Hendrik Jan van Randen
Christian Bercker
Julian Fieml

Einführung in UML

Analyse und Entwurf von Software

Springer Vieweg

Einführung in UML

Lizenz zum Wissen.

Sichern Sie sich umfassendes Technikwissen mit Sofortzugriff auf tausende Fachbücher und Fachzeitschriften aus den Bereichen: Automobiltechnik, Maschinenbau, Energie + Umwelt, E-Technik, Informatik + IT und Bauwesen.

Exklusiv für Leser von Springer-Fachbüchern: Testen Sie Springer für Professionals 30 Tage unverbindlich. Nutzen Sie dazu im Bestellverlauf Ihren persönlichen Aktionscode **C0005406** auf *www.springerprofessional.de/buchaktion/*

Jetzt 30 Tage testen!

Springer für Professionals.
Digitale Fachbibliothek. Themen-Scout. Knowledge-Manager.

- Zugriff auf tausende von Fachbüchern und Fachzeitschriften
- Selektion, Komprimierung und Verknüpfung relevanter Themen durch Fachredaktionen
- Tools zur persönlichen Wissensorganisation und Vernetzung

www.entschieden-intelligenter.de

Springer für Professionals

Springer

Hendrik Jan van Randen • Christian Bercker
Julian Fieml

Einführung in UML

Analyse und Entwurf von Software

Springer Vieweg

Hendrik Jan van Randen
Vorden, Niederlande

Julian Fieml
rechenwerk GmbH
Essen, Deutschland

Christian Bercker
rechenwerk GmbH
Essen, Deutschland

Übersetzung aus dem Niederländischen mit freundlicher Genehmigung des Autors. Titel der niederländischen Originalausgabe: Inleiding UML, Pearson Benelux B.V., 2013.

ISBN 978-3-658-14411-1 ISBN 978-3-658-14412-8 (eBook)
DOI 10.1007/978-3-658-14412-8

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

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist Teil von Springer Nature
Die eingetragene Gesellschaft ist Springer Fachmedien Wiesbaden GmbH
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Strasse 46, 65189 Wiesbaden, Germany

Vorwort der Übersetzer

Seit Jahrhunderten stimmen Bauherren und Architekten Gebäudeanforderungen mithilfe von Grundrissen, Schnitten und Ansichten ab. Wie Bauzeichnungen für die Gebäudeplanung bilden UML-Diagramme im Kontext der Softwareentwicklung mittlerweile das zentrale Entwurfs- und Kommunikationsmittel. Anders als Bauzeichnungen, die bis zu einem gewissen Grad auch von Laien intuitiv verstanden werden können, bedarf das Verstehen von UML-Diagrammen zusätzlicher Erläuterung.

Das vorliegende Buch liefert hierzu den passenden Einstieg. Dabei richtet es sich auch an Leser ohne Programmierkenntnisse, die zum Beispiel an einem Softwareentwicklungsprojekt mitarbeiten. Mit Blick auf das notwendige Methodenwissen vermittelt das Buch eine solide Basis, die eine zielorientierte Zusammenarbeit unterstützt. Bestimmte Problemstellungen (z. B. das objektrelationale Mapping, die Problematik von parallelen Status im Aktivitätsdiagramm, das Erstellen einer CRUD-Matrix), die in Entwicklungsprojekten üblicherweise eine Rolle spielen, werden fokussiert behandelt und klar erörtert. Das Buch zeigt exemplarisch anhand der einfach nachzuvollziehenden Entwicklung eines Webshops, welche konzeptionellen Schritte der eigentlichen Programmierung vorausgehen und diese begleiten. Um das Bild zu vervollständigen, werden neben UML weitere im Entwicklungsprozess anwendbare Methoden und Werkzeuge vorgestellt.

Der Mehrwert von UML-Diagrammen steigt mit der Anzahl der Projektbeteiligten, die sie interpretieren und mitgestalten können. Zur Etablierung einer gemeinsamen Sprache empfiehlt sich die Lektüre dieses Buches für alle, die sich noch nicht mit UML auskennen und in ein Softwareentwicklungsprojekt eingebunden sind.

Essen, im Mai 2016

Christian Bercker
Julian Fieml

Einleitung

Das Ziel der Analyse und des Entwurfs von Software ist es, ein möglichst klares Bild von der Software zu erhalten und dieses vermitteln zu können.

Ein Mittel, um dieses Ziel zu erreichen, besteht aus einer Sammlung von Abbildungen, einem sogenannten **Modell**, das kurz und bündig folgendes beschreibt:

- Die **Daten**, die mit der Software betrachtet und verändert werden können.
- Die **Geschäftsprozesse**, durch die die Daten laufen.
- Die **Rollen** und **Zugriffsrechte**, die Nutzer von der Software haben.
- Die **Anwenderschnittstelle** (Masken/Berichte der Software).
- Die **Navigation** zwischen den Fenstern der Anwenderschnittstelle.

Das Buch zeigt, wie diese Art von Modell erstellt werden kann.

Die meistgenutzten Modelliertechniken sind Teil der international standardisierten Modelliersprache **Unified Modeling Language (UML)**. UML wird durch eine internationale Organisation verwaltet, die **Object Management Group (OMG)**.

Um diese Modelliertechniken in einen Kontext zu setzen, werden in diesem Buch auch nicht-UML-bezogene Aspekte des Analyse- und Entwurfsprozesses beschrieben.

Bei den Modelliertechniken (Diagrammen), die Bestandteil von UML sind, wird dies zu Beginn des jeweiligen Kapitels angegeben.

Inhaltsverzeichnis

Anforderungskatalog/Lastenheft

1

Zusammenfassung

Jede Anwendung wird auf Basis einer Anzahl von Kriterien/Anforderungen entworfen, die meistens durch den Auftraggeber festgelegt sind. Dieses Kapitel gibt eine Übersicht zu den Informationen, die der Modellierer benötigt, um mit dem Entwurf einer Anwendung beginnen zu können.

1.1 Ziel und Anwendungsbereich

Bevor mit dem Entwurf der Anwendung begonnen werden kann, müssen das Ziel und der **Anwendungsbereich** (**scope**), der die zu unterstützenden Aufgaben umfasst, festgelegt werden.

Beispiel

Das Ziel eines zu entwickelnden Webshops ist, dass Kunden Produkte über das Internet bestellen.

Der Umfang der Aufgabe (der Anwendungsbereich) beschreibt kurz und bündig, was die Anwendung macht und was sie nicht macht.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_1

Beispiel

Der Anwendungsbereich eines zu entwickelnden Webshops umfasst das Bestellen und Bezahlen durch den Kunden, das Verschicken aus dem Lager und die Nutzerverwaltung.

Die Aufgaben Vorratshaltung, Retouren, Reklamationen, Finanzierung, Vorsteuerberechnung und das Erstellen von Rechnungen fallen aus dem Anwendungsbereich heraus. Die Kunst ist es, den Anwendungsbereich so klein wie möglich zu definieren, um die Entwicklungskosten und -zeit so niedrig wie möglich zu halten.

1.2 Anforderungsspezifikation

Meistens wird der Anwendungsbereich in einer **Anforderungsspezifikation** (**requirements**) weiter ausgearbeitet.

Beispiel

Die Anforderungsspezifikation des Webshops enthält u. a. folgende Anforderungen:

- Kunden können sich selbst als Kunden über das Internet registrieren.
- Eine Bestellung wird verschickt, nachdem der Kunde bezahlt hat.
- Ein Kunde kann via iDeal/PayPal bezahlen.
- Ein Produktverwalter bestimmt die Auswahl der Produkte und ihren Preis.
- Alle Preise und Beträge enthalten die gesetzliche Mehrwertsteuer.

Der Anwendungsbereich und die Anforderungsspezifikation beschreiben, was die Anwendung macht, ohne darauf einzugehen, wie die Anwendung es ausführt. Das „wie“ wird im **Entwurf** beschrieben.

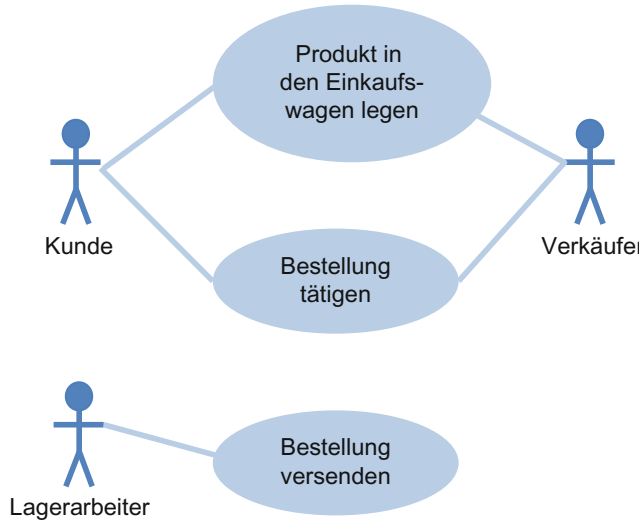
1.3 Use Cases (Anwendungsfälle)

Oft enthält die Anforderungsspezifikation eine Anzahl von **Use Case-Diagrammen**. Anhand dargestellter Verbindungslinien zeigen diese, welche **Akteure** (Nutzerrollen, dargestellt durch Strichmännchen) welche **Use Cases** (Szenarien, dargestellt durch Ellipsen) ausführen können.

Ein Use Case (Anwendungsfall) beschreibt allein die Interaktion, nicht die interne Verarbeitung durch das System.

Weil sie an die entsprechende Erfahrungswelt anschließen, können Use Cases gut mit Anwendern besprochen werden. Der Name eines Use Cases besteht allgemein aus einem selbstständigen Namenwort (Nomen) und einem Tätigkeitswort (Verb).

Beispiel



In diesem Beispiel können Kunden und Verkäufer Produkte in den Einkaufswagen legen und Bestellungen tätigen. Ein Lagermitarbeiter kann Bestellungen versenden.

1.4 Unklarheit in der Anforderungsspezifikation

Falls Undeutlichkeiten oder Unvollständigkeiten in den Anforderungsspezifikationen existieren, ist es Aufgabe des Modellierers, diese Undeutlichkeiten und Unvollständigkeiten zu beseitigen. Hierfür sind Auftraggebern und Fachspezialisten die richtigen Fragen zu stellen.

Um die Nachvollziehbarkeit zu gewährleisten, ist es hilfreich, die Fragen und Antworten systematisch in einer Art Tagebuch aufzuzeichnen. Das macht es später einfach, die richtigen Personen zu finden, wenn noch weitere Detailfragen beantwortet werden müssen oder um nachzuvollziehen, warum eine bestimmte Anforderung gestellt wurde. Letzteres ist insbesondere relevant, wenn in einem späteren Entwicklungsstadium die Bedeutung verschiedener Anforderungen verglichen und abgewogen werden muss. Das Gleiche trifft zu, wenn gegenseitige Konflikte bestehen, d. h. wenn Anforderungen sich gegenseitig behindern (oder beeinträchtigen) oder wenn sich herausstellt, dass nicht genug Zeit und/oder Geld zur Verfügung steht, um alle Anforderungen umzusetzen.

1.5 Übung zur Anforderungsspezifikation

Um die Übung in diesem Buch so realistisch wie möglich zu gestalten, passt du die Übung am besten an ein Softwareentwicklungsprojekt an, das die Organisation, in der du arbeitest, tatsächlich realisieren (lassen) will.

Falls du (noch) nicht in einer Organisation arbeitest, in der dies möglich ist, wähle beispielsweise eine Organisation, in der Menschen arbeiten, die du kennst und mit denen du darüber sprechen kannst.

Wähle die zu realisierende Softwareentwicklung (Individualentwicklung), die du in allen Übungen des Buches nutzen wirst.

Beschreibe für die erste Übung kurz und bündig:

- Das **Ziel** der Softwareentwicklung.
- Den **Anwendungsbereich**. Definiere diesen so klein wie möglich, um die Kosten und die Durchlaufzeit für die Umsetzung der Entwicklung so gering wie möglich zu halten.
- Die **Anforderungsspezifikation** des Softwareprodukts inklusive der **Use Cases**.

Daten in einem Klassendiagramm abbilden

2

Zusammenfassung

Die Basis eines jeden Entwurfs ist die Beschreibung der **Daten** (**data**), die mit der zu entwerfenden Anwendung abgerufen und/oder verarbeitet werden sollen. Dieses Kapitel zeigt auf, wie die Beschreibung der Daten in Form eines UML-**Klassendiagramms** (**class diagram**) umgesetzt werden kann.

2.1 Klasse

Eine **Klasse** (**class**) beschreibt eine Kategorie von Objekten, die in der Anwendung eine Rolle spielen.

Dargestellt wird eine Klasse als Rechteck, das den Namen der Klasse enthält. Direkt darunter befindet sich ein weiteres Rechteck mit der gleichen Breite. Dieses zusätzliche Rechteck enthält Angaben zu den Attributen. Ferner kann in der Darstellung ein weiteres Rechteck mit Angaben zu Operationen angrenzen. Im Kontext des Klassendiagramms werden Operationen in diesem Buch jedoch nicht weiter behandelt.

Beispiel

Im Entwurf eines Webshops, in dem ein Kunde eine Bestellung für ein Produkt anlegen kann, spielen u. a. folgende Klassen eine Rolle:

Der Name der Klasse steht immer im Singular (z. B. Kunde, nicht Kunden).

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_2

2.2 Objekt

Mit der zu entwerfenden Anwendung werden **Objekte** (**objects**) der Klasse generiert. Diese Objekte werden auch **Instanzen** (**instances**) genannt.

Beispiel

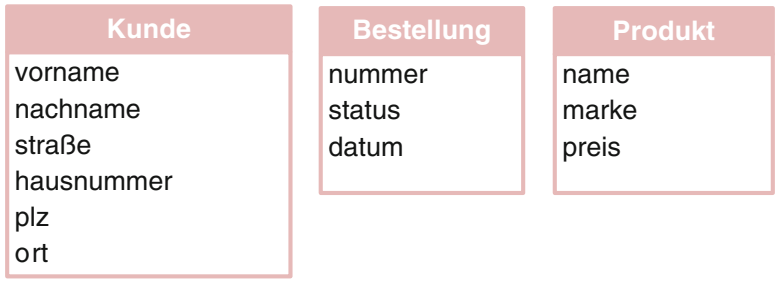
In der Webshop-Anwendung können von der Klasse „Kunde“ zum Beispiel die Objekte „Piet Jansen“, „Gerard van de Gaag“ und „Joke Hining“ generiert werden.

Von der Klasse „Bestellung“ können die Objekte „Bestellung Nr. 125“ vom Kunden „Piet Jansen“, „Bestellung Nr. 23“ und so weiter generiert (instanziiert) werden.

2.3 Attribut

Die Attribute (**attributes**) einer Klasse beschreiben Merkmale, die die Objekte der Klasse aufweisen. Die Attribute einer Klasse stehen untereinander im unteren Rechteck der Klasse.[1]

Beispiel



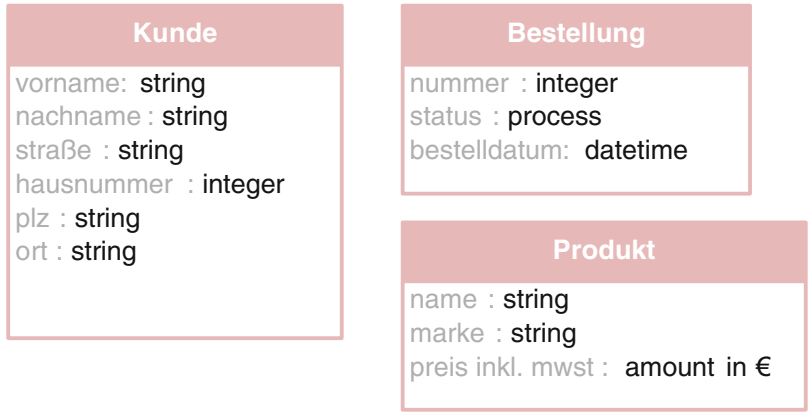
2.3.1 Attributtyp

Die Art der Daten bzw. die möglichen Werte, die in einem Attribut gespeichert werden können, werden durch den **Attributtypen** (**type**) definiert.

Der **Attributtyp** steht, getrennt durch einen Doppelpunkt, nach dem Namen des Attributs. Attributtypen werden in UML immer auf Englisch geschrieben.

[1] In einigen der folgenden Beispieldarstellungen werden Elemente, die zuvor bereits behandelt wurden, in einer helleren Farbe wiedergegeben, sodass neue Elemente leicht zu erkennen sind.

Beispiel



Tab. 2.1 listet häufig gebrauchte Attributtypen auf.

Tab. 2.1 Auflistung Attributtypen

Typ	Werte, die ein Attribut dieses Typs annehmen kann
amount in €	Geldbetrag, Valuta genannt
boolean	Ja oder Nein
date	Datum
Datetime oder timestamp	Zeitpunkt inklusive Datum
float	Zahl, die Nachkommastellen haben kann
integer	Ganzzahl
process	Geschäftsprozess (wie später in Kapitel 3.8 beschrieben)
string	Text (meistens wird eine Telefonnummer am besten als string definiert, damit z. B. die vorlaufende 0 gespeichert werden kann)
time	Zeitpunkt, Uhrzeit ohne Datum

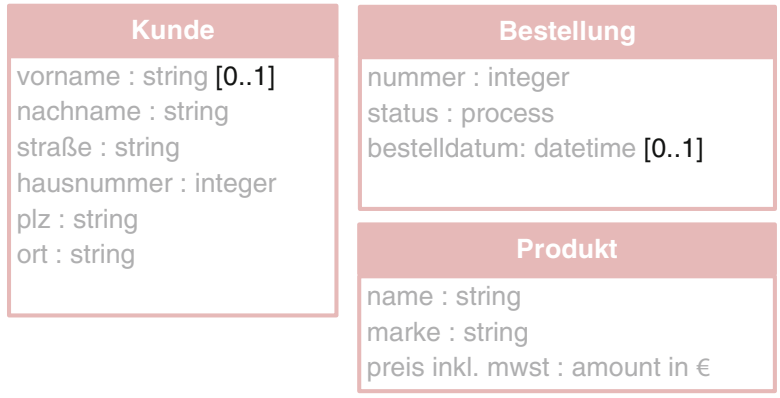
2.3.2 Pflichtfeld und optionales Attribut

Pflichtfelder (**mandatory**) müssen in jedem Objekt der Klasse einen Wert haben.

Optionale Attribute (**optional**) müssen hingegen nicht in jedem Objekt der Klasse einen Wert haben.

Optionale Attribute werden durch das Setzen von [0..1] hinter den Typen gekennzeichnet. Attribute, hinter denen diese Angabe nicht steht, sind Pflichtfelder.

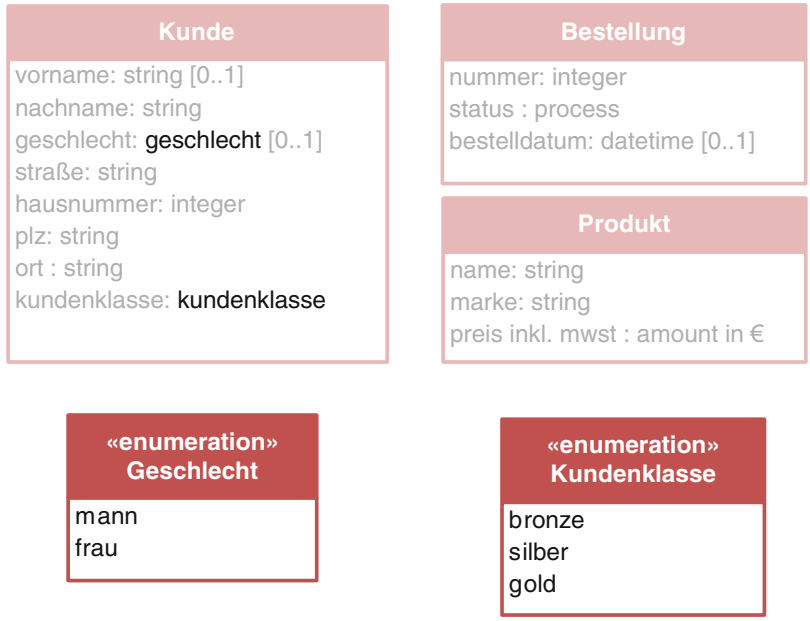
Beispiel



2.3.3 Aufzählung

Eine **Aufzählung** (**enumeration**) beschreibt eine Liste, die alle Werte umfasst, die ein Attribut annehmen kann.

Beispiel



In diesem Beispiel kann das Geschlecht eines Kunden „mann“ oder „frau“ sein. Da es als optionales Attribut definiert ist, muss es jedoch nicht für jeden Kunden ausgefüllt werden. Ferner ist jeder Kunde einer bestimmten Kundenkategorie zugeordnet. Diese kann „bronze“, „silber“ oder „gold“ sein.

Auch wenn es nicht sein muss, sind oftmals der Name der Aufzählung und der Name des betroffenen Attributs identisch.

2.4 Assoziation

Eine **Assoziation** (**association**) zwischen Klassen wird durch eine verbindende Linie zwischen den Klassen wiedergegeben.

Beispiel

Ein Kunde legt eine Bestellung an; eine Bestellung besteht aus Produkten.

Die Assoziationen und die Attribute einer Klasse beschreiben zusammen die **Eigenschaften** (**properties**), die Objekte der Klasse haben können.

Die Assoziation in der obigen Darstellung zeigt, dass Kunde eine Eigenschaft von Bestellung ist.

2.4.1 Multiplizität

Die **Multiplizität** (**multiplicity**) gibt an, wie viele Objekte der Klasse, neben der die Multiplizität steht, mit einem Objekt auf der anderen Seite der Assoziation verbunden sein können.

Beispiel

In diesem Beispiel gehört jede Bestellung zu einem Kunden und ein Kunde kann mehrere Bestellungen durchgeführt haben.

Ein Produkt kann in mehreren Bestellungen vorkommen und mit einer Bestellung können mehrere Produkte bestellt werden.

Tab. 2.2 Beispiele für Multiplizitäten

Multiplizität:	Anzahl der Objekte, die verbunden sein können:
1	Genau 1
0..1	0 oder 1
*	0 oder mehr (Abkürzung für 0..*)
1..*	1 oder mehr

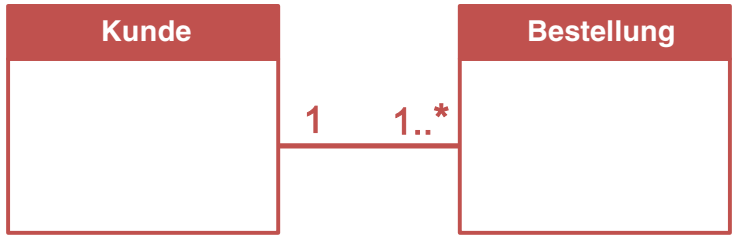
Es können auch andere Ober- und Untergrenzen definiert werden. Die Multiplizität 2..4 gibt beispielsweise an, dass immer mindestens 2 und höchstens 4 Objekte miteinander in Beziehung stehen können. Tab. 2.2 zeigt Beispiele für Multiplizitäten.

Die Daten in der Anwendung müssen jederzeit ohne Ausnahme die angegebenen Multiplizitäten einhalten. Im oben genannten Beispiel ist es daher nicht möglich, dass eine Bestellung ohne Kunde existiert.

2.4.2 Auftreten des Henne-Ei-Problems

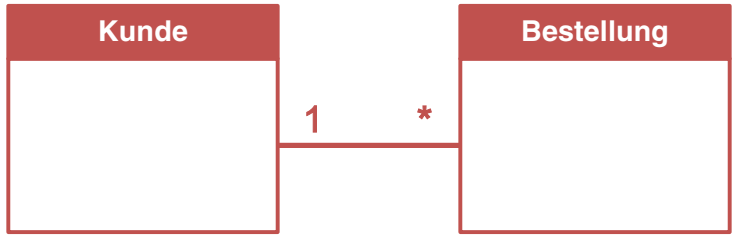
Bei der Festlegung der Multiplizitäten muss der Modellierer darauf achten, dass kein Henne-Ei-Problem entsteht.

Beispiel



Im Beispiel ist jemand kein echter Kunde, solange er nicht mindestens eine Bestellung getätigt hat. Das bedeutet, dass kein Kunde vor der ersten Bestellung existiert und eine Bestellung ohne Kunden nicht existieren kann. Entsprechend müssen Kunde und erste Bestellung genau gleichzeitig angelegt werden.

Wenn es dem gewünschten Verhalten hingegen entspricht, dass ein Kunde angelegt wird und erst danach seine erste Bestellung folgt, dann hat ein Kunde 0 oder mehrere Bestellungen [0..*]:



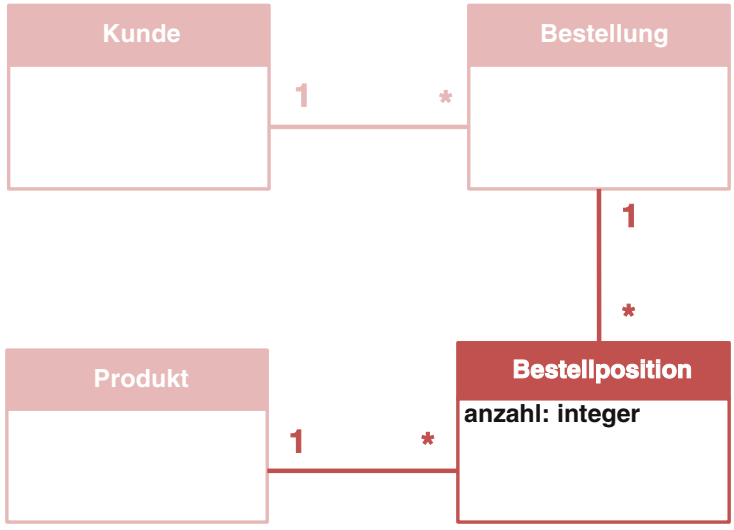
Wie das gewünschte Verhalten aussieht, ist eine Entwurfsentscheidung, die bewusst getroffen und deren Folgen sorgfältig abgewogen werden sollten. Um die Nachvollziehbarkeit zu gewährleisten, sollte der Entscheidungsprozess dabei ausreichend dokumentiert werden.

2.4.3 Zwischenklasse

Eine **Zwischenklasse** umfasst zusätzliche Informationen zu jedem Exemplar (jeder Instanz), der Verbindung zwischen zwei anderen Klassen.

Beispiel

Im Webshop muss der Anwender in der Bestellung angeben können, wie viele Exemplare von jedem Produkt bestellt werden. Hierfür ist eine Zwischenklasse notwendig, die wir Bestellposition nennen. Eine Bestellung hat für jedes bestellte Produkt eine Bestellposition, in der die Anzahl der bestellten Produkte angegeben wird:



Manchmal wird anstelle einer Zwischenklasse eine Assoziationsklasse genutzt. Eine Assoziationsklasse bietet jedoch weniger Möglichkeiten zur Handhabung von Multiplizitäten. Aus diesem Grund wird die Nutzung nicht empfohlen und hier nicht weiter behandelt.

2.4.4 Komposition

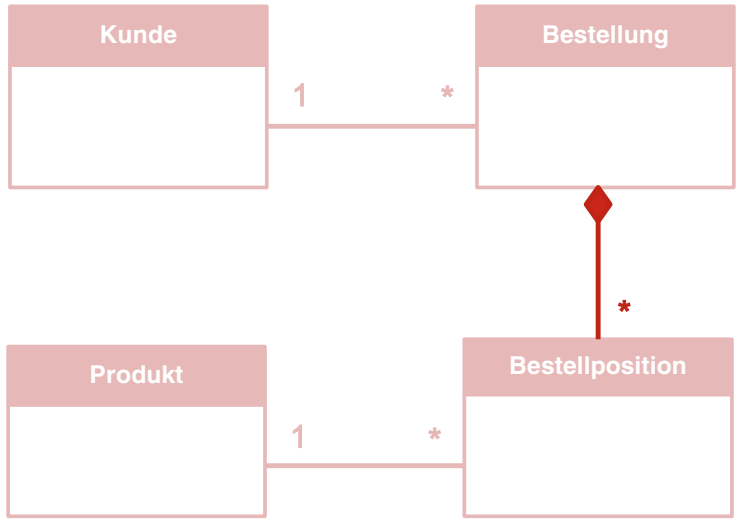
Eine **Komposition** (**composite**) wird bei Teil-Ganzes-Beziehungen (**part-whole-association**) gebraucht. Im folgenden Beispiel ist die Bestellposition Teil einer Bestellung. Eine Komposition legt zugleich fest, dass die Multiplizität auf der Seite des Ganzen 1 ist. Im Beispiel bedeutet das, dass es für jede Bestellposition eine Bestellung gibt.

Wenn das Ganze gelöscht wird, dann wird der über die Kompositionsbeziehung verbundene Teil automatisch ebenfalls gelöscht. Eine Komposition wird durch eine ausgefüllte Raute dargestellt.

Eine Komposition gibt an, dass eine **kaskadierende Löschregel** (**delete rule**) existiert.

Eine Multiplizität von 1 ohne Komposition repräsentiert eine **beschränkende Löschregel** (**restricting**).

Beispiel



Im Beispiel sorgt die Komposition bei der Bestellung dafür, dass beim Löschen der Bestellung automatisch und ohne Rückfrage alle dazugehörigen Bestellpositionen ebenfalls gelöscht werden.

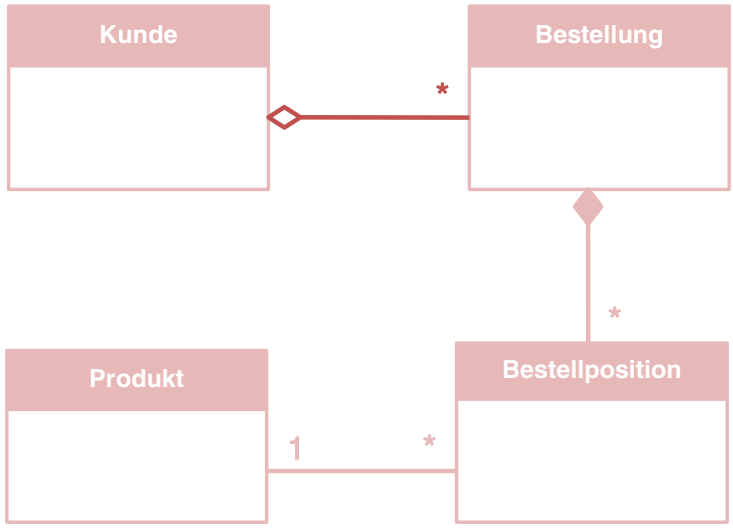
Die Multiplizität von 1 ohne Komposition beim Kunden sorgt dafür, dass ein Kunde, der Bestellungen getätigt hat, nicht mehr gelöscht werden kann. Denn, falls der Kunde in diesem Fall gelöscht werden würde, hätten die Bestellungen keinen Kunden mehr und die Multiplizität von 1 wäre nicht mehr eingehalten.

2.4.5 Aggregation

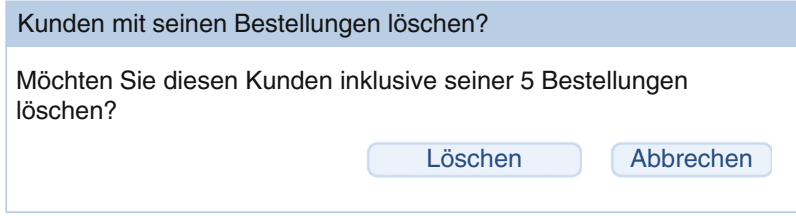
Eine **Aggregation** (**aggregation**) wird durch eine nicht ausgefüllte Raute dargestellt. Die Bedeutung einer Aggregation ist in UML nicht formal definiert. Eine Aggregation kann beispielsweise wie folgt interpretiert werden:

Wenn ein Anwender versucht, ein Ganzes zu löschen, gibt das System dem Anwender die Wahl zwischen „auch Teile löschen“ (wie bei der Komposition) und „nichts löschen“ (wie bei einer Multiplizität von 1 ohne Komposition).

Beispiel



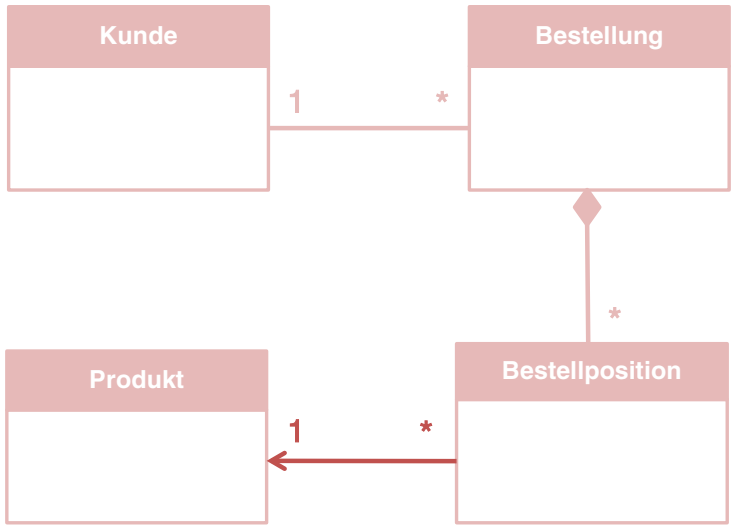
Wenn in diesem Beispiel ein Anwender probiert, einen Kunden zu löschen, und der Kunde hat eine Bestellung getätigt, dann hat der Anwender folgende Wahl:



2.4.6 Navigation

Die **Navigierbarkeit** (**navigability**) einer Assoziation wird durch einen Pfeil in die Richtung, in die navigiert werden kann, dargestellt.

Beispiel



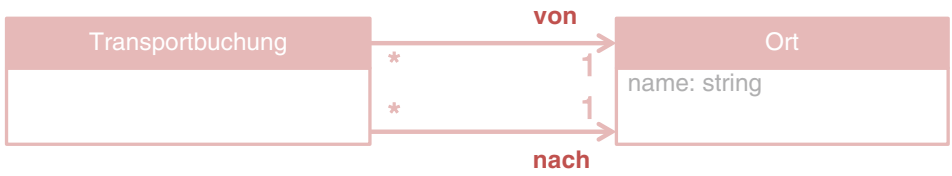
In diesem Beispiel ist die Beziehung allein von der Bestellposition zum Produkt navigierbar. Das heißt: Wenn der Anwender eine Bestellposition betrachtet, ist das betroffene Produkt sichtbar. Wenn der Anwender hingegen ein Produkt betrachtet, sind die Bestellpositionen, die sich auf dieses Produkt beziehen, nicht sichtbar.

2.4.7 Assoziationsname

Meistens ergibt sich die Bedeutung einer Assoziation aus sich selbst heraus. Manchmal ist es jedoch notwendig anzugeben, welche Eigenschaft einer Klasse durch die Assoziation beschrieben wird.

Beispiel

In der Buchungsanwendung eines Reisebüros hat die Klasse „Transportbuchung“ eine Assoziation zur Klasse „Ort“, um anzugeben, von wo aus die Reise beginnt. Außerdem besitzt sie eine Assoziation zur Klasse „Ort“ für das Ziel der Reise.



Der Name eines Assoziationsendes gibt an, wie das Objekt an dem Assoziationsende aus Sicht des anderen Objekts (das am jeweils anderen Assoziationsende steht) genannt wird.

2.5 Einfachheit oder Flexibilität

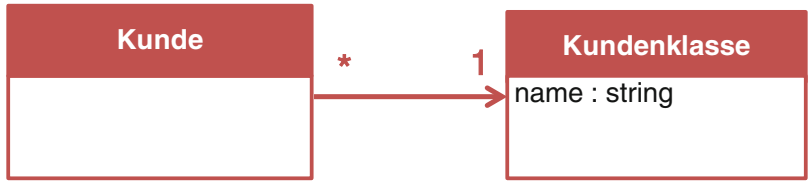
Flexibilität führt meistens zu einer größeren und komplexeren Anwendung. Ein Modellierer muss darum stets sorgfältig abwägen, ob er einerseits Flexibilität oder anderseits Einfachheit bieten möchte.

Beispiel Einfachheit

Wenn die Kundenklasse eines Kunden „bronze“, „silber“ oder „gold“ sein kann, dann lässt sich das am einfachsten mit einer **Aufzählung** (**enumeration**) modellieren:

Beispiel Flexibilität

Wenn für den Anwendungsadministrator (der die Funktionen einer Anwendung verwaltet, Wartungsarbeiten durchführt oder, wenn nötig, Einstellungen in der Anwendung anpasst) jedoch die Möglichkeit bestehen muss, die existierenden Kundenklassen zu ändern, dann wird die Klasse „Kundenklasse“ besser als **Referenzklasse** (**reference class**) modelliert.



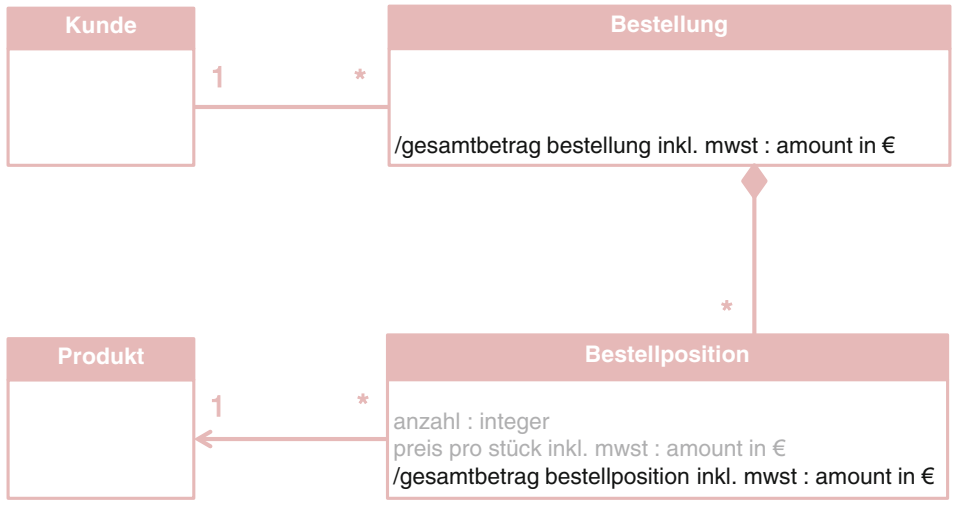
Zu einer Referenzklasse wird über eine Assoziation zu einer anderen Klasse referenziert. Objekte einer Referenzklasse werden häufig durch Anwendungsadministratoren gepflegt.

In diesem Fall führt Flexibilität zu Komplexität, weil Masken entworfen und implementiert werden müssen, mit denen der Anwendungsadministrator die Objekte der Kundenklasse pflegen kann. Dabei muss z. B. auch entschieden werden, ob eine bestehende Kundenklasse geändert oder gelöscht werden kann und, was in einem solchen Fall mit den Kunden geschieht, die dieser Kundenklasse zugeordnet sind.

2.6 Abgeleitetes Attribut

Von einem **abgeleiteten Attribut** (**derived attribut**) wird der Wert automatisch berechnet oder aus den Werten anderer Attribute ermittelt. Abgeleitete Attribute werden durch das Vorsetzen eines/gekennzeichnet.

Beispiel



In diesem Bespiel wird der Gesamtbetrag der Bestellung sowie der Gesamtbetrag der Bestellposition automatisch berechnet.

Falls die Bezeichnung des abgeleiteten Attributs nicht für sich selbst spricht, wird unterhalb des Klassendiagramms beschrieben, wie die Werte aus dem abgeleiteten Attribut berechnet werden.

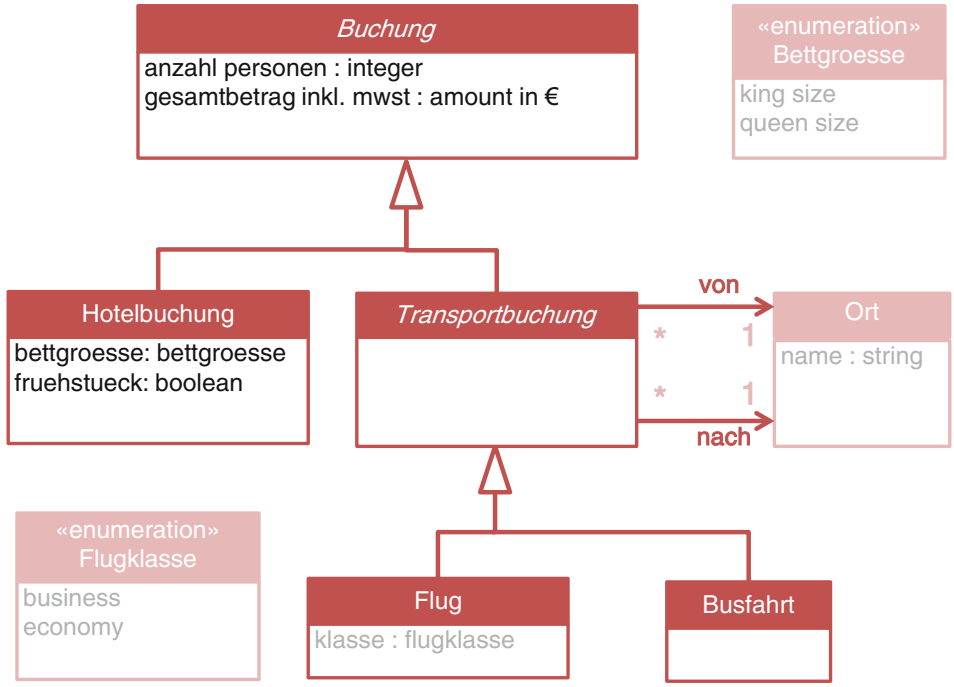
2.7 Vererbung

Wenn diverse Klassen gemeinsame Eigenschaften (Attribute, Assoziationen) haben, dann können diese gemeinsamen Eigenschaften durch eine **Oberklasse** (**super class**) modelliert werden. Eine Oberklasse wird auch als **Generalisierung** (**generalization**) bezeichnet.

Die **Unterklassen** (**subclasses**) erben (inherit) die Attribute und Assoziationen dieser Oberklasse(n). Eine Unterklasse wird auch als **Spezialisierung** (**specialization**) bezeichnet.

Die **Vererbung** wird durch einen dreieckigen Pfeil, der zur Oberklasse zeigt, dargestellt. Die Oberklasse wird immer oberhalb ihrer Unterklassen dargestellt.

Beispiel



In diesem Beispiel sind die Klassen „Hotelbuchung“ und „Transportbuchung“ Spezialisierungen der Klasse „Buchung“. Die Klassen „Flug“ und „Busfahrt“ sind Spezialisierungen der Klasse „Transportbuchung“.

Die Klassen „Hotelbuchung“, „Flug“ und „Busfahrt“ erben die Personenzahl und den Gesamtpreis von der Klasse „Buchung“. Die Klassen „Flug“ und „Busfahrt“ erben die Assoziationen „von“ und „nach“ von der Klasse „Transportbuchung“. Diese führen von der Klasse „Transportbuchung“ zur Klasse „Ort“.

Oft sind Oberklassen **abstrakt**. Das bedeutet, dass es keine anderen Instanzen der Oberklasse gibt als die Instanzen ihrer Unterklassen. Der Name einer abstrakten Klasse steht in kursiv.

Im Beispiel trifft dieser Fall auf die Klassen „Buchung“ und „Transportbuchung“ zu.

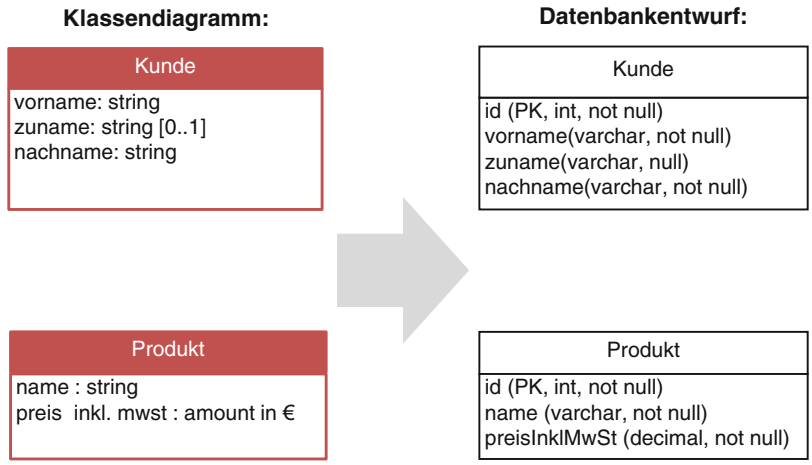
2.8 Klassendiagramm vs. Datenbankentwurf

▶ Dieser Abschnitt ist besonders relevant für Leser, die mit dem Entwerfen von Datenbanken vertraut sind. Andere Leser können diesen Abschnitt überschlagen und direkt mit dem Abschn.2.9 beginnen.

Ein Entity-Relationship-Diagramm, das den Entwurf einer Datenbank visualisiert, gleicht einem Klassendiagramm, aber es ist nicht das Gleiche. Informationen zu N:M-Beziehungen und Vererbung gehen daraus beispielsweise nicht unmittelbar hervor.

Das Umsetzen von einem Klassendiagramm in einen Datenbankentwurf erfolgt durch eine **objektrelationale Abbildung**. Hierbei werden die meisten Klassen durch eine Tabelle abgebildet (gemappt) und die meisten Attribute durch eine Tabellenspalte skizziert.

Beispiel



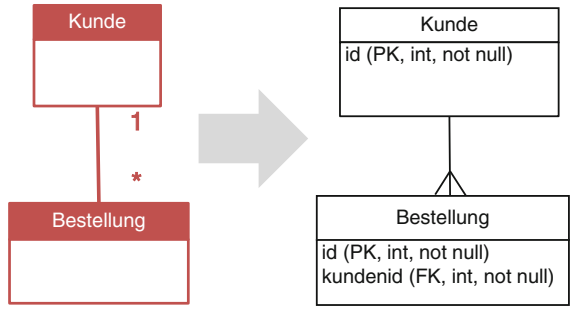
Die Primärschlüssel (gekennzeichnet durch PK) im Datenbankentwurf kommen im Klassendiagramm nicht vor.

2.8.1 Objektrelationale Abbildung von Beziehungen

Wie Beziehungen in einer Datenbank abgebildet werden, hängt von ihrer Multiplizität ab.

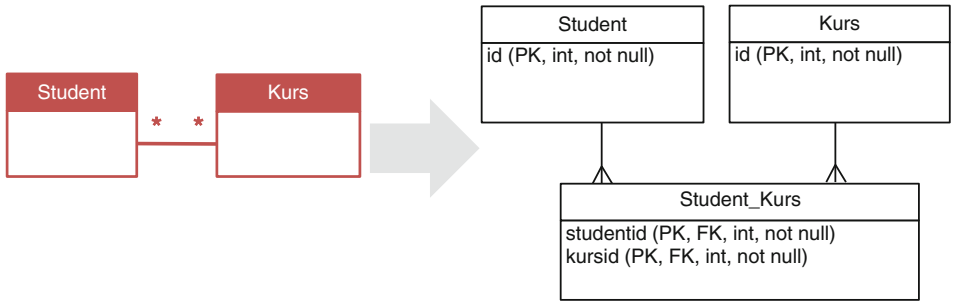
Beispiel 1:N Beziehung

1:N Beziehungen werden über einen Fremdschlüssel (gekennzeichnet durch FK) abgebildet.



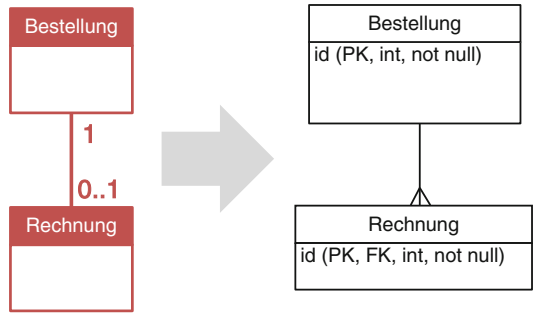
Beispiel N:M Beziehung

N:M Beziehungen werden durch eine Verbindungstabelle mit zwei Fremdschlüsseln abgebildet.



Beispiel 1:1 Beziehungen

1:1 Beziehungen werden über einen gemeinsamen Primary Key (PK) abgebildet, der in einer der Tabellen gleichzeitig auch Fremdschlüssel ist.



2.8.2 Objektrelationale Abbildung von Vererbung

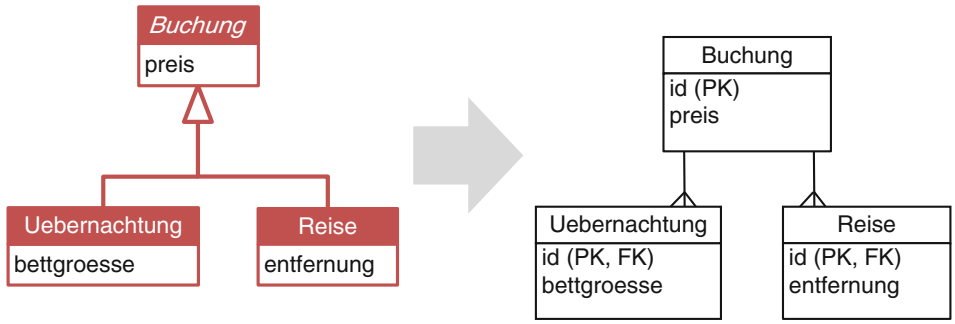
Vererbung kann auf drei verschiedene Arten (delegate, roll down, roll up) im Datenbankentwurf umgesetzt werden.

Welche Variante gewählt wird, ist abhängig von den Abfragen, die am häufigsten aufgerufen werden, und den Präferenzen des Datenbankmodellierers.

Um die weiter unten aufgeführten Beispiele einfach zu halten, werden die Spaltentypen und die Kennzeichnung, ob es sich um ein Mussfeld handelt, weggelassen.

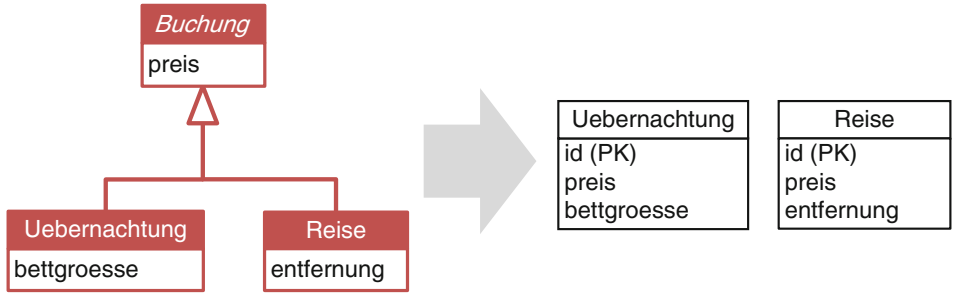
Delegate Jede Klasse wird durch eine Tabelle abgebildet. Die Tabelle einer Unterklasse erhält einen Fremdschlüssel zur Oberklasse. Dieser Fremdschlüssel ist gleichzeitig auch Primärschlüssel der Unterklasse.

Beispiel



Roll Down Jede Unterklasse wird durch eine eigene Tabelle abgebildet. Von der Oberklasse werden die Attribute und Assoziationen nach „unten gerollt“, in die Tabelle der Unterklasse, und dort abgebildet.

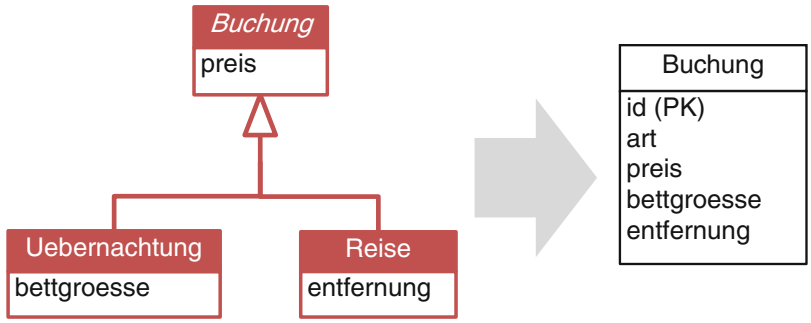
Beispiel



Roll Up Die Oberklasse wird durch eine Tabelle abgebildet. Für die Unterklassen gibt es keine Tabellen.

Von jeder Unterklasse werden die Attribute und Assoziationen in die Tabelle der Oberklasse nach „oben gerollt“. Die Tabelle der Oberklasse hat eine zusätzliche Spalte, die angibt, von welcher Unterklasse das Objekt ist, das durch den Datensatz repräsentiert wird.

Beispiel



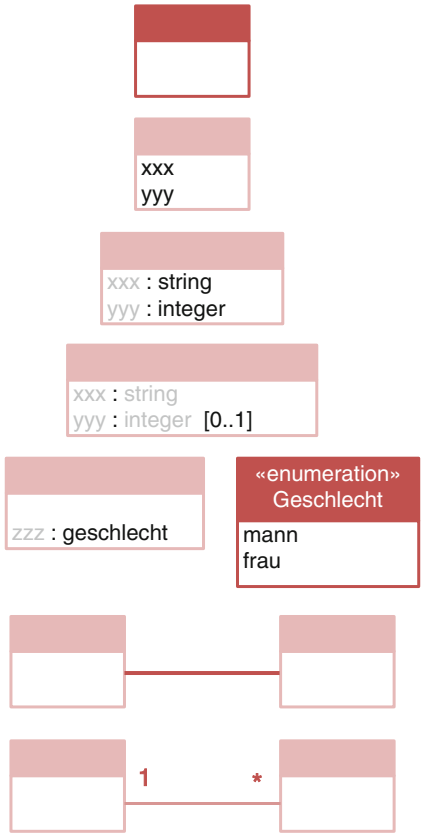
2.8.3 Wofür ein Klassendiagramm?

Es gibt viele Gründe, warum anstelle eines direkten Datenbankentwurfs zunächst ein Klassendiagramm erstellt werden sollte.

- In einem Klassendiagramm können Aspekte dargestellt werden, die in einem Datenbankentwurf nicht explizit gezeigt werden können:
  - Vererbung und Abstraktheit von Klassen
  - Komposition
  - Abgeleitete Attribute
  - Spezielle Attributtypen (wie Währung)
- Der Modellierer muss sich nicht darum kümmern, auf welche Art und Weise die Vererbung in der Datenbank umgesetzt wird.
- Das Klassendiagramm ist übersichtlicher, weil es eine Anzahl von Aspekten unberücksichtigt lässt:
  - Verbindungstabellen bei N:M-Verbindungen
  - Primärschlüssel und Fremdschlüssel

2.9 Übersicht zu den Elementen eines Klassendiagramms

Mithilfe folgender Symbole zeigt ein UML-Klassendiagramm, welche Daten in einer Anwendung relevant sind:



Klassen geben an, welche Arten von Objekten in einer Anwendung eine Rolle spielen.

Attribute geben die Merkmale der Klasse wieder.

Attributtypen definieren, welche Werte ein Attribut haben kann.

[0..1] ein optionales Attribut muss nicht immer einen Wert haben.

Eine Aufzählung umfasst eine genaue Liste von möglichen Werten, die ein Attribut annehmen kann.

Assoziationen zeigen, wie Klassen miteinander verbunden sind.

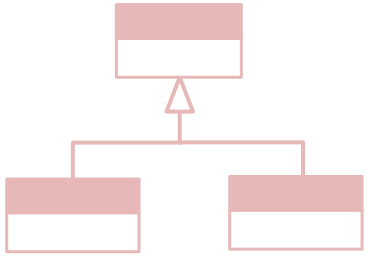
Multiplizität gibt an, wie viele Objekte (von der Klasse wo die Multiplizität steht) mit einem Objekt auf der anderen Seite der Assoziation verbunden sein können.

Eine Komposition gibt an, dass die verbundenen Objekte (Teile) automatisch auch gelöscht werden, wenn das Objekt an dem die Raute steht (Ganzes), gelöscht wird.

Die Navigierbarkeit gibt an, aus welcher Richtung entlang einer Assoziation eine andere Klasse sichtbar ist.

Der Name von einem Assoziationsende gibt sofern nötig an, was eine verbundene Klasse aus Sicht der Klasse am anderen Ende der Assoziation ist.

Von einem abgeleiteten Attribut wird der Wert automatisch berechnet bzw. abgeleitet. Dies wird mit einem / vor dem Attributnamen ausgedrückt..



Vererbung drückt aus, dass die Unterklassen auch die Eigenschaften von der Oberklasse haben..

2.10 Übung zum Klassendiagramm

Erstelle von der Anwendung, für die du zuvor die Anforderungsspezifikation aufgestellt hast, ein Klassendiagramm, indem du folgende Schritte vollziehst:

- Zeichne die Klassen, die in der Anwendung eine Rolle spielen.
- Zeichne die Assoziationen zwischen diesen Klassen mit Multiplizitäten.
- Gib den Klassen Attribute.
- Gib die Typen dieser Attribute an.
- Vervollständige das Klassendiagramm, indem du die Techniken anwendest, die du in diesem Kapitel gelernt hast.

2.11 Checkliste zum Klassendiagramm

Fragen

- Drückt der Name jeder Klasse deutlich aus, was die Objekte der Klasse sind?
- Drückt der Name jedes Attributs deutlich aus, welche Daten in diesem Attribut gespeichert werden?
- Sind die Multiplizitäten der Assoziationen korrekt?
- Steht die Multiplizität auf der richtigen Seite der Assoziation?

Wenn die Antwort auf eine der Fragen „Nein“ lautet, dann ist dieser Aspekt vom Klassendiagramm falsch und muss geändert werden.

Wiedergabe von Geschäftsprozessen in einem Aktivitätsdiagramm

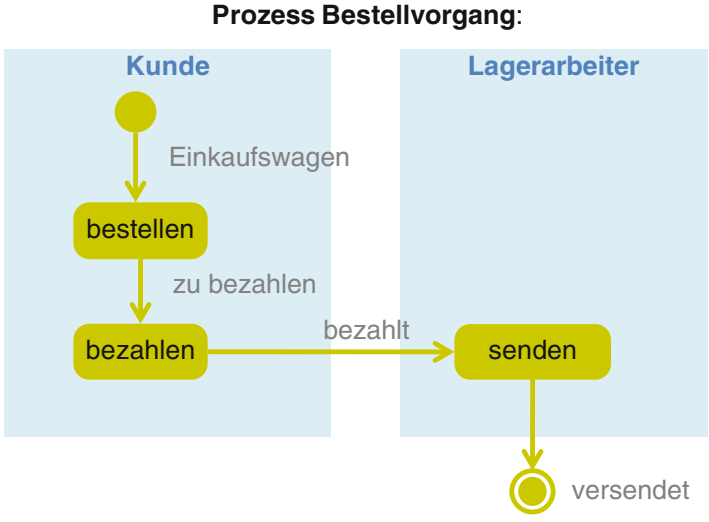
3

Zusammenfassung

Ein **Geschäftsprozess** (**business process**) beschreibt den Prozess, der von den Objekten einer Klasse durchlaufen wird. Dieses Kapitel beschreibt die Modellierung eines Geschäftsprozesses mithilfe des UML-**Aktivitätsdiagramms** (**activity diagram**).

Über dem Aktivitätsdiagramm steht, welche Objekte den Geschäftsprozess durchlaufen.

Beispiel



Ein Aktivitätsdiagramm kann auch genutzt werden, um neben Geschäftsprozessen andere Arten von Prozessen zu beschreiben.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_3

3.1 Aktion

Eine **Aktion** (**action**) wird in einem Aktivitätsdiagramm als Rechteck mit abgerundeten Ecken dargestellt, das den Namen der **Aktion** enthält.

3.2 Zustand

Das Ausführen einer **Aktion** sorgt dafür, dass der **Zustand** (**state**) eines Objekts, das den Geschäftsprozess durchläuft, vom **Eingangszustand** (**source state**) der **Aktion** in den **Zielzustand** (**target state**) der **Aktion** übergeht. Ein Zustand wird als Pfeil dargestellt, neben dem der Name des Zustands steht.

Beispiel

Im Geschäftsprozess des zuletzt genannten Beispiels verändert die Aktion „bestellen“ den Zustand einer Bestellung: von „Einkaufswagen“ in „zu bezahlen“.

Wie die Namen der Zustände lauten, ist eine Entscheidung des Modellierers. Im vorherigen Beispiel hätte der Modellierer den Zustand nach der **Aktion** „bestellen“ zum Beispiel auch „bestellt“ statt „zu bezahlen“ nennen können. In diesem Fall ist die Bezeichnung „zu bezahlen“ gewählt worden, um dem Kunden anhand des Zustands aufzuzeigen, was er tun muss. Der erste Zustand im Beispiel wird „Einkaufswagen“ genannt, weil es in einem Webshop üblich ist, noch nicht ausgeführte Bestellungen „Einkaufswagen“ zu nennen. Die Zustandsbezeichnungen sind in diesem Beispiel so gewählt, dass sie für den Kunden möglichst informativ sind.

Jeder Zustand muss einen Namen haben, damit es jederzeit möglich ist, zu sehen, welchen Zustand ein Objekt hat.

3.3 Schwimmbahn

Eine **Schwimmbahn** (**swimlane**) gibt als Verantwortungsbereich an, wer eine **Aktion** ausführen kann. Oberhalb der Schwimmbahn steht der Name der Anwenderrolle oder der Rolle eines anderen **Akteurs** (z. B. ein bestimmtes Computersystem), der die Aktionen ausführen kann.

Beispiel

Im ersten Beispiel des Kapitels zeigt die linke Schwimmbahn, welche Aktionen durch den „Kunden“ ausgeführt werden können. Die rechte Schwimmbahn zeigt, welche Aktionen ein „Lagerarbeiter“ ausführen kann.

In einem Prozess können Schwimmbahnen vertikal oder horizontal laufen (im zuletzt genannten Fall steht die Anwenderrolle links in der Schwimmbahn).

3.4 Startknoten

Der **Startknoten** (**initial node**) eines Geschäftsprozesses wird als ausgefüllter Kreis dargestellt, aus dem ein Zustandspfeil hervorgeht.

Das Ausführen legt ein Objekt der zum Geschäftsprozess gehörenden Klasse an und gibt ihm den Zielzustand des Startknotens.

Ein Startknoten liegt in der Schwimmbahn der Rolle, die den Geschäftsprozess durchlaufende Objekte anlegen kann.

Beispiel

In der ersten Beispiel-Grafik dieses Kapitels gibt der Startknoten in der Schwimmbahn der Anwenderrolle „Kunde“ an, dass ein Kunde eine Bestellung anlegen kann. Die so angelegte Bestellung hat dann den Zustand „Einkaufswagen“.

Ein Prozess hat immer einen oder mehrere Startknoten.

Aus einem Startknoten kommt immer genau ein Pfeil. Es läuft niemals ein Pfeil zu einem Startknoten hin.

3.5 Endknoten

Der **Endknoten** (**final node**), der den Endzustand eines Geschäftsprozesses wiedergibt, wird durch zwei ineinander liegende Kreise dargestellt, von denen der innere ausgefüllt ist.

Hier eine Eselsbrücke, um den Unterschied der zeichnerischen Darstellung zwischen Startknoten und **Endknoten** zu behalten: Der **Endknoten** sieht aus wie der Abfluss in einer Spüle. Und so wie das Wasser letztendlich in den Abfluss fließt, fließt der Geschäftsprozess zum Endknoten.

Anders als bei den anderen Zuständen steht der Name des Endzustands neben dem **Endknoten** und nicht neben dem Pfeil, der zum **Endknoten** führt.

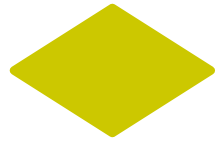
Ein Geschäftsprozess kann keinen oder mehrere **Endknoten** haben.

Wenn ein Prozess keinen **Endknoten** hat, bedeutet das, dass ein Objekt, das den Prozess durchläuft, niemals einen endgültigen Zustand einnimmt, der sich nicht mehr verändern kann.

Zu einem **Endknoten** gehen immer ein oder mehrere Pfeile. Es geht nie ein Pfeil aus dem **Endknoten** hervor.

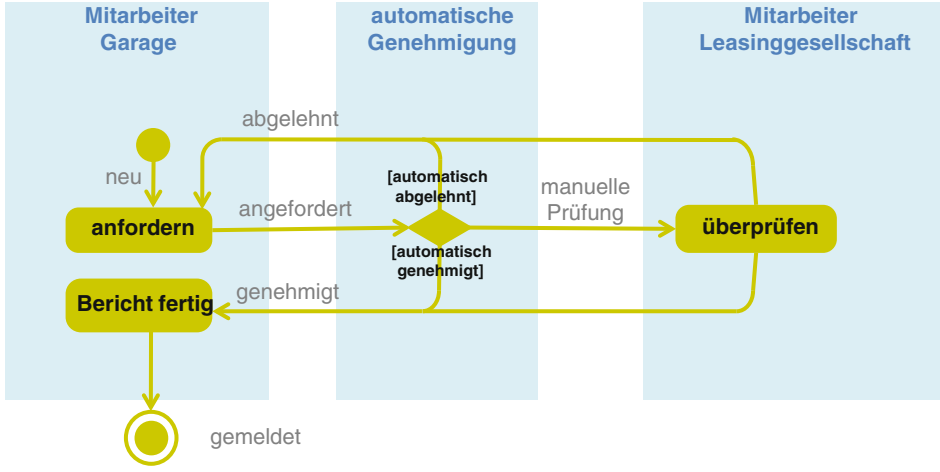
3.6 Entscheidungsknoten

Eine **automatische Entscheidung** (**automated choice**) wird in einem Geschäftsprozess als Verzweigung durch eine Raute **Entscheidungsknoten** (**decision node**) dargestellt, die zwei oder mehr Ausgänge aufweist.



Beispiel

Prozess der Reperatur-und Wartungsaktivitäten:



Wenn an dem Ausgang eines Entscheidungsknotens eine **Bedingung** (**guard**) steht, dann steht diese Bedingung in eckigen Klammern [].

Wenn die Eigenschaften eines Objekts der Bedingung an einem Ausgang genügen, dann bekommt es den Zustand des entsprechenden Ausgangs zugewiesen.

Wenn die Eigenschaften eines Objekts keiner der spezifizierten Bedingungen genügen, dann läuft der Pfeil ohne Bedingung durch.

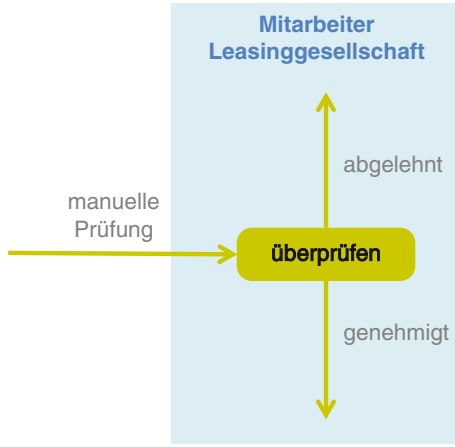
Die Bedingungen müssen insgesamt so definiert sein, dass immer eindeutig bestimmt ist, welchen Zustand das Objekt nach dem Durchlaufen des Entscheidungsknotens einnimmt.

Wenn ein Diagramm zu wenig Platz bietet, um zu beschreiben, was eine Bedingung (z. B. „automatisch genehmigt“) genau beinhaltet, wird dies unterhalb des Diagramms näher erläutert.

3.7 Manuelle Wahl

Eine **manuelle Wahl** wird als eine manuell ausgeführte Aktion mit mehreren Ausgängen modelliert. In diesem Fall wählt die Person, die die Aktion ausführt, welchen Zustand das Objekt erhält. Hierbei kann sie zwischen den Zuständen wählen, die als Pfeile aus der Aktion hervorgehen.

Beispiel



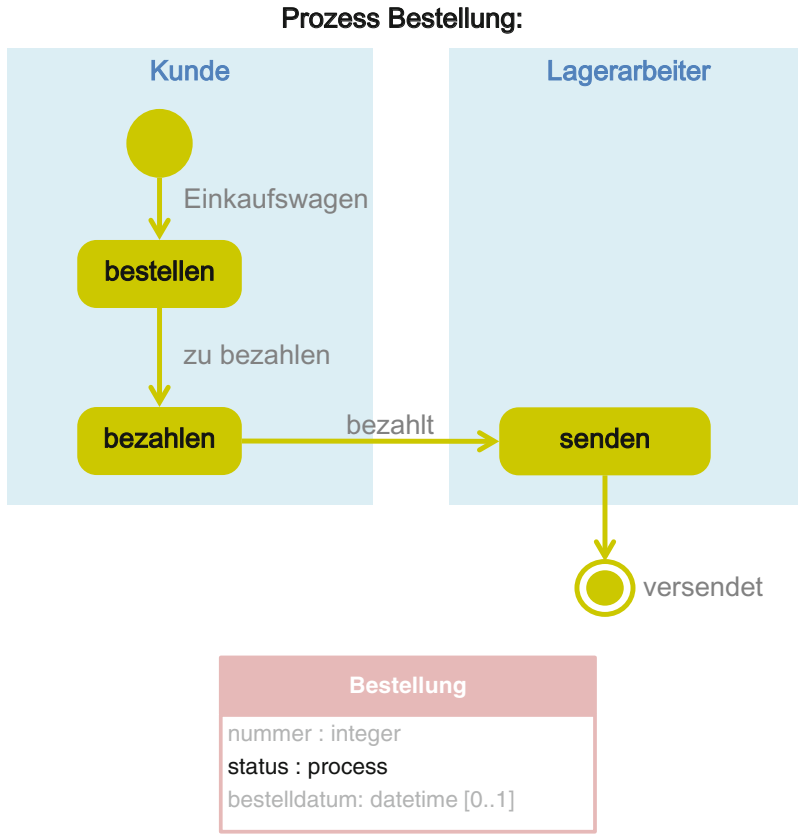
Beim Ausführen der Aktion „beurteilen“ wählt der Mitarbeiter der Leasinggesellschaft, ob der Ausgangszustand von der Aktion „genehmigt“ oder „abgelehnt“ ist.

3.8 Beziehung mit dem Klassendiagramm

Klassen, deren Objekte einen Geschäftsprozess durchlaufen, umfassen ein Zustandsattribut. Für jedes Objekt der Klasse gibt der Wert von diesem Attribut den Zustand an, den das Objekt aufweist.

Der Typ des Zustandsattributs ist „process“, denn der Geschäftsprozess bestimmt, welche Werte dieses Attribut annehmen kann (nämlich die Zustände vom Geschäftsprozess und den Aktivtäten, die einige Zeit dauern können).

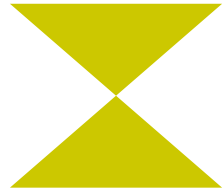
Beispiel



In diesem Beispiel hat das Zustandsattribut einer Bestellung die Werte „Einkaufswagen“, „zu bezahlen“, „bezahlt“, und „versendet“.

3.9 Timer

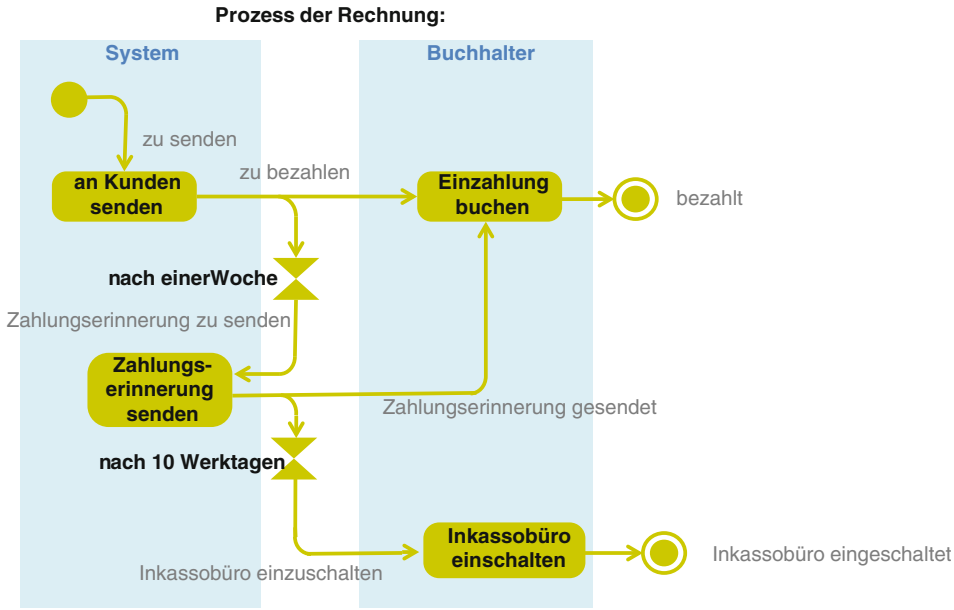
Wenn etwas zu einem bestimmten Zeitpunkt passieren muss, dann wird dies durch ein Timer-Symbol dargestellt.



Eselsbrücke: Das Timer-Symbol erinnert an eine Sanduhr.

Neben dem Symbol steht eine Zeitangabe (z. B. die Dauer, in der ein Verarbeitungsschritt vollzogen werden muss, nachdem das Objekt den Eingangsstatus des Timer-Symbols erhalten hat).

Beispiel



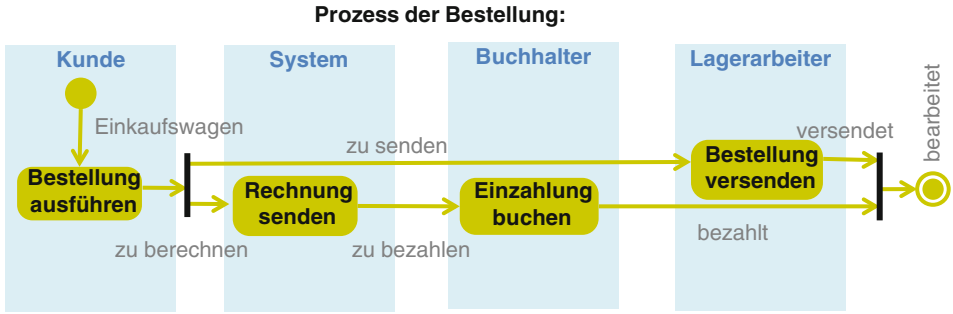
Wenn die Rechnung eine Woche nach dem Übergang in den Zustand „zu bezahlen“ immer noch diesen Zustand aufweist, geht sie in den Zustand „Zahlungserinnerung zu senden“ über.

Wenn die Rechnung den Zustand „Zahlungserinnerung gesendet“ länger als 10 Werktage aufweist, geht sie in den Zustand „Inkassobüro einzuschalten“ über.

3.10 Parallele Flüsse

Parallele Flüsse von Aktionen werden mit folgenden Symbolen modelliert:

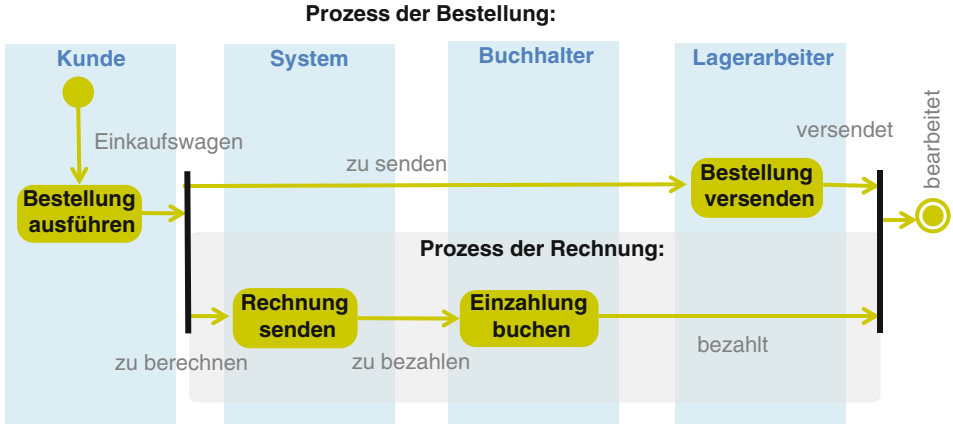
Beispiel



In diesem Beispiel müssen die Bezahlung und das Verschicken der Bestellung nicht aufeinander folgen, sondern können gleichzeitig geschehen.

Hieraus entsteht ein Problem: Im Beispiel kann eine Bestellung z. B. gleichzeitig die Zustände „zu bezahlen“ und „zu senden“ aufweisen. In diesem Fall ist unklar, wie der Zustand wiedergegeben werden soll. Deshalb ist es besser, dort, wo parallele Flüsse starten, ein Objekt von einer anderen Klasse anzulegen und dieses Objekt einen gesonderten Geschäftsprozess durchlaufen zu lassen.

Beispiel



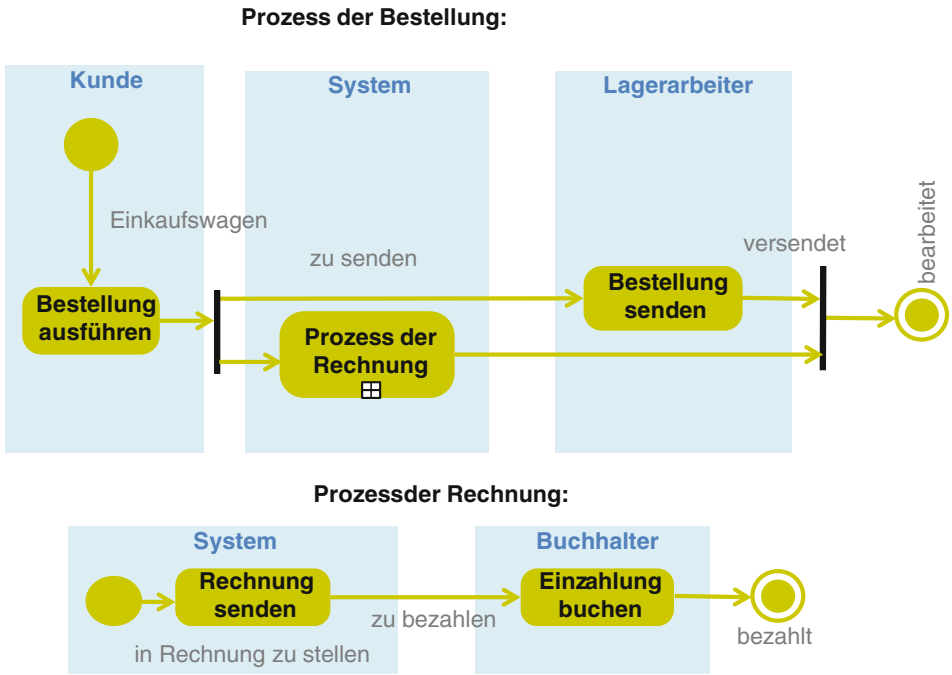
3.11 Hauptprozess und Teilprozess

Wenn ein Aktivitätsdiagramm droht, zu voll und unübersichtlich zu werden, ist es möglich, vom **Hauptprozess** auf einen **Teilprozess** zu verweisen. Dies erfolgt durch folgendes Symbol:

[1]

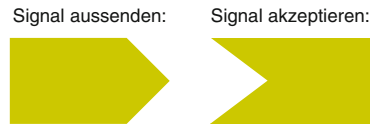
[1] Formal ist dieses Symbol kein Teil von UML sondern von BPMN Business Process Modeling Notation.

Beispiel

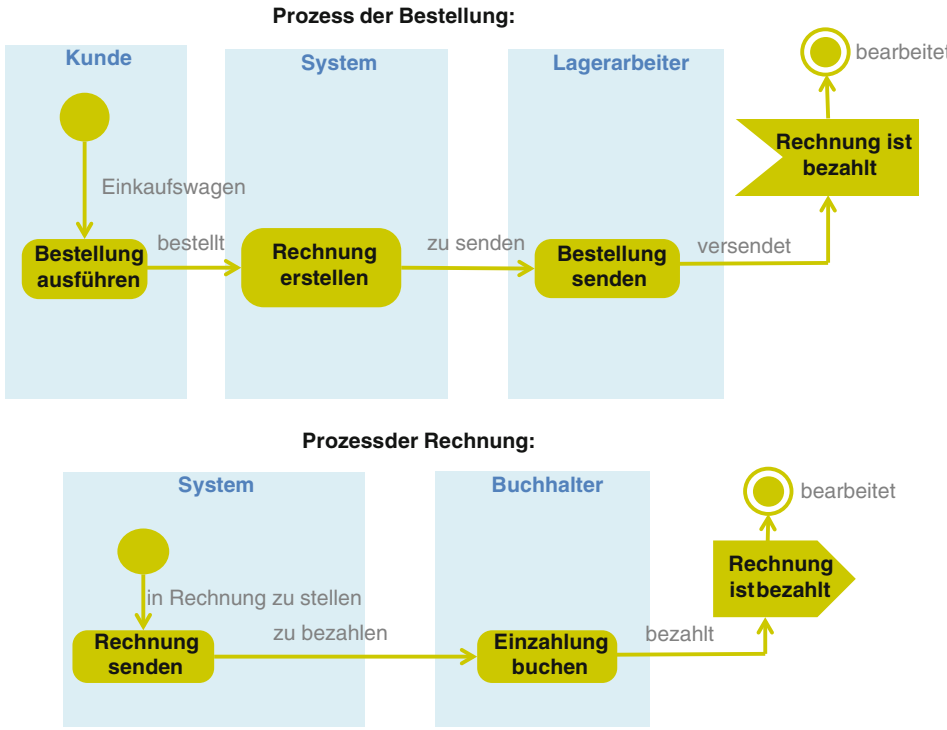


3.12 Signal

Ein Prozess kann ein **Signal aussenden** (**broadcast signal**), um sich mit einem anderen Prozess, der das Signal **akzeptiert** (**accept**), zu synchronisieren. In beiden Fällen steht der Name des Signals im Symbol.



Beispiel



Der Prozess einer Rechnung sendet das Signal „Rechnung ist bezahlt“ aus. Dieses Signal wird durch den Bestellprozess akzeptiert, wodurch die Bestellung in den Zustand „bearbeitet“ übergeht.

Über folgende Assoziationen wird die richtige Bestellung durch das ausgesendete Signal identifiziert.

3.12.1 Signal akzeptieren, wenn das Objekt einen bestimmten Zustand hat

Sollte das Akzeptieren eines Signals erst dann Einfluss auf den Prozess haben, wenn das Objekt einen bestimmten Zustand hat, so ist dieser Zustand der Eingangszustand des signal-akzeptierenden Symbols.

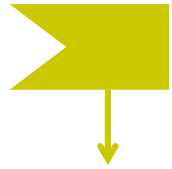
Beispiel

Ein Beispiel hierfür steht in Abschn. 3.12. Das Signal „Rechnung ist bezahlt“ hat erst Einfluss auf den Bestellprozess, wenn die Bestellung den Zustand „versendet“ hat.

Wenn das Signal bereits akzeptiert wurde, bevor das Objekt den **Eingangszustand** des signal-akzeptierenden Symbols aufweist, dann läuft, wenn das Objekt diesen Eingangszustand schließlich erhält, der Prozess direkt durch zum **Ausgangsstatus** des signal-akzeptierenden Symbols.

3.12.2 Signal unabhängig vom Zustand akzeptieren

Wenn das Akzeptieren eines Signals immer Einfluss auf den Prozess hat, das heißt, unabhängig vom Zustand in dem sich das Objekt befindet, dann hat das signal-akzeptierende Symbol keinen Eingangsstatus.

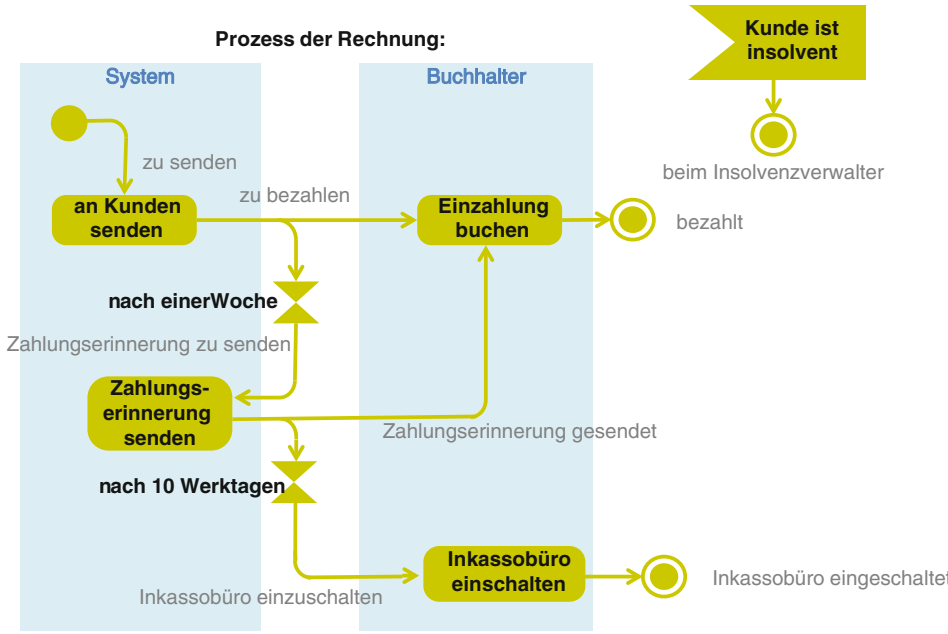


Dann bekommt das Objekt in dem Moment, in dem das Signal akzeptiert wird, sofort den Ausgangszustand des signal-akzeptierenden Symbols. Es sei denn, das Objekt befindet sich bereits in einem Endzustand.

Beispiel

Wenn ein Kunde insolvent ist, erhält die Rechnung den Zustand „beim Insolvenzverwalter“ unabhängig vom Zustand, den die Rechnung aufwies, bevor das Signal

„Kunde ist insolvent“ akzeptiert wurde. Es sei denn, die Rechnung hat bereits einen der Endzustände „bezahlt“ oder „Inkassobüro eingeschaltet“ erreicht.



3.12.3 Signale in der UML-2.5- Spezifikation

▶ Dieser Abschnitt ist besonders für Leser relevant, die sich für die formale UML-Spezifikation interessieren. Andere Leser können diesen Abschnitt überschlagen.

Formal heißt das folgende Symbol in der Version 2.5 der UML- Spezifikation „Send Signal Action“. Die Bezeichnung „Broadcast Signal“ drückt aber deutlicher aus, dass dieses Signal nicht an einen spezifischen Empfänger verschickt wird, sondern durch jeden Empfänger, der daran interessiert ist, akzeptiert werden kann.

Formal heißt das folgende Symbol in der Version 2.5 der UML-Spezifikation „Accept Event Action“. Die Bezeichnung „Signal“ drückt aber deutlicher aus, was (im anderen

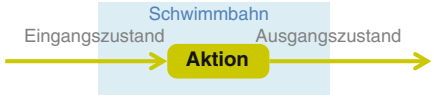
Prozess) ausgesendet wurde. Dabei handelt es sich um ein Signal und nicht um ein **Ereignis** (**event**).

3.13 Übersicht zu den Elementen eines Aktivitätsdiagramms

Ein UML-Aktivitätsdiagramm stellt die Aktion usw. in einem Geschäftsprozess (oder einem anderen Prozess) mittels folgender Symbole dar:

Aktionen geben an, welche Verarbeitungsschritte während des Geschäftsprozesses ausgeführt werden.

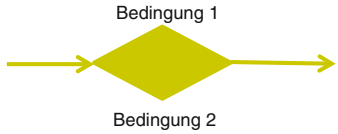
Ein **Zustand** gibt an, welchen Status das Objekt, das den Prozess durchläuft, nach dem Ausführen einer vorhergehenden Aktion und vor dem Ausführen einer nachfolgenden Aktion aufweist.



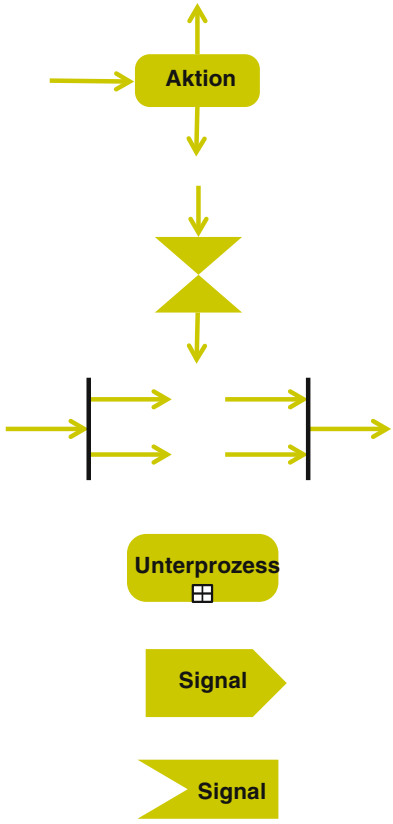
Eine **Schwimmbahn** zeigt an, wer die Aktionen innerhalb der Schwimmbahn bzw. in dem Verantwortungsbereich ausführen kann.

Aus dem **Startknoten** geht der Anfangszustand hervor, den das Objekt, das den Geschäftsprozess durchläuft, direkt nach dem Anlegen aufweist.

Der **Endknoten** gibt an, welchen Zustand das Objekt am Ende des Prozesses aufweist, wenn sich der Zustand nicht mehr verändern kann.



Ein **Entscheidungsknoten** gibt an, welchen Zustand das Objekt erhält. Dies wird durch die Bedingungen bei den Ausgängen des Entscheidungsknotens gesteuert.



Eine **manuelle Wahl** gibt dem Anwender die Möglichkeit, manuell zu wählen, welchen Zustand ein Objekt bekommt.

Ein **Timer** drückt aus, wann ein Objekt automatisch einen bestimmten Zustand erhält.

Parallele Flüsse sorgen dafür, dass ein Objekt gleichzeitig verschiedene Aktionen durchlaufen und gleichzeitig verschiedene Zustände aufweisen kann.

Ein **Hauptprozess** kann einen **Unterprozess** aufrufen.

Ein Prozess kann ein **Signal aussenden…**

und ein anderer Prozess kann dieses **Signal akzeptieren.**

3.14 Übung zum Aktivitätsdiagramm

- Erstelle eine Auflistung von **Klassen** (aus dem Klassendiagramm, das du in Kap. 2 erstellt hast), deren Objekte einen **Geschäftsprozess** durchlaufen.
- Zeichne für jede der Klassen genau die **Aktionen**, die im Geschäftsprozess ausgeführt werden, als **abgerundete Rechtecke**.
- Zeichne zwischen diesen Aktionen **Pfeile** und schreibe an jeden Pfeil den **Zustand**, den das Objekt aufweist.
- Zeichne für jede Anwenderrolle, die **Aktionen** in dem Prozess ausführt, eine **Schwimmbahn**.
- Platziere die **Aktionen**, die Anwender mit dieser Rolle ausführen, in den entsprechenden **Schwimmbahnen**.

3.15 Checkliste zum Aktivitätsdiagramm

Die häufigsten Fehler beim Zeichnen von Aktivitätsdiagrammen können durch die Beantwortung folgender Fragen aufgespürt werden:

Fragen

- Steht über dem Geschäftsprozess die Information, von welcher Klasse die Objekte sind, die ihn durchlaufen?
- Hat jeder Pfeil und jeder Endknoten einen Namen, der deutlich macht, welchen Zustand das Objekt, das den Geschäftsprozess durchläuft, nach der Ausführung der Aktion aufweist?
- Hat jeder Aktionsblock einen Namen, der deutlich beschreibt, welcher Verarbeitungsschritt dort ausgeführt wird?
- Steht der Startknoten in der Schwimmbahn des Akteurs, der ein Objekt, das den Geschäftsprozess durchläuft, anlegen kann?

Wenn die Antwort auf eine dieser Fragen „Nein“ lautet, dann ist dieser Aspekt des gezeichneten Geschäftsprozesses falsch und muss entsprechend geändert werden.

Konsistenz der Anwendung

4

Zusammenfassung

Es ist wichtig, dass eine Anwendung durchgehend konsistent und eindeutig aufgebaut ist. Das beinhaltet, dass gleichartige Funktionalitäten auch tatsächlich auf dieselbe Art und Weise funktionieren müssen. Dieses Kapitel zeigt beispielhaft einige Kriterien für das Entwerfen konsistenter Anwendungen, damit diese sich erwartungskonform verhalten.

4.1 Durchgängig einheitliche Terminologie

Der Name einer Klasse, eines Attributs, einer Aktivität, eines Zustands usw. muss für den Leser des Entwurfs und den Anwender der Software deutlich machen, was gemeint ist. Zusammen bilden diese Namen eine **durchgängig einheitliche Terminologie**, die überall im Entwurf und in der zu realisierenden Software genutzt wird.

Beispiel

Undeutliche Attribute:

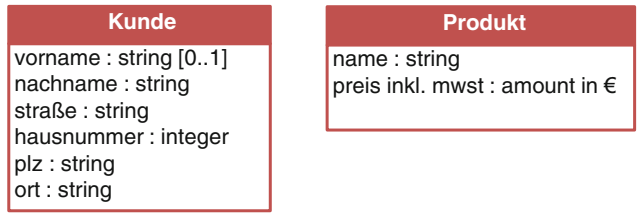
In diesem Beispiel ist unklar, ob der Name des Kunden den Vornamen, den Nachnamen oder beides umfasst. Bei der Adresse ist unklar, ob diese mit oder ohne Angabe der PLZ registriert werden muss.

Beim Preis ist es nicht eindeutig, ob er inklusive oder exklusive MwSt. gemeint ist. Ferner bleibt die Währung (z. B. Euro oder Dollar) des Preises unklar.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_4

Beispiel

Deutliche Attribute:



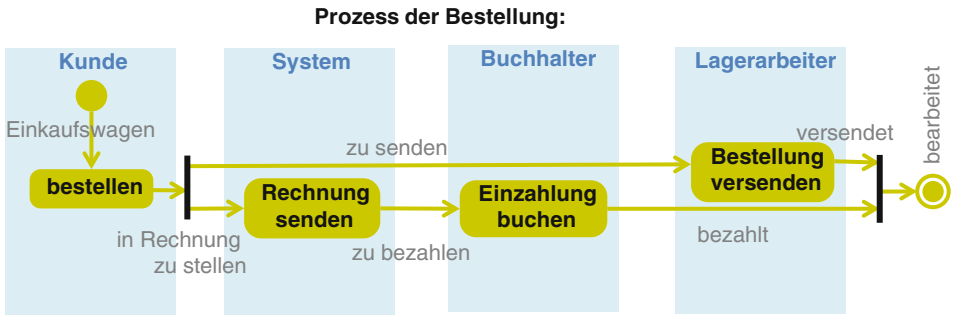
In diesem Beispiel wird deutlich, was durch den Anwender an welcher Stelle eingefügt werden soll.

Auch mit Blick auf die Groß- bzw. Kleinschreibung ist eine Konvention festzulegen, die konsequent einzuhalten ist.

4.2 Konzeptionelle Integrität

Alle in der Anwendung verwendeten Konzepte müssen klar und eindeutig und überall konsequent angewendet werden. Dies gilt für Klassen, Zustände von Objekten, die Bedienung der Nutzeroberfläche usw.

Beispiel



Im Beispiel ist das Konzept „Zustand von einer Bestellung“ nicht eindeutig, weil in diesem Fall eine Bestellung gleichzeitig mehrere Zustände haben kann.

Die konzeptionelle Integrität lässt sich hier durch die Definition eines getrennten Rechnungsprozesses verbessern. Die Nutzerfreundlichkeit der Anwendung wird maßgeblich durch die **konzeptionelle Integrität** beeinflusst.

Auch während der Implementierung und dem Testen der Software ist die konzeptionelle Integrität sehr wichtig. Konsistente Modelle sorgen dafür, dass die Softwarearchitekten

und Programmierer wissen, was sie implementieren müssen und dass die Tester wissen, wie sie die Testfälle zu gestalten haben.

4.3 Übung zur Konsistenz der Anwendung

- Sorge dafür, dass alle Bezeichnungen in der Anforderungsspezifikation, den Klassendiagrammen und den Geschäftsprozessen, die du auf Basis der vorherigen Kapitel erstellt hast, deutlich machen, was sie beschreiben.
- Sorge dafür, dass die Terminologie in deiner Anforderungsspezifikation, in deinen Klassendiagrammen und Geschäftsprozessen konsistent ist.

Anwenderrolle und Zugriffsrecht

5

Zusammenfassung

In den meisten Anwendungen sind jedem Anwender eine oder mehrere **Anwenderrollen** (**user roles**), meistens abgekürzt mit **Rollen** (**roles**), zugeordnet. Diese Rollen bestimmen, für welche Verarbeitungsschritte der Anwender autorisiert ist. Dieses Kapitel zeigt, wie die Anwenderrollen und ihre Zugriffsrechte modelliert werden können.

5.1 CRUD-Matrix

Eine **CRUD**-Matrix (CRUD steht für **C**reat, **R**ead, **U**pdate, **D**elete) umfasst für jede Anwenderrolle eine Spalte und für jede Klasse eine Zeile.

Bei jedem Schnittpunkt von einer Spalte (Rolle) und einer Zeile (Klasse) bedeutet ein:

- **C**-, dass Anwender mit dieser Rolle Objekte der Klasse **anlegen** können (**create**).
- **R**-, dass Anwender mit dieser Rolle Objekte der Klasse **lesen** können (**read**).
- **U**-, dass Anwender mit dieser Rolle Objekte der Klasse **ändern** können (**update**).
- **D**-, dass Anwender mit dieser Rolle Objekte der Klasse **löschen** können (**delete**).

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_5

Beispiel

	Kunde	Produkt-manager	Lager-arbeiter
Kunde	C R U D eigen	R	R
Bestellung	C R U[1] D[1] eigen	R	R
Bestellposition	C R U[1] D[1] eigen	R	R
Produkt	R	C R U D	R

[1] falls der Zustand der Bestellung „Einkaufswagen" ist

Diese CRUD-Matrix legt u. a. fest, dass ein Kunde seine eigenen Bestellungen und Bestellpositionen anlegen, lesen, ändern und löschen kann. Die Fußnote gibt an, dass das Ändern und Löschen nur möglich ist, wenn die Bestellung den Status „Einkaufswagen" hat.

Eine Spalte kann anstelle einer Anwenderrolle auch ein anderes System darstellen. Die CRUD-Matrix zeigt dann an, von welcher Klasse das System Objekte anlegen, lesen, ändern und löschen kann.

Der Zusatz „eigen" in dieser CRUD-Matrix drückt aus, dass die Zugriffsrechte nur für eigene Objekte gelten (wie im Beispiel Objekte der Klasse „Kunde" von dem Anwender, der in der Rolle „Kunde" agiert). Falls Klasse und Rolle nicht gleich sind (wie im Beispiel Objekte der Klasse „Bestellung" von dem Anwender, der in der Rolle „Kunde" agiert), muss eine indirekte Beziehung zwischen der betroffenen Klasse und der Anwenderrolle bestehen, die deutlich macht, auf welche Objekte der Klasse sich der Zusatz „eigen" bezieht. Im obigen Beispiel betrifft das die Beziehungen zwischen den Klassen „Kunde", „Bestellung" und „Bestellposition".

5.2 Attribut in einer CRUD-Matrix

Wenn die Zugriffsrechte für ein Attribut von den Zugriffsrechten für den Rest der Klasse abweichen, dann erhält dieses Attribut eine eigene Zeile in der CRUD-Matrix.

Beispiel

	Kunde	Produkt-manager	Lager-arbeiter
Kunde	C R U D (eigen)	R	R
Bestellung	C R U[1] D[1] (eigen)	R	R
Bestellposition	C R U[1] D[1] (eigen)	R	R
Preis inkl. MwSt.	R (eigen)	R	R
Produkt	R	C R U D	R

[1] falls der Zustand der Bestellung „Einkaufswagen" ist

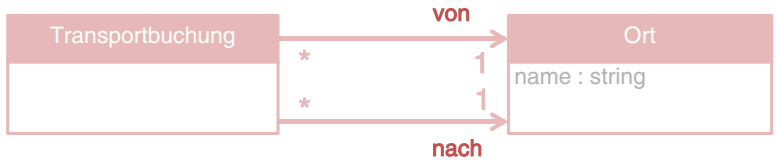
In diesem Beispiel kann ein Kunde eine Bestellposition (von seiner eigenen Bestellung mit dem Status „Einkaufswagen") ändern. Den Preis des Produkts in der Bestellposition kann er aber nicht ändern, sondern nur lesen. Deshalb hat das Attribut „Preis inkl. MwSt." eine eigene Zeile in der CRUD-Matrix.

Von den nicht explizit in der CRUD-Matrix genannten Attributen sind die **C**reate-, **R**ead- und **U**pdate-Rechte immer die gleichen wie die der Klasse. Eine Create-Berechtigung bedeutet, dass der Anwender den Wert des Attributs beim Anlegen des Objekts auswählen kann. **D**elete ist auf Attribute nicht anwendbar, weil beim Löschen stets alle Attributwerte des Objekts gelöscht werden.

5.3 Assoziation in einer CRUD-Matrix

Wenn die Zugriffsrechte für eine Assoziation von den Zugriffsrechten für den Rest der Klasse abweichen, dann erhält diese Assoziation (genau wie beim Attribut) eine eigene Zeile in der CRUD-Matrix.

Beispiel



	Kunde	Verkäufer	Disponent
Kunde	C R U D eigen	R	R
Transportbuchung	C R D[1] eigen	R	R
von	R U eigen	R	R
nach	R U eigen	R	R
Ort	R	R	R

[1] falls der Zustand der Reise „noch nicht definitiv" ist

In diesem Beispiel kann ein Kunde die „von"- und „nach"-Assoziationen seiner eigenen Transportbuchung ändern. Andere Eigenschaften der Transportbuchung kann er nicht ändern.

Eine Assoziation wird in einer CRUD-Matrix durch den Namen des Assoziationsendes auf der anderen Seite (zur anderen Klasse) angegeben oder (wenn der Name am Assoziationsende nicht explizit genannt ist) durch den Namen der Klasse auf der jeweils anderen Seite der Assoziation.

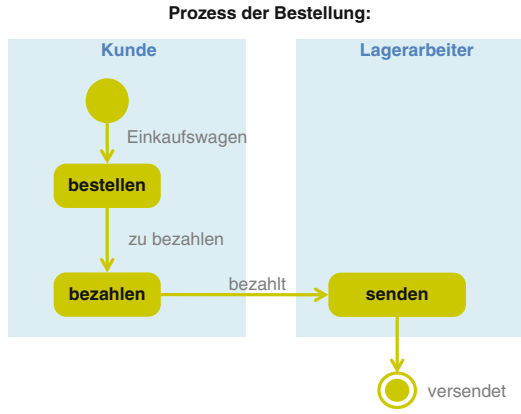
Von den nicht explizit in der CRUD-Matrix genannten Assoziationen sind die **C**reate-, **R**ead- und **U**pdate-Rechte immer die gleichen wie die der Klasse. Eine Create-Berechtigung bedeutet, dass der Anwender den Wert dieser Assoziation beim Anlegen des Objekts auswählen kann. **D**elete ist auf Assoziationen nicht anwendbar, weil beim Löschen stets alle Assoziationen des Objekts gelöscht werden.

5.4 Schwimmbahn

Neben den Zugriffsrechten, die ermöglichen, Objekte anzulegen, zu lesen, zu ändern und zu löschen, kann eine Anwenderrolle einem Anwender auch die Rechte geben, Aktionen in einem Geschäftsprozess auszuführen.

Im Aktivitätsdiagramm wird diese Autorisierung – wie in Abschn. 3.3 beschrieben – durch Schwimmbahnen modelliert.

Beispiel



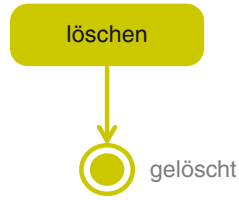
In diesem Beispiel gibt die Anwenderrolle „Kunde“ einem Anwender die Rechte, eine Bestellung anzulegen, zu bestellen und zu bezahlen. Die Anwenderrolle „Lagerarbeiter“ gibt einem Anwender das Recht, eine Bestellung zu verschicken.

5.5 Löschen

Welcher Anwender ein Objekt löschen kann, lässt sich mit einem **D** (**D**elete) in der CRUD-Matrix angeben.

Wenn die Autorisierung, ein Objekt zu löschen, stark davon abhängt, welchen Zustand das Objekt hat, kann das Löschen auch im Geschäftsprozess statt in der CRUD-Matrix modelliert werden.

In die Schwimmbahn der Rolle mit der entsprechenden Berechtigung wird dabei eine Aktivität „Löschen“ positioniert, die das Objekt in den Endzustand „gelöscht“ überführt:

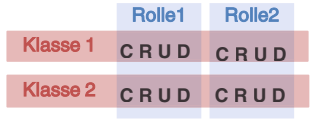


5.6 Übersicht zu Anwenderrolle und Zugriffsrecht

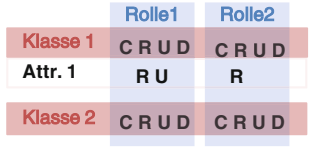
Anwenderrollen und Zugriffsrechte werden durch folgende Symbole wiedergegeben:

Anwenderrollenwerden in einer CRUD-Matrix als Spalten dargestellt.

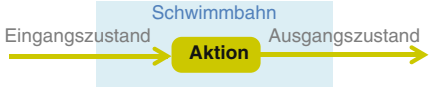
Klassen aus den Klassendiagrammen werden in einer CRUD-Matrix als Zeilen dargestellt.



In der **CRUD-Matrix** steht in den Schnittpunkten von Zeilen und Spalten, ob Anwender mit dieser Rolle die Objekte der Klasse anlegen (**C**reate), lesen (**R**ead), ändern (**U**pdate) oder löschen (**D**elete) können.



Attribute und **Assoziationen**, für die sich die Zugriffsrechte von den Zugriffsrechten der übrigen Eigenschaften der Klasse unterscheiden, haben in der CRUD-Matrix eigene Zeilen.



Eine **Schwimmbahn** zeigt, welche Rolle die Aktionen in der Schwimmbahn ausführen können.

Wenn das Zugriffsrecht „**Löschen**“ stark davon abhängt, in welchem **Zustand** sich das Objekt befindet, kann das Löschen auch innerhalb des Geschäftsprozesses abgebildet werden – anstatt in der CRUD-Matrix berücksichtigt zu werden.

5.7 Übung zu Anwenderrolle und Zugriffsrecht

- Zeichne eine Matrix mit einer Zeile pro Klasse aus dem Klassendiagramm und einer Spalte für jede Rolle aus der Anwendung.
- Gib an jedem Schnittpunkt der Matrix an, welche der Verarbeitungsschritte Anlegen, Lesen, Ändern und Löschen mit dieser Rolle hinsichtlich der Objekte der Klasse ausgeführt werden können.
- Wenn einige dieser Verarbeitungsschritte nur auf eigene Objekte angewendet werden dürfen, lässt sich das durch die Kennzeichnung „eigen“ ausdrücken.
- Wenn es weitere Beschränkungen bzgl. der Zugriffsrechte gibt, können diese mithilfe von zusätzlichen Fußnoten berücksichtigt werden.

5.8 Checkliste zu Anwenderrolle und Zugriffsrecht

Die häufigsten Fehler beim Modellieren von Anwenderrollen und Zugriffsrechten können durch die Beantwortung folgender Fragen identifiziert werden:

Fragen

- Existiert für jeden Anwender des Systems eine Rolle in der CRUD-Matrix?
- Gibt die CRUD-Matrix für jede Klasse an, wer Objekte anlegen, lesen, ändern und löschen kann?
- Gibt es für jede Klasse eine Rolle, die Objekte anlegen und löschen kann? Falls nicht – ist das eine bewusste Wahl?

Wenn die Antwort auf eine dieser Fragen „Nein“ lautet, dann ist dieser Aspekt in der Modellierung der Zugriffsrechte zu korrigieren.

Zustandsautomat

6

Zusammenfassung

In diesem Kapitel wird der **UML-Zustandsautomat** (**state machine**) beschrieben. Dieser zeigt die **Zustände** (**states**) auf, die ein Objekt aufweisen kann und die **Zustandsübergänge** (**state transitions**) zwischen diesen Zuständen. Darüber hinaus wird in diesem Kapitel der Unterschied zwischen Zustandsautomat und Aktivitätsdiagramm verdeutlicht.

6.1 Zustand und Zustandsübergang

Der **Anfangszustand** (**initial state**) wird durch einen ausgefüllten Kreis dargestellt:

Der **Endzustand** (**final state**) wird durch einen ausgefüllten Kreis, der von einem weiteren Kreis umschlossen ist, dargestellt:

Zustände werden durch abgerundete Rechtecke dargestellt:

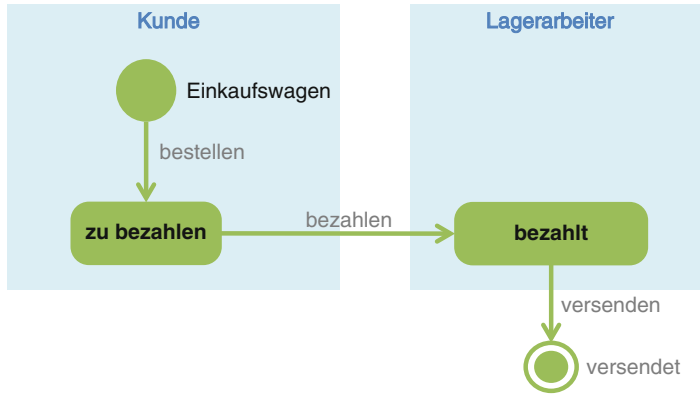
H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_6

Zustandsübergänge werden durch Pfeile dargestellt:

In einem Zustandsautomat zeigt eine Schwimmbahn an, welche Anwenderrolle ein Objekt (das einen Zustand in der entsprechenden Schwimmbahn hat) in einen anderen Zustand überführen kann.

Beispiel

Zustandsautomat von einer Bestellung:



Wenn in diesem Beispiel die Bestellung den Zustand „Einkaufswagen“ oder „zu bezahlen“ hat, kann der Kunde den Zustand durch den Zustandsübergang „bestellen“ bzw. „bezahlen“ verändern.

Wenn die Bestellung den Zustand „bezahlt“ hat, kann ein Lagermitarbeiter die Bestellung mit dem Zustandsübergang „versenden“ in den Zustand „versendet“ überführen.

In einem Zustandsautomat wird nicht gezeigt, wer das Objekt, das die Zustände durchläuft, anlegen kann. Weil dies in der CRUD-Matrix (siehe Kapitel 5) beschrieben wird, stellt diese fehlende Information jedoch kein Problem dar.

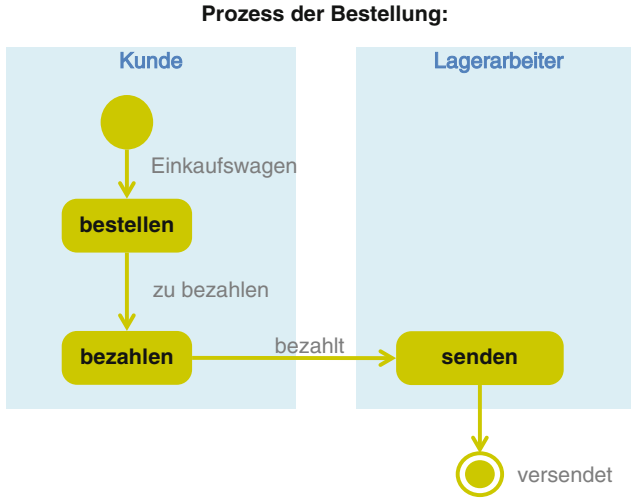
6.2 Zustandsautomat vs. Aktivitätsdiagramm

Ein Zustandsautomat zeigt die gleichen Informationen wie ein Aktivitätsdiagramm an, betont dabei aber die Zustände anstelle der Aktionen. Ob du ein Aktivitätsdiagramm oder einen Zustandsautomat erstellst, hängt davon ab, was betont werden soll.

Es macht jedoch keinen Sinn, vom selben Prozess sowohl einen Zustandsautomat als auch ein Aktivitätsdiagramm zu zeichnen. Ein Zustandsübergang in einem Zustandsautomat korrespondiert mit einer Aktion in einem Aktivitätsdiagramm. Kurz: Wenn ein Zustandsautomat in ein Aktivitätsdiagramm umgewandelt wird, dann wird jeder Pfeil ein abgerundetes Rechteck und jedes abgerundete Rechteck wird ein Pfeil.

Beispiel

Dieses Aktivitätsdiagramm beschreibt den gleichen Geschäftsprozess wie das Zustandsdiagramm in der vorherigen Abbildung.



Anwenderschnittstelle

7

Zusammenfassung

Die Anwenderschnittstelle besteht aus den Programmfenstern und der Navigation zwischen diesen Programmfenstern. Dieses Kapitel zeigt, wie Anwenderschnittstellen modelliert werden können. Im Kontext von Webanwendungen wird auch von Browserfenstern und Webseiten gesprochen. Aus Gründen der Vereinfachung und Vereinheitlichung findet im Folgenden der Begriff Programmfenster Verwendung.

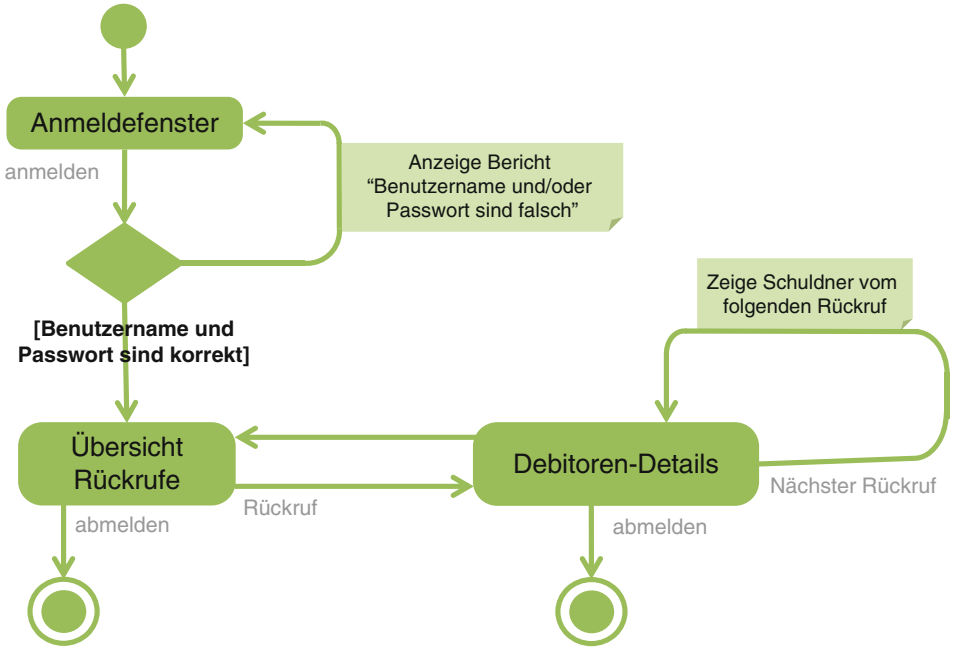
7.1 Dialogstruktur

Ein **Diagramm** zur **Dialogstruktur** (**screen flow diagram**) zeigt, wie ein Anwender zwischen den Programmfenstern einer Anwendung navigieren kann.

Ein Diagramm, das die Dialogstruktur wiedergibt, entspricht einem UML-**Zustandsautomat**, wenn jeder Zustand, in dem die Anwenderschnittstelle sich befinden kann, ein Programmfenster ist. Ein Programmfenster wird dabei durch ein abgerundetes Rechteck dargestellt, das den Namen des Programmfensters enthält.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_7

Beispiel



Nach dem Anmelden sieht der Debitorenverwalter eine Liste der Rückrufe, die er noch durchführen muss.

Das Startsymbol zeigt auf das Programmfenster, mit dem die Anwendung startet (Login-Fenster). Die anderen Pfeile geben die Navigation zwischen den Programmfenstern wieder.

Der Name eines Pfeils entspricht dem Namen der Schaltfläche, die der Anwender betätigt, wenn er die Navigation ausführt.

Bei Pfeilen ohne Namen hat die Schaltfläche den gleichen Namen wie das Programmfenster, zu dem der Anwender navigiert.

Das Symbol für den Endzustand gibt an, dass dieser Navigationsschritt die Anwendung beendet.

Dass das Anmeldungsfenster gezeigt wird, wenn der Anwender sich abgemeldet hat, wird dadurch ausgedrückt, dass der Abmeldungspfeil zum Anmeldungsfenster zeigt. In diesem Fall hätte die Dialogstruktur keinen Endzustand.

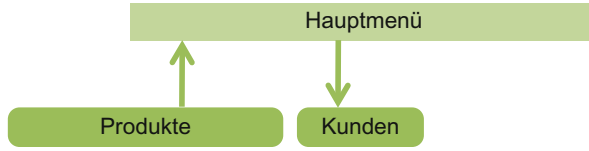
Genau wie in einem Aktivitätsdiagramm wird eine automatisch ablaufende Entscheidung durch eine Raute (Entscheidungsknoten) dargestellt, an deren Ausgängen Bedingungen stehen.

Beispiel

7.1.1 Hauptmenü

Über das **Hauptmenü** (**main menu**) kann der Anwender direkt zu anderen Programmfenstern navigieren.

Beispiel

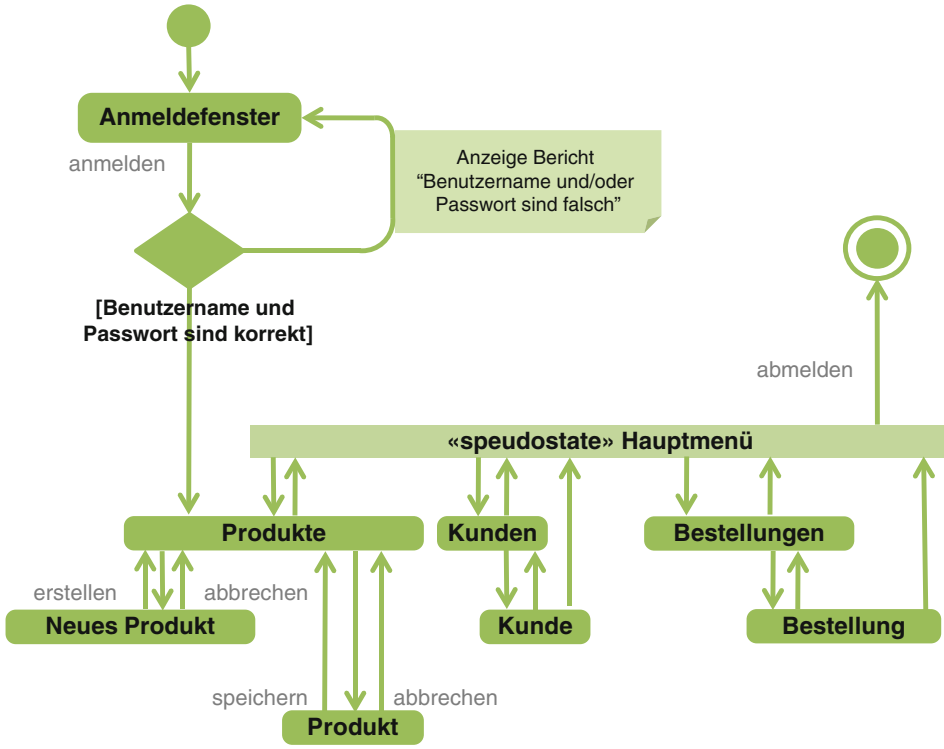


Über das Hauptmenü kann der Anwender vom Programmfenster „Produkte“ zum Programmfenster „Kunden“ navigieren.

Um darzustellen, dass das Hauptmenü kein Programmfenster (und insofern kein Zustand der Anwenderschnittstelle) ist, wird das Hauptmenü anders als ein Programmfenster dargestellt (z. B. durch hellere Farben und kursive Buchstaben).

Beispiel

Anwenderschnittstelle: Dialogstruktur des Produktverwalters



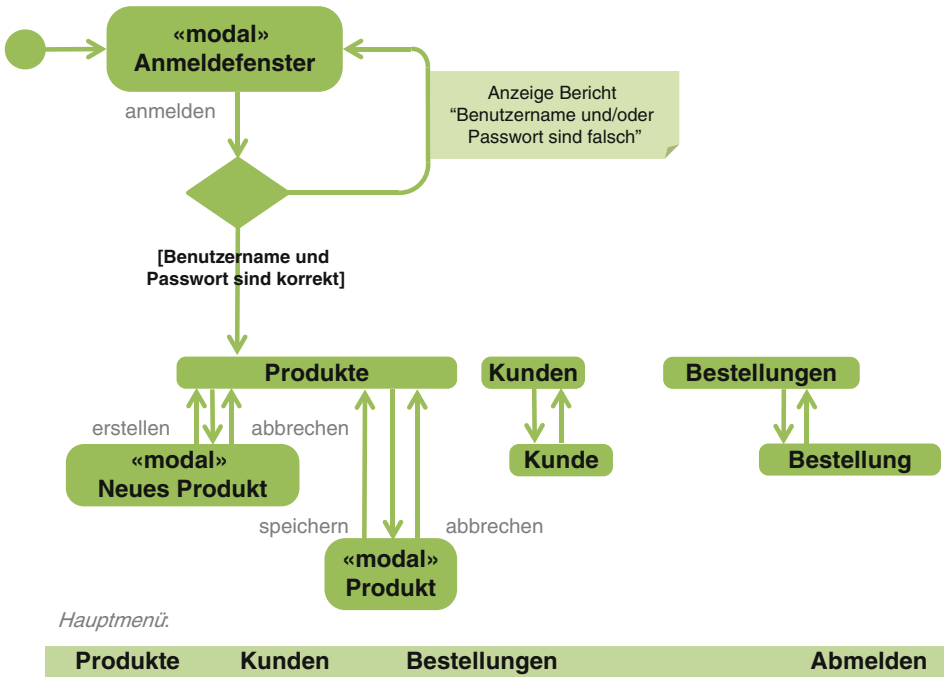
Jedem Programmfenster, zu dem der Anwender über das Hauptmenü navigieren kann, entspricht eine Schaltfläche, ein Hyperlink oder ein Menüpunkt im Hauptmenü.

7.1.2 Modales Fenster und nicht-modales Fenster

Ein **modales Fenster** (**modal screen**) sorgt dafür, dass sich die Anwendung in einem bestimmten **Modus** (**mode**) befindet. Solange das modale Programmfenster geöffnet ist, kann der Anwender zu keinem anderen Programmfenster navigieren. Zu einem modalen Fenster kann der Anwender auch nicht über das Hauptmenu navigieren. Ein Programmfenster, in dem das Hauptmenü hingegen verfügbar ist, wird entsprechend **nicht-modales Fenster** genannt (**non modal screen**).

Um die Navigation über das Hauptmenü zu modellieren, kann anstelle der Notation aus dem vorherigen Abschnitt auch der **Stereotyp** «**modal**» in Kombination mit einer Abbildung des Hauptmenüs verwendet werden.

Beispiel



Dieses Beispiel zeigt die gleiche Ablauflogik wie das vorangegangene Beispiel. Dieses Mal findet jedoch die Notation mit dem Stereotyp «modal» Anwendung.

In dieser Notation gibt es Programmfenster mit dem **Stereotyp** «**modal**». Die Programmfenster ohne den **Stereotyp** «**modal**» sind nicht-modal und haben darum Zugang zum Hauptmenü.

Das Hauptmenü wird hier durch eine explizite Darstellung modelliert. Die Darstellung des Hauptmenüs zeigt, zu welchen Fenstern der Anwender über das Hauptmenü navigieren kann und wie das Hauptmenü eingeteilt ist. Jeder Menüpunkt in der Darstellung trägt den Namen des Fensters, zu dem der Menüpunkt den Anwender navigiert.

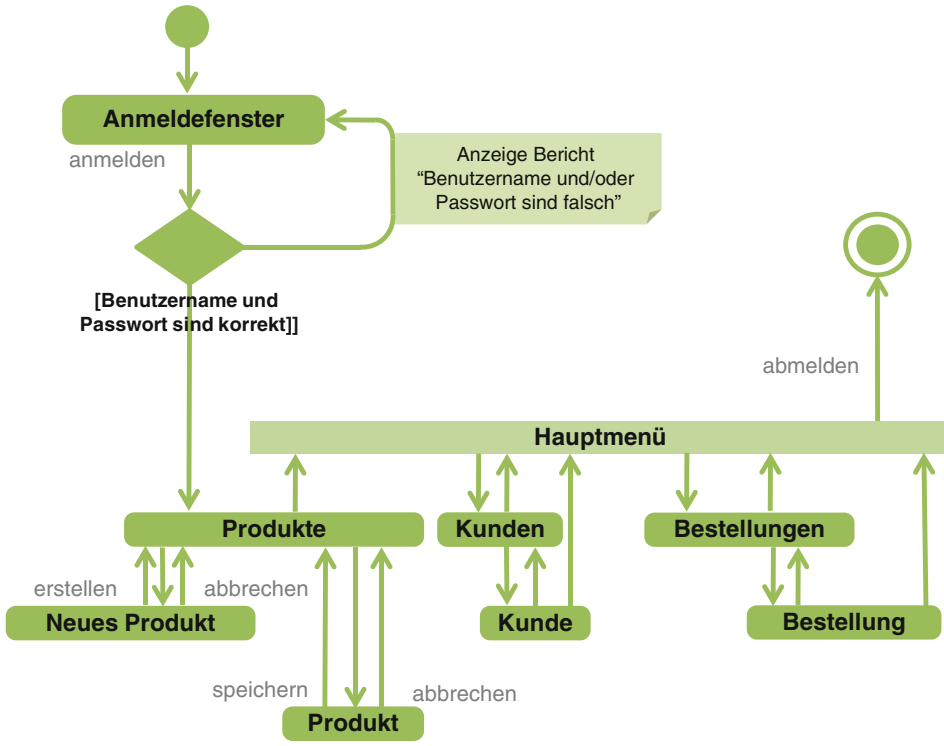
Der Nachteil der Notation mit «modal» ist, dass die Navigation nicht direkt aus den Pfeilen ersichtlich ist und dass auch nicht ersichtlich ist, was passiert, wenn der Anwender sich abmeldet. Der Vorteil ist, dass die Anzahl der Pfeile geringer ist. Letztendlich entscheidet der Modellierer, welche Notation er verwendet.

7.1.3 Anwenderrolle und Darstellung der Dialogstruktur

Wenn sich die Dialogführung in Abhängigkeit von der Rolle des Anwenders unterscheidet, dann wird der Name der Rolle deutlich im Titel der Darstellung der Dialogstruktur angegeben.

Beispiel

Dialogstruktur des Produktverwalters:

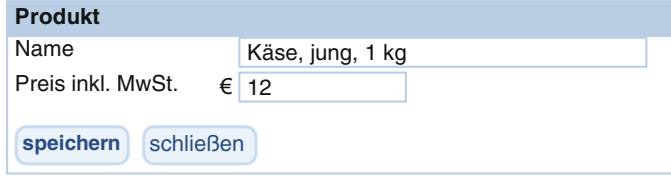


7.1.4 Pop-up oder Darstellung im Hauptfenster der Anwendung?

Als Pop-up

Ein Pop-up verdeckt (in der Regel teilweise) das Programmfenster, von dem der Anwender kommt, und lässt sich durch den Anwender verschieben.

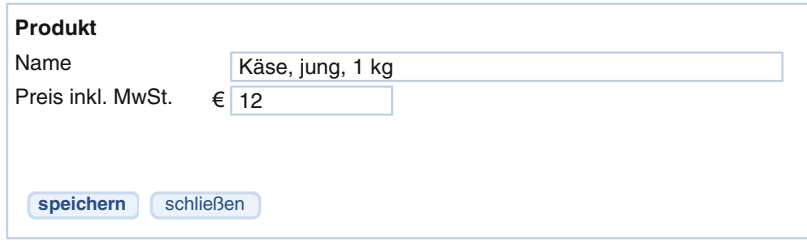
Beispiel



Im Hauptfenster

Das Programmfenster, zu dem der Anwender navigiert, wird hierbei im Hauptfenster der Anwendung dargestellt und das Programmfenster, von dem der Anwender kommt, ist nicht mehr sichtbar.

Beispiel



In beiden Fällen kann das Fenster, zu dem der Anwender navigiert, modal oder nicht-modal sein.

Der Vorteil eines Pop-ups ist, dass der Anwender das Fenster, von dem er kommt, noch sehen kann. Mehrere Pop-ups übereinander machen eine Anwendung jedoch schnell unübersichtlich.

Oft ist ein Pop-up vor allem dann praktisch, wenn der Anwender vom Pop-up immer (oder zumindest häufig) zurück zu dem Fenster navigiert, von dem er gekommen ist. In diesem Fall geschieht das automatisch durch das Schließen des Pop-ups.

Wenn der Anwender von einem Pop-up irgendwo anders hin navigiert, muss der Modellierer abwägen, ob das Pop-up automatisch geschlossen wird oder geöffnet bleibt. Auf der einen Seite erfordert das Arbeiten mit Pop-ups deshalb vom Modellierer besondere Sorgfalt. Auf der anderen Seite kann der sinnvolle Einsatz von Pop-ups helfen, die Nutzererfahrung zu verbessern. Im funktionalen Entwurf sind Pop-ups an ihrem Titelbalken erkennbar, mit dem sie über den Bildschirm verschoben werden können.

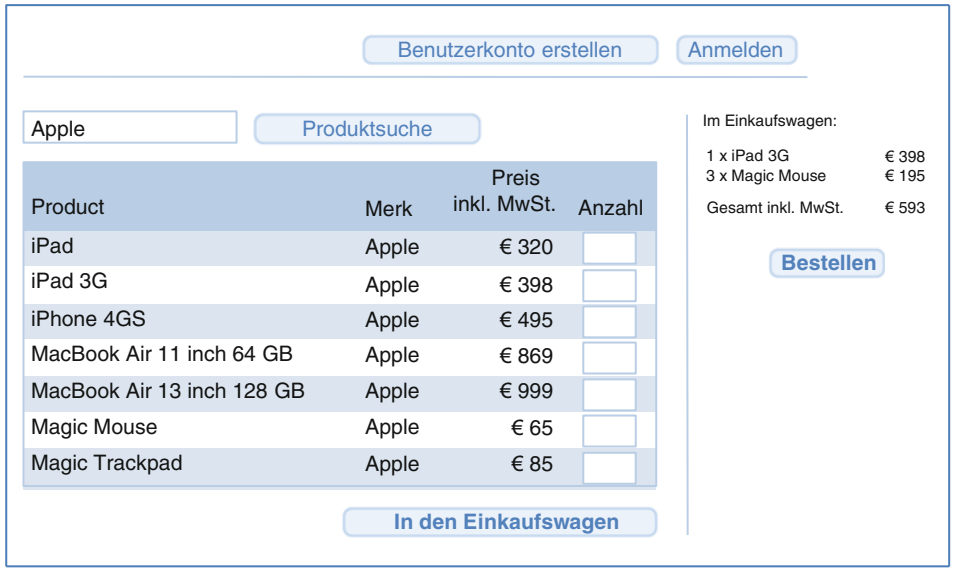
7.2 Programmfenster

Der Entwurf umfasst für jedes Programmfenster eine Abbildung, die zeigt, was auf dem Programmfenster sichtbar ist.

Der Name dieser Abbildung ist der Name des Programmfensters, der mit dem betroffenen abgerundeten Rechteck in der Darstellung der Dialogstruktur korrespondiert. Sofern notwendig, wird dieser um die Anwenderrolle ergänzt, die mit dem Programmfenster arbeiten kann.

Beispiel

Startfenster des Kunden:



Falls die Abbildung des Programmfensters nicht für sich selbst spricht oder das Verhalten sich nicht aus der Darstellung der Dialogstruktur ergibt, werden diese Angaben durch zusätzlichen Text unter der Abbildung des Programmfensters ergänzt.

7.2.1 Steuerelement für Attribute

Wenn ein Programmfenster die Möglichkeit bietet, Werte von Attributen einzufügen, sind die verwendeten **Steuerelemente** (**widget**, **control**) abhängig vom Attributtyp.

Beispiel

Typen mit Attributen:	Bildschirmelement	
amount	textfeld mit währungszeichen	€
boolean	kontrollkästchen	☐
date	datumsauswahl	freitag 6 april 2012 :::
datetime o. timestamp	datums-/zeitauswahl	freitag 6 april 2012 ::: 9:15
ennumeration	optionsfelder oder klappliste	◉ wahl 1 ○ wahl 2 ○ wahl 3 oder wahl 1 wahl 2
integer, float	textfeld	
string	textfeld	
time	zeitauswahl	9:15

Ob für eine Aufzählung dabei Optionsfelder oder eine Klappliste verwendet werden, ist abhängig vom verfügbaren Platz im Programmfenster.

- Optionsfelder haben den Vorteil, dass der Anwender alle Wahlmöglichkeiten sofort sieht und nur einmal klicken muss.
- Ein Dropdown-Listenfeld (Klappliste) hat den Vorteil, dass es weniger Platz benötigt.

7.2.2 Steuerelement für Assoziationen mit einzahliger Multiplizität

Einzahlige Multiplizität (**single mulitplicity**) bedeutet, dass die Assoziation maximal auf ein Objekt verweist (0..1 oder 1).

Beispiel

Verkäufer wählen:

Einem Kunden ist (maximal) **1 Verkäufer** zugeordnet.

Es stehen folgende alternative Steuerelemente zur Verfügung, um das Objekt zu wählen:

Situation:	Bildschirmelement	
Wenige Elemente auszuwählen und **genug Platz** verfügbar	**optionsfelder**	◉ wahl 1 ○ wahl 2 ○ wahl 3
Wenige Elemente auszuwählen und **wenig Platz** verfügbar	**klappliste**	wahl 1 wahl 2 wahl 3
Viele Elemente auszuwählen	**textfeld mit auswahlschaltfläche**	wahl 1 …

Die Betätigung der Auswahlschaltfläche zeigt ein Programmfenster, in dem der Anwender (ggf. unterstützt durch eine Suchfunktion) ein Objekt auswählen kann.

Beispiel Verkäufer wählen

Für ein Programmfenster mit Kundendaten gibt es folgende Alternativen, um die Auswahl des Verkäufers zu modellieren:

- Wenn die Anzahl der Verkäufer gering und im Programmfenster genug Platz vorhanden ist, dann wird für jeden Verkäufer ein Optionsfeld angezeigt.
- Wenn die Anzahl der Verkäufer gering und im Programmfenster nur wenig Platz vorhanden ist, dann wird ein Dropdown-Listenfeld (Klappliste) angezeigt, aus dem der Verkäufer ausgewählt werden kann.
- Wenn die Anzahl der Verkäufer groß ist, wird ein Textfeld für den auszuwählenden Verkäufer und eine Schaltfläche angezeigt. Nach dem Klicken auf die Schaltfläche erscheint ein Programmfenster, in dem der Verkäufer gesucht und ausgewählt werden kann.

7.2.3 Steuerelement für Assoziationen mit mehrzähliger Multiplizität

Mehrzählige Multiplizität (**multiple multiplicity**) bedeutet, dass die Assoziation auf ein oder mehrere Objekte verweist (multiplicity * oder 1..*).

Beispiel Kunde wählen

Ein Verkäufer ist mit **0 oder mehr Kunden** verbunden.

Das Interaktionselement zum Anzeigen und Auswählen von Objekten kann auf Basis folgender Teilelemente realisiert werden:

- Einer Liste, um die ausgewählten Objekte anzuzeigen.
- Einer Schaltfläche, um Objekte der Liste hinzuzufügen.
- Einer Schaltfläche, um Objekte aus der Liste zu entfernen.

Beispiel Kunde auswählen

Das rote Kreuz in jeder Spalte ist eine Schaltfläche, um den Kunden zu entfernen.

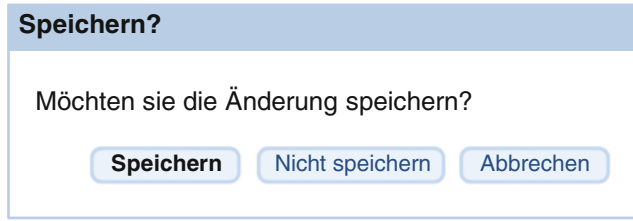
Mögliche alternative Lösungen sind:

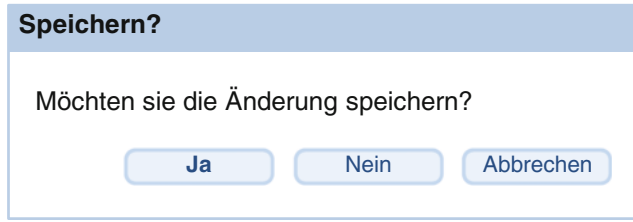
1. Eine Liste, die alle Objekte enthält, die ausgewählt werden können. Ein Kontrollkästchen je Objekt gibt an, ob das Objekt ausgewählt ist oder nicht.
2. Eine Liste, die verbundene Objekte enthält und daneben eine Liste, die nicht-verbundene Objekte enthält sowie Schaltflächen, mit denen Objekte von einer Liste in die andere Liste verschoben werden können.

7.2.4 Information in der Bezeichnung von Schaltflächen

Im folgenden Beispiel ist das erste Meldungsfenster besser geeignet als das zweite.

Beispiel



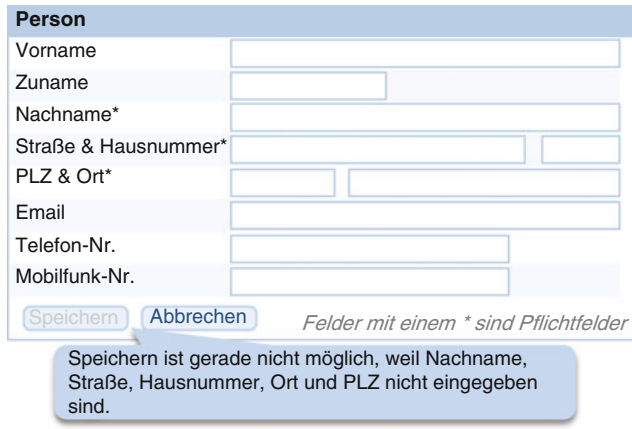


Im ersten Fall erkennt der Anwender direkt anhand der Bezeichnung der Schaltfläche, was ein Anklicken bewirkt. Im zweiten Fall ist er gezwungen, den gesamten Text oberhalb der Schaltfläche zu lesen und den Sinn zu interpretieren. Das kostet ihn mehr Zeit und Energie.

7.2.5 Nicht verwendbare Schaltfläche

Eine Schaltfläche, die für den Anwender nur manchmal verwendbar ist, wird, wenn sie nicht verwendbar ist, am besten ausgegraut. Wenn der Anwender in diesem Fall mit der Maus über die Schaltfläche fährt oder auf die Schaltfläche klickt, kann ein Erläuterungstext angezeigt werden, der erklärt, warum die Schaltfläche gerade nicht verwendbar ist.

Beispiel



Eine Schaltfläche, die für einen bestimmten Anwender nie verwendbar ist, wird am besten ausgeblendet, weil der Anwender nichts davon hat, diese Schaltfläche zu sehen.

7.2.6 Übersichtlichkeit oder Vollständigkeit

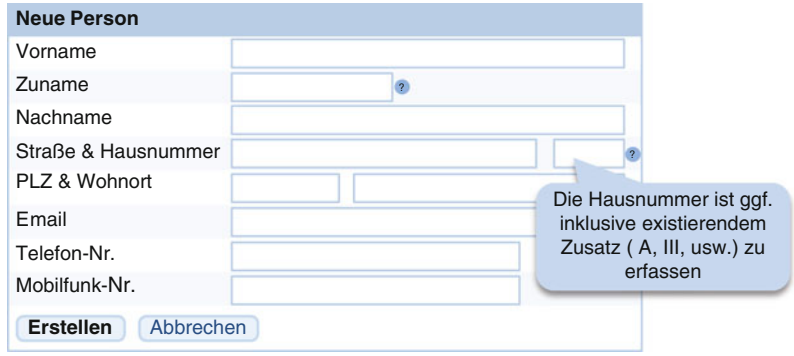
Es ist visuell ansprechend, wenn Programmfenster nicht zu viele Elemente enthalten. Insbesondere Anwendern, die seltener mit der Anwendung arbeiten, hilft es, sich besser zu Recht zu finden.

Anwender wollen bei der Durchführung ihrer Aufgaben so wenig wie nötig scrollen und navigieren müssen. Das gilt vor allem für Anwender, die häufig mit der Anwendung arbeiten. Es ist Aufgabe des Modellierers, das richtige Gleichgewicht zu finden.

7.2.7 Hilfefunktion

Es gibt verschiedene Möglichkeiten, neuen und unerfahrenen Anwendern unterstützende Informationen zu Steuerelementen zu geben, ohne dass die Erklärungstexte mit Blick auf erfahrene Anwender kostbaren Raum in den Programmfenstern verschwenden.

1. Zeige die Erklärung in einem Meldungsfenster, wenn der Anwender auf ein kleines Hilfesymbol klickt, das neben dem zu erklärenden Steuerelement steht, oder mit seiner Maus darüberfährt. Der Vorteil ist, dass neue Anwender direkt erkennen, dass Erläuterungen verfügbar sind. Gleichzeitig benötigt diese Hilfefunktion wenig Platz, wie das folgende Beispiel zeigt.



2. Zeige die Erklärung, wenn der Mauszeiger über das Steuerelement gehalten wird (Tooltip). Der Vorteil dieser Möglichkeit ist, dass sie keinen zusätzlichen Platz kostet und die Harmonie der Fensterdarstellung fördert, weil keine zusätzlichen Fensterelemente (Hilfesymbole) notwendig sind.
3. Zeige die Erklärung in einem Hilfefenster. Dieses Hilfefenster ist auf Wunsch gleichzeitig mit dem eigentlichen Programmfenster selbst sichtbar. Der Nachteil ist, dass es Zeit und Aufwand kostet, im Hilfefenster die Erklärung zu einem bestimmten Fensterelement zu finden.

7.3 Interaktion

Der Anwender interagiert mit der Anwendung in vielfältiger Weise. Der Modellierer muss dabei stets Entscheidungen treffen, wie die Interaktion stattfinden soll. In den nachfolgenden Abschnitten wird eine Auswahl von Punkten, die die Interaktion des Anwenders betreffen, näher betrachtet.

7.3.1 Häufigste Verarbeitungsschritte

Im Folgenden werden Gestaltungsfragen erläutert, die aus dem Sachverhalt resultieren, dass bestimmte Verarbeitungsschritte häufiger durchgeführt werden als andere.

Der Anwender arbeitet von oben nach unten

Während des Einfügens von Werten in Steuerelemente arbeitet der Anwender normalerweise von oben nach unten.

Die Schaltfläche, die Anwender nach dem Ausfüllen betätigen, sollte am unteren Rand des Programmfensters stehen. Um diese Schaltfläche so leicht wie möglich auffindbar zu machen, kann sie am besten unterhalb (des Labels) des Steuerelements stehen, in das der Anwender zuletzt Werte eingefügt hat. Positioniere diese Schaltfläche entsprechend nicht rechts im Programmfenster, wenn sich dieses Steuerelementlinks im Programmfenster befindet.

Auffallende und weniger auffallende Steuerelemente

Anwender finden sich am einfachsten zurecht, wenn die Schaltflächen, die zur Durchführung des primären Prozesses verwendet werden, auffallender sind als die übrigen Schaltflächen.

Beispiel

In diesem Beispiel fällt die Schaltfläche „Speichern“ durch die fett dargestellten Buchstaben auf.

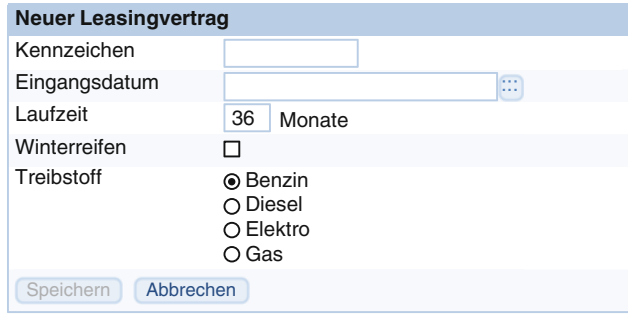
Eine Schaltfläche kann durch die Farbe oder die Dicke der Buchstaben auffallen oder durch den Einsatz weniger auffallender Steuerelemente wie Hyperlinks.

7.3.2 Nutzung von Standardwerten und Mussfeldern

Wenn der Anwender in ein Steuerelement häufig den gleichen Wert einfügt, spart es Zeit, wenn der Wert automatisch eingefügt wird. Dieser automatisch eingefügte Wert wird als

Standardwert (**default value**) des Steuerelements bezeichnet. Wenn es nötig ist, kann der Anwender den automatisch eingefügten Wert ändern.

Beispiel

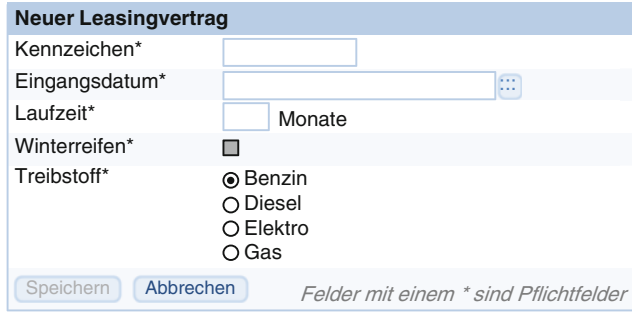


In diesem Beispiel ist die Laufzeit standardmäßig 36 Monate, das Winterreifen-Kontrollkästchen ist standardmäßig nicht ausgewählt und der Treibstoff ist standardmäßig Benzin.

Wenn die Werte automatisch eingetragen sind, gibt es jedoch eine erhöhte Wahrscheinlichkeit, dass der Anwender nicht daran denkt, die Werte zu ändern, wenn dort eigentlich andere Werte eingetragen werden müssten. Die Wahrscheinlichkeit für falsche Werte nimmt deshalb etwas zu.

Um den Anwender dazu zu bringen, eine explizite Auswahl zu treffen, kann es darum besser sein, die Steuerelemente nicht von vornherein mit Standardwerten zu füllen.

Beispiel



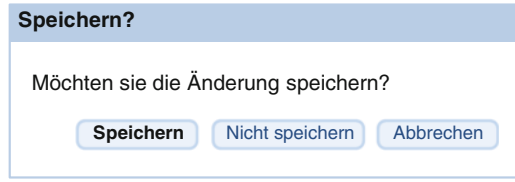
In diesem Beispiel sind für die Laufzeit, Winterreifen und Treibstoffsorte keine Werte eingetragen. Solange diese Werte nicht eingetragen sind, bleibt die Schaltfläche „Speichern“ ausgegraut.

Es liegt im Ermessen des Modellierers, zwischen den Zielen der Benutzerfreundlichkeit und der Datenqualität abzuwägen: Benutzerfreundlichkeit durch das automatische Einfügen von Standardwerten oder Sicherheit für richtige Werte aufgrund des notwendigen Ausfüllens durch den Anwender.
Bei Textfeldern sorgen die meisten Webbrowser durch die Funktion „**autocomplete**“ dafür, dass es Anwendern erleichtert wird, einen zuvor eingetragenen Wert erneut einzutragen.

7.3.3 Modale Frage und Mitteilung

Eine wichtige Frage oder Mitteilung, die der Anwender lesen soll, bevor er in der Dialogführung fortfährt, wird in einem modalen Fenster gezeigt, das der Anwender schließen muss, bevor er fortfährt.

Beispiel



7.3.4 Nicht-modale Frage und Mitteilung

Eine Frage oder Mitteilung, die rein informativen Charakter hat und den Anwender bei seiner Arbeit nicht stören soll, wird als **nicht-modale Frage** oder als **nicht-modale Mitteilung** angezeigt.

Meistens wird eine nicht-modale Mitteilung an einer bestimmten Stelle im oberen Fensterbereich gezeigt und zwar so, dass sie keine anderen Informationen überdeckt.

Beispiel

Nach dem Speichern von Daten sieht der Anwender folgendes:

Die Änderungen in den Kundendaten von Nico de Haas sind gespeichert.

7.3.5 Explizites und implizites Speichern

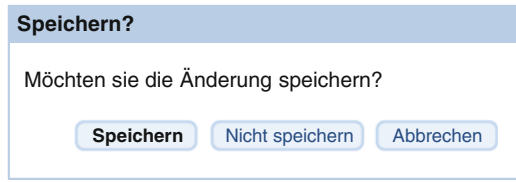
Für das Speichern von Daten kann der Modellierer zwischen folgenden Möglichkeiten wählen:

1. **Explizit**, wenn der Anwender die Schaltfläche „Speichern“ betätigt. Dies hat den Vorteil einer erhöhten Kontrollier- und Nachvollziehbarkeit. Wenn der Anwender z. B. zusammenhängende Änderungen noch nicht vollständig eingetragen hat (z. B. schon den Straßennamen, aber noch nicht die PLZ), macht die Schaltfläche „Speichern“ ihm deutlich, dass er dabei ist, Änderungen zu vollziehen.
2. **Implizit**, automatisch wenn der Anwender z. B. ein Fenster verlässt. Der Vorteil ist, dass der Anwender nicht so häufig klicken muss.

Um die Erwartungskonformität zu fördern, sollte die gewählte Vorgehensweise, sofern möglich, überall in der Anwendung eingesetzt werden.

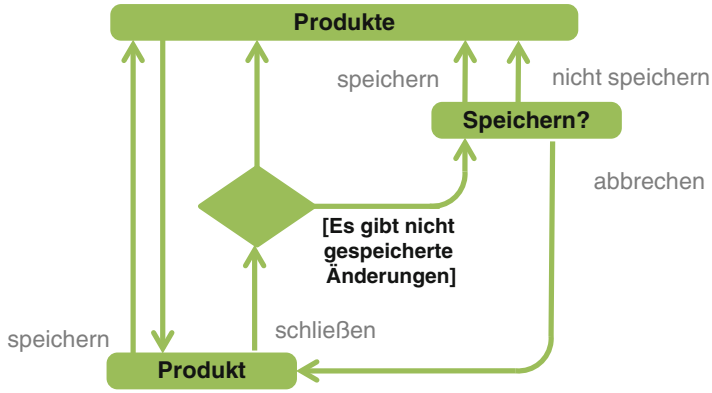
Wenn das Speichern explizit passiert und ein Anwender (ohne die vollzogenen Änderungen zu speichern) zu einem anderen Fenster navigiert, dann sollte ein Mitteilungsfenster abfragen, ob die Änderungen gespeichert werden sollen.

Beispiel



Beispiel Detailfenster eines Produkts

Wenn die Änderungen explizit gespeichert werden, ist die Dialogführung wie folgt:



Wenn das Mitteilungsfenster „Speichern?“ in jeder Darstellung der Dialogstruktur gezeichnet würde, hätte dies eine unnötige Komplexität zur Folge. Deswegen reicht es aus, dieses Verhalten an einer Stelle im funktionalen Entwurf zu beschreiben und anzumerken, dass es überall in der Anwendung so umgesetzt wird.

7.3.6 Löschen: Bestätigen oder rückgängig machen

Um das Risiko zu verringern, dass ein Anwender Daten versehentlich endgültig löscht, hat der Modellierer folgende Möglichkeiten:

1. Rückgängig machen: Die Option einführen, das Löschen von Daten rückgängig zu machen. Das hat den Vorteil, dass der Anwender weniger Mausklicks ausführen muss, um etwas zu löschen. Ein anderer Vorteil ergibt sich, wenn diese Funktion Bestandteil einer umfassenden Funktion „Rückgängig machen“ (**undo**) ist. Wenn auch nicht notwendigerweise in Kombination mit einem Audit Trail, der in Kapitel 10 dieses Buches beschrieben wird.
2. Vor dem endgültigen Löschen eine Bestätigung erfragen. Das hat den Vorteil, dass es einfacher (und entsprechend kostengünstiger) umzusetzen ist.

Um die Erwartungskonformität zu fördern, sollte die gewählte Vorgehensweise überall in der Anwendung Einsatz finden.

Auf die Möglichkeit, das Löschen von Daten rückgängig zu machen, kann in einer nicht-modalen Meldung hingewiesen werden, die erscheint, wenn der Anwender Daten gelöscht hat.

Beispiel

Die Möglichkeit, das Löschen von Daten rückgängig zu machen, kann auch durch einen elektronischen Papierkorb umgesetzt werden, aus dem der Anwender Daten wieder hervorholen kann.

7.3.7 Tastatur anstelle der Maus

Viele erfahrene Nutzer verwenden lieber die Tastatur als die Maus, weil sie dann während der Eingabe die Hände nicht von der Tastatur nehmen müssen. Es ist weniger zeitaufwendig, Finger auf der Tastatur als einen Mauszeiger mit der Maus auf einen Punkt des Bildschirms zu bewegen.

Beispielsweise lässt sich die Tabulatortaste verwenden, um zum folgenden Steuerelement (rechts vom aktuell aktiven Fensterelement) zu springen. Wenn sowohl rechts als auch unter dem aktuellen Steuerelement andere Steuerelemente stehen, kann der Modellierer mit Spalten arbeiten:

- Wenn das Steuerelement rechts vom aktuell aktiven Steuerelement in einer anderen Spalte steht, dann setzt die Tabulatortaste den Fokus auf das Steuerelement unterhalb des aktiven Steuerelements.
- Ansonsten setzt die Tabulatortaste den Fokus auf das Steuerelement rechts vom aktuell aktiven Steuerelement.

Steuerelemente, die der Anwender aktivieren möchte, die aber nicht rechts oder unterhalb des aktiven Steuerelements liegen, können auch über eine Tastenkombination erreicht werden. Diese Tastenkombinationen sollten dann gut sichtbar sein, z. B. als Name des Steuerelements.

Beispiel

Es ist üblich, dass der Buchstabe der Taste, die (zusammen mit der Alt-Taste) den Fokus auf ein Steuerelement setzt, in der Bezeichnung (wie im Beispiel) unterstrichen ist.

Um die Darstellung „ordentlich“ und „aufgeräumt“ zu halten, kann die Unterstreichung nur angezeigt werden, wenn die Alt-Taste gedrückt ist.

7.4 CRUD-Muster

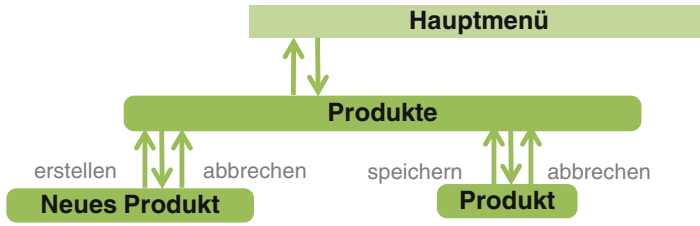
Dort, wo ein Anwender in einer Anwendung Objekte anlegen, lesen, ändern und löschen kann, wird häufig eine gleichartige Kombination von Steuerelementen und Fenstern genutzt. Diese Kombination von Steuerelementen und Fenstern bildet ein **CRUD-Muster** (**CRUD pattern**).

Beispiel CRUD-Muster für Produkte

Über das Hauptmenü kann der Anwender eine Übersicht von Produkten aufrufen:

Beispiel

Produkte	+ Neues Produkt…	
Käse, reif, 1 Kilogramm	€ 14,-	X
Käse, jung, 1 Kilogramm	€ 12,-	X
Käse, alt, 1 Kilogramm	€ 16,-	X
Quark, halbfett, 500 Gramm	€ 3,-	X
Quark, vollfett, 500 Gramm	€ 4,-	X
Milch, 3,5% Fett, 1 Liter	€ 1,50	X
Joghurt, 1 Liter	€ 2,-	X



Von diesem Listenfenster aus kann ein Erfassungsfenster aufgerufen werden, in dem sich ein neues Produkt anlegen lässt.

Neues Produkt

Name: Schlagsahne, vollfett, 250 Gramm

Preis inkl. Mwst.: € 3

Erstellen | Abbrechen

Vom folgenden Listenfenster aus können auch Details zu einem Produkt betrachtet und geändert werden.

Produkt

Name: Käse, jung, Kilogramm

Preis inkl. Mwst.: € 12

Speichern | Abbrechen

Darüber hinaus umfasst das Listenfenster auch Schaltflächen, um Produkte zu löschen.

Oft gleicht das Programmfenster, um ein neues Objekt anzulegen (Erfassungsfenster), dem Programmfenster, das dazu dient, vorhandene Objekte zu betrachten und zu ändern. Es hängt von der Entscheidung des Modellierers ab, ob er in seinem Modell beide Varianten des Programmfensters darstellt oder ob er nur eine Variante darstellt und beschreibt, wo die Unterschiede liegen.

Pop-up: Vom Listenfenster aus wird ein Pop-up-Fenster mit Detailfeldern des Objekts gezeigt. Dies gilt für ein bestehendes Objekt, das im Listenfenster zum Bearbeiten selektiert wurde ebenso wie für ein neu anzulegendes Objekt.

Navigieren: Vom Listenfenster aus kann der Anwender zu einem Programmfenster mit Detailfeldern des Objekts navigieren. Dies gilt für ein bestehendes Objekt, das im Listenfenster zum Bearbeiten selektiert wurde ebenso wie für ein neu anzulegendes Objekt.

Master/Detail: Von der Übersichtsliste im Listenfenster (**Master**) kann ein Objekt selektiert werden. Die Details des ausgewählten Objekts werden unter- oder oberhalb der Übersichtsliste im selben Programmfenster (im **Detail**) angezeigt und können dort auch geändert werden.

Inline Editing: Im Listenfenster können Objekte direkt geändert werden. Zum einen können die Felder direkt als bearbeitbare Steuerelemente (Textfelder, Dropdown-Listenfelder usw.) angezeigt werden. Das hat den Vorteil, dass der Anwender direkt erkennen kann, dass sie änderbar sind. Es ist jedoch auch möglich, die Anwendung so zu gestalten, dass aus den Feldern erst dann bearbeitbare Steuerelemente (Textfelder, Dropdown-Listenfelder usw.) werden, wenn der entsprechende Datensatz ausgewählt wurde oder der Mauszeiger über dem entsprechenden Feld schwebt. Die Liste sieht dann weniger unruhig aus.

7.5 Konsistente Anwendererfahrung

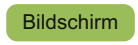
Beim Entwerfen von Anwenderschnittstellen ist es wichtig, eine konsistente Nutzererfahrung zu bieten, die sich ergibt, wenn sich die Anwenderschnittstelle erwartungskonform verhält.

Entwurfsentscheidungen – von der Farbwahl über das Positionieren von Schaltflächen bis hin zum genauen Ablauf des Speicherns von Daten – sollen, sofern möglich, konsequent in allen Programmfenstern der Anwendung angewendet werden. Nur so erhält der Anwender eine konsistente Nutzererfahrung, die für die Benutzerfreundlichkeit und Qualität der Anwendung essenziell ist.

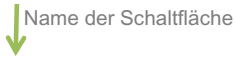
In vielen Fällen existieren organisationsspezifische **Gestaltungsrichtlinien** (**style guides**). In der Regel kommt dies der Konsistenz zugute. Häufig existieren auch Vorgaben für das Betriebssystem, für das die Anwendung entworfen wird.

7.6 Übersicht zur Anwenderschnittstelle

Die Darstellung der Dialogstruktur zeigt, wie ein Anwender zwischen den Programmfenstern der Anwendung navigiert. Die Darstellung der Dialogstruktur ist ein UML-Zustandsautomat und kann folgende Symbole umfassen:



Jedes Programmfenster entspricht einem Zustand, in dem die Anwenderschnittstelle sich befinden kann. Diese wird durch ein abgerundetes Rechteck dargestellt.

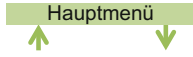


Ein Pfeil zeigt die Navigation von einem Programmfenster zu einem anderen Programmfenster und impliziert gleichzeitig einen Zustandsübergang. Neben dem Pfeil steht der Name der Schaltfläche, die der Anwender betätigen muss, um die Navigation auszuführen..

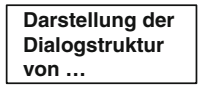
Das Symbol Anfangszustand zeigt zum Programmfenster, mit dem die Anwendung beginnt..

Das Symbol Endzustand gibt an, dass durch diese Navigation, die Anwendung beendet wird.

Eine automatisch ablaufende Entscheidung wird in der abgebildeten Dialogstruktur genauso dargestellt wie in einem Geschäftsprozess: als eine Raute, an deren Ausgängen Bedingungen stehen (Entscheidungsknoten).



Vom Hauptmenü aus kann der Anwender direkt zu anderen Fenstern navigieren.



Oberhalb der Darstellung der Dialogstruktur zeigt der Titel die Rolle der Anwender, die entsprechend der abgebildeten Dialogführung durch die Anwendung navigieren können.

Für jedes Programmfenster umfasst der Entwurf eine Abbildung, die zeigt, was im Programmfenster sichtbar ist (**Mock up**).

Die meisten Steuerelemente in einem Programmfenster bieten dem Anwender die Möglichkeit, Werte von Attributen und Assoziationen zu sehen und zu ändern.

Welche Steuerelemente hierfür notwendig sind, hängt vom Attributtyp des betroffenen Attributs und der Multiplizität der betroffenen Assoziation ab.

7.7 Übung zur Anwenderschnittstelle

- Zeichne das Hauptmenü deiner Anwendung.
- Zeichne für jedes Programmfenster deiner Anwendung ein abgerundetes Rechteck.
- Zeichne Pfeile, die zeigen, wie der Anwender zwischen diesen abgerundeten Rechtecken navigiert.

- Vervollständige die Darstellung der Dialogstruktur mit Techniken, die du in diesem Kapitel gelernt hast.
- Erstelle von den Programmfenstern deiner Anwendung grobe Skizzen. Achte darauf, dass der Name jedes Programmfensters mit dem Namen eines abgerundeten Rechtecks aus der Darstellung der Dialogstruktur korrespondiert.

7.8 Checkliste zur Anwenderschnittstelle

Die häufigsten Fehler beim Entwerfen einer Anwenderschnittstelle können durch die Beantwortung folgender Fragen identifiziert werden:

Fragen

- Ist der Name jedes abgerundeten Rechtecks in der Darstellung der Dialogstruktur der gleiche wie der Name der groben Skizze des entsprechenden Programmfensters?
- Können alle in der CRUD-Matrix modellierten Verarbeitungsschritte durch die berechtigten Anwender über die Anwenderschnittstelle ausgeführt werden?
- Können alle in den Geschäftsprozessen modellierten Verarbeitungsschritte durch die berechtigten Anwender ausgeführt werden?
- Sind alle Assoziationen innerhalb der Anwenderschnittstelle sichtbar? Sind sie dort, wo nötig, auch änderbar?
- Werden alle Attribute innerhalb der Anwenderschnittstelle angezeigt? Sind sie entsprechend der CRUD-Matrix zu bearbeiten?

Wenn die Antwort auf eine dieser Fragen „Nein“ lautet, dann ist dieser Aspekt des Entwurfs der Anwenderschnittstelle zu korrigieren.

Geschäftsregeln

8

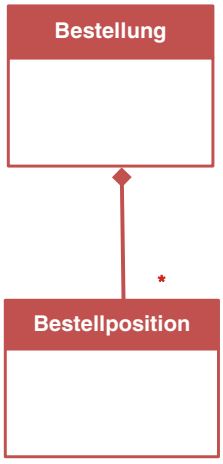
Zusammenfassung

Geschäftsregeln sind Regeln, die die Daten, die den Geschäftsprozess durchlaufen, aus fachlicher Sicht erfüllen müssen. In diesem Kapitel wird gezeigt, wie sich diese mittels UML-Klassendiagrammen, Aktivitätsdiagrammen und Dialogstrukturen abbilden lassen.

8.1 Modellbestandteil oder gesonderte Beschreibung

Einige **Geschäftsregeln** (**business rules**) sind Bestandteil von Modellen, die bereits in diesem Buch beschriebenen wurden.

Beispiel



H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_8

Eine Bestellposition gehört zu einer Bestellung (Geschäftsregel 1).

Das Löschen einer Bestellung löscht automatisch auch seine Bestellpositionen (Geschäftsregel 2).

Beide Geschäftsregeln werden durch die Kompositionsbeziehung ausgedrückt und sind Bestandteil des Klassendiagramms.

Andere Geschäftsregeln müssen ggf. gesondert beschrieben werden.

Beispiel

Wenn ein Kunde in einer Bestellung Waren im Wert von mehr als 1.000 Euro bestellt, bekommt er einen Nachlass von 2 Prozent.

Geschäftsregeln dieser Art lassen sich im funktionalen Entwurf unterhalb des Modells beschreiben, mit denen sie am stärksten verbunden sind (in diesem Fall unter dem Geschäftsprozess einer Bestellung). Alternativ können sie auch in einem gesonderten Teil mit Geschäftsregeln beschrieben werden.

8.2 Bedingung in einem Geschäftsprozess

Eine Bedingung, die erfüllt sein muss, damit ein Geschäftsprozess von einer automatisch ablaufenden Entscheidung ausgehend einen bestimmten Zustand erreicht, steht zu Beginn des betreffenden Zustands am Ausgang des Entscheidungsknotens.

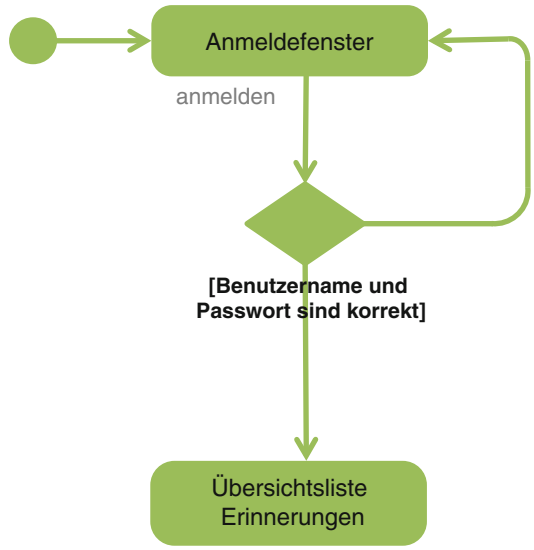
Beispiel

Wenn hier die Ware die Bedingung „ausreichend auf Vorrat“ erfüllt, dann verändert sich der Zustand automatisch in „zu versenden“. Unter dem Prozessdiagramm oder an einer anderen Stelle innerhalb des funktionalen Entwurfs muss genau beschrieben werden, was die Bedingung „ausreichend auf Vorrat“ beinhaltet (also wie die automatisch ablaufende Entscheidung kontrolliert, ob ausreichend Vorrat vorhanden ist).

8.3 Bedingung in der Darstellung der Dialogstruktur

Eine Bedingung, die erfüllt sein muss, damit innerhalb der Dialogführung der Übergang zwischen zwei bestimmten Programmfenstern erfolgen kann, steht am entsprechenden Ausgang der automatisch ablaufenden Entscheidung, d. h. am Anfang des Pfeils zum Zielfenster.

Beispiel

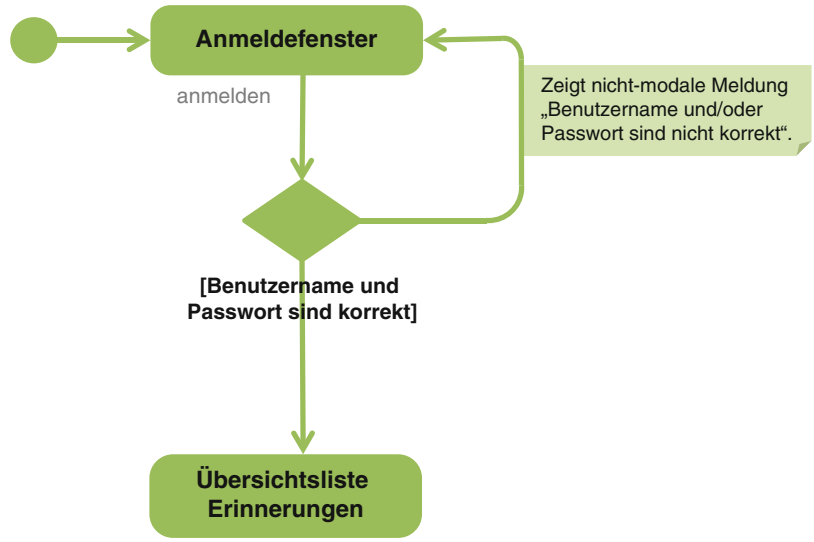


Wenn der Benutzername und das Passwort korrekt sind, wird das Programmfenster „Übersichtsliste Erinnerungen“ gezeigt.

8.4 Automatisches Ereignis in der Darstellung der Dialogstruktur

Wenn beim Übergang zwischen zwei Programmfenstern innerhalb der Dialogstruktur ein bestimmtes Ereignis auftreten muss (z. B. das Anzeigen einer nicht-modalen Meldung am oberen Fensterrand), wird dies durch eine rechteckige Notiz am betreffenden Pfeil dargestellt.

Beispiel



Wenn der Benutzername und/oder das Passwort nicht korrekt sind, wird der Übergang „zurück zum Anmeldefenster“ durchlaufen. Das Ereignis, das hier auftritt, ist das Anzeigen der nicht-modalen Meldung.

8.5 Zeitpunkt, zu dem eine Bedingung einzuhalten ist

Wenn eine Geschäftsregel eine Bedingung ist, die Daten einhalten müssen, hat der Modellierer verschiedene Zeitpunkte zur Auswahl, um die Einhaltung durchzusetzen.

1. **Jederzeit und direkt:**
   Die Anwenderschnittstelle erzwingt die permanente Einhaltung der Bedingung.
   Wenn umsetzbar, dann ist das häufig die benutzerfreundlichste Art, um die Einhaltung einer Bedingung sicherzustellen.

   Beispiel:
   Ein Datum ist als Standardwert vorgegeben und kann nur über das verfügbare Kalender-Steuerelement geändert werden. Hierdurch ist es z. B. nicht möglich, dass der 31. Februar eingegeben wird.

   Das ist häufig die benutzerfreundlichste Art und Weise, um die Einhaltung einer Bedingung sicherzustellen. Aber sie ist nicht in jedem Fall umsetzbar.

2. **Vor dem Speichern:**
   Daten können nur dann gespeichert werden, wenn sie die vorgegebene Bedingung erfüllen.

Beispiel:
Wenn der Anwender versucht, Daten zu speichern und diese eine vorgegebene Bedingung nicht erfüllen, zeigt eine Meldung den Fehler an und die Daten werden nicht gespeichert.

Der Vorteil ist, dass die gespeicherten Daten die Bedingung immer erfüllen. Der Nachteil ist, dass der Anwender die Daten nur speichern kann, wenn er Zeit hat, die Daten so anzupassen, dass sie alle Bedingungen erfüllen.

3. **Vor der Verarbeitung:**
   Daten können auch dann gespeichert werden, wenn sie vorgegebene Bedingungen nicht erfüllen. Erst bei der weiteren Verarbeitung werden die Bedingungen durchgesetzt.

   Beispiel:
   Die Aktion „bestellen“ kann in einem Webshop erst ausgeführt werden, wenn die Bedingung, dass jede Bestellung eine Bestellposition umfasst, erfüllt ist.

Es ist eine Entscheidung des funktionalen Modellierers, zu welchem Zeitpunkt die Bedingung erfüllt sein muss. Ggf. kann sich dieser Zeitpunkt auch von dem Zeitpunkt unterscheiden, in dem der Anwender die Meldung erhält, dass eine Bedingung nicht eingehalten wird.

Um die konzeptionelle Integrität der Anwendung zu fördern, empfiehlt es sich, verschiedene Bedingungen zur gleichen Zeit durchzusetzen.

Die Multiplizität wird nicht innerhalb eines Klassendiagramms dargestellt, wenn der Zeitpunkt der Durchsetzung ein anderer ist als „jederzeit und direkt“. In diesen Fällen wird die Multiplizität als gesonderte Geschäftsregel beschrieben. Die im Klassendiagramm angegebenen Multiplizitäten gelten immer „jederzeit und direkt“.

8.6 Übung zu Geschäftsregeln

- Beschreibe deine Geschäftsregeln für die Anwendung, für die du zuvor die Anforderungsspezifikation, das Klassendiagramm usw. erstellt hast.

Kopplung zwischen Systemen und Komponenten

9

Zusammenfassung

UML bietet verschiedene Möglichkeiten, um Verknüpfungen zwischen Systemen darzustellen. In diesem Kapitel wird gezeigt, wie Schnittstellen und Abhängigkeiten mithilfe von Komponentendiagrammen und Paketdiagrammen modelliert werden sowie den zeitlichen Ablauf von Objekten und Methoden innerhalb eines Sequenzdiagramms.

Ein Informationssystem (meistens abgekürzt durch System) wird durch folgendes Symbol dargestellt.

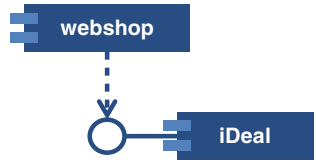
Eine Schnittstelle eines Systems, die ein anderes System nutzen kann, wird durch folgendes Symbol dargestellt.

9.1 Nutzung einer Schnittstelle und Abhängigkeit

Wenn ein System die Schnittstelle eines anderen Systems nutzt, wird das durch eine gestrichelte Linie, die vom System zum benutzten System führt, dargestellt.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_9

Beispiel



Dieser Webshop verwendet die Schnittstelle des iDeal-Systems für die Bezahlung.

Der gestrichelte Pfeil drückt gleichzeitig eine **Abhängigkeit** (**dependency**) aus. In diesem Beispiel ist der Webshop abhängig vom iDeal-System. Das bedeutet, dass:

- der Modellierer des Webshops von der Existenz des iDeal-Systems weiß.
- der Modellierer des iDeal-Systems nicht wissen muss, dass der Webshop realisiert wird.

Eselsbrücke: *Der Pfeil weist in die Richtung, in die das abhängige System schaut. Das andere System schaut nicht zurück.*

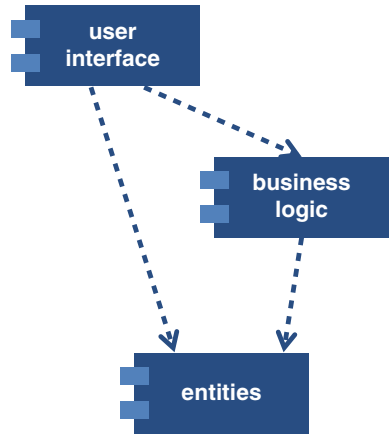
9.2 Komponentendiagramm

▶ Dieser Abschnitt ist besonders für Programmierer und Software-Architekten interessant.

Ein UML-**Komponentenabhängigkeitsdiagramm** (**component dependency diagramm**) (abgekürzt Komponentendiagramm) zeigt, welche Komponenten abhängig von anderen Komponenten sind.

In diesem Diagramm weisen die gestrichelten Pfeile von den abhängigen Komponenten zu den Komponenten, von denen sie abhängig sind.

Beispiel



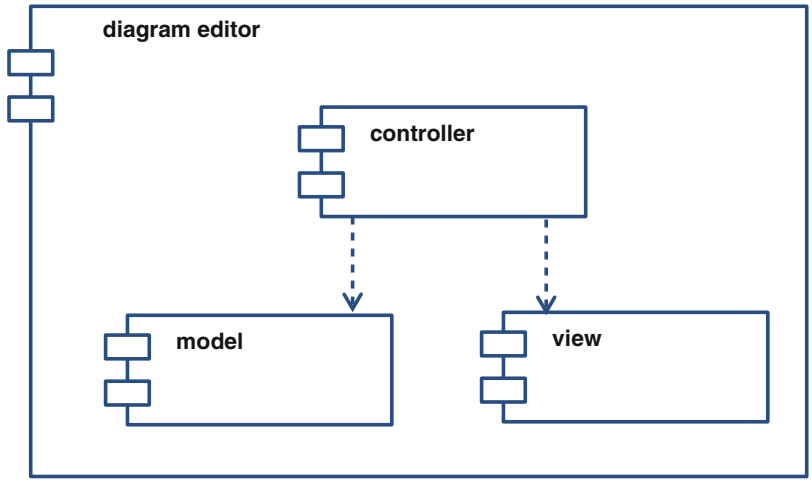
Die Komponenten „user interface“ und „business logic“ sind beide abhängig von der Komponente „entities“.

Zirkuläre Abhängigkeiten führen in der Regel zu schwer verständlichem „Spaghetticode“ und sollten deshalb vermieden werden.

Um eine klar geschichtete Architektur (sie entsteht, wenn es keine zirkulären Abhängigkeiten gibt) zu zeigen, werden abhängige Komponenten oberhalb der Komponenten dargestellt, von denen sie abhängig sind.

Der Fall, dass eine Komponente wiederum andere Komponenten enthält, kann dadurch ausgedrückt werden, dass im Symbol der enthaltenden Komponente Symbole der enthaltenen Komponenten dargestellt werden.

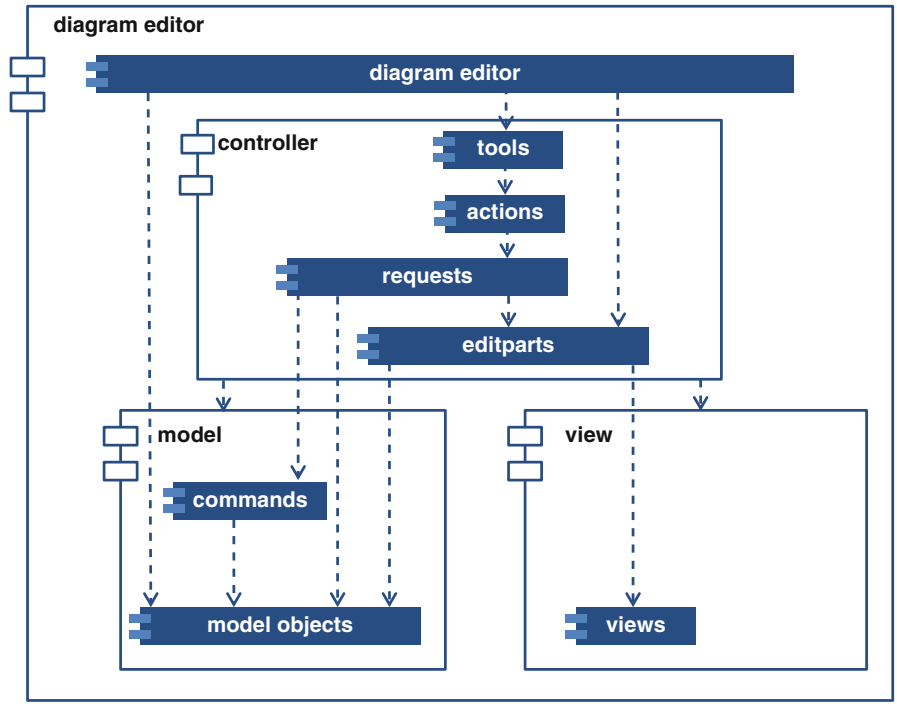
Beispiel Komponentendiagramm



In diesem Beispiel umfasst die Komponente diagram editor die Komponenten controller, model und view.

Beispiel des dazugehörenden detaillierten Komponentendiagramms

Das oben stehende Beispiel wird im Folgenden detailliert ausgearbeitet, sodass sichtbar wird, welche Komponenten Bestandteil der Komponenten controller, model und view sind.



9.3 Paketdiagramm

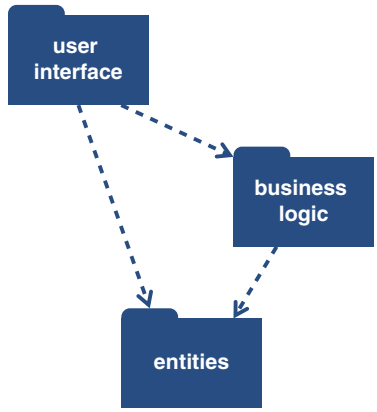
▶ Dieser Abschnitt ist besonders für Software-Architekten und Programmierer relevant.

Ein UML-**Paketdiagramm** (**package dependency diagram**) zeigt, welche Pakete (in einer Programmiersprache wie C#.NET und Java) von anderen Paketen abhängig sind – ein ähnlicher Prozess wie beim Komponentendiagramm, das die Abhängigkeitsbeziehungen zwischen Komponenten zeigt.

Das heißt: Ein Paket ist von einem anderen Paket abhängig, wenn eine Klasse oder eine Schnittstelle im abhängigen Paket auf eine Klasse oder eine Schnittstelle im Paket verweist, von dem es abhängig ist.

Beispiel

In diesem Diagramm weisen die gestrichelten Pfeile von den abhängigen Paketen zu den Paketen, von denen sie abhängig sind.



Die Pakete user interface und business logic sind beide vom Paket entities abhängig. Außerdem ist das Paket user interface vom Paket business logic abhängig.

9.4 Sequenzdiagramm

▶ Dieser Abschnitt ist besonders für Programmierer und Software-Architekten von Interesse, die mit einer objektorientierten Programmiersprache arbeiten.

Ein UML-**Sequenzdiagramm** zeigt, welche Objekte voneinander welche Methoden in welcher Reihenfolge aufrufen.

Die **Zeitlinien** laufen von oben nach unten. Vertikale Blöcke auf der Zeitlinie verdeutlichen, von wann bis wann die Ausführung einer Methode dauert.

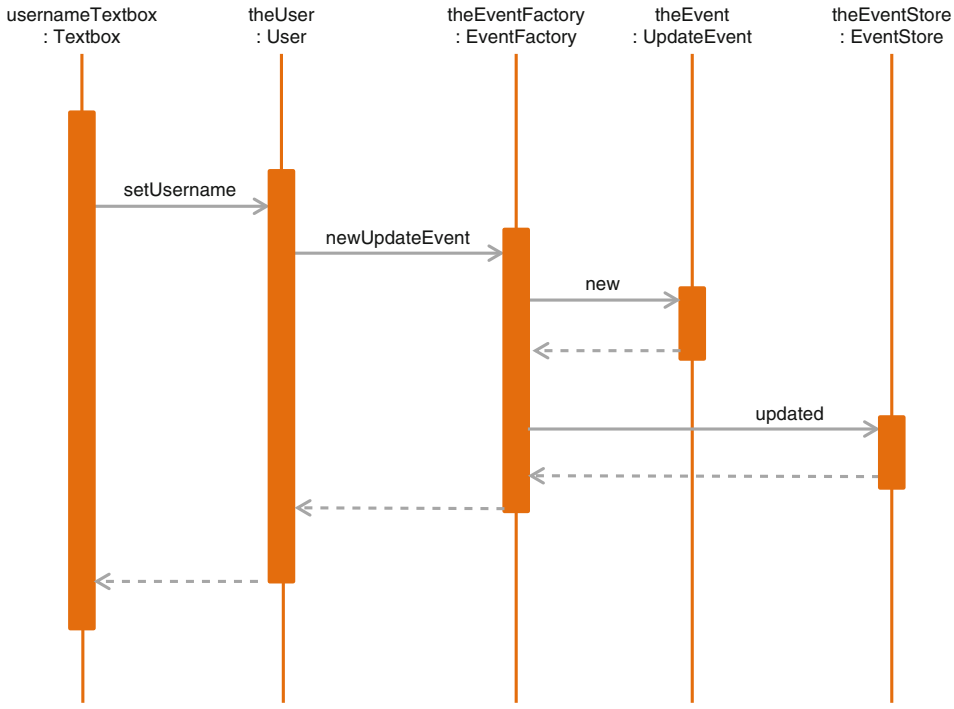
Oberhalb der Zeitlinie steht der Name des Objekts, auf dem die Methoden aufgerufen werden. Unterhalb des Objektnamens steht (hinter einem Doppelpunkt) der Name der Klasse, von der das Objekt eine Instanz ist.

Ein Pfeil gibt den Aufruf einer Methode wieder.

9.4.1 Synchroner Methodenaufruf

Bei einem **synchronen Methodenaufruf** (**synchronous call**) wartet die aufrufende Methode, bis die aufgerufene Methode das Ergebnis liefert. Ein gestrichelter Pfeil zeigt die Lieferung des Ergebnisses.

Beispiel

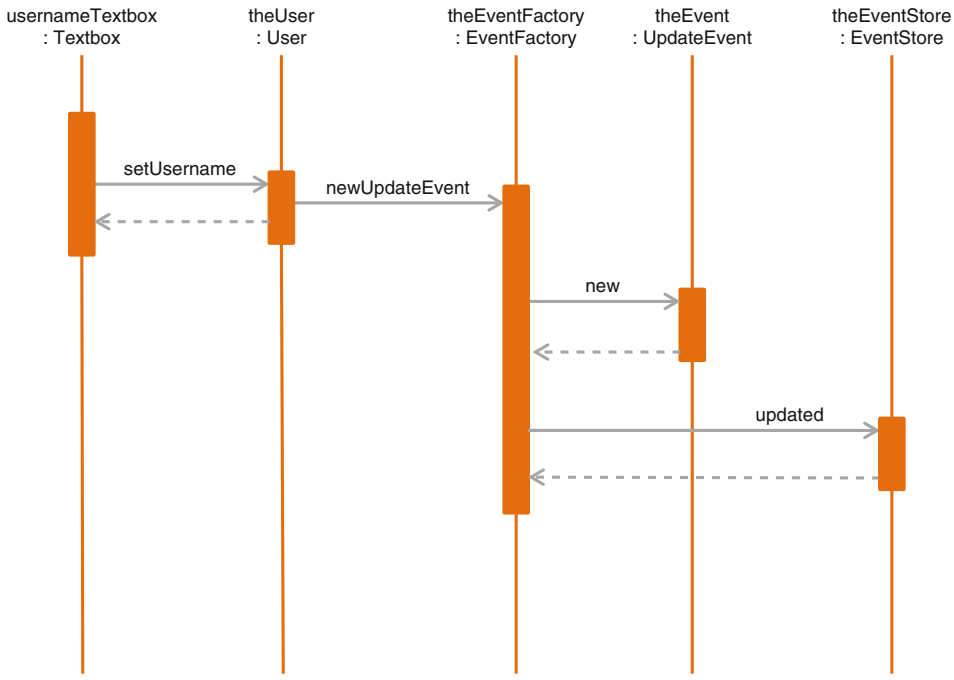


In diesem Beispiel ruft das Objekt „usernameTextBox“ (der Klasse „TextBox“) die Methode „setUsername“ des Objekts „theUser“ (der Klasse „User“) usw. auf. Alle Aufrufe in diesem Beispiel sind synchron.

9.4.2 Asynchroner Methodenaufruf

Bei einem **asynchronen Methodenaufruf** (**asynchronous call**) wartet die aufrufende Methode nicht, bis die aufgerufene Methode beendet ist und ein Ergebnis liefert.

Beispiel



In diesem Beispiel ruft die Methode „setUsername“ des Objekts „theUser“ die Methode „newUpdateEvent“ des Objekts „theEventFactory“ asynchron auf.

Die übrigen Aufrufe in diesem Beispiel sind synchron.

9.5 Kommunikationsdiagramm

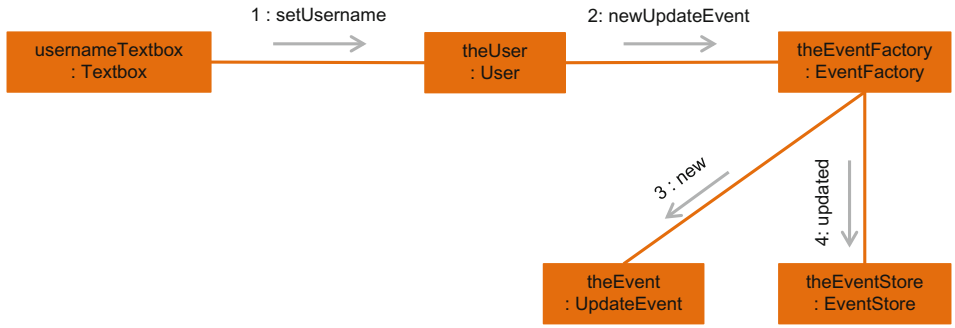
▶ Dieser Abschnitt ist besonders für Programmierer und Software-Architekten relevant, die mit einer objektorientierten Sprache arbeiten.

Ein UML-**Kommunikationsdiagramm** zeigt beispielhaft, wie und in welcher Reihenfolge Objekte miteinander kommunizieren.

Abgesehen von der Ergebnislieferung und der Laufzeit der Methode enthält ein Kommunikationsdiagramm die gleichen Informationen wie ein Sequenzdiagramm. Sie werden jedoch auf andere Art und Weise wiedergegeben.

Jedes Rechteck in einem Kommunikationsdiagramm repräsentiert ein Objekt. Innerhalb des Rechtecks stehen (vor dem Doppelpunkt) ein optionaler Objektname und (hinter dem Doppelpunkt) der Name der Klasse deren Instanz das Objekt ist.

Beispiel



Dieses Kommunikationsdiagramm zeigt die gleichen Methodenaufrufe wie im vorangegangen Beispiel das Sequenzdiagramm.

Das Aufrufen einer Methode wird durch einen Pfeil mit einer Nummer, die die Reihenfolge der Aufrufe im Diagramm angibt, wiedergegeben. Dahinter stehen ein Doppelpunkt und der Name der aufgerufenen Methode.

Eine Linie zwischen zwei Objekten gibt an, dass beide miteinander kommunizieren.

9.6 Übung zur Kopplung zwischen Systemen

- Zeichne für die Anwendung, für die du in den vorherigen Kapiteln die Anwendungsspezifikation, das Klassendiagramm usw. erstellt hast, welche Kopplungen diese Anwendung mit anderen Systemen hat.

9.7 Checkliste zur Kopplung zwischen Systemen

Der häufigste Fehler beim Modellieren einer Kopplung zwischen Systemen ist, dass der Pfeil in die falsche Richtung zeigt. Diese Checkliste besteht deshalb nur aus einer Frage:

Frage

- Zeigt der Pfeil zu der Schnittstelle, die vom System benötigt wird?

Die Antwort auf diese Frage muss „Ja“ sein.

Audit Trail und Rückgängig

10

Zusammenfassung

Ein **Audit Trail** ist ein wirksames Instrument, um Betrug zu verhindern und aufzuspüren. Deshalb spielen Audit Trails in Systemen zur Unterstützung des kaufmännisch-administrativen Bereichs eine wichtige Rolle. Dieses Kapitel zeigt, wie Audit Trails für elementare Ereignisse modelliert werden, die auch als Basis dienen, um diese wieder rückgängig zu machen.

10.1 Audit Trail

In einem Audit Trail wird jedes **Ereignis** (**event**) registriert, bei dem ein Anwender (oder ein externes System über eine Schnittstelle) ein Objekt anlegt, ändert oder löscht. Auf diese Weise wird innerhalb des Audit Trails gespeichert, wer wann was gemacht hat und in welcher Anwendersession dies geschah.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_10

Beispiel Audit Trail

Ereignis	Wann:	Von wem
Produkt ‘Quark, vollfett, 500 Gramm‘ geändert	29-02-12 13:14	Rainer Voll
Kunde Klaus Müller erstellt	29-02-12 13:18	(anonym)
Kunde Peter Vogel geändert Straße von ‘Heinestr.’ in ‘Neuestr.’ Hausnummer von ‘12’ in ‘30’ PLZ von ‘12345’ in 78945’	29-02-12 13:35	Peter Schnee
Produkt ‘Camenbert 125 Gramm’ erstellt	29-02-12 13:45	Anna Kaiser
Bestellung 1351 von Klaus Schröder versendet	29-02-12 13:56	Ulla Bremer
Kunde Heidi Vollmer geändert	29-02-12 14:14	Peter Schnee
Produkt ‘Joghurt, vollfett, 1 Liter’ entfernt	29-02-12 13:34	Rainer Voll

Die Ereignisse in einem Audit Trail können aus verschiedenen elementaren Ereignissen bestehen. So ein elementares Ereignis kann das Anlegen eines Objekts, das Ändern einer Objekteigenschaft oder das Löschen eines Objekts sein.

Bei sehr sensiblen Daten werden in einem Audit Trail auch Lesezugriffe gespeichert (geloggt), sodass später nachvollziehbar ist, welche Anwender welche sensiblen Daten wann gesehen haben.

In der Realität beinhalten Anwendungen oft unrichtige und veraltete Daten. Ein Audit Trail bietet die Möglichkeit, zu sehen, wo die Ursachen liegen. Im Anschluss kann versucht werden, die Ursachen zu beseitigen, z. B. indem jemand, der unrichtige Daten eingegeben hat, entsprechende Instruktionen und/oder eine Schulung erhält.

10.2 Rückgängig

Wenn ein Audit Trail zu jedem elementaren Ereignis die vollständigen Informationen enthält, um dieses Ereignis rückgängig zu machen, kann der Audit Trail auch dafür genutzt werden, der Anwendung die Funktionalität eines Multilevel-undo (und redo) zu geben, wie sie in zahlreichen Office-Anwendungen (z. B. Word, Excel) und Bildbearbeitungsprogrammen (z. B. Photoshop) vorhanden sind.

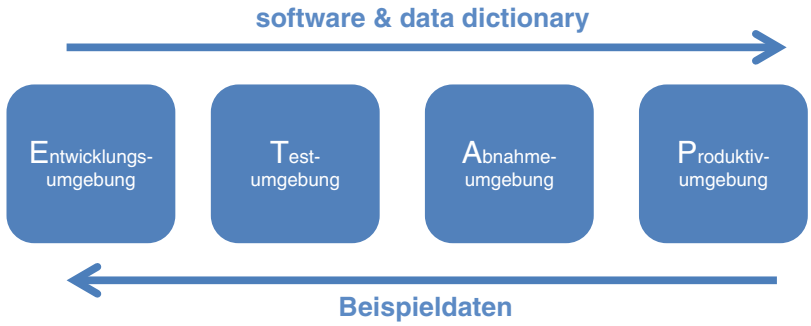
Ändern

Preis ändern rückgängig machen	**ctrl + Z**
Produkt löschen wiederholen	**ctrl + Y**

Die Rückgängig-Funktion kann auch in einem Programmfenster integriert werden, das den Audit Trail zeigt. Dort kann jemand, der über die entsprechende Autorisierung verfügt, Ereignisse im Audit Trail rückgängig machen oder wiederholen.

10.3 Nutzung eines Audit Trails in einer DTAP-Street

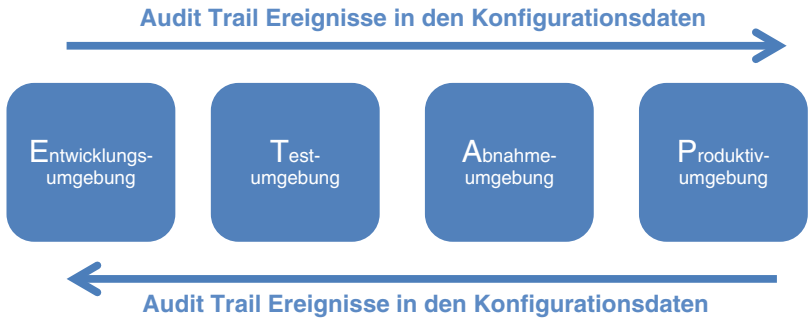
Um Software, die entwickelt (**D**evelopped) wurde, zu prüfen bzw. zu testen (**T**) und abzunehmen bzw. zu akzeptieren (**A**), bevor sie in die Produktion (**P**) geht, wird häufig eine DTAP-Street verwendet.



Die Produktivumgebung ist das System (Server + Datenbank), in dem die Anwendung in der Praxis eingesetzt wird. Die anderen Umgebungen in der DTAP-Street sind mehr oder weniger Kopien dieser Produktivumgebung.

Einführungen und Änderungen der Software fließen durch die DTAP-Street von der Entwicklung zur Produktion: Wenn ein Teil der Anwendung entwickelt ist, wird er in der Testumgebung geprüft. Danach wird er in die Abnahmeumgebung übernommen und, wenn dort alles richtig funktioniert, erfolgt die Installation und der Einsatz in der Produktivumgebung.

Um mit realistischen Beispieldaten arbeiten zu können, fließen die Daten (wenn auch wegen des Datenschutzes anonymisiert) durch die DTAP-Street von der Produktiv- in die Abnahme-, Test- und manchmal auch in die Entwicklungsumgebung.

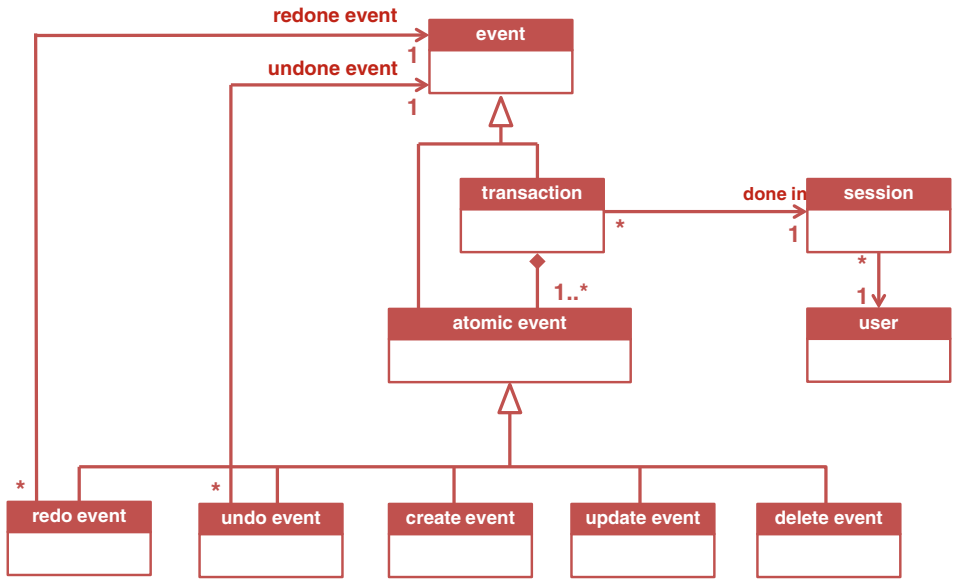


Auf die gleiche Art können Einträge und Änderungen in den Einstellungen der Konfigurations- und Stammdaten, die (z. B. durch den funktionalen Administrator) direkt in der Produktivumgebung vorgenommen wurden, über die geloggten Ereignisse im Audit Trail in die Abnahme-, Test- und Entwicklungsumgebung übernommen werden. Auf diese Weise kann jederzeit mit den aktuellsten Einstellungen aus der Produktivumgebung getestet werden.

10.4 Klassenmodell eines Audit Trails mit Rückgängig-Funktionalität

▶ Dieser Abschnitt ist besonders für Leser relevant, die sich etwas tiefer in die Materie graben möchten. Andere können diesen Abschnitt ruhigen Gewissens überschlagen.

Das Klassenmodell eines Audit Trails sieht in grober Skizze wie folgt aus:



Weitere Details dieses Klassenmodells betreffen eher technische als funktionale Angelegenheiten und werden deshalb in diesem Buch nicht beschrieben.

Beim Ausarbeiten dieser Details muss beispielsweise für diverse Eigenschaften der create-, update- und delete-Ereignisse bestimmt werden, ob sie statisch oder dynamisch typisiert werden. Weiter muss festgelegt werden, ob das Modell hierfür explizit als Instanz eines Metamodells gespeichert werden soll und ob in einem create- oder delete-Ereignis eine Kopie des angelegten oder gelöschten Objekts gespeichert werden soll. Der Audit Trail kann dabei Bestandteil einer event-sourcing-Architektur sein. Wenn auch die Modelle auf diese Weise gebaut werden, spricht man von „**eventsourced rapid application development**“.

Für jede dieser Alternativen gibt es verschiedene Lösungen, die jeweils ihre eigenen (vor allem technischen) Vor- und Nachteile haben.

Vom Modell zur funktionierenden Software 11

Zusammenfassung

Die gezeichneten Modelle dienen an erster Stelle dazu, Teile des funktionalen Entwurfs festzulegen und diese Informationen mit dem Auftraggeber und den Teammitgliedern zu teilen und abzustimmen. Im Laufe dieses Austauschs werden die Modelle verbessert und verfeinert. Dieses Kapitel zeigt, wie sich Modelle innerhalb eines **Scrum-Teams** einsetzen lassen.

Oft werden in der Vergangenheit erstellte Modelle dabei nur dann angepasst, wenn hieraus auch wirklich ein Mehrwert resultiert. Es ist nicht immer notwendig, Modelle up-to-date zu halten, wenn sich die Software ändert.

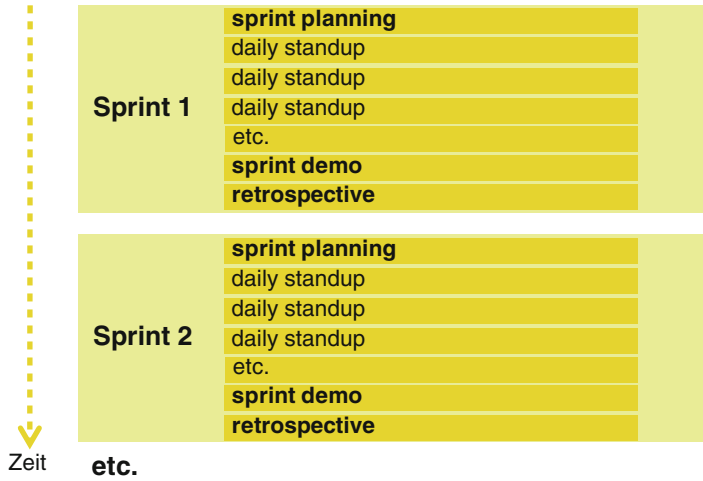
11.1 Scrum-Methode: agil und lean

Bei der Scrum-Methode wird die Umsetzung in eine Serie von **Sprints** aufgeteilt. Am Ende eines jeden Sprints wird ein lauffähiges Produkt geliefert, das jeweils mehr Funktionalitäten umfasst als das Ergebnisprodukt des Sprints davor.

Die Sprints dauern meistens einige Wochen.

Zu Beginn eines Sprints wird in einer **Sprintplanung** festgelegt, was am Ende des Sprints als Ergebnis geliefert werden muss.

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8_11



Jeder Tag beginnt mit einem kurzen **daily-standup**-Treffen. Hierbei stehen die Teammitglieder im Kreis zusammen und berichten der Reihe nach kurz:

- Was sie am vorherigen Tag getan haben.
- Was sie heute tun werden.
- Welche Probleme sie eventuell erwarten.

Am Ende des daily-standup-Treffens fragt der **Scrum-Master** (das ist der Leiter des Scrum-Prozesses), wer noch Fragen hat. Anschließend geht jeder an die Arbeit.

Eine ausgiebige Besprechung einzelner Sachverhalte findet während des daily-standup-Treffens nicht statt. Sofern notwendig erfolgt dies in gesonderten Treffen. Deswegen bleiben die daily-standup-Treffen immer relativ kurz.

Am Ende eines jeden Sprints zeigt das Entwicklungsteam dem Auftraggeber in einem ebenfalls relativ kurzen **sprint-demo**-Treffen (manchmal auch **sprint-review** genannt), was entwickelt wurde.

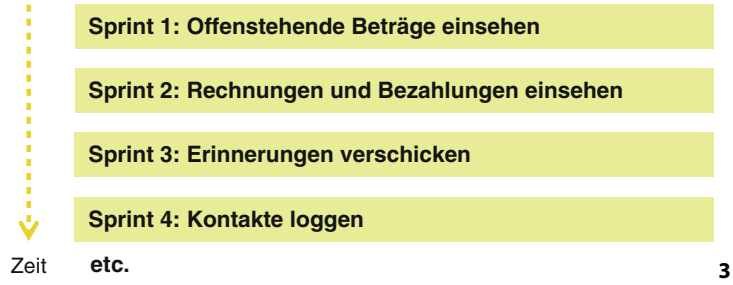
Nach dem sprint-demo-Treffen wird in einem retrospective-Treffen analysiert, wie der Sprint verlaufen ist und wie der Entwicklungsprozess mit Blick auf zukünftige Sprints verbessert werden kann. Auf diese Weise formt sich ein lernendes Team.

Die Sprints sind zeitlich streng begrenzt: Das sprint-demo-Treffen wird nicht verschoben, wenn das für den Sprint geplante Ergebnis nicht fertiggestellt ist, sondern es wird das gezeigt, was erreicht wurde. Die nicht vervollständigten Arbeiten werden in dem sprint-planning-Treffen besprochen und dann meistens für den folgenden Sprint eingeplant.

Um ein Bild davon zu erhalten, wann welche Funktionalität verfügbar ist, kann ein **Iterationsplan** (**iteration plan**) erstellt werden. In ihm wird für jeden zukünftigen Sprint angegeben, was man erwartet, als Ergebnis abzuliefern. Mit Vorliebe wird das Ergebnis dabei als „knackiges“ Thema formuliert.

Beispiel

Iterationsplan für die Entwicklung eines Debitorenverwaltungsmoduls



Während der sprint-planning-Treffen wird der Iterationsplan, wo es nötig ist, hinzugezogen.

Wegen des schnellen Ablieferns von Ergebnissen (alle paar Wochen) und des sukzessiven Plananpassens wird dieses Vorgehen in der Entwicklung **agil** (**agile**) genannt.

Das Abliefern nutzbarer Ergebnisse (lauffähiger Programmversionen) und das aktive Lernen aus vorangegangenen Phasen passt gut zu der **Lean**-Methode, die unter anderem in Toyota-Autofabriken Anwendung findet.

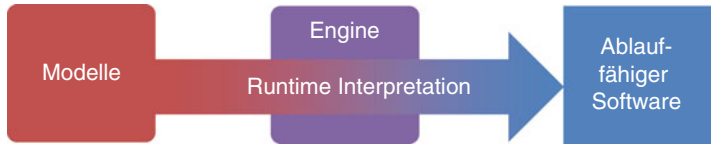
11.2 Modellgetriebene Entwicklung

Bei der **modellgetriebenen Entwicklung** (**model driven development**, oft abgekürzt als **MDD**) werden Modelle automatisch in ablauffähige Prototypen oder in Teile der zu realisierenden Software umgesetzt.

Das kann durch das Generieren von Codes aus dem Modell geschehen:

Die **Schablonen** (**templates**) werden, oft unter Verwendung von open-source-Tools, so erstellt, dass der generierte Code in die vom Auftraggeber vorgesehene Architektur passt.

Es ist ebenso möglich, die Modelle mithilfe einer Engine zu interpretieren. Das geschieht z. B. bei der Nutzung kommerziell verfügbarer **RAD**-Tools (rapid application development).



11.3 Übung zum Iterationsplan

- Erstelle eine Liste der vorbereitenden Arbeitspakete, die nötig sind, damit mit der Umsetzung (Programmierung) der entworfenen Software gestartet werden kann.
- Erstelle eine Liste der Arbeitspakete, die nötig sind, um die entworfene Software umzusetzen (Programmierschritte).
- Setze die oben genannten Arbeitspakete in eine sinnvolle Reihenfolge, in der sie ausgeführt werden können.
- Wähle eine Länge für die Sprints, z. B. zwei Wochen.
- Verteile die oben genannten Arbeitspakete so, dass in jedem Sprint ausreichend Zeit ist, um die geplanten Arbeitspakete zu realisieren. Gib, wenn möglich, jedem Sprint ein aussagekräftiges Thema als Titel.

Nachwort

Ich hoffe, dass dieses Buch dich beim Analysieren und Entwerfen inspiriert und unterstützt.

Was nicht in den beschriebenen Diagrammen ausgedrückt werden kann, kannst du in kurzen bündigen Texten beschreiben.

Die Qualität wird in hohem Maß durch die Kompaktheit bestimmt. Es ist schwieriger einen langen Text zu lesen, zu behalten und zu interpretieren als einen kurzen.

Berichte können am besten anhand von Beispielen beschrieben werden (**specification by example**).

Ich möchte meinem Kollegen Wiebe de Witte herzlich für sein wertvolles Feedback danken, auf dessen Basis ich das Buch verbessern konnte.

Weiter möchte ich meinen Seminarteilnehmern, Kollegen und Auftraggebern für die vielen Praxisinformationen danken, mit denen ich die beschriebenen Techniken weiter verfeinern konnte.

Mehr Information kannst du auf folgender Webseite finden: www.einfuehrunguml.de.

Ich wünsche dir viel Modellierspaß!

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8

Sachverzeichnis

H.J. van Randen et al., *Einführung in UML*,
DOI 10.1007/978-3-658-14412-8