	ja	ja
Deployment-Diagramm	ja	ja	ja	ja	ja
Activity-Diagramm	ja	ja	ja	ja	ja
Communication-Diagramm	ja	ja	ja	ja	ja
Sequence-Diagramm	ja	ja	ja	ja	ja
Code-Generierung	Java, C#, C++, SQL	C++, Java, C#, PHP	C#, C++, Java, PHP, SQL	C#, C++, Java, VB.Net	C#, C++, Java, PHP, Ruby
Team-Unterstützung	ja	ja	nein	ja	nein
Reverse Engineering	ja	ja	nein	ja	ja
Roundtrip Engineering	ja	ja	nein	ja	ja

9.5 Grafische Tools



Für die Verständigung unter den Entwicklern und späteren Anwendern der Software haben sich die UML-Diagramme und weitere grafische Zeichenelemente besser als umfangreiche Textbeschreibungen bewährt. Da die Erstellung von Quellcode aus den Diagrammen die Komplexität der CASE-Tools stark erhöht und zum Teil nicht benötigt wird (z. B. weil die Sprache, in der die Anwendung geschrieben werden soll, nicht unterstützt wird), kommen auch Tools ohne Quellcode-Generierung zum Einsatz.

Diese Tools sind preiswerter und haben den Vorteil, dass die Namensgebung z. B. von Klassen, Objekten und Schnittstellen keinen Konventionen unterliegt. Die Implementierung der Anwendung aus den erstellten Modellen können Sie bei diesen grafischen Tools nur manuell vornehmen.

Solche grafischen Tools sind zum Beispiel die Windows-Anwendung **Visio** von Microsoft oder der **Violet UML Editor** von Cay S. Horstmann und Alexandre de Pellegrin, der für Windows oder auch Online unter der GPL-Lizenz verfügbar ist.

9.6 Adressen im Internet

Unter nachfolgenden Internetadressen erhalten Sie weiterführende Informationen zu einigen der gängigen hier aufgezählten CASE-Tools und grafischen Modellierungswerkzeugen:

Produktname	Firma	Internetauftritt
Modelio	Modeliosoft / Open Source	http://www.modeliosoft.com/
Visual Paradigm for UML	Visual Paradigm	http://www.visual-paradigm.com/
ArgoUML	University of California / Open Source	http://argouml.tigris.org/
objectiF	microTool GmbH Berlin	https://www.microtool.de/modellgetriebene-entwicklung-mit-objectif/
StarUML	MKLab, Co.	http://www.staruml.io/
Microsoft Visio	Microsoft	http://www.microsoft.com/office/visio/
Violet UML Editor	Cay S. Horstmann und Alexandre de Pellegrin	http://alexdp.free.fr/violetumleditor/page.php/

Aufgrund der Vielzahl vorhandener UML-CASE-Tools empfiehlt es sich, vor Installation/Einsatz eines konkreten Tools im Internet eine Übersicht der aktualisierten CASE-Tools einzusehen. Eine dieser Übersichten steht unter <https://de.wikipedia.org/wiki/UML-Werkzeug> zur Verfügung.

10

Einführung in die Anwendung objectiF

10.1 Grundfunktionen von objectiF

objectiF ist ein CASE-Tool, das von der Firma microTOOL in Berlin entwickelt wurde. Dieses Tool kann Sie bei der Softwareentwicklung in allen drei Phasen (Analyse, Design und Implementierung) unterstützen. Zusätzlich stellt dieses Tool Funktionen zur Generierung der Dokumentation der Entwicklungsarbeit bereit, diese kann wahlweise im Word oder HTML erzeugt werden. Unterstützung zur praxisnahen Erstellung eines Pflichtenheftes ist ebenfalls enthalten.

Über den ScreenPainter, der in objectiF integriert ist, können Sie Dialogfenster gestalten und dem Auftraggeber präsentieren. Jedes Dialogfenster wird in einem Szenario organisiert. Durch die generierten HTML-Seiten wirken die Dialogfenster realitätsnah und geben einen lebendigen Ausblick auf die spätere Version des Programms. Da die Implementierung dieser Dialogfenster nicht in die Anwendung übernommen wird, erhalten sie den Namen Wegwerf-Prototypen. Die Ausführung der Szenarios und damit die Erstellung der passenden Dialogfenster ist für jeweils eine Kommunikationslinie zwischen Akteur und Anwendungsfall möglich.

Die Anwendung ist in der Lage, nach der Erstellung des Designmodells automatisch Quellcode zu generieren. Dabei geben Sie die Dateien vor, in denen die Quellcode-Fragmente gespeichert werden. Mithilfe des integrierten Quellcode-Editors können Sie die komplette Anwendung erstellen. Die zur Verfügung stehenden Klassen werden über einen Explorer zur Auswahl angeboten. Sie können per Vorgabe entscheiden, aus welchem Programmiersystem Sie die Klassen einbinden möchten. Dabei stehen ANSI C++, C#, Visual Basic.Net, die MFC und Java zur Auswahl. Über ein Zusatztool können auch Delphi und der C++Builder von Borland eingebunden werden.

objectiF gestattet die Einbindung mehrerer Entwicklungsumgebungen, sodass Sie die Programmentwicklung mit dem Quellcode-Editor, dem Debugger und dem Compiler/Interpreter dieser Entwicklungsumgebungen realisieren können.

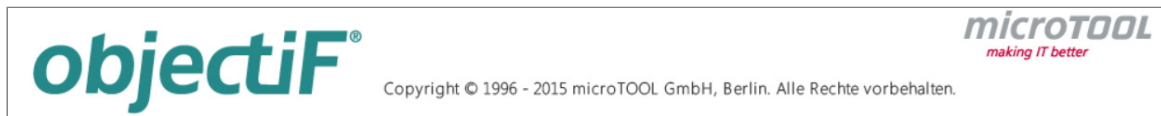
Haben Sie das CASE-Tool nicht zu Beginn der Entwicklung eingesetzt, liegen Ihnen auch von diesen Phasen keine Informationen vor. Das Tool erlaubt Ihnen aber das Analysieren des bereits erstellten Quellcodes. Es erstellt automatisch das Klassenmodell Ihres Programms, mit dem Sie die Entwicklung weiterführen können.

Diese Funktion bringt Ihnen besonders bei der Wartung älterer Programme Vorteile. Die Generierung des Modells aus dem Quellcode wird als Reverse Engineering bezeichnet.

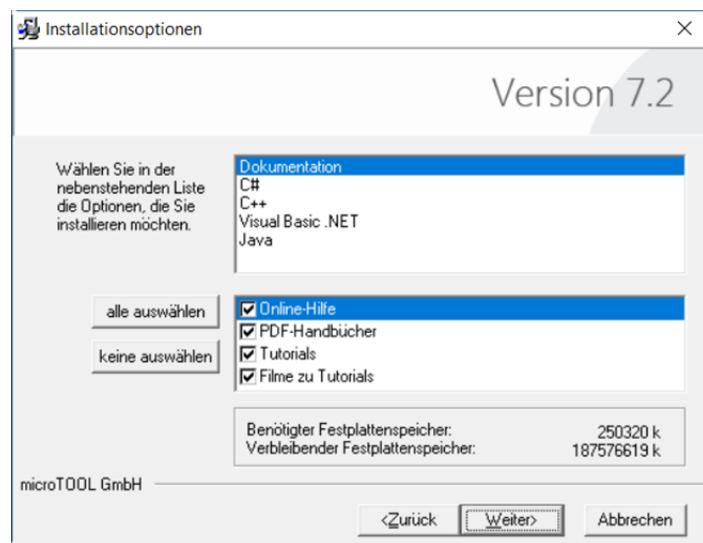
objectiF installieren

objectiF wird in der aktuellen und in diesem Buch verwendeten Version 7.2 als 30-Tage-Testversion auf einer Vielzahl von Internetseiten (beispielsweise http://www.chip.de/downloads/objectiF-Enterprise-Trial-Edition_18828757.html) zum Download bereitgestellt.

- Installieren Sie das Programm objectiF durch Ausführen der ausgewählten und heruntergeladenen Setup-Datei.

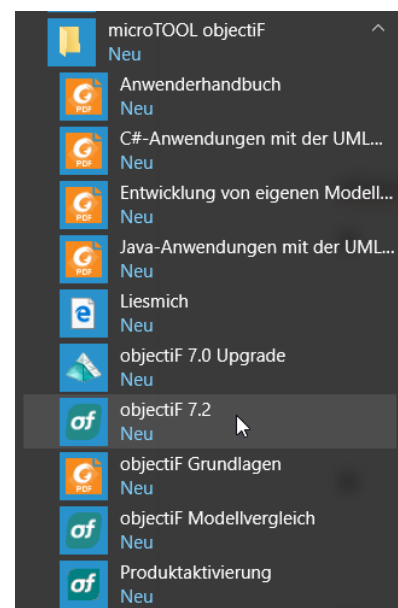


- Geben Sie im nachfolgenden Dialogfenster den gewünschten Installationsumfang vor.

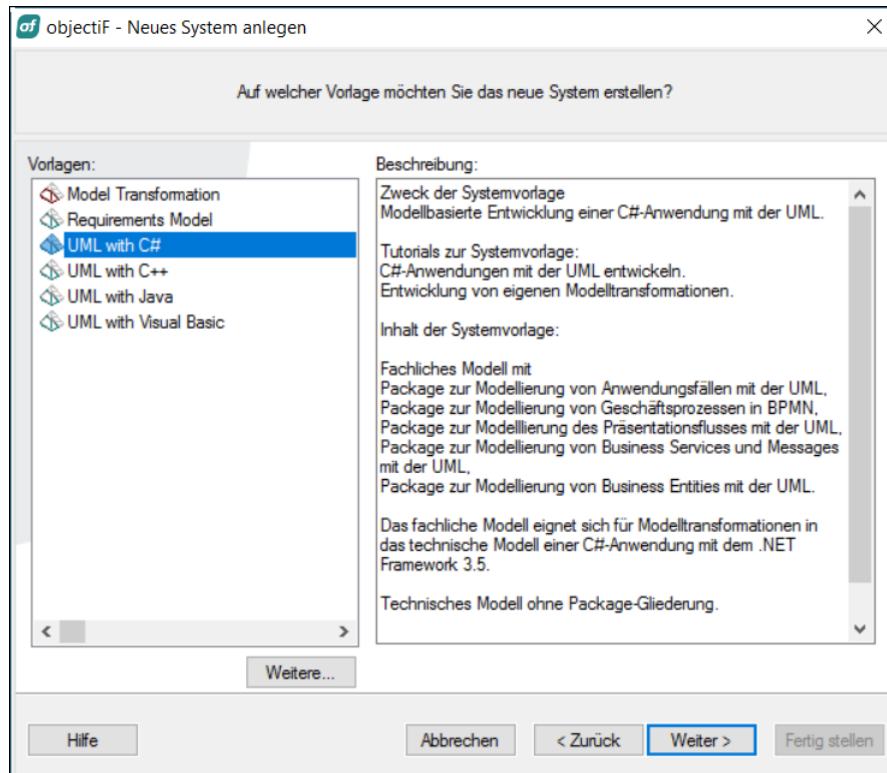


objectiF starten

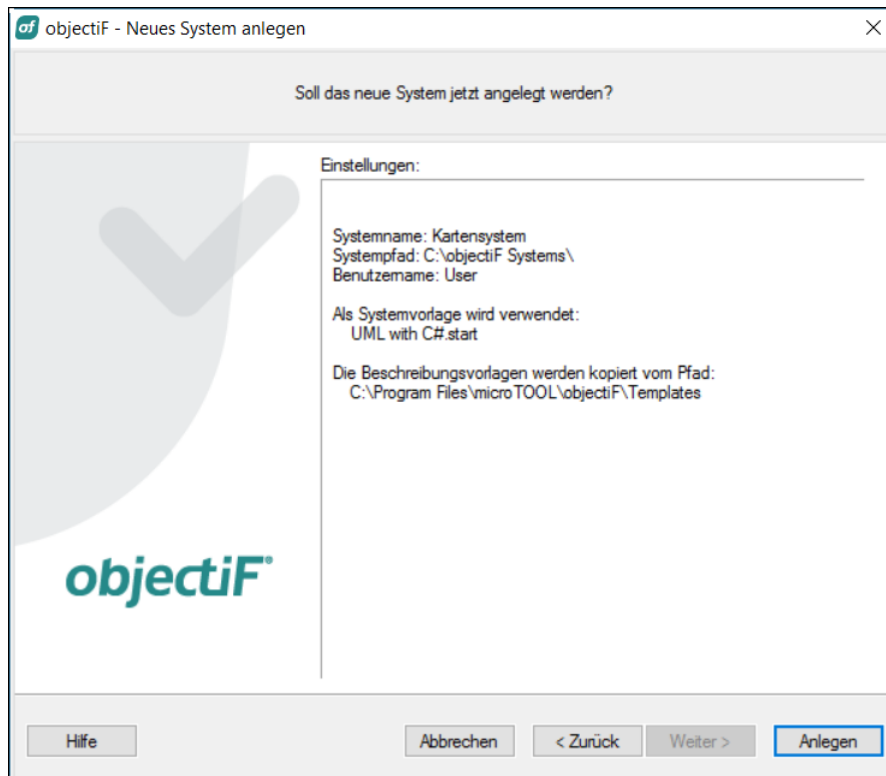
- Wählen Sie den Menüpunkt *Start - Alle Programme - microTOOL objectif - objectif 7.2*.
objectiF wird gestartet.
Beim ersten Start der Anwendung wird eine einmalige Aktivierung bzw. Registrierung gefordert.



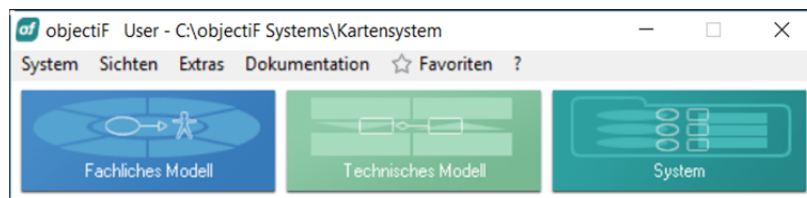
- ▶ Wählen Sie in dem geöffneten Dialogfenster das Optionsfeld *Ein neues System anlegen* aus (in objectiF wird ein Projekt durchgängig als „System“ bezeichnet).
- ▶ Betätigen Sie die Schaltfläche *Weiter*.
- ▶ Wählen Sie die passende Vorlage – z. B. *UML with C#* –, aus der das neue System erstellt werden soll.
- ▶ Betätigen Sie die Schaltfläche *Weiter*.



- ▶ Vergeben Sie Namen, Anwenderkennung und Speicherort für das zu erstellende System.
- ▶ Betätigen Sie die Schaltfläche *Weiter*.
- ▶ Bestätigen Sie das Einfügen der angebotenen Beschreibungsmuster aus dem objectiF-Template-Verzeichnis.
- ▶ Betätigen Sie die Schaltfläche *Weiter*.
- ▶ Überprüfen Sie die bisher ausgewählten und angezeigten Installationsoptionen und bestätigen Sie diese mit der Schaltfläche *Anlegen*.



Nach dem Anlegen eines Projektes startet die Anwendung. Auf dem Bildschirm wird das Hauptfenster von objectiF geöffnet. Sie können jetzt das angelegte Projekt modellieren.



In der aktuellen objectiF-Version kann die Modellierung in der plattformunabhängigen Sicht **Fachliches Modell** bzw. in der plattformspezifischen Sicht **Technisches Modell** erfolgen. Der grundlegende Funktionsumfang und Bedienmodus ist durchgängig kontextsensitiv und in beiden Sichten weitestgehend identisch. In den folgenden Abschnitten wird die Modellierung aus plattformunabhängiger Sicht beschrieben. Modellierte Komponenten können von einer Sicht in die andere kopiert bzw. verschoben werden.

Grundlegende Vorgehensweise für die Analyse

- ✓ Erstellen der Pakete
- ✓ Erstellen der Akteure und Anwendungsfälle
- ✓ Erstellen der Anwendungsfalldiagramme, in denen die Beziehungen der Anwendungsfälle mit den Akteuren erstellt werden
- ✓ Erstellen der essenziellen Anwendungsfallbeschreibungen
- ✓ Erstellen der Aktivitätsdiagramme
- ✓ Modellieren des Objektflusses in den Aktivitätsdiagrammen

Grundlegende Vorgehensweise für das Design

- ✓ Erstellen der Klassen
- ✓ Erstellen der Attribute und Methoden der Klassen
- ✓ Erstellen von Klassendiagrammen
- ✓ Modellierung der Beziehungen zwischen den Klassen im Klassendiagramm
- ✓ Erfassen der Zustandsautomaten der Klassen in Zustandsdiagrammen
- ✓ Erstellen eines Sequenzdiagramms

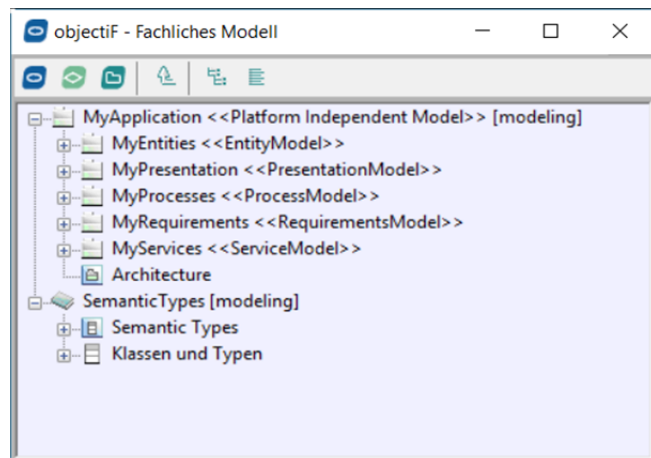
Grundlegende Vorgehensweise für die Implementierung

- ✓ Automatisches Generieren der Klassendefinitionen in den vorgesehenen Dateien
- ✓ Manuelles Einfügen der Implementierungen der Methoden

10.2 Analysemodell erstellen

- ▶ Betätigen Sie die Schaltfläche *Fachliches Modell* auf dem Hauptfenster von objectiF.

Es wird ein Fenster geöffnet, das einen Baum mit den möglichen Modellierungselementen als Einträgen anzeigt. Da Sie noch keine Pakete für Ihre Anwendung angelegt haben, ist nur ein (globales) Paket vorhanden und es werden die Elementkategorien dieses Paketes angezeigt. Um eine Gliederung bzw. eine Aufteilung in kleinere Einheiten zu erreichen, werden zunächst Pakete für die neue Anwendung erstellt. Dieser Schritt ist notwendig, um den Entwicklerteams eine fest umrissene Arbeitsaufgabe, die einen Teil der Anwendung umfasst, in Form eines oder mehrerer Pakete übergeben zu können.

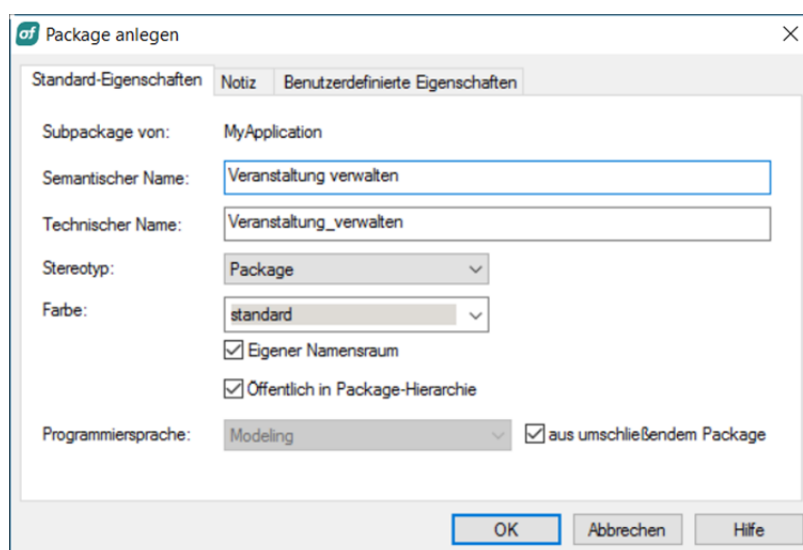
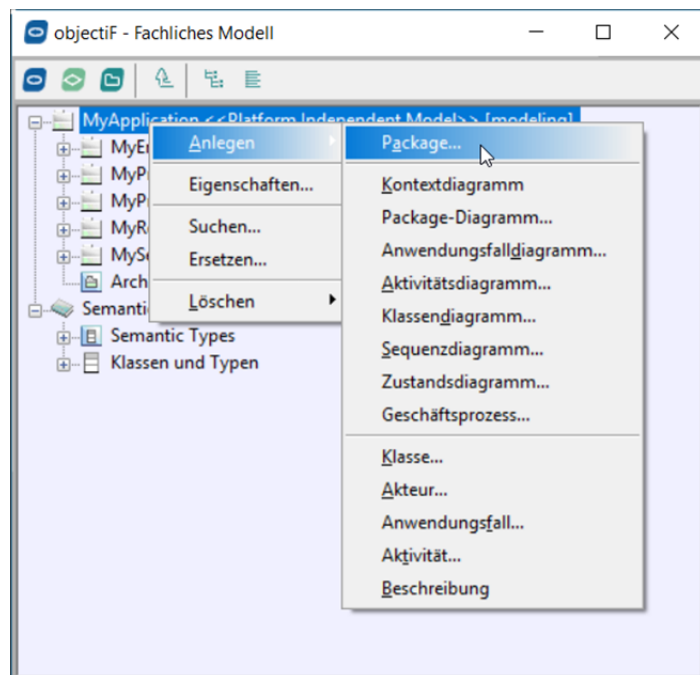


Sie können die Pakete auch nach der Erarbeitung der Elemente, die eine Darstellung in der UML ermöglichen, erstellen und die zum Paket gehörigen Elemente aus dem globalen Paket in das entsprechende Paket per Drag & Drop verschieben. Da oft eine Arbeitsteilung in Entwicklergruppen in einem frühen Stadium der Entwicklung vollzogen wird, ist die Paketerstellung auch in diesem Buch an die erste Stelle gesetzt worden.

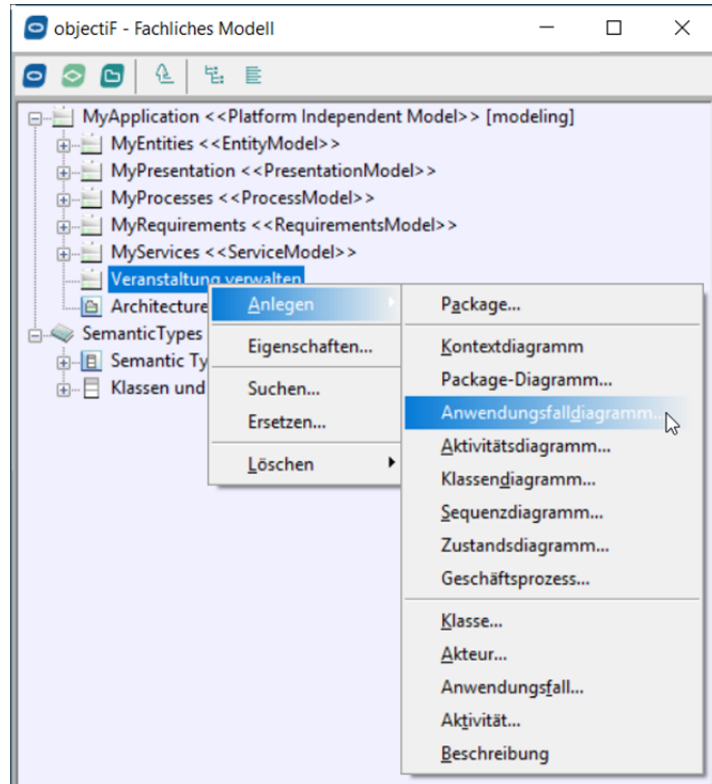
Anlegen von Paketen

- ▶ Öffnen Sie das Dialogfenster *Fachliches Modell*.

- ▶ Wählen Sie aus dem Kontextmenü des Fensters den Menüpunkt *Anlegen - Package* aus.
- ▶ Geben Sie einen Namen in das Eingabefeld *Semantischer Name* (entspricht dem Namen des Paketes in den Diagrammen der UML) für das neue Paket ein.
Im Eingabefeld *Technischer Name* wird ein nach den Namenskonventionen der Programmiersprache passender Name erstellt (z. B. ohne Leerzeichen und Umlaute).
- ▶ In dem Eingabefeld der Registerkarte *Notiz* können Sie Informationen über das Paket eingeben.
- ▶ Bestätigen Sie Ihre Angaben mit der Schaltfläche *OK*.



Das neu erstellte Paket wurde in das Diagramm eingetragen. Um ein Diagramm oder Element für ein Paket zu erstellen, markieren Sie das Paket und wählen aus dem Kontextmenü *Anlegen* das gewünschte Diagramm bzw. Element.



10.3 Anwendungsfälle

Ein Anwendungsfall ist eine Folge von Aktivitäten meist eines einzelnen Akteurs, mit denen ein fachlich relevantes sichtbares Ergebnis erzielt wird. Zur Modellierung der Anwendungsfälle eines Paketes benötigen Sie Akteure und Anwendungsfälle, die in ein Anwendungsfalldiagramm eingezeichnet werden. Beide können Sie über das Dialogfenster *Fachliches Modell* erstellen. Alle Angaben dienen ausschließlich der Information und beeinflussen die später von objectiF generierten Quellcode-Fragmente nicht.

Anlegen der Akteure für das Paket

Die Akteure sind die Projektbeteiligten. Dies können Personen, z. B. Anwender, oder auch Fremdsysteme sein, die Informationen bereitstellen oder Eingabewerte liefern.

- ▶ Klicken Sie im Kontextmenü *Anlegen* des gewünschten Paketes auf den Eintrag *Akteur*. Es wird ein Dialogfenster zur Eingabe eines neuen Akteurs geöffnet.
- ▶ Tragen Sie den Namen des Akteurs in das Eingabefeld *Akteur* ein.
- ▶ Wählen Sie das Stereotyp für den Akteur aus der Liste des Kombinationsfelds *Stereotyp* aus (z. B. *Secondary Actor* für ein Fremdsystem oder *Primary Actor* für einen Bediener).
- ▶ Markieren Sie das Kontrollfeld *abstrakt*, wenn dieser Akteur als Rolle für weitere Akteure verwendet wird, d. h., wenn von diesem weitere Akteure abgeleitet werden.

- ▶ Im Eingabefeld *Notiz* können Sie Informationen eingeben, die die Aufgabe des Akteurs in der Modellierung näher beschreiben.
- ▶ Schließen Sie das Dialogfenster mit der Schaltfläche *OK*.

Das Ergebnis der Eingabe können Sie im Diagramm betrachten, wenn Sie den Eintrag *Akteure* unter dem Eintrag des Paketes erweitern. Dort werden die angelegten Akteure des Paketes angezeigt.

Anlegen der Anwendungsfälle

Ein Anwendungsfall kann in einem Paket untergebracht werden. Mit den folgenden Schritten erstellen Sie einen Anwendungsfall.

- ▶ Wählen Sie aus dem Kontextmenü des gewünschten Paketes den Menüpunkt *Anlegen - Anwendungsfall* aus.
Es wird ein Dialogfenster zur Eingabe eines neuen Anwendungsfalls geöffnet.
- ▶ Tragen Sie den Namen des Anwendungsfalls in das Eingabefeld *Anwendungsfall* ein.
- ▶ Wählen Sie das Stereotyp für den Anwendungsfall aus der Liste des Kombinationsfelds *Stereotyp* aus (z. B. *UseCase* für einen Systemanwendungsfall oder *Business* für einen Business-Anwendungsfall).
- ▶ Markieren Sie das Kontrollfeld *abstrakt*, wenn von diesem Anwendungsfall spezielle Anwendungsfälle abgeleitet werden.
- ▶ Im Eingabefeld *Notiz* können Sie Informationen eingeben, die den Anwendungsfall genauer spezifizieren.

Die angelegten Anwendungsfälle werden unmittelbar nach Eingabe unterhalb des entsprechenden Paketes angezeigt.

Änderung der Eigenschaften für Akteure und Anwendungsfälle

Möchten Sie die Eigenschaften eines Akteurs oder eines Anwendungsfalls verändern, gehen Sie wie folgt vor:

- ▶ Erweitern Sie im Fenster *Fachliches Modell* den Eintrag *Anwendungsfälle* oder *Akteure*.
- ▶ Klicken Sie doppelt auf den Namen des Akteurs oder Anwendungsfalls, dessen Eigenschaften Sie ändern wollen.
Das Dialogfenster *Eigenschaften* wird geöffnet.
- ▶ Nehmen Sie die Änderungen in den entsprechenden Eingabefeldern des Dialogfensters analog zu den Eingaben beim Anlegen eines Anwendungsfalls bzw. Akteurs vor (vgl. Abschnitt *Anwendungsfälle bzw. Akteure anlegen*).
- ▶ Bestätigen Sie die Angaben mit der Schaltfläche *OK*.



Beachten Sie, dass unbedachte Änderungen an den Einstellungen von Akteuren oder Anwendungsfällen zu Widersprüchen mit der bisherigen Modellierung führen können. Sie können gegebenenfalls das erstellte Modell zerstören.

Anlegen des Anwendungsfalldiagramms

Im Anwendungsfalldiagramm wird der Zusammenhang zwischen den Akteuren und den Anwendungsfällen untereinander und miteinander modelliert.

Sie können die Zusammenhänge in mehreren Diagrammen modellieren. Es ist nicht sinnvoll, alle Akteure und alle Anwendungsfälle in ein Diagramm einzuzeichnen.

- ▶ Wählen Sie aus dem Kontextmenü des gewünschten Paketes den Menüpunkt *Anlegen - Anwendungsfalldiagramm* aus.
Es wird ein Dialogfenster zur Eingabe eines neuen Anwendungsfalldiagramms geöffnet.
- ▶ Geben Sie einen Namen für den Anwendungsfall in das Eingabefeld *Anwendungsfalldiagramm* ein.
- ▶ Geben Sie Informationen zur Erläuterung des Anwendungsfalls, z. B. eine Abgrenzungsbeschreibung dieses Anwendungsfalls von allen weiteren, in das Eingabefeld *Notiz* ein.
- ▶ Markieren Sie das Kontrollfeld *Öffnen*, wenn Sie direkt nach der Erstellung den Anwendungsfall im Grafik-Editor modellieren wollen.
- ▶ Bestätigen Sie die Eingaben mit der Schaltfläche *OK*.

Einfügen von Anwendungsfällen in das Diagramm

Für das Einfügen von Anwendungsfällen in ein Anwendungsfalldiagramm stehen Ihnen zwei Möglichkeiten zur Verfügung:

- ✓ Einfügen eines Anwendungsfalls aus der Liste der bereits vorhandenen Anwendungsfälle,
- ✓ Einfügen eines neu zu erstellenden Anwendungsfalls.

Die zweite Lösung ist die aufwendigere und nimmt mehr Zeit in Anspruch als die erste Variante. Deshalb ist zu empfehlen, die zweite Variante nur für die neu hinzukommenden Anwendungsfälle zu verwenden.

- ▶ Klicken Sie doppelt auf den Namen des gewünschten Diagramms, um den Grafik-Editor mit dem Diagramm zu öffnen.
- ▶ Klicken Sie im linken Fensterbereich auf das Symbol  und anschließend in das Editor-Fenster.
Daraufhin wird ein Dialogfenster mit einer Liste der bereits erstellten Anwendungsfälle angezeigt.
- ▶ Markieren Sie den gewünschten Anwendungsfall und betätigen Sie die Schaltfläche *Einfügen*.
Das Symbol (Ellipse) für den Anwendungsfall wird in das Diagramm eingezeichnet.
- ▶ Fügen Sie gegebenenfalls weitere Anwendungsfälle in das Diagramm ein, indem Sie im Dialogfenster einen weiteren Anwendungsfall markieren und die Schaltfläche *Einfügen* betätigen.
Die Anwendungsfall-Symbole werden übereinander etwas versetzt im Diagramm angezeigt.
- ▶ Schließen Sie das Dialogfenster mit der Schaltfläche *Schließen*.

Wenn Sie mehrere Anwendungsfälle aus der Liste des Dialogfensters *Anwendungsfälle auswählen* auswählen, werden alle ausgewählten Anwendungsfälle in das Diagramm eingefügt. Betätigen Sie dazu die Tasten **Strg** oder , während Sie mit der Maus die Einträge markieren:

- ✓  – um mehrere einzelne in der Liste aufgeführte Anwendungsfälle zu markieren,
- ✓  – um in der Liste nacheinander aufgeführte Anwendungsfälle zu markieren.

Sie können auch einen Anwendungsfall neu erstellen und diesen dem Diagramm hinzufügen.

- ▶ Klicken Sie im linken Fensterbereich auf das Symbol  und anschließend in das Editor-Fenster.
Damit wird das Dialogfenster *Anwendungsfall anlegen* geöffnet.
- ▶ Legen Sie die Eigenschaften des Anwendungsfalls im Dialogfenster fest.
- ▶ Bestätigen Sie Ihre Eingaben mit der Schaltfläche *OK*.

Der neue Anwendungsfall wird nach dem Schließen des Dialogfensters im Diagramm angezeigt. Er wird automatisch in der Baumansicht des Fensters *Fachliches Modell* an der gleichen Stelle angezeigt, an der die zuvor erstellten Anwendungsfälle sichtbar waren.

Einfügen von Akteuren in das Diagramm

Akteure können wie die Anwendungsfälle in das Diagramm eingefügt werden. Die Schaltflächen ,  zum Einfügen von Akteuren finden Sie im linken Bereich des Grafik-Editors. Das Einfügen entspricht den Arbeitsschritten, wie sie für die Anwendungsfälle aufgeführt sind.

Einfügen von Notizen

Mithilfe einer Notiz können Sie den Elementen im Diagramm Informationen zuordnen. Dazu sind die folgenden Schritte notwendig:

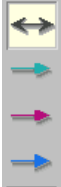
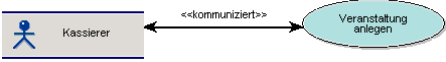
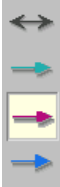
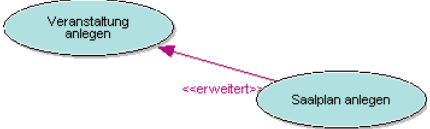
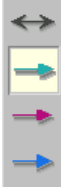
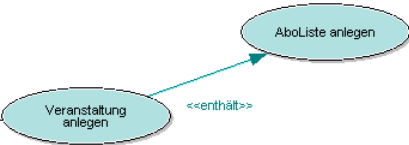
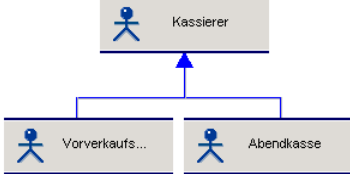
- ▶ Öffnen Sie das Anwendungsfalldiagramm im Grafik-Editor.
- ▶ Klicken Sie im linken Fensterbereich des Grafik-Editors auf die Schaltfläche .
- ▶ Bewegen Sie die Maus auf das Diagramm und betätigen Sie die linke Maustaste, um die Notiz im Diagramm abzulegen.
- ▶ Klicken Sie doppelt in das Rechteck der Notiz, um die Information dem Notizsymbol hinzuzufügen.



objectiF verfügt nicht über die Möglichkeit, Notizen durch eine gestrichelte Linie mit Elementen des Diagramms zu verbinden. Diese Möglichkeit ist jedoch in der UML-Spezifikation vorgesehen.

Beziehungen hinzufügen

Sie können in das Diagramm folgende Beziehungen einfügen. Diese können nur zwischen dargestellten Elementen angelegt werden.

Symbol	Beziehung	Symbol	Beziehung
	 Es wird modelliert, dass der Akteur Kassierer mit dem Anwendungsfall Veranstaltung anlegen in Verbindung steht (kommuniziert).		 Der Anwendungsfall Veranstaltung anlegen wird mit dem Anwendungsfall Saalplan anlegen erweitert, d. h., unter bestimmten Bedingungen wird innerhalb des Anwendungsfalls Veranstaltung anlegen der Anwendungsfall Saalplan anlegen abgearbeitet. Die Bedingung hat den Namen Erweiterungspunkt (engl. extension point)
	 Der Anwendungsfall Veranstaltung anlegen enthält den Anwendungsfall AboListe anlegen, d. h., er ist ein Teil des Anwendungsfalls (wie eine Art Unterprogramm).		 Die Generalisierung bedeutet, dass die Akteure Vorverkaufsstelle und Abendkasse Spezialisierungen des Akteurs Kassierer sind. Kassierer ist somit eine Rolle – die konkreten Akteure, die diese Rolle übernehmen, sind die Vorverkaufsstelle und die Abendkasse.

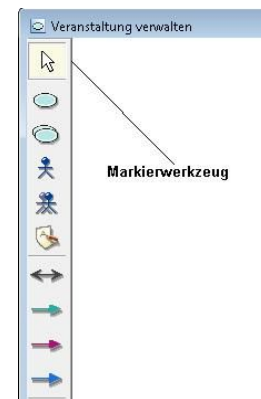
- ▶ Klicken Sie auf das gewünschte Symbol im linken Fensterbereich des Grafik-Editors.
- ▶ Klicken Sie im Diagramm ein Element an, das an der Beziehung beteiligt ist.
Das Symbol des Mauszeigers ändert sich.
- ▶ Ziehen Sie die Maus auf das zweite Element der Beziehung. Dabei wird eine bewegliche „elastische“ Linie gezeichnet.
- ▶ Klicken Sie auf dieses zweite Element, um die Beziehung anzulegen.

Über das Kontextmenü einer eingezeichneten Beziehung können Sie die Beschriftung aus- und einblenden. Die Beschriftungen können unabhängig vom Element im Diagramm platziert werden.

Anordnen von Akteuren, Anwendungsfällen und Notizen

Sie können im Grafik-Editor die Anordnung der Akteure, Anwendungsfälle und Notizen verändern. Die erstellten Beziehungen werden dabei nicht getrennt, sondern über „elastische“ Linien erhalten.

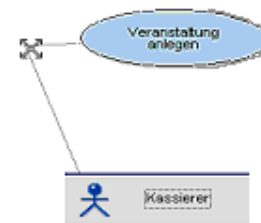
- ▶ Wählen Sie im linken Fensterbereich des Grafik-Editors das Markierungswerkzeug durch Klicken auf die Schaltfläche  aus.
- ▶ Verschieben Sie die Maus über das Diagrammelement, das Sie verschieben möchten.
- ▶ Ziehen Sie das Element bei gedrückter linker Maustaste an die gewünschte Position.
- ▶ Legen Sie das Element im Diagramm ab.



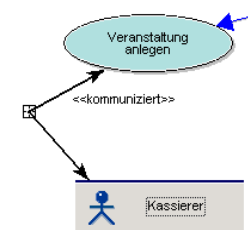
Anordnen von Beziehungen

Die grafische Anordnung der erstellten Beziehungen kann geändert werden. Dabei werden durch die „elastischen“ Beziehungslinien die neue Position und der grafische Verlauf der Beziehung sichtbar.

- ▶ Wählen Sie das Markierungswerkzeug aus.
- ▶ Verschieben Sie die Maus auf das Symbol der Beziehung, bis sich der Mauszeiger in ein Zeichen  oder  verwandelt.
- ▶ Ziehen Sie die Beziehungslinie (linke Maustaste gedrückt halten) in die neue Position.
- ▶ Geben Sie die Maustaste frei, um das Symbol der Beziehung abzulegen.



Die Stelle auf dem Symbol der Beziehung, die Sie mit der Maus ausgewählt haben, nimmt die Form eines Eckpunktes an, den Sie verschieben können. Um den gewünschten Verlauf und den gewünschten Ansatzpunkt an den Ellipsen der Anwendungsfälle und an den Rechtecken der Akteure zu erhalten, ist eine mehrfache Anwendung des Werkzeuges erforderlich.



objectiF kann die Symbole der Beziehungen automatisch anordnen.

- ▶ Wählen Sie aus dem Kontextmenü des Diagramms den Menüpunkt *Beziehungen anordnen* aus.
Danach werden alle Beziehungen von objectiF automatisch angeordnet.

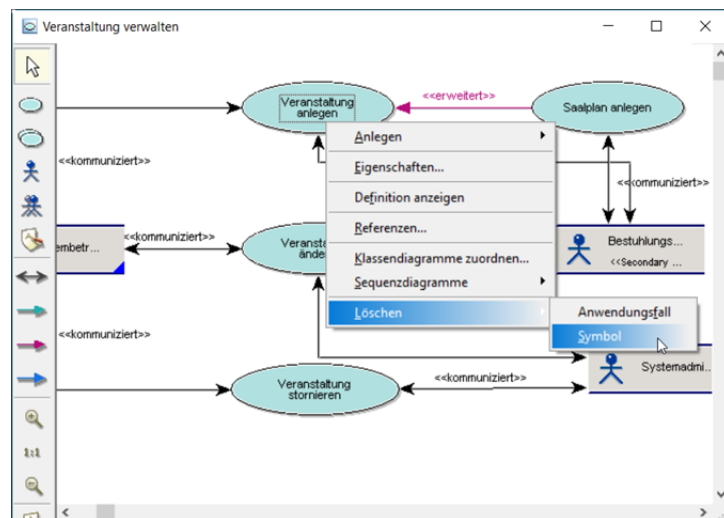
Löschen von Elementen

Über das Kontextmenü des zugehörigen grafischen Objekts kann es aus dem Diagramm gelöscht werden. Dabei stehen Ihnen zwei Optionen zur Auswahl:

- ✓ Löschen des grafischen Objekts aus dem Diagramm,
- ✓ Löschen der Modellierung des Elements.

Während die erste Möglichkeit nur das Symbol aus dem Diagramm entfernt, das eigentliche Element aber in seinen Beziehungen erhalten bleibt, werden mit der zweiten Möglichkeit das Element und alle seine Beziehungen aus dem Modell entfernt.

- ▶ Verschieben Sie den Mauszeiger auf ein Element des Diagramms.
- ▶ Betätigen Sie die rechte Maustaste, um das Kontextmenü des Elementes anzuzeigen.
- ▶ Wählen Sie den Kontextmenüpunkt *Löschen*.
- ▶ Klicken Sie auf den Untermenüpunkt *Symbol*, um nur das Symbol und nicht das Element zu löschen.



oder Markieren Sie den Untermenüpunkt, der den Namen der Elementkategorie des Elements enthält (z. B. Anwendungsfall), um das Element selbst und alle Beziehungen zu diesem zu löschen. (Das Element wird auch aus der Liste des Paketes entfernt.)

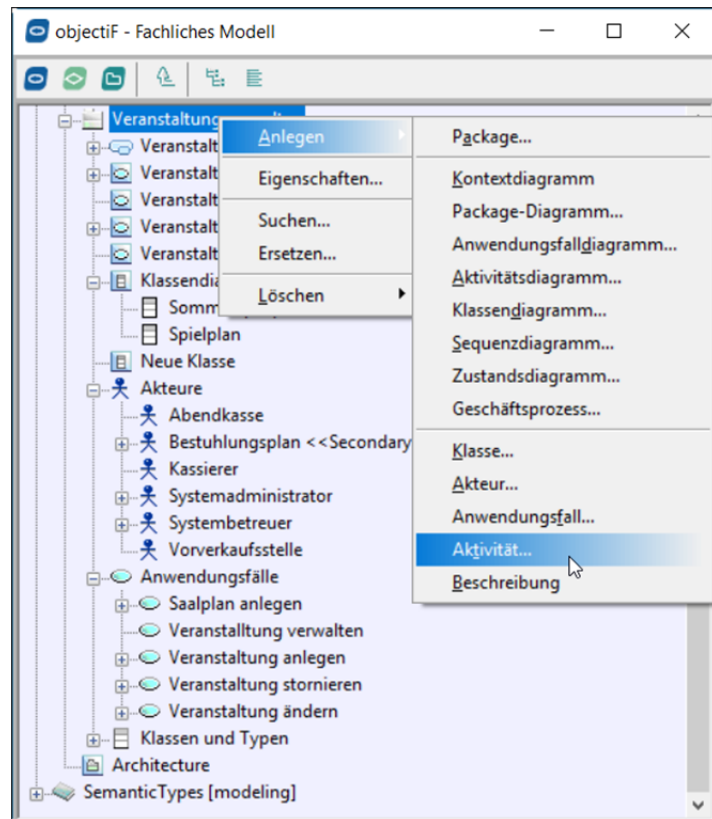
Über den Menüpunkt *Eigenschaften* des Kontextmenüs eines Elements im Anwendungsfalldiagramm können Sie ein Dialogfenster zur Bearbeitung der Eigenschaften anzeigen; der Kontextmenüpunkt *Referenzen* öffnet ein Fenster, in dem in einer Liste alle Diagrammnamen angezeigt werden, in denen das Element enthalten ist. Sie können durch Auswahl eines Listeneintrages zu diesem Diagramm wechseln.

10.4 Aktivitäten modellieren

Nachdem Sie die Anwendungsfälle mit ihren Beziehungen modelliert haben, können Sie in den Aktivitätsdiagrammen den Ablauf eines Anwendungsfalls modellieren. Legen Sie dazu die Aktivitäten an, die innerhalb des jeweiligen Anwendungsfalls abgearbeitet werden.

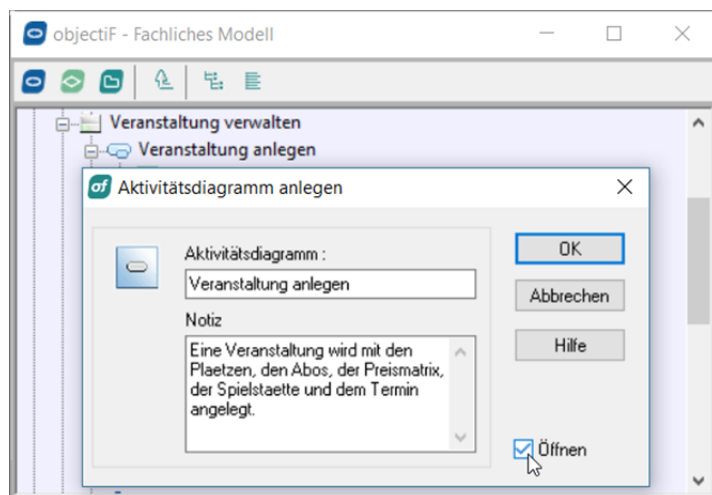
Anlegen von Aktivitäten

- ▶ Wählen Sie im Fenster *Fachliches Modell* aus dem Kontextmenü des gewünschten Paketes den Menüpunkt *Anlegen - Aktivität* aus.
Es wird ein Dialogfenster zur Eingabe einer neuen Aktivität geöffnet.
- ▶ Geben Sie einen Namen für die Aktivität in das Eingabefeld *Aktivität* ein.
- ▶ Erstellen Sie eine Information im Eingabefeld des Registers *Notiz*, um die Aufgabe der Aktivität zu beschreiben.
- ▶ Bestätigen Sie Ihre Eingaben mit der Schaltfläche *OK*.
Das Dialogfenster wird geschlossen und die Aktivität wird angelegt.



Erstellen eines Aktivitätsdiagramms

- ▶ Wählen Sie im Fenster *Fachliches Modell* aus dem Kontextmenü der gewünschten Aktivität den Menüpunkt *Anlegen - Aktivitätsdiagramm* aus.
Es wird ein Dialogfenster zur Eingabe eines neuen Aktivitätsdiagramms geöffnet.
- ▶ Geben Sie einen Namen für das Aktivitätsdiagramm in das Eingabefeld *Aktivitätsdiagramm* ein.
- ▶ Markieren Sie das Kontrollfeld *Öffnen*, um das Grafik-Editor-Fenster mit dem Diagramm sofort nach dem Anlegen zu öffnen.
- ▶ Geben Sie in das Eingabefeld *Notiz* eine Information über den Inhalt des neu erstellten Diagramms ein.



Erstellen Sie ein Aktivitätsdiagramm, bevor die zugehörige Aktivität angelegt wurde, wird diese automatisch – ohne Beschreibung – erzeugt.

Ablaufmodellierung

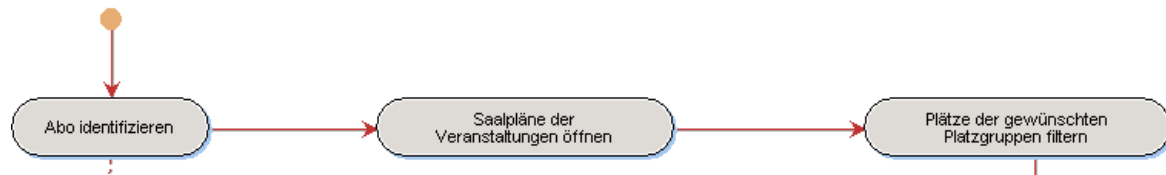
Das Grafik-Editor-Fenster, in dem die Aktivitätsdiagramme bearbeitet werden, enthält im linken Fensterbereich eine Werkzeugleiste mit Modellierungselementen.

Symbol	Beschreibung
	Dieses Werkzeug legt im Diagramm an der Position des Mauszeigers über das Dialogfenster <i>Eigenschaften</i> eine neue Aktivität an.
	Mit diesem Werkzeug können Sie eine Aktivität, die bereits angelegt wurde, in das Diagramm einfügen. Klicken Sie dazu in das Diagramm. Es wird ein Dialogfenster mit einer Liste der bereits angelegten Aktivitäten angezeigt. Wählen Sie die gewünschte Aktivität aus.
	Mit diesem Werkzeug fügen Sie den Startpunkt des Programmflusses in das Diagramm ein.
	Der Programmfluss endet bei diesem Symbol.
	Wird der Programmfluss in mehrere Wege aufgetrennt oder werden mehrere bereits aufgetrennte Wege wieder zusammengeführt, wählen Sie dieses Werkzeug aus und fügen Sie das Symbol in das Diagramm ein.
	Das Symbol für das Splitten und das Synchronisieren von gleichzeitig ablaufenden Programmsträngen können Sie mit diesem Werkzeug dem Diagramm hinzufügen.
	Dieses Werkzeug zeichnet den Programmfluss in Form einer Linie von Aktivität zu Aktivität ein. Die Linie erhält an einem Ende einen Pfeil, um die Richtung des Programmflusses darzustellen.
	In einem Aktivitätsdiagramm können Sie den Objektfluss einarbeiten. Dieses Werkzeug erstellt die Objektsymbole und die gestrichelt gezeichneten Objektflusslinien. Der Objektzustand kann über das Kontextmenü des Objektsymbols eingetragen werden.
	Um den Programmfluss zwischen den Aktivitäten mehrerer Anwendungsfälle modellieren zu können und dennoch eine Abgrenzung der Aktivitäten zu erreichen, können Sie sogenannte SwimLanes (Schwimmbahnen) anlegen. Dies sind senkrechte Linien zur Trennung der Aktivitäten unterschiedlicher Anwendungsfälle.
	Dieses Werkzeug erstellt eine Notiz.

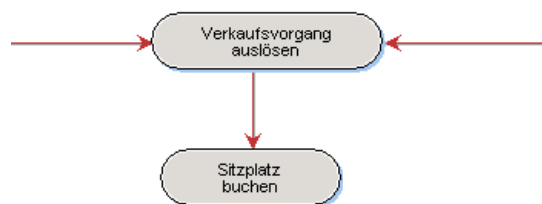
Das Verfahren zum Einfügen von Symbolen in den Grafik-Editor des geöffneten Diagramms ist in allen Diagrammen gleichermaßen zu handhaben (vgl. Abschnitt *Anwendungsfälle*).

Anordnen der Aktivitäten und Verbindungslinien

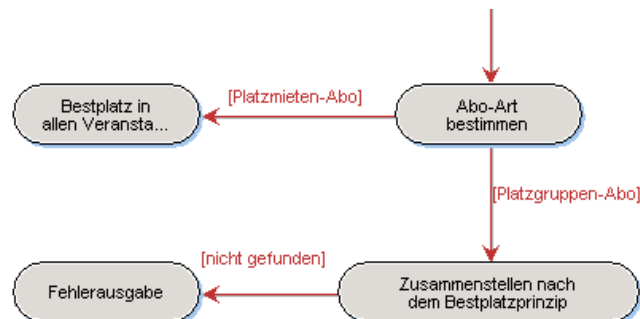
Mit dem Werkzeug für die Verbindungslinien können Sie die Elemente in der Reihenfolge der Abarbeitung miteinander verbinden. Dabei wird zu jeder Aktivität eine Linie mit Pfeilrichtung auf die Aktivität und eine Linie, die von der Aktivität ausgeht, erstellt.



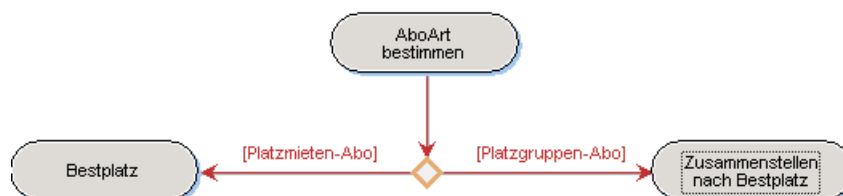
Wenn der Programmfluss zusammengeführt wird, sind mindestens drei Verbindungslinien (zwei, deren Pfeil auf die Aktivität zeigt, und eine, die von der Aktivität ausgeht) einzuzeichnen.



Gehen mehrere Verbindungslinien von einem Anwendungsfall weg, findet eine Aufteilung der Programmwege statt. Den Verbindungslinien können Sie als Beschreibung die Bedingung der Verzweigung hinzufügen.



Die Zusammenführung und das Aufspalten der Programmwege können Sie auch über das Symbol  realisieren.



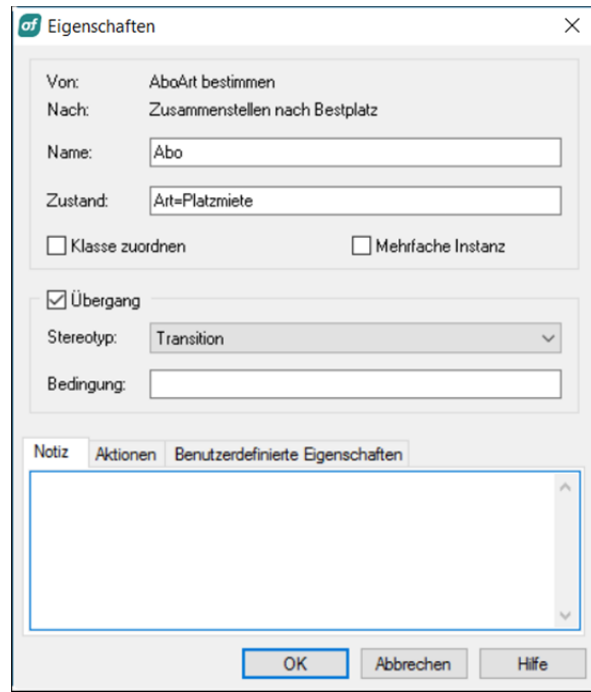
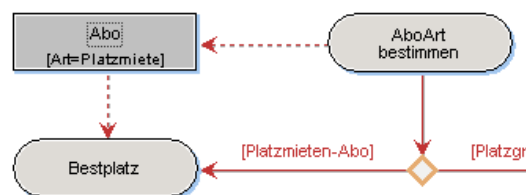
Objektfluss in das Aktivitätsdiagramm einzeichnen

Sie können in einem Aktivitätsdiagramm den Objektfluss in Abhängigkeit vom Programmfluss modellieren. Dabei ändert ein Objekt seinen Zustand, wenn der Programmfluss von einer zur nächsten Aktivität weitergeleitet wird.

- ▶ Klicken Sie im linken Teil des Grafik-Editors auf das Werkzeug .
- ▶ Verschieben Sie den Mauszeiger auf eine Aktivität und klicken Sie mit der linken Maustaste.
- ▶ Verschieben Sie den Mauszeiger auf die Aktivität, bei der der neue Zustand des Objekts zum Beginn der Aktivität erreicht ist.
- ▶ Klicken Sie das Symbol dieser Aktivität an. Es wird das Dialogfenster *Eigenschaften* geöffnet.
- ▶ Geben Sie einen Namen für das Objekt in das Eingabefeld *Name* ein.
- ▶ Beschreiben Sie den Zustand des Objekts, indem Sie einem Attribut (z. B. Art) einen Wert (z. B. Platzmiete) zuweisen.
- ▶ Markieren Sie das Kontrollfeld *Klasse zuordnen* und gegebenenfalls *Mehrfache Instanz*, wenn Sie eine Klasse auswählen möchten, von der das Objekt erstellt wurde.
- ▶ Bestätigen Sie Ihre Eingaben mit der Schaltfläche *OK*.

Sie können das Kontrollfeld *Übergang* deaktivieren, wenn Sie keine Bedingung für den Zustandswechsel festlegen möchten.

Das Dialogfenster wird geschlossen und der Objektfluss wird in das Diagramm eingezeichnet.

Löschen von Diagrammelementen

Das Löschen erfolgt über das Kontextmenü des Elements im Diagramm. Sie haben die Möglichkeit, das Element aus der Modellierung zu löschen oder nur das grafische Symbol aus dem Diagramm zu entfernen. Im Modell bleibt in diesem Fall das Element jedoch erhalten (vgl. Abschnitt *Anwendungsfälle - Löschen von Elementen*).

10.5 Klassendiagramme

Mit der Erstellung von Klassendiagrammen erfolgt der Schritt von der Analyse zum Design der Anwendung. Sie modellieren die Datenstruktur, die in Klassen und deren Beziehungen besteht und den realen Prozess abbildet. Für die Erstellung der Klassendiagramme stehen Ihnen folgende Möglichkeiten zur Verfügung:

- ✓ Sie erstellen selbst die Klassen und verändern so das bereits erarbeitete Modell aus der Sichtweise des Entwurfs.
- ✓ Sie verwenden Entwurfsmuster, in deren Klassen Sie die bereits in der Analysephase modellierten Prozesse einbetten.

Beide Methoden können auch kombiniert eingesetzt werden; objectiF unterstützt beide Varianten. Zunächst wird auf die erste eingegangen.

Anlegen eines Klassendiagramms

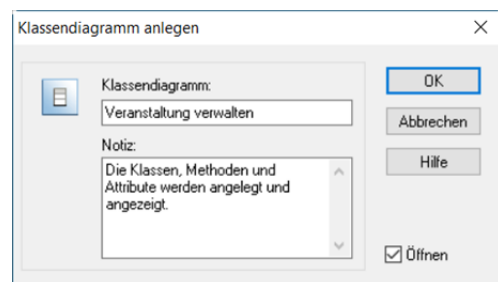
Wie bei den Diagrammen der Analyse können Sie auch bei diesem Entwicklungsschritt die bereits gefundenen Klassen anlegen, bevor Sie das Diagramm erstellen. Empfehlenswert ist die Erstellung der Klassen im Diagramm. Um die Übersicht zu erhalten, können die Klassen mit ihren Methoden und Attributen in einem Diagramm erstellt werden. In einem zweiten Diagramm können Sie dann mit diesen Klassen die Beziehungen der Klassen untereinander modellieren.

Mit folgenden Schritten legen Sie ein neues Klassendiagramm an:

- ▶ Wählen Sie aus dem Kontextmenü des gewünschten Paketes den Menüpunkt *Anlegen - Klassendiagramm* aus.

Es wird ein Dialogfenster zur Eingabe eines neuen Klassendiagramms geöffnet.

- ▶ Geben Sie einen Namen für das Klassendiagramm ein.
- ▶ Möchten Sie Informationen, die den Inhalt des Diagramms näher erläutern, hinzufügen, können Sie in das Eingabefeld *Notiz* entsprechende Angaben einfügen.
- ▶ Markieren Sie das Kontrollfeld *Öffnen*, damit der Grafik-Editor mit dem Klassendiagramm sofort nach der Erstellung geöffnet wird.
- ▶ Bestätigen Sie die Ausführung der Erstellung mit der Schaltfläche *OK*.

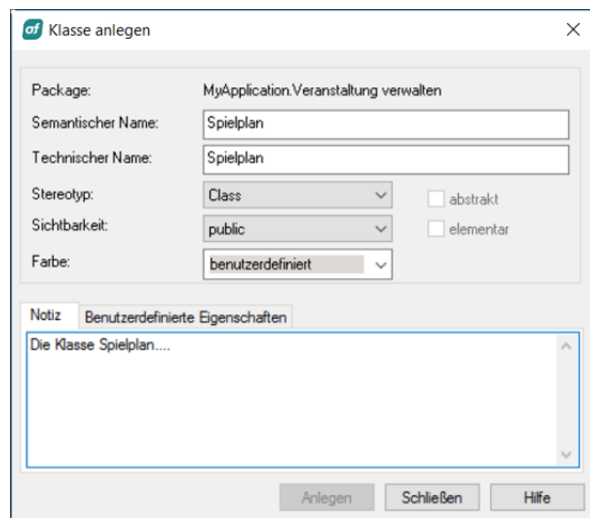


Anlegen von Klassen

Die ersten Klassen des Entwurfs sind oft bereits in den Anwendungsfällen zu finden. Kandidaten sind die Substantive, welche Sie in den Schritten der essenziellen Anwendungsfallbeschreibungen finden, wenn Sie in der Analyse die Beschreibungen angelegt haben.

Sie legen dann die gefundenen Klassen über das im Vorfeld angelegte Klassendiagramm mit folgenden Schritten an:

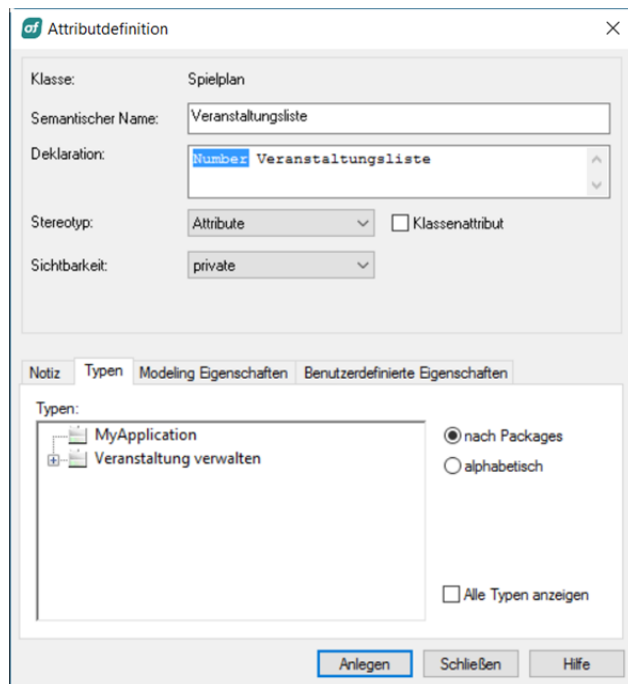
- ▶ Klicken Sie auf das Symbol  im linken Bereich des Grafik-Editors.
- ▶ Bewegen Sie den Mauszeiger auf die Stelle im Klassendiagramm, an der Sie das Klassensymbol anlegen möchten.
- ▶ Klicken Sie mit der linken Maustaste, um das Symbol abzulegen.
Es öffnet sich ein Dialogfenster zum Anlegen einer neuen Klasse.
- ▶ Geben Sie in das Eingabefeld *Semantischer Name* den Klassennamen ein, der in den Diagrammen angezeigt wird. Sie können in diesem Feld auch deutsche Umlaute verwenden.
- ▶ Ändern Sie gegebenenfalls den vom System generierten technischen Namen, der aus dem semantischen Namen abgeleitet und somit ein für die Programmiersprache geeigneter Name ist.
- ▶ Markieren Sie das Kontrollfeld *abstrakt*, wenn Sie eine abstrakte Klasse erstellen möchten. Das System überwacht in diesem Fall, dass Sie von dieser Klasse keine Objekte erstellen.
- ▶ Wählen Sie das Attribut *public*, um den Aufruf von Methoden der Klasse für andere Pakete zu ermöglichen.
- ▶ Möchten Sie Informationen, die die Funktionalität der Klasse näher erläutern, hinzufügen, können Sie in das Eingabefeld *Notiz* entsprechende Angaben einfügen.



Attribute hinzufügen

Nach dem Erstellen der Klasse im Klassendiagramm können Sie der Klasse Attribute zuordnen.

- ▶ Wählen Sie aus dem Kontextmenü des Klassensymbols den Menüpunkt *Aufklappen* (dieser Punkt ist nur im Menü enthalten, wenn das Symbol noch nicht aufgeklappt ist).
Das Klassensymbol wird erweitert und zwei zusätzliche Bereiche für Methoden und Attribute werden angezeigt.
- ▶ Klicken Sie doppelt in den oberen der zwei neuen Bereiche. Das Dialogfenster *Attributdefinition* wird angezeigt.
- ▶ Geben Sie in das Eingabefeld *Semantischer Name* einen Namen für das Attribut ein. Sie können in diesem Feld auch deutsche Umlaute verwenden.
- ▶ Ändern Sie gegebenenfalls den Namen im Eingabefeld *Deklaration*, der vom System aus dem semantischen Namen abgeleitet wurde und einen für die installierte Anwendungsversion zugelassenen Namen hat. Dabei wird z. B. der Datentyp `number` per Vorgabe an das Attribut vergeben.
- ▶ Wählen Sie in dem Kombinationsfeld *Sichtbarkeit* den Sichtbarkeitsbereich des Attributes aus, in dem es in der Klassendefinition erstellt wird.
- ▶ Markieren Sie gegebenenfalls das Kontrollfeld *Klassenattribut*, um ein statisches Attribut zu erstellen.
- ▶ Bestätigen Sie Ihre Angaben mit der Schaltfläche *Anlegen*.
oder Markieren Sie den Datentyp im Eingabefeld *Deklaration*. Wählen Sie einen Datentyp für das Attribut über die Baumansicht des Registers *Typen* aus.
- ▶ Bestätigen Sie Ihre Angaben mit der Schaltfläche *Anlegen*.
- ▶ Schließen Sie das Dialogfenster mit der Schaltfläche *Schließen*.

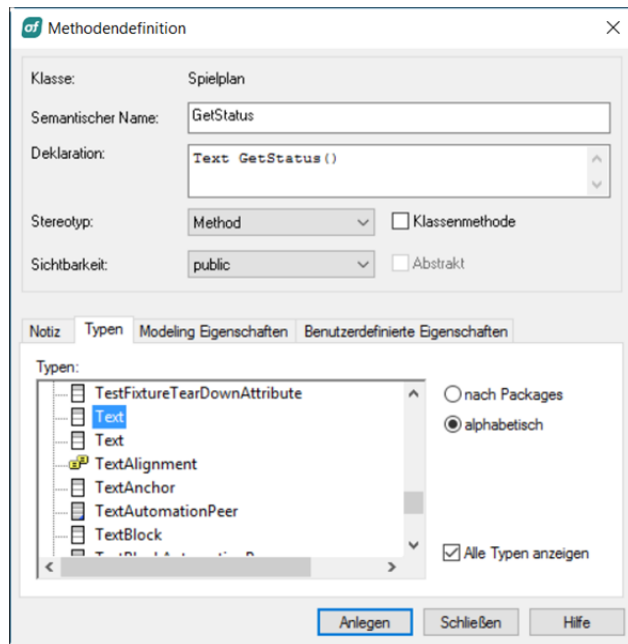


Möchten Sie alle verfügbaren Typen in der Baumansicht des Registers *Typen* anzeigen, markieren Sie das Kontrollfeld *Alle Typen anzeigen*. Durch die Markierung des Optionsfeldes *alphabetisch* können Sie einen Typ nach dem Namen in der alphabetisch geordneten Baumansicht finden.

Operationen hinzufügen

Neben den Methoden zum Setzen und Lesen eines Attributwertes können Sie der Klasse weitere Methoden hinzufügen.

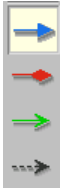
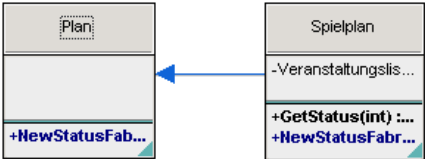
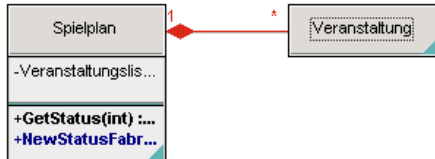
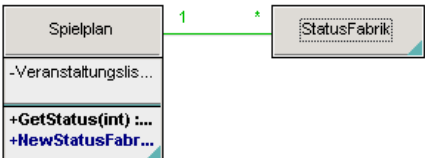
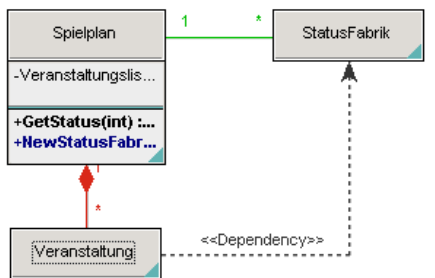
- ▶ Wählen Sie den Menüpunkt *Aufklappen* im Kontextmenü des Klassensymbols im Klassendiagramm (dieser Punkt ist nur im Menü enthalten, wenn das Symbol noch nicht aufgeklappt ist).
- ▶ Klicken Sie doppelt mit der linken Maustaste in den unteren Bereich des Symbols der Klasse. Das Dialogfenster *Methodendefinition* wird angezeigt.
- ▶ Geben Sie in das Eingabefeld *Semantischer Name* einen Namen für die Methode ein. Das System erstellt für Ihre Methode aus diesem Namen den Namen für die Deklaration.
- ▶ Ändern Sie gegebenenfalls den Datentyp des Rückgabewertes der Methode und die Datentypen der Parameter. Setzen Sie den Cursor dazu in die Parameterliste im Eingabefeld *Deklaration*.
- ▶ Klicken Sie doppelt auf den Namen des gewünschten Typs im Register *Typen*. Der Typ wird im Eingabefeld *Deklaration* eingefügt.
- ▶ Wiederholen Sie das Einfügen der Typangaben für jeden Parameter und den Rückgabewert der Methode.
- ▶ Markieren Sie das Kontrollfeld *Klassenmethode*, um eine statische Methode zu erstellen.
- ▶ Markieren Sie das Kontrollfeld *Abstrakt*, um eine abstrakte Methode zu erstellen.
- ▶ Wählen Sie aus dem Kombinationsfeld *Sichtbarkeit* den Bereich der Klasse aus, in dem die Methode deklariert werden soll.
- ▶ Legen Sie durch Auswahl der Kontrollfelder der Liste *Modeling Eigenschaften* die Implementierungsattribute für die Methode fest (z. B. *virtual*).
- ▶ Bestätigen Sie Ihre Eingaben über die Schaltfläche *Anlegen*.
- ▶ Schließen Sie das Dialogfenster *Methodendefinition* mit der Schaltfläche *Schließen*.



Klassenbeziehungen

Wenn Sie Klassenbeziehungen modellieren wollen, können Sie ein weiteres Klassendiagramm mit diesen Klassen anlegen oder in das bereits angelegte Klassendiagramm die Beziehungen einzeichnen. Sie können für die modellierten Beziehungen über das System die Attribute in den Klassendefinitionen anlegen lassen, die für die Beziehungen notwendig sind.

Wenn das System die Attribute für die Beziehung anlegt, werden diese auch vom System automatisch auf dem aktuellen Stand gehalten.

Symbol	Beziehung	Symbol	Beziehung
	 Die dargestellte Generalisierung bedeutet, dass die Klasse Spielplan von der Klasse Plan abgeleitet wird. Weiterhin finden Sie in der abgeleiteten Klasse die Methode NewStatusFabrik der Basisklasse Plan wieder, die in der abgeleiteten Klasse Spielplan überschrieben wird.		 Die Klasse Spielplan und die Klasse Veranstaltung sind durch eine Aggregation verbunden. Durch die Kardinalitätsangabe * und das Rhombussymbol wird ausgedrückt, dass ein Objekt der Klasse Spielplan ein Container ist, der beliebig viele Objekte der Klasse Veranstaltung aufnehmen kann. Die Kardinalitätsangabe 1 bedeutet, dass jede Veranstaltung zu genau einem Spielplan gehört (verwaltet wird).
	 Die Klasse Spielplan kommuniziert mit der Klasse StatusFabrik. Die bestehende Assoziation sorgt dafür, dass Objekte beider Klassen die Schnittstellen-Methoden der jeweils anderen Klasse aufrufen können.		 Die Klasse Veranstaltung ist abhängig von der Klasse StatusFabrik; d. h., unterschiedlichen Objekten der Klasse Veranstaltung können unterschiedliche Objekte der Klasse StatusFabrik zugeordnet sein.

Zum Anlegen einer Beziehung zwischen zwei Klassen gehen Sie wie folgt vor:

- ▶ Wählen Sie ein Beziehungswerkzeug auf der linken Seite des Grafik-Editors.
- ▶ Klicken Sie auf das Klassensymbol der ersten Klasse. Der Mauszeiger ändert sich.
- ▶ Klicken Sie auf das Klassensymbol der zweiten Klasse. Das Symbol für die Beziehung wird im Diagramm angezeigt.

Wenn Sie eine Generalisierung anlegen wollen, klicken Sie zuerst auf die Basisklasse. Wenn Sie eine Kombination anlegen wollen, klicken Sie zuerst auf die Container-Klasse. Bei einer Assoziation wird keine Navigationsrichtung mit der Reihenfolge angelegt. Diese können Sie über die Eigenschaften der Assoziation konfigurieren.

Die UML kennt zur Aggregation noch eine strengere Form, die Komposition. Die Beziehung, die Sie im Klassendiagramm eintragen, ist zunächst eine Komposition, bei der die Container-Klasse nur mit den eingefügten Objekten existiert.

Möchten Sie eine Aggregation modellieren, gehen Sie wie folgt vor:

- ▶ Wählen Sie auf der Werkzeugleiste das Markierungswerkzeug  aus.
- ▶ Klicken Sie doppelt auf das Symbol der Komposition, die beide Klassen verbindet.
Es wird das Dialogfenster *Eigenschaften* geöffnet.
- ▶ Entfernen Sie die Markierung im Kontrollfeld *Komposition*.
- ▶ Bestätigen Sie die Angaben über die Schaltfläche *OK*.

Für eine Aggregation wird ein ähnliches Symbol wie für die Komposition verwendet. Der Unterschied besteht darin, dass der Rhombus nicht ausgefüllt wird.



Automatisches Anlegen der Beziehungsattribute

Mit den folgenden Schritten können Sie die Beziehungsattribute für die verbundenen Klassen automatisch vom System anlegen lassen.

- ▶ Klicken Sie mit dem Markierungswerkzeug doppelt auf das Symbol der Beziehung.
Das Dialogfenster *Eigenschaften* wird geöffnet.
- ▶ Wählen Sie das Register *Navigierbarkeit* aus.
- ▶ Markieren Sie die Kontrollfelder *Navigierbar* und *Automatische Attributbehandlung*.
- ▶ Schließen Sie das Dialogfenster *Eigenschaften* mit der Schaltfläche *OK*.

Es wird für die Klasse ein Attribut entsprechend der modellierten Assoziation erstellt und automatisch bei einer Änderung der Assoziation aktualisiert.

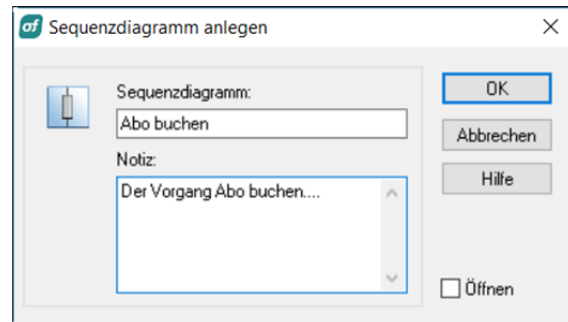
Die Navigierbarkeit ist die Richtung der Beziehung zweier Klassen, d. h., sie gibt an, welche Klasse die zweite an der Beziehung beteiligte Klasse referenziert. Dabei wird berücksichtigt, von welcher Klasse aus hauptsächlich Botschaften gesendet werden. Diese Klasse enthält ein Attribut, dessen Wert das zweite Objekt referenziert (einen Zeiger auf das zweite Objekt speichert). Die Navigation ist von diesem Objekt in Richtung des zweiten Objekts gerichtet (gerichtete Assoziation). Die Richtung wird mit einem Pfeil an dem Beziehungssymbol gekennzeichnet.

10.6 Sequenzdiagramme

- ▶ Wählen Sie im Fenster *Fachliches Modell* aus dem Kontextmenü des gewünschten Paketes den Menüpunkt *Anlegen - Sequenzdiagramm* aus.

Es wird ein Dialogfenster zur Anlage eines neuen Sequenzdiagramms geöffnet.

- ▶ Geben Sie einen Namen für das Diagramm in das Eingabefeld *Sequenzdiagramm* ein.
- ▶ Füllen Sie das Eingabefeld *Notiz* mit Informationen, die das Modell im Sequenzdiagramm genauer charakterisieren.
- ▶ Markieren Sie das Kontrollfeld *Öffnen*, wenn der Grafik-Editor mit dem neu erstellten Diagramm nach dem Schließen dieses Dialogfensters geöffnet werden soll.
- ▶ Schließen Sie das Dialogfenster mit der Schaltfläche *OK*.



Das Grafik-Editor-Fenster, in dem die Sequenzdiagramme bearbeitet werden, enthält im linken Fensterbereich eine Werkzeugleiste mit Modellierungselementen.

Symbol	Beschreibung
	Mit diesem Werkzeug können Sie die Elemente des Sequenzdiagramms markieren. Im Kontextmenü der Elemente können Sie über den Menüpunkt <i>Eigenschaften</i> die Elemente konfigurieren.
	Mit dem Objektwerkzeug können Sie Objekte in das Diagramm einfügen. Die Objekte erhalten Lebenslinien, die in der Farbe Grau dargestellt werden, wenn die Klasse, über die das Objekt erstellt wurde, im Modell noch nicht existiert.
	Sie können mit diesem Werkzeug eine Notiz in das Diagramm einfügen und darin Informationen zur genaueren Beschreibung eines Modellierungsdetails angeben.
	Mit diesem Werkzeug fügen Sie eine Systemgrenze in das Diagramm ein. Von diesem Element kann der erste Aufruf einer Operation erfolgen (Auslöser). Die Systemgrenze kann auch Botschaften empfangen.
	Das Botschaftswerkzeug ermöglicht, die Kommunikation zweier Objekte zu modellieren. Sie können mit diesem Werkzeug eine Botschaft in das Diagramm einfügen. Diese Botschaft kann eine bereits angelegte Methode der Klasse aufrufen, wodurch das Objekt der Klasse auf die Botschaft reagiert.
	Sendet ein Objekt einem zweiten Objekt eine Botschaft und wird mit dieser Botschaft das zweite Objekt erzeugt (z. B. mit der Methode <code>new</code>), verwenden Sie dieses Werkzeug. Die Lebenslinie des erzeugten Objekts wird bis zum Empfang dieser Botschaft gestrichelt gezeichnet.
	Sendet ein Objekt einem zweiten Objekt eine Botschaft und wird mit dieser Botschaft das zweite Objekt zerstört (z. B. mit der Methode <code>delete</code>), verwenden Sie zur Erstellung dieser Botschaft dieses Werkzeug.

10.7 Zustandsdiagramme

Sie können für eine Klasse ein Zustandsdiagramm erstellen. Dieses Diagramm wird über das Kontextmenü der Klasse erstellt.

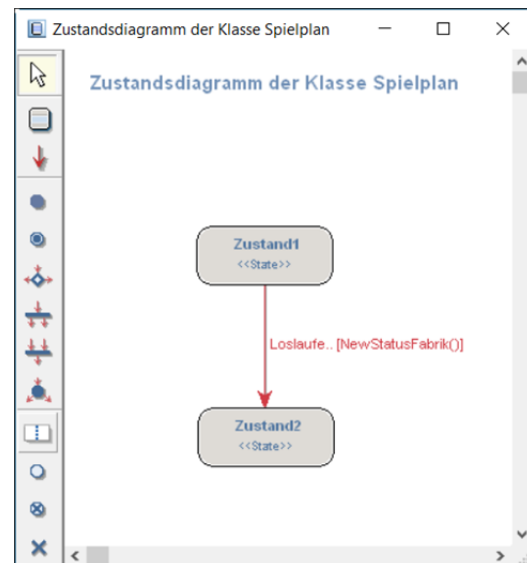
- ▶ Wählen Sie aus dem Kontextmenü einer Klasse den Menüpunkt *Anlegen - Zustandsdiagramm* aus.

Nach dem Erstellen des Diagramms wird der Grafik-Editor mit diesem Diagramm geöffnet. Er enthält im linken Fensterbereich eine Werkzeugleiste mit Modellierungselementen, von denen Sie u. a. folgende in das Diagramm einfügen können:

Symbol	Beschreibung
	Mit diesem Werkzeug erstellen Sie einen Zustand eines Objekts der Klasse, für die das Zustandsdiagramm angelegt wurde.
	Wählen Sie dieses Werkzeug, um einen Zustandsübergang zu modellieren.
	Fügen Sie dem Diagramm eine Notiz hinzu, um Informationen zu Modellierungsdetails anzugeben.

Zustand anlegen

- ▶ Wählen Sie das Werkzeug zur Erstellung eines Zustandes aus dem linken Bereich des Grafik-Editors aus.
- ▶ Klicken Sie auf die gewünschte Position im Diagramm, an der das Zustandssymbol angezeigt werden soll.
- ▶ Geben Sie in das Eingabefeld *Name* des Dialogfeldes *Zustand anlegen* einen Namen für den Zustand ein.
- ▶ Bestätigen Sie Ihre Eingaben über die Schaltfläche *OK*.

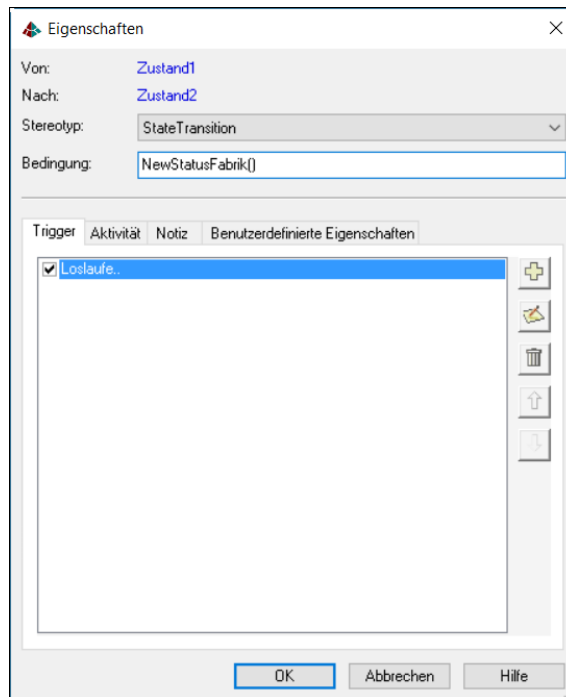


Zustandsübergang anlegen und konfigurieren

- ▶ Wählen Sie das Werkzeug zur Erstellung eines Zustandsübergangs aus dem linken Bereich des Grafik-Editors aus.
- ▶ Klicken Sie auf das Symbol eines im Diagramm angelegten Zustands, von dem aus der Übergang erfolgen soll.
- ▶ Verschieben Sie den Mauszeiger auf das Symbol des Zustands, der nach dem Übergang erreicht wird.
- ▶ Klicken Sie auf das Symbol des Endzustands, um den Zustandsübergang anzulegen; der Zustandsübergang wird als Linie erstellt.

- ▶ Wählen Sie aus dem Kontextmenü des erstellten Zustandsübergangs den Menüpunkt *Eigenschaften*.
- ▶ Geben Sie die für den Zustandsübergang relevanten Bedingungen, Trigger bzw. Aktivitäten vor bzw. wählen Sie diese aus bereits vorhandenen aus.
- ▶ Schließen Sie das Dialogfenster *Eigenschaften* nach der Eingabe mit der Schaltfläche *OK*.

Der Übergang wurde als Linie zwischen den zwei betroffenen Zuständen angelegt und mit einem Pfeil versehen, der in Richtung des Endzustands weist. Neben dem Übergangspfeil wird der Name der auslösenden Bedingung bzw. Methode angegeben.

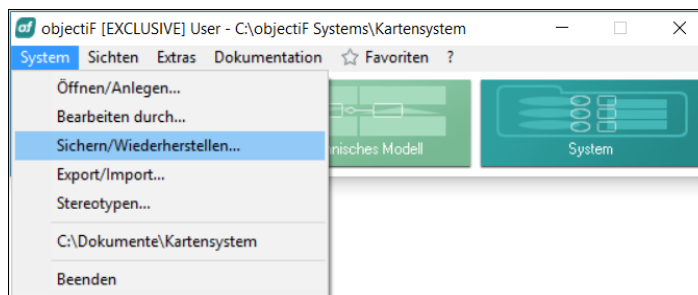


10.8 Speichern

Alle Gestaltungselemente werden sofort nach dem Anlegen automatisch gespeichert. In keinem der Diagramme müssen Sie die Speicherung explizit auslösen. Sie finden die angelegten Dateien im Verzeichnis `C:\objectiF Systems\<Ihr Projektname>`. Wenn Sie bei Projekterstellung ein anderes Verzeichnis angegeben haben, finden Sie das Projektverzeichnis in dem von Ihnen angegebenen Verzeichnis. Das Verzeichnis enthält mehrere Unterordner.

Ordnername	Inhalt
<i>workspace</i>	In diesem Ordner sind mehrere Unterordner vorhanden, in denen u. a. die Quellcode-Dateien gespeichert sind, die objectiF für Sie aus den modellierten Klassen Ihres Projektes generiert hat. Zusätzlich sind die für die Weiterbearbeitung in der jeweiligen Entwicklungsumgebung (z. B. Visual Studio .Net) erforderlichen Dateien enthalten.
<i>backup</i>	In diesem Ordner wird die Sicherungsdatei des jeweiligen Projekts (Dateinamenerweiterung <i>.obs</i>) aufbewahrt.
<i>template</i>	Dieser Ordner enthält die Dokumentvorlagen bzw. die bei der Modellierung erstellten Word-Dokumente.

Um ein Projekt zu archivieren oder wiederherzustellen, nutzen Sie den entsprechenden objectiF-Menüpunkt.



Beispieldatei: *KartensystemSicherung.obs*

A		Entitätsklassen	83	Komponente	94
Ablaufmodellierung	147	Entwurfsmuster	46	Komponentendiagramm	68, 86
Abnahmephase	5	Entwurfsphase	5	Komponenteninstanz	95
Abstrakte Fabrik	47	Erbauer	47	Komposition	34, 80
Abstraktion	12, 21	Erweiterungsphase	5	Kompositum	47, 65
Adapter	47, 49	Erzeugendes Muster	47	Konstruktoren	19
Aggregation	34, 80	Exemplartyp	36	Koordinator	38
Akteur	70	F		L	
Aktivitäten modellieren	145	Fabrikmethode-Muster	47	Laufzeitobjekte	94
Aktivitätsdiagramm	68, 74, 146	Fachliches Modell	136	Lebenslinien	87
Analysemuster	38	Faktsammlung	101	Liste	36
Analysephase	5	Fassade	47, 53	M	
Analyseprozess	31	Fliegengewicht	47	Memento	47, 54
Anwendungsbereich	128	Framework	43	Modelio	130
Anwendungsfalldiagramm	68, 69	Fremdsystem	70	Modellierung	67
Anwendungsfälle	145	G		Modularität	13, 22
ArgoUML	130	Generalisierung	73, 80	N	
Assoziation	34	Glossar	101	Nachrichten	26
Attribute	12, 19	Gruppe	39	Namensraum	78
Auslöser	30	Gruppenhistory	40	Notation	11
B		H		Notizen	72
Balancierter Makroprozess	31	Hierarchie	13	Nutzungsbeziehung	33
Basisklasse	23	History	39	O	
Baugruppe	37	I		objectiF	96, 97, 130, 133
Befehl	47	Implementierungsphase	5	objectiF installieren	134
Beobachter	47, 57	Information Hiding	12	objectiF starten	134
Besucher	47, 61	Interpreter	47	Objekte	19
Bezeichner, Namensvergabe	26	Iterator	47	OO-Prinzipien	21
Beziehungen, Notation von	16	K		OO-Vorgehensmodell	11
Brücke	47, 50	Kapselung	21	OO-Vorgehensmodell, Phasen	14
C		Kardinalität	33	P	
CASE-Tools	127	Kardinalität, Notation von	17	Paketdiagramm	68, 78
CASE-Tools im Vergleich	130	Kartenverkaufssysteme	96	Pakete	78
Code-Generierung	129	Klasse	18	Parametrisierbare Klassen	80
Computer	94	Klasse, abgeleitete	23	Planungsphase	5
D		Klassenbeziehungen	33	Polymorphie	24
Datenkapselung	12	Klassenbibliotheken	42	Programmiersprachen,	
Dekorierer	47, 51	Klassendiagramm	68	objektorientierte	11
Design Patterns	46	Klassifizierung	24	Prototyp	47
Destruktoren	20	Knoten	94	Proxy	47
Dynamisches Modell	30	Knoteninstanz	94	Prozesse	94
E		Kommunikationsdiagramm	68, 89		
Einführungsphase	5				
Einsatzdiagramm	94				

R

Referenzbeziehung	33
Reverse Engineering	129
Rollen	38
Roundtrip	129

S

Schablonenmethode	47
Schnittstelle	19, 84
Schnittstellenmethoden	19
Schnittstellenobjekte	83
Secondary	70
Sequenzdiagramm	68, 87
Simulation	45
Singleton	47, 48
Softwareentwicklung, Phasen	5
Spezialisierung	80
StarUML	130
Statische Analyse	32
Statisches Modell	30
Stereotypen	80
Strategie	47
Strukturelles Muster	47
Stückliste	37

Synchronisation	74
Szenario	35

T

Technisches Modell	136
Teilprozess	94
Teilsystem	78, 94
Template-Klassen	80
Ticketverkauf	96, 97
Transitionen	75
Typisierung	14

U

UML – Unified Modeling Language	11, 15, 72
UML-Notation	15
Unterklasse	23

V

Vererbung	23
Vererbungshierarchie	80
Verhaltensmuster	47

Vermittler	47
Verteilungsdiagramm	69, 94
Verzweigung	74
Vielgestaltigkeit	24
Visual Paradigm for UML	130
Vorgehensmodelle	24

W

Wartungsphase	5
Wechselnde Rollen	38
Wiederverwendung	42

Z

Zeichensprache	67
Zustand	19, 47
Zuständigkeitsketten	47
Zustandsänderungen	91
Zustandsautomaten	29, 35
Zustandsdiagramm	69, 91