

Funktions- und Verhaltensmodellierung

1. Motivierung und Übersicht

Das Klassenmodell (auch Domänenklassenmodell genannt) der objektorientierten Programmierung ist statisch und auf die Struktur des Anwendungssystems fokussiert.

Zur vollständigen Beschreibung des Systems ist die Funktionalität eines Anwendungssystems von großer Bedeutung, muss also in jedem Fall bei einem Programmentwurf mit berücksichtigt werden. Die Trennung bei der Modellierung von Struktur und Funktionalität ist ein wichtiger Schritt in Richtung Komplexitätsreduktion.

Die *extern beobachtbaren Funktionen*, also das, was ein Anwendungssystem dem Benutzer anbietet oder anbieten soll, wird mit Anwendungsfällen spezifiziert, die in das Anwendungsfallsdiagramm münden. Das ablaforientierte Verhalten von Anwendungsfällen und im Klassenmodell definierten komplexen Operationen modelliert man mit Interaktionsdiagrammen. Diese Diagramme stellen insbesondere dar, wie die an einem Anwendungsfall oder einer Operation beteiligten Objekte den Ablauf durch den Austausch von Nachrichten koordinieren.

Das zustandsorientierte Verhalten von Exemplaren einer Klasse, also das komplexe Zusammenspiel von Zuständen und Operationen einer Instanz, wird mit Zustandsdiagrammen präzisiert.

Diese Zusammenfassung basiert auf dem Skript von **Herrn Prof. Dr. H.W. Six**, Lehrgebiet Software Engineering, der [FernUniversität in Hagen](#).

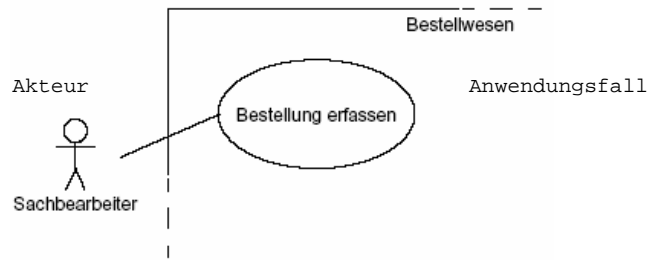
2. Grundlegende Definitionen

Innerhalb eines Geschäftsprozesses interagieren menschliche oder maschinelle „Benutzer“ mit dem Anwendungssystem, um bestimmte Aufgaben durchzuführen und konkrete Ziele zu erreichen. Im Normalfall wird es mehrere solche Benutzer geben, die in Bezug auf das Anwendungssystem eine bestimmte Rolle spielen.

Ähnlich, wie eine Klasse eine Abstraktion konkreter Objekte darstellt, abstrahiert ein Akteur eine bestimmte Rolle konkreter Benutzer der realen Welt.

Ein **Anwendungsfall** formuliert eine in sich abgeschlossene *Teilfunktionalität* des Anwendungssystems, die für mind. einen Akteur ein bestimmtes Ergebnis *innerhalb* des Geschäftsprozesses erbringt. Dabei werden nur extern beobachtbare Funktionen mit Hilfe eines Anwendungsfalls spezifiziert.

Jeder Anwendungsfall stellt eine „*Familie*“ von möglichen konkreten Abläufen dar. Bei der „Instanziierung“ eines Anwendungsfalls wird seine allg. beschriebene Funktionalität für einen speziellen, vorgegebenen Zustand des Anwendungssystems konkret präzisiert. Eine Instanziierung, d.h. ein konkreter Ablauf eines Anwendungsfalls, wird als **Szenario** bezeichnet.



Ähnlich wie eine Klasse eine Abstraktion konkreter Objekte darstellt, abstrahiert ein **Akteur** (*actor*) eine bestimmte Rolle konkreter Benutzer der realen Welt. Dabei interagieren menschliche aber auch maschinelle Benutzer innerhalb eines Geschäftsprozesses. Ein konkreter Benutzer kann dabei durchaus mehrere Rollen einnehmen.

Die Kommunikation der Akteure mit dem System wird durch *Assoziationen* zwischen Akteuren und Anwendungsfällen modelliert, wobei die Richtung des Informationsflusses durch entsprechende Navigierbarkeiten der Assoziation modelliert werden kann.

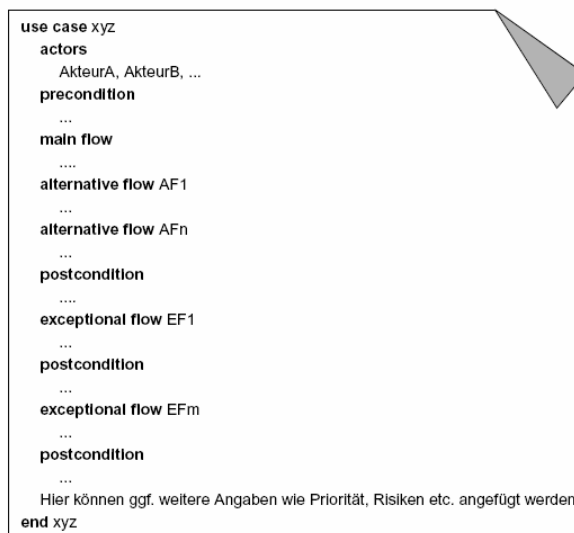
Jeder Anwendungsfall formuliert die möglichen Abläufe und Interaktionen, die zwischen Akteuren und dem Anwendungssystem im Rahmen der Teilfunktionalität stattfinden.

Anwendungsfälle bilden ein besonders wichtiges Modellierungskonzept in der Anforderungsermittlung. Wegen ihrer Einfachheit und Informalität stellen sie *die zentrale Kommunikationsbasis* zwischen Anwendern (Benutzer, Auftraggeber) und Analytikern dar. Die Anwendungsfälle ergeben sich aus Gesprächen mit entsprechenden Sachbearbeitern bzw. durch die Analyse bereits vorhandener Ablaufbeschreibungen. Dabei kommt den Szenarien eine entscheidende Rolle zu.

Um die Beschreibung des „normalen“ Ablaufs (**main flow**) eines Anwendungsfalls möglichst einfach zu halten, werden Abweichungen vom normalen Ablauf in eigenen Abschnitten aufgeführt (*alternative and exceptional flows*):

Bei den alternativen Abläufen (**alternative flows**) wird der Anwendungsfall wie beim normalen Ablauf erfolgreich beendet, und es ist auch hier die Nachbedingung des normalen Ablaufs erfüllt.

Anders verhält es sich bei sog. Ausnahmesituationen (**exceptional flows**). Hier ist der erfolgreiche Ablauf des Anwendungsfalls gefährdet und es wird eine gesonderte Nachbedingung angegeben.



Als *Beispiel* könnte man eine Geldabhebung am Bankschalter heranziehen. Geldabhebungen innerhalb des dem Kunden gewährten Dispokredits modelliert man sinnvoller Weise als mainflow. Würde der Kunde seinen Kreditrahmen mit der gewünschten Abhebung um einen Betrag kleiner 2.000€ überschreiten, so kann dies der Schaltermitarbeiter u.U. genehmigen. Der Kunde erhält nach Bewilligung sofort das Geld ausgezahlt. Dies könnte man als alternative flow modellieren.

Will der Kunde allerdings sein Konto um einen Betrag größer als 2.000€ überziehen, so muss der Auftrag erst von der Kreditabteilung genehmigt werden. Die Geldauszahlung kann nicht sofort erfolgen, deshalb würde man dieses Szenario wohl als exceptional flow modellieren.

Graphisch werden Anwendungsfälle im **Anwendungsfalldiagramm** als Ellipsen dargestellt, in denen der Name des jeweiligen Anwendungsfalls aufgeführt wird. Weiter unten sind Beispiele visualisiert.

Die *Vorbedingungen* eines Anwendungsfalls grenzt die Situation ein, in denen der Anwendungsfall ausgeführt werden darf; die *Nachbedingung* präzisiert das Ergebnis des Anwendungsfalls. Ähnlich also wie bei der *pre-* und *postcondition* eines Programmes, um bspw. die partielle Korrektheit zu verifizieren.

Beziehungen zwischen Anwendungsfällen:

Bei der Anforderungsermittlung für große Anwendungssysteme können prinzipiell zwei (konkurrierende) Fälle eintreten, die eine weitere Strukturierung der Anwendungsfälle notwendig machen:

- 1) Die Vielzahl der Anwendungsfälle macht die Anforderungsspezifikation unüberschaubar.
- 2) Einzelne Anwendungsfälle werden sehr komplex.

Im Fall vieler kleiner, logisch zusammengehöriger Anwendungsfälle können diese in Paketen zusammengefasst werden.

Vorteil:

Verbesserter Überblick;

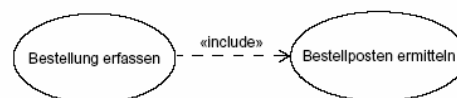
Nachteil:

Redundanz, schlechte Strukturierung bei komplexen Anwendungsfällen, Abhängigkeiten zwischen den Anwendungsfällen können nur schlecht herausgestellt werden.

Lösung:

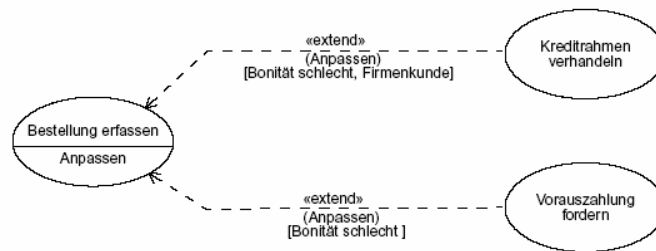
Die include- und extend-Beziehungen zwischen Anwendungsfällen.

Die **include-Beziehung** wird verwendet, wenn verschiedene Anwendungsfälle *dieselbe Teilfunktionalität* beinhalten. Diese Teilfunktionalität wird zur Redundanzvermeidung in einem separaten Anwendungsfall beschrieben, der von den benutzenden Anwendungsfällen (Basis-Anwendungsfälle) verwendet wird.



Eine **extend-Beziehung** von einem Anwendungsfall B zu einem Anwendungsfall A bedeutet, dass AF A unter *bestimmten Bedingungen* durch die im AF B beschriebene Funk-

tionalität erweitert werden kann. Mit anderen Worten: die Funktionalität des AF B ist eine optionale Erweiterung des AF A.



Sowohl die include- als auch die extend-Beziehung fügen zusätzliche Funktionalitäten in eine Basisfunktionalität ein.

Im Falle der include-Beziehung komplettieren die zusätzlichen Funktionalitäten stets die des Basisanwendungsfalls, d.h., ohne die zusätzliche Funktionalität wäre der dienstnehmende Anwendungsfall nur eingeschränkt (wenn überhaupt) funktionstüchtig.

Im Falle der extend-Beziehung ist sie optional (nur anzuwenden, wenn exakt definierte Umstände eintreten), d.h. der Basisanwendungsfall ergibt auch ohne die zusätzliche Funktionalität ein sinnvolles Ergebnis.

Die include-Beziehung vermeidet Redundanz und erlaubt die Wiederverwendung von Teilfunktionalitäten, während die extend-Beziehung die Flexibilität und einfache Erweiterbarkeit des Anwendungsfallmodells sicherstellen soll.

Darüber hinaus können Anwendungsfälle auch in einer *Generalisierungsbeziehung* zueinander stehen, wobei der speziellere Anwendungsfall die Aufgabe des allgemeineren Anwendungsfalls umfasst, aber einige Teilaufgaben auf eine speziellere Art und Weise bearbeitet.

Der speziellere Anwendungsfall kann überall dort Anwendung finden, wo auch der allg. Anwendungsfall angewendet wird. Die Umkehrung ist im Allgemeinen nicht korrekt.

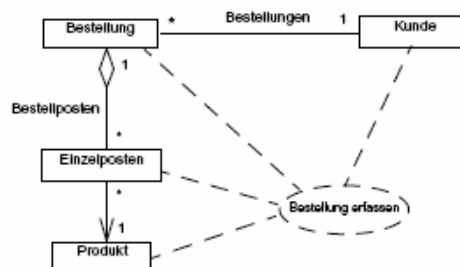
„Vererbt also der allgemeine Anwendungsfall mehreren spezielle Anwendungsfälle, so setzt sich der allg. Anwendungsfall gerade aus der Schnittmenge der Funktionalitäten der speziellen Anwendungsfälle zusammen!“

Konkrete Szenarien von Anwendungsfällen sind ohne Kenntnis der zugehörigen Objekte (und ihrer Verbindungen) schwerlich zu verstehen. Eine Möglichkeit die Objekte und Ihre Verbindungen im Kontext einer Szene zu visualisieren sind szenarienspezifizierte Objektdiagramme.

Natürlich kann mit einem konkreten Objektdiagramm nur genau eine konkrete Situation, also nur ein bestimmter Zeitpunkt in einem Szenario illustriert werden.

Die Menge aller Klassen von denen Instanzen in irgendeinem Objektdiagramm für den Anwendungsfall vorkommen kann als sog. **Mechanismus** dargestellt werden.

Ein Mechanismus ist ein partielles Klassendiagramm, in dem der Anwendungsfall selbst gestrichelt angedeutet und durch gestrichelte Linien mit allen an ihm beteiligten Klassen verbunden wird.



3. Interaktionsdiagramme

Interaktionsdiagramme beschreiben den Ablauf von Funktionen und präzisieren somit deren Semantik.

So wird z.B. dargestellt, wie Objekte bei der Ausführung einer im Klassenmodell definierten Operation, durch den Austausch von Nachrichten, zusammenarbeiten oder wie Akteure in einem Szenario eines Anwendungsfalls mit dem Anwendungssystem interagieren.

Die UML bietet zwei Arten von Interaktionsdiagrammen an:

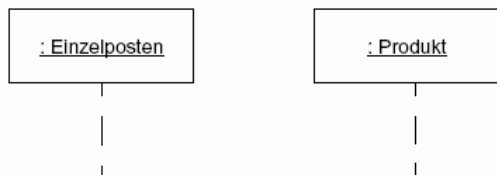
- (1) Das Sequenzdiagramm
- (2) Kollaborationsdiagramm (franz. „Zusammenarbeit“)

Während sich ersteres stärker auf das *ablauforientierte Verhalten* konzentriert, hebt letzteres –quasi als Erweiterung des Objektdiagramms – den *Zusammenhang zwischen der strukturellen und der ablauforientierten Modellierung* hervor.

Beide Diagramme visualisieren prinzipiell einen konkreten Ablauf also ein Szenario.

3.1 Sequenzdiagramme

Mit Hilfe von Sequenzdiagrammen modelliert man den konkreten Ablauf von (komplexen) Operationen im Klassenmodell unter Einbeziehung der beteiligten Objekte. Dabei wird jedes beteiligte Objekt als Rechteck am oberen Ende einer gestrichelten *vertikalen Linie* dargestellt, welche die *Zeit* symbolisiert, die von oben nach unten fortschreitet. Diese Linie veranschaulicht die Lebensdauer des Objektes und wird deshalb auch die *Lebenslinie* des Objektes genannt.



Der Aufruf der Operation wird als Pfeil mit geschlossener, ausgefüllter Spitze von der Lebenslinie des dienstnutzenden zu der des dienstleistenden Objektes wiedergegeben.

Jeder Pfeil wird mit dem Namen der aufgerufenen Operation beschriftet.

Wendet ein Objekt eine Operation auf sich selbst an, so nennt man dies eine *Selbstdelegation*.

Zwei Arten von Kontrollinfos, die in Interaktionsdiagrammen dargestellt werden können, sind besonders erwähnenswert:

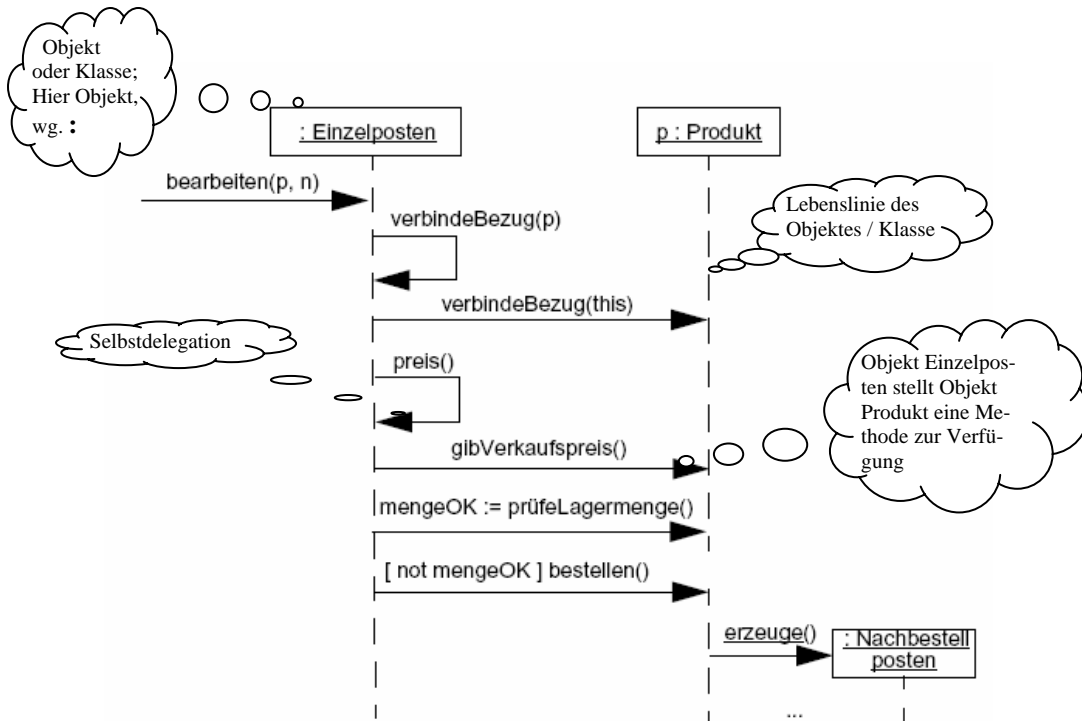
- Bedingungen und
- Iterationen.

Eine Bedingung zeigt in Form eines booleschen Ausdrucks an, in welchen Situationen eine Operation aufgerufen wird. Im SD wird die Bedingung in eckige Klammern vor dem Aufruf angegeben, z.B. [prüfen == true] prüfe().

Die zweite wesentliche Kontrollinformation betrifft Iterationen, bei denen Operationen wiederholt aufgerufen werden, z.B. wenn innerhalb einer Operationsausführung alle Objekte angesprochen werden sollen, die mit dem die Iteration ausführenden Objekt bez. einer mehrwertigen Assoziation verbunden sind.

Wird eine *Operation wiederholt aufgerufen*, so wird dies in der Form **[Iterationsausdruck] Operationsaufruf* angegeben, also z.B. *[i:1..10] messwertErfassen(i)*.

Der Iterationsausdruck ist optional – fehlt er, so ist die Anzahl der Iteration unbestimmt.



Schließlich können die Sequenzdiagramme noch die *Erzeugung und Zerstörung von Objekten* kenntlich machen. Im ersten Fall zeigt der Pfeil auf das Objektsymbol, im zweiten Fall wird die Lebenslinie mit einem Kreuz abgeschlossen:

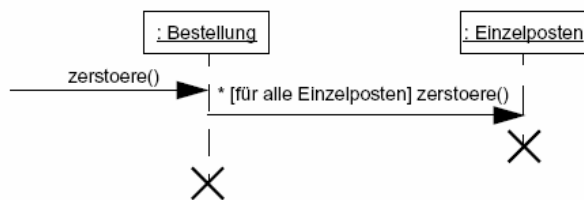


Abb. 14.5 Iteration eines Aufrufs

Synchrone Nachrichten und Aktivierungen

Das Veranlassen einer Operationsausführung kann auch als das Senden einer Nachricht (message) an ein Objekt verstanden werden.

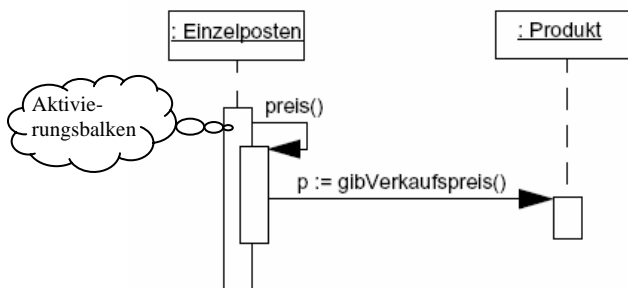
Jede Nachricht gibt dabei das dienstleistende und das dienstnutzende Objekt sowie den Namen und die Parameter der Operationen an. Das Ereignis „Empfang einer Nachricht“ bewirkt beim dienstleistenden Objekt die Ausführung der entsprechenden Operation.

Oft wartet der Aufrufer (Dienstnutzende) einer Operation untätig, bis das Ergebnis der Operation zurückgeliefert wird, und fährt erst dann mit seiner Beschäftigung fort. In diesem Falle spricht man von einer **synchrone Nachricht (synchronous message)**.

In Interaktionsdiagrammen werden synchrone Nachrichten durch geschlossene, gefüllte Pfeile dargestellt.

Wir präzisieren jetzt im SD die Ablaufkontrolle bei der Ausführung von Operationen und möchten darstellen, ob ein Objekt zu einem bestimmten Zeitpunkt gerade Operationen ausführt oder auf Rückmeldungen eines Operationsaufrufs wartet. Das Zeitintervall, in welchem ein Objekt auf das Ergebnis eines Operationsaufrufes wartet, wird durch ein Rechteck auf seiner Lebenslinie dargestellt und als *Aktivierungsbalken* bezeichnet.

Der Fall, dass ein Objekt auf die Rückmeldung einer aufgerufenen Operation wartet, tritt in unseren Anwendungen häufig auf, so dass Aktivierungsbalken ein SD schnell überladen können. Wir machen also nur dann davon Gebrauch, wenn es zum Verständnis der Situation wichtig ist.



Die Abbildung zeigt eindeutig, dass die Operation `gibVerkaufspreis()` von der Operation `preis()`, also während der Selbstdelegation aufgerufen und auch wieder abgeschlossen wird.

Zu einem Zeitpunkt besitzt genau ein Objekt die Ablaufkontrolle!

Wir lassen nun die Voraussetzung fallen, dass zu einem bestimmten Zeitpunkt nur jeweils genau ein Objekt die Ablaufkontrolle besitzt, und gehen stattdessen davon aus, dass *mehrere Objekte gleichzeitig Operationen* ausführen können.

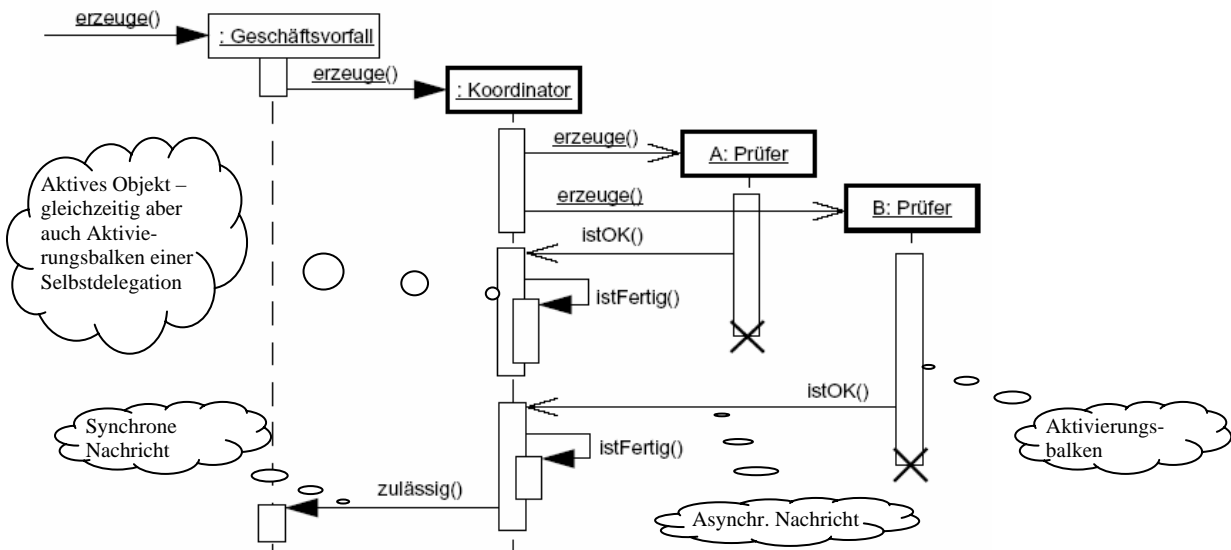
Eine Nachricht kann man sich nun als Brief vorstellen, nach dessen Absenden der Briefschreiber (Dienstanforderer) mit seiner Beschäftigung fortfährt. Die weiterlaufende Beschäftigung wird erst dann unterbrochen, wenn er die Antwort auf seinen Brief benötigt, dieser aber noch nicht eingegangen ist.

In einem solchen Fall muss explizit modelliert werden, dass der Sender nach dem Senden einer Nachricht weiter aktiv ist, also beide Objekte die Ablaufkontrolle innehaben. Wir sprechen dabei von **asynchronen Nachrichten** und bezeichnen die beiden Objekte als *aktive Objekte*.

Aktive Objekte werden in Interaktionsdiagrammen durch ein fett gezeichnetes Rechteck markiert (ebenso die entsprechenden Klassen im Klassendiagramm). Asynchrone Nachrichten werden durch Pfeile mit offener Spitze angegeben.

Bei einer asynchronen Nachricht werden zwei Fälle unterschieden:

- (a) Aktivierung eines Objektes: Hier zeigt der offene Pfeil auf den Anhang des Aktivierungsbalkens des angesprochenen Objektes.
- (b) Nachricht an Objekt, dass bereits aktiviert worden ist: Hierbei endet der (offene) Pfeil irgendwo innerhalb des Aktivierungsbalkens des angesprochenen Objektes.



3.2 Kollaborationsdiagramme

Die zweite Form von Interaktionsdiagrammen stellen Kollaborationsdiagramme (=:KD) dar. Man kann sie als Objektdiagramme betrachten, die zusätzlich ablaufforientiertes Verhalten beschreiben. Dabei wird der Nachrichtenaustausch an bestimmte Objektverbindungen gebunden.

Nachrichten, die zwei Objekte austauschen, werden wie in Sequenzdiagrammen durch Pfeile visualisiert, die ggf. durch Kontrollinformation angereichert sind.

Da Objekte (wie im Objektdiagramm) beliebig angeordnet werden können und *keine Zeitachse existiert*, muss die Abfolge der Nachrichten durch eine *geeignete Nummerierung* verdeutlicht werden.

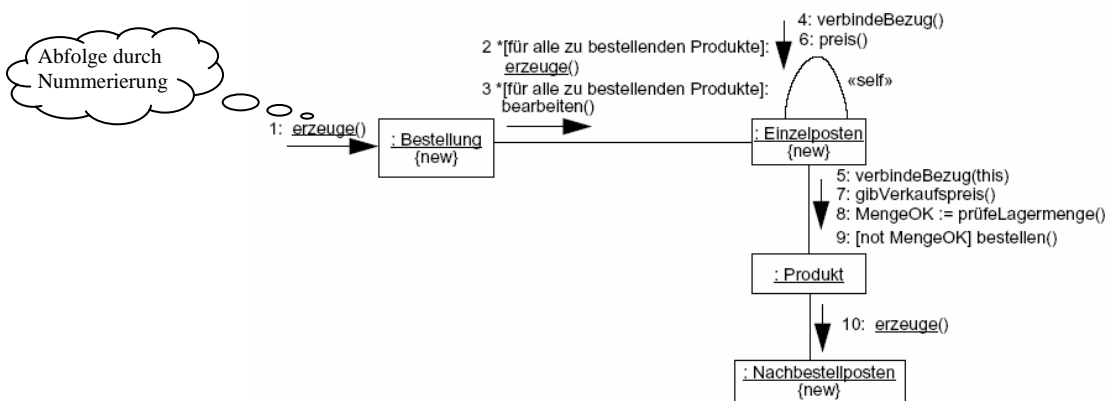


Abb. 14.11 Kollaborationsdiagramm mit Ordinalzahl-Nummerierung

Objekte, die während des dargestellten Ablaufs erzeugt oder zerstört werden, erhalten die Kennung `{new}` bzw. `{destroy}`. Wird ein Objekt während des Ablaufs erzeugt und zerstört, erhält es die Kennung `{transient}`.

4. Zustandsdiagramme

Das ablauforientierte Verhalten von Operationen und Anwendungsfällen wurde mit Hilfe von Interaktionsdiagrammen modelliert.

Diese Modellierung reicht aber nicht mehr aus, wenn Objekte mit einem komplexen Zusammenspiel von Zuständen und Operationen auftreten.

Dieses so genannte zustandsorientierte Verhalten von Objekten modelliert man mit Hilfe von Zustandsdiagrammen, welche aus Zuständen und Ereignissen bestehen.

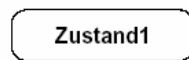
Der **Zustand** eines Objektes zu einem bestimmten Zeitpunkt ist durch die vorliegende Kombination seiner konkreten Attributwerte und Verbindungen zu anderen Objekten determiniert.

Beschränkung auf die wesentlichen Zustände und Eigenschaften:

Da in der Praxis die Anzahl der möglichen Kombinationen von Attributwerten und Verbindungen eines Objektes schnell explodieren kann, beschränkt man sich meist auf viel weniger Zustände. Dabei ist zu berücksichtigen, dass sich Objekte oftmals für viele der Kombinationen gleich verhalten und bestimmte Eigenschaften, welche sich nicht auf das Verhalten auswirken, außer Acht zu lassen.

UML Notation:

Zustände werden in der UML als abgerundete Rechtecke dargestellt, in denen der Bezeichner des Zustandes angegeben werden kann.



Darstellung eines Zustands in der UML

Objekte ändern ihre Attributwerte durch die Ausführung von Operationen. Wir verallgemeinern jetzt diese Sichtweise und betrachten den Empfang einer Nachricht als ein Ereignis.

Als **Ereignis** bezeichnet man das Eintreten eines bestimmten Sachverhalts zu einem bestimmten Zeitpunkt, das im Kontext der Modellierung eine *vernachlässigbare* Zeitdauer besitzt.

Ereignisse werden mit Ereignisnamen versehen.

Wir unterscheiden zwischen

- Aufrufereignissen
- Änderungsereignissen
- Zeitereignissen
- Signalereignissen

Auf Signalereignisse gehen wir hier nicht ein.

Ein **Aufrufereignis** ist der Empfang einer synchronen oder asynchronen Nachricht, welche die Ausführung einer Operation veranlasst.

Änderungsereignisse zeigen Änderungen an, die im Rahmen des zustandsorientierten Verhaltens relevant sind.

Zeitereignisse sind Ereignisse, welche an einen zeitlichen Ablauf bzw. an eine zeitliche Periode gebunden sind.

Unterscheiden sich mehrere Ereignisse lediglich in Details wäre es unpraktisch, ebenso viele Ereignisnamen wie Ereignisse zu erfinden und zu verwalten. In solchen Situationen empfiehlt es sich, ein Ereignis mit Ereignisattributen auszustatten, so dass seine „Instanzen“ anhand ihrer konkreten Werte voneinander unterscheidbar sind.

UML Notation:

Da Ereignisse über einen Namen und ggf. Attribute verfügen, werden sie in der UML graphisch wie Klassen dargestellt, obwohl sie natürlich keine Klassen sind und weder über Operationen noch über Assoziationen verfügen.

Zustandsübergänge und Bedingungen

Im Zustandsdiagramm wird die Reaktion auf des Eintreten eines Ereignisses als Zustandsübergang (Transition) modelliert: Ein Objekt in einem bestimmten (Ausgangs-)Zustand geht bei Eintritt eines bestimmten Ereignisses in einen bestimmten (Folge-)Zustand über. Sind Ausgangs- und Folgezustand eines Zustandsübergangs identisch, spricht man von einem reflexiven Zustandsübergang.

UML Notation:

Ein Zustandsübergang wird in der UML als Pfeil mit offener Spitze vom Ausgangszustand zum Folgezustand dargestellt, der mit einem Namen des den Zustandsübergang auslösenden Ereignisses beschriftet werden kann.

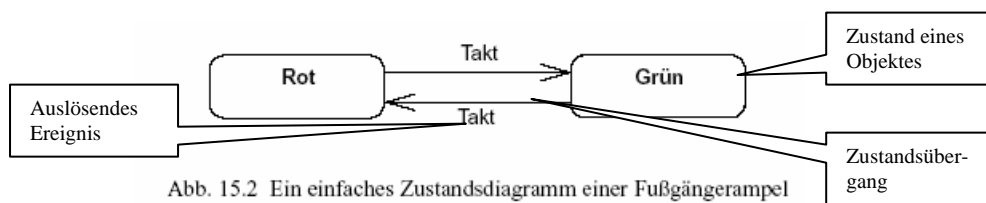


Abb. 15.2 Ein einfaches Zustandsdiagramm einer Fußgängerampel

Oft ist es sinnvoll, modellieren zu können, dass ein und dasselbe Ereignis für denselben Objektzustand unterschiedliche Übergänge auslösen kann (bedingte Anweisungen). Welcher Übergang stattfindet, hängt dann z.B. von den konkreten Werten der Ereignisattribute ab. Zur Modellierung solcher Sachverhalte können Zustandsübergänge mit Bedingungen versehen werden.

Eine Bedingung, auch **Wächterbedingung** genannt, ist ein prädikatenlogischer Ausdruck, der sich u.A. auf Parameter des auslösenden Ereignisses sowie auf Attribute und Verbindungen des betroffenen Objektes beziehen kann.

UML Notation:

Die Wächterbindung eines Zustandsübergangs wird im Zustandsdiagramm in eckigen Klammern hinter dem Ereignisnamen angegeben.

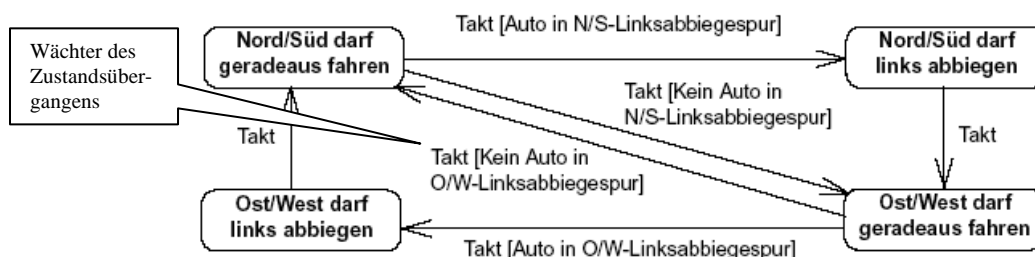


Abb. 15.3 Zustandsdiagramm mit Bedingungen

Aktionen und Aktivitäten

Objekte, die nur ihren Zustand wechseln können, sind meist uninteressant. Bei Eintritt eines Ereignisses wird während des ausgelösten Zustandsübergangs gewöhnlich eine bestimmte Aktion oder eine *Aktionsfolge* durchgeführt und ggf. weitere Ereignisse ausgelöst.

Ereignisse können auch nur Aktionen auslösen ohne Zustände zu ändern. Hierbei unterscheiden wir

- die Eingangsaktionen und
- die Ausgangsaktionen.

Eingangsaktionen werden unmittelbar vor dem „Betreten“ des Zustandes, eine Ausgangsaktion unmittelbar vor jedem „Verlassen“ des Zustandes ausgeführt.

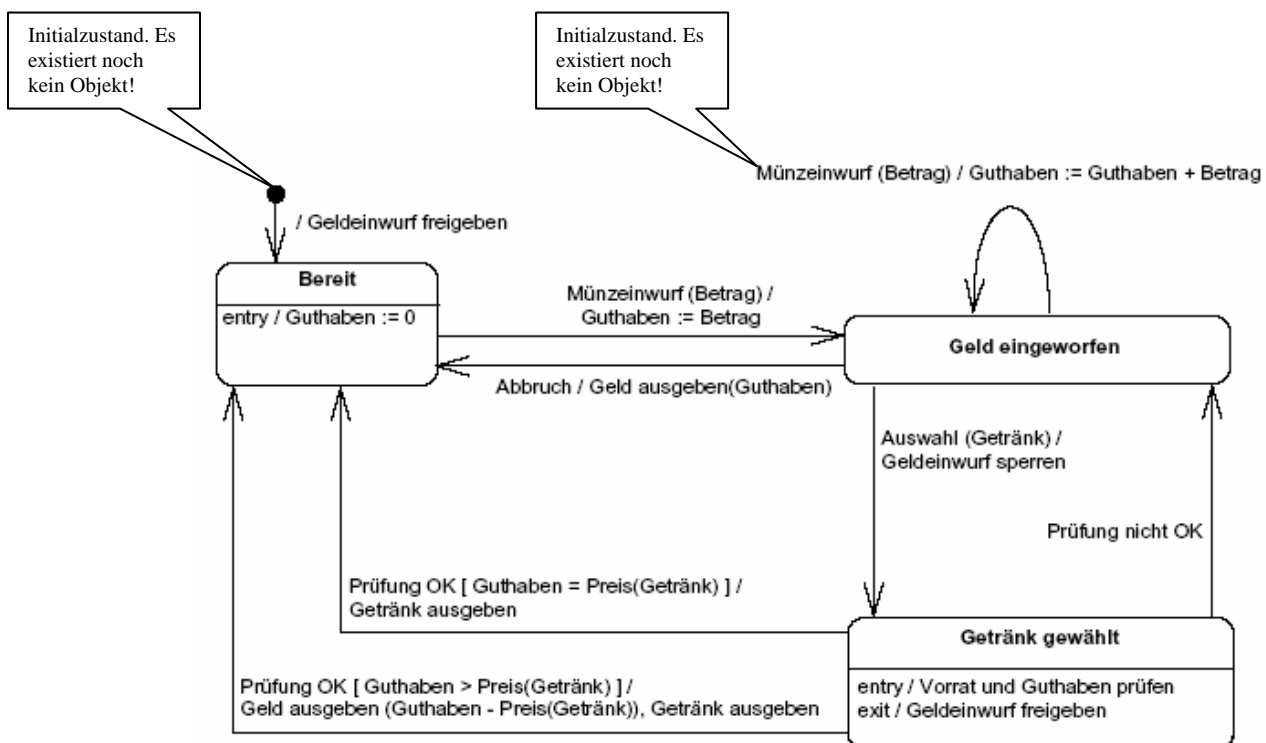


Abb. 15.5 Zustandsdiagramm für einen Getränkeautomaten.

Die vollständige UML-Syntax für die Beschriftung von Zustandsübergängen beinhaltet vier Teile, die alle optional sind:

Ereignis [Wächterbedingung] / Aktionsfolge ^Sendeaktion.

Eine **Aktionsfolge** besteht aus einem oder mehreren, durch Kommata getrennten Aktionsnamen, ggf. mit Parameterübergaben.

Eine **Sendeaktion** versendet eine Nachricht. In der UML-Syntax wird sie in der Form *Zielausdruck.Nachrichtename(Argumente)* angegeben, wobei der Zielausdruck zu einem Objekt, einer Menge von Objekten oder dem Gesamtsystem ausgewertet werden kann. In den letzten beiden Fällen handelt es sich um eine sog. Broadcast-Nachricht, die gleichzeitig an mehrere, dem Sender nicht unbedingt bekannte, Empfänger gesendet wird.