UML (Unified Modeling Language)

- UML ist grafische Modellierungssprache für **objektorientierte Softwareentwicklung**
  - für Spezifikation, Dokumentation, Visualisierung von (objektorientierten) Software-Systemen
  - de facto-Standard
  - unterstützt alle Phasen des Entwicklungsprozesses
- **Klassendiagramm**: Grafische Darstellung der statischen Struktur eines Softwaresystems
- Enthält Klassen, Attribute, Methoden und Beziehungen

- **UML 2 unterscheidet folgende Diagrammarten**

a) **Strukturdiagramme** (für statische Systemsicht)	b) **Verhaltensdiagramme** (für dynamische Systemsicht)
• **Klassendiagramm** • **Objektdiagramm** • **Paketdiagramm** • **Verteilungsdiagramm** • Komponentendiagramm • Kompositionsstrukturdiagramm • Profildiagramm	• **Use Case Diagramm (Anwendungsfalldiagramm)** • **Sequenzdiagramm** • Kommunikationsdiagramm • **Aktivitätsdiagramm** • **Zustandsdiagramm** • Timing-Diagramm (Zeitverlaufsdiagramm) • Interaktionsübersichtsdiagramm

Inhaltsverzeichnis

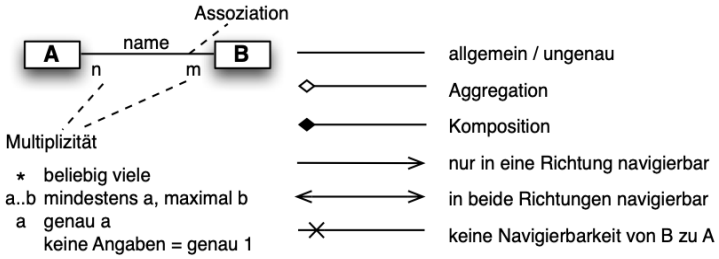
1 Strukturdiagramm (für statische Systemsicht)

1.1 UML-Klassendiagramm

Multiplizität

gibt an, mit wie vielen Objekten man in Beziehung steht (sog. **Kardinalität**)

- Notation: einfache Zahl (0,*) oder Intervall (0,n)
- n = (0,n) = (0,*)
- Das * steht dabei für eine beliebige Zahl



Beziehungen

Abhängigkeit

Schwächste Form der Beziehung - **“nutzt vorübergehend”**

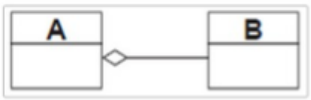
- Temporäre Abhängigkeit (z. B. nur für die Dauer eines Methodenaufrufes) notiert man in UML mit der **gestrichelten Linie**.

Assoziation

- **"benutzt ein/e" || "ist zugeordnet zu" || "hat eine Beziehung zu" || “kennt”**
- relativ lose Kopplung, dauerhaft oder temporär
- *Beispiele A/B: Mann/Frau Person/Computer Tafel/Kreide*

Aggregation

- **"besitzt ein/e"**
- Stärkere Beziehung als Assoziation, assoziiert Besitz
- *Beispiele A/B: Auto/Fahrer Restaurant/Kunde Mannschaft/Spieler*

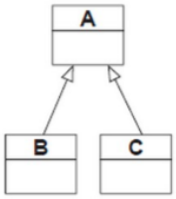


Komposition

- **“besteht aus“ || „Ist ein Teil von“**
- stärkste Form der Beziehung
- 1 kann auf der Seite der Komposition weggelassen werden!
- Sie beschreibt, wie sich etwas Ganzes aus Einzelteilen zusammensetzt.
- *Beispiele A/B: Mensch/Herz Buch/Kapitel Gebäude/Raum*

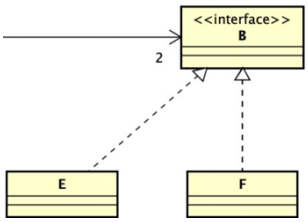
Vererbung

- **"Ist ein"**
- Generalisierung, Spezialisierung
- *Beispiele A/B,C,...: Fahrzeug/Auto,Bus,Bahn,... Beruf/Politiker,Professor,Maurer,...*



Interface

- Klassenrechtecksymbol, das das Schlüsselwort «interface» enthält.
- Diese Notation wird auch als interne oder Klassenansicht bezeichnet

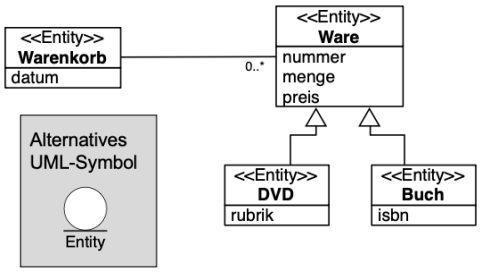


1.2 UML-Klassendiagramm (Class Diagram)

- Ziel: Modell der Gegenstände/Konzepte der fachlichen Domäne, deren Beziehungen und Operationen
- Modellierung der statischen Sicht durch **UML-Klassendiagramme**
  - Modelle enthalten ausschließlich **fachliche** Klassen
  - zeigen welche Klassen und Methoden es gibt aber nicht deren Zusammenspiel

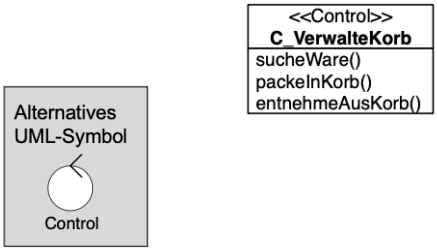
Klassentypen

1. **Entity**-Klassen (**Entitäts**klassen)
   - repräsentieren **Objekte** der realen Welt, z.B. Person, Kunde, Adresse, Vertrag, Buch
   - modellieren **Daten** (= Attribute und Beziehungen), die i.d.R. **persistent** abgespeichert werden
   - bieten Methoden für konsistenten Zugriff auf Daten (= Attribute)

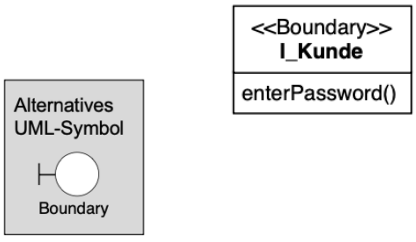


- Entities sind datentragend, Enthv§lt fachliche Logik
- Methoden bieziehen sich nur auf Daten

2. **Control**-Klassen (**Steuerungs**klassen)
   - bieten nur **Dienste** (Geschäftslogik / fachliche Operationen) an, d.h. keine (Daten-)Attribute
   - auch Methoden, die keiner Entity zugeordnet werden können (Entity-/Klassen-übergreifend)
   - realisieren komplexe **Abläufe**, nutzen hierzu mehrere Entity- sowie andere Control-Klassen

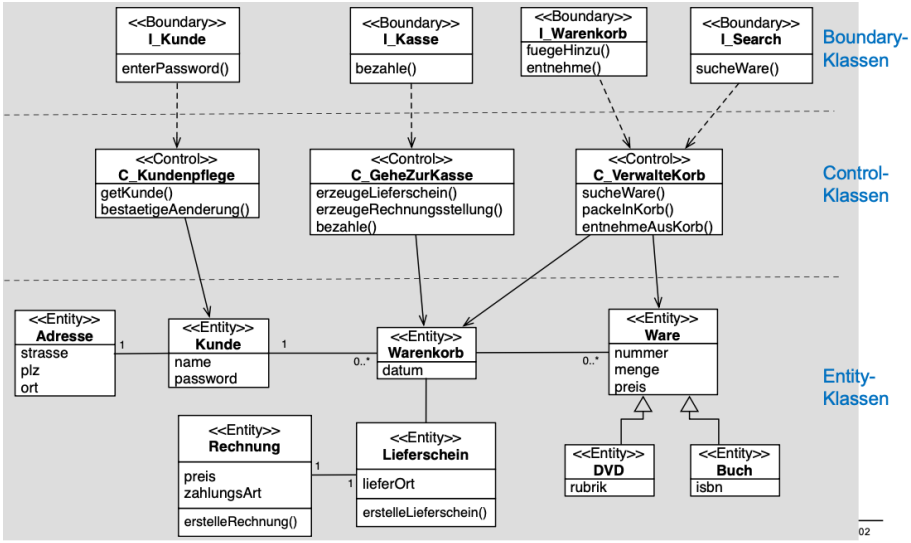


3. **Boundary**-Klassen (**Anwenderschnittstellen**klassen)
   - beinhalten **Operationen** für Interaktion zwischen Akteur und System
   - bilden **fachliche Schnittstelle** (stellen fachl. Operationen bereit), d.h. enthalten selbst **keine** Logik, Funktionalität, Abläufe
     - Akteur darf **nur** über Boundary-Klassen mit System kommunizieren
     - delegieren Aufrufe an Control-Klassen



- Schnittstelle zwischen Fachlichkeit und GUI / anderes System, also die Akteure die damit interagieren
- **Ziel: Logik wird verbirgt. Damit kann man Logik ändern ohne die Schnittstelle zu ändern**
- Schnittstelle & Implementierung getrennt
  (Wenig Redundanz, aber die ist gewollt. I_Klasse bezahle() ruft C_GeheZurKasse bezahle() auf)

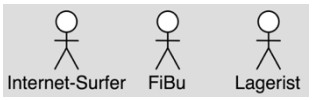
Internet-Buchhandlung (nur Ausschnitt !) (i.w. Klassen für Kunden-bezogene Use Cases)



2 Verhaltensdiagramme (für dynamische Systemsicht)

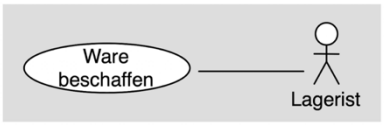
2.1 Use Case Diagramm (Anwendungsfalldiagramm)

UML-Notation für Akteure



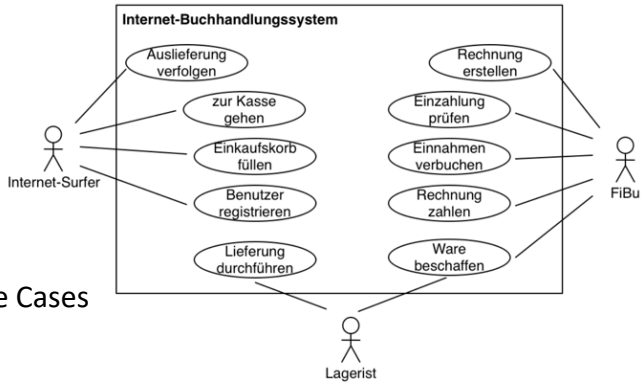
UML-Notation für Use Cases

- Oval für Use Case
- Linie für Kommunikationsbeziehung
- Use Case hat Namen, der i.d.R. ein Verb enthält



Use Case Diagramm (Anwendungsfalldiagramm)

- Überblick über alle Use Cases
- beschreibt Zusammenspiel mehrerer Use Cases untereinander und mit den Akteuren
- System durch Rechteck modelliert, dass alle UCs umschließt



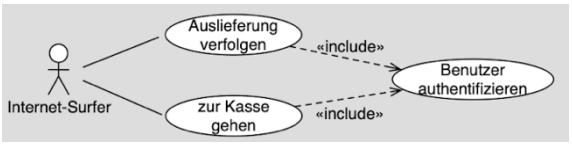
Problem: sinnvolle **Granularität** von Use Cases

- wie viel Funktionalität ist in einem Use Case beschrieben?
- wenige (15-20) große Use Cases versus viele (100) kleine Use Cases
- häufiger Fehler: zu viele kleine Use Cases

Strukturierung von Use Cases mit Stereotypen

(a) **<<include>>** – Beziehung verweist auf einen anderen (Sub-)Use Case

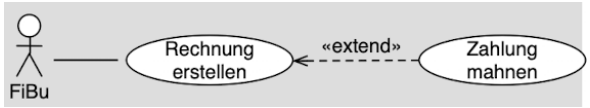
- Vermeidung von redundanten Ereignisflüssen



(b) **<<extend>>** – Beziehung für **optionalen** Ereignisfluss

- spezielle Use Cases für umfangreiche Ausnahmen, Varianten und Sonderfälle
- wenn Use Case den Erweiterungspunkt erreicht, wird entspr. Ereignisfluss Ausgeführt

Vorsicht: Strukturierung macht UC-Diagramm unübersichtlicher !



Sehr große Iterationen

Vorteil:

- Alle Mitarbeiter beschäftigt
- Weniger Redundanz

Nachteil:

- Testen der Ergebnisse zieht sich
- Wahrscheinlichkeit für Misserfolg steigt, da zu viel Use Cases parallel gemacht werden

Sehr kleine Iterationen

Vorteil:

- Schnell Ergebnisse zum Präsentieren

Nachteil:

- Mitarbeiter ohne Beschäftigung
- Sehr viel redundante Arbeit

2.1.1 Beschreibung von Use Cases

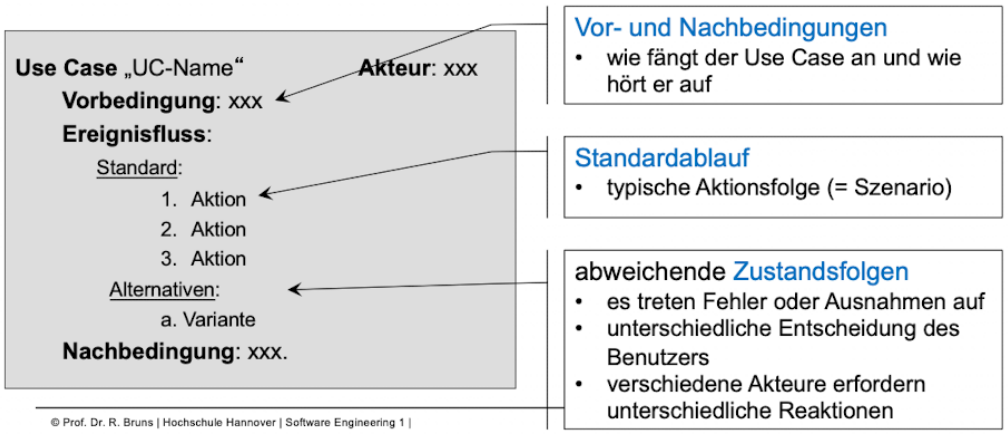
– Beschreibung des Ereignisflusses eines Anwendungsfalls als Folge von Aktionen im Detail
– Festlegung der Verantwortlichkeiten von System und Akteur

- ein Use Case umfasst sehr viele Abläufe (= Szenarien)
  - zunächst typische Szenarien beschreiben, danach Varianten/ Abweichungen
  - Use Cases werden im Projektverlauf verfeinert; weitere Szenarien beschreiben
- Beschreibungsmittel
  - ggf. Verwendung von weiteren UML-Diagrammen

Standard-Formular (use case template) nutzen

• **notwendige Informationen**: Name, Ziel, Akteure, auslösendes Ereignis, Vor-/Nachbedingungen, Beschreibung des Standardfalls sowie Erweiterungen, Ausnahmen etc.

• **Beispielformular**



Use Case 1: „zur Kasse gehen“ **Akteur**: Internet-Surfer

Vorbedingung: Benutzer ist bereits mit seinen persönlichen Daten registriert

Ereignisfluss:

Standard:

1. Kunde will zahlen, dazu klickt er auf das Warenkorb-Icon.
2. Er erhält eine Übersicht der Waren mit der Menge, dem zu bezahlenden Betrag und dem Lieferdatum.
3. Er bestätigt die ausgesuchten Waren und erhält nach Eingabe von Namen und Passwort seine persönlichen Daten mit der Lieferadresse und der Zahlungsart (Lastschrift oder Kreditkarte).
   Ihm werden einige Ziffern seines Kontos / seiner Kreditkartennummer angezeigt.
4. Er bestätigt Lieferanschrift und Zahlungsart und initiiert damit die Lieferung und Rechnungsstellung. Damit ist der Use Case beendet.

Alternativen:

– zu 3: er entfernt Waren direkt aus dem Warenkorb
– zu 4: a) er ändert die Lieferadresse für ein Geschenk.
  b) er ändert Zahlungsart und spezifiziert die entsprechenden Daten

Nachbedingung: Use Case endet, wenn die Lieferung der Waren veranlasst ist.

UC1: zur Kasse gehen Akteur: Internet-Surfer

Vorbedingung: Benutzer ist bereits mit seinen persönlichen Daten registriert

Szenario 1:

Verantwortlichkeit des Akteurs	Verantwortlichkeit des Systems
1. Kunde will zahlen, dazu klickt er auf das Warenkorb-Icon	2. Zeigt Übersicht der Waren mit der Menge, dem zu bezahlenden Betrag und dem Lieferdatum.
3. Er bestätigt die ausgesuchten Waren und gibt Name u. Passwort ein.	4. Zeigt persönliche Daten mit der Lieferadresse und der Zahlungsart an (Lastschrift oder Kreditkarte). Einige Ziffern seines Kontos/ seiner Kreditkartennr werden angezeigt
5. Er bestätigt Lieferanschrift und Zahlungsart	6. Initiiert Lieferung und Rechnungserstellung. UC endet

Alternativen:

- zu 3.: er entfernt Waren direkt aus dem Warenkorb
- zu 5.: a) er ändert die Lieferadresse für ein Geschenk.
  b) er ändert die Zahlungsart und spezifiziert die entsprechenden Daten

Nachbedingung: Use Case endet, wenn die Lieferung der Waren veranlasst ist.

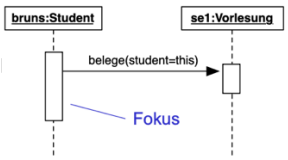
2.2 Sequenzdiagramm

UML-Sequenzdiagramm (sequence diagram)

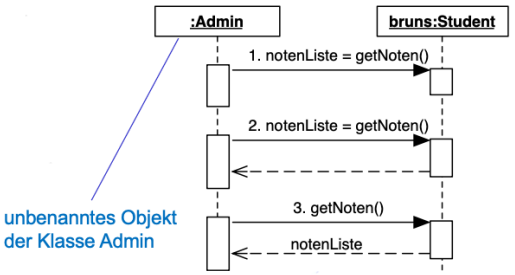
- Sequenzdiagramm muss mit Klassendiagramm konsistent sein
- **Aufbau dynamischer Modelle**
  - spielen genau ein Szenario (eines Use Cases) durch
  - als Abfolge von Methodenaufrufen zwischen konkreten Objekten
    - enthalten die am Use Case partizipierenden **Objekte**
  - **Akteur** (aus Use Case) kann ein Szenario initialisieren
- modelliert **Interaktionen** zwischen **Objekten**
- ein Objekt ruft eine Methode eines anderen (ihm bekannten) Objektes auf

Notation

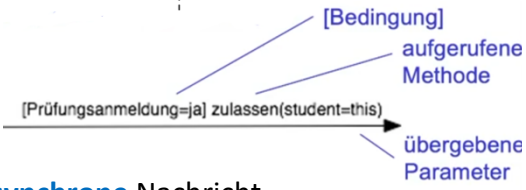
- **Fokus** (Aktionssequenz) - Box
  - zeigt an, wie lange eine Methode abgearbeitet wi
  - verdeutlicht Kaskadierung von Methoden



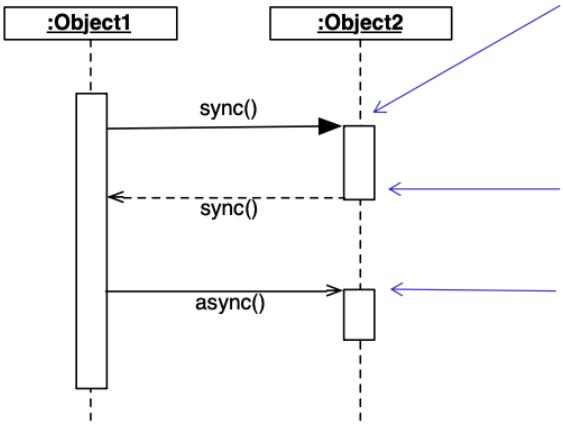
- **Rückgabewert** (reply message)
  1. In Textform an Pfeil
  2. als eigener Pfeil
  3. Mit Variable/Wert an Pfeil



- **Bedinungen (guard)** an Pfeile der Methodenaufrufe



- Interaktion zwischen Objekten durch **synchrone** oder **asynchrone** Nachricht



(a) synchrone Nachricht: (gefüllte Pfeilspitze)
Senderobjekt wartet, bis Empfängerobjekt die Verarbeitung komplett beendet hat

Empfänger schickt Antwortnachricht

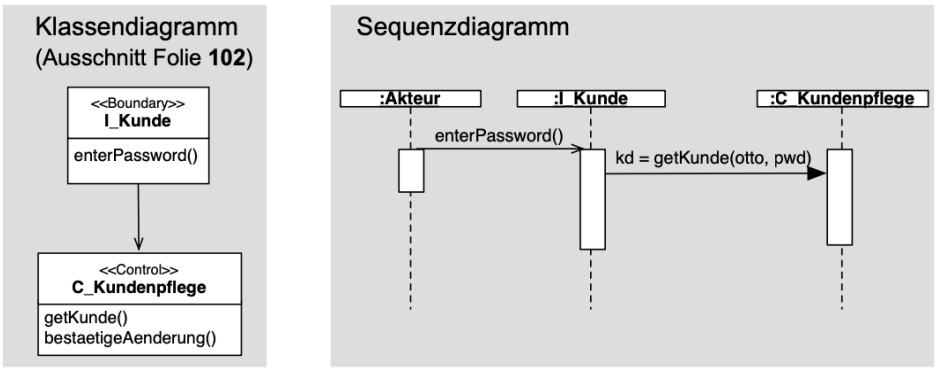
(b) asynchrone Nachricht: (offene Pfeilspitze)
Sender wartet nicht auf Verarbeitungsende durch Empfänger, sondern setzt parallel eigene Verarbeitung fort

Beispiel Sequenzdiagramm 1

Akteur kommuniziert ausschließlich über Boundary-Klasse, diese nur mit Control-Klassen, diese mit anderen Control- oder Entity- Klassen

- Akteur interagiert mir GUI und ruft die Boundary-methoden auf. Es gibt kein Akteur/Kunden Objekt
- Control-Objekte werden beim Systemstart initialisiert

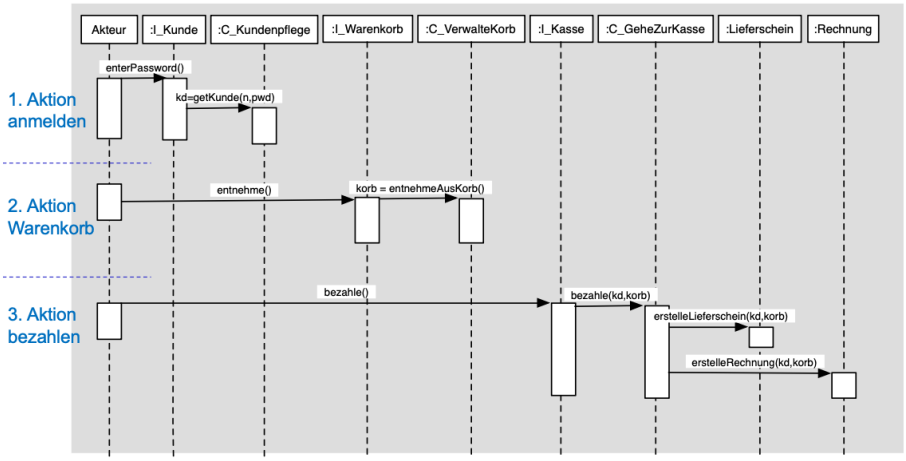
Sequenzdiagramm für Anmeldung im UC: „Benutzer meldet sich im System an“.



Beispiel Sequenzdiagramm 2

- Akteur interagiert mir GUI und ruft die boundary-methoden auf. Es gibt kein Akteur/Kunden Objekt
- Control-Objekte werden beim Systemstart initialisiert

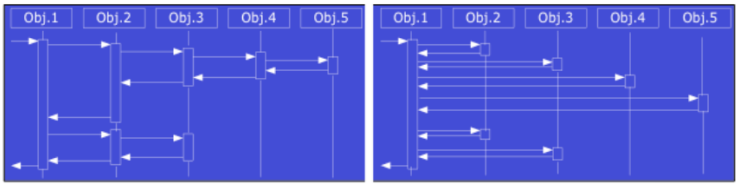
Use Case „zur Kasse gehen“ (nur Ausschnitt!) (siehe UC-Beschreibung Folie 74)



UML-Sequenzdiagramm

Vorteile:

- betonen den **zeitlichen Aspekt** des dynamischen Verhaltens
- **Reihenfolge** und **Verschachtelung** der Methoden bzw. **Zusammenspiel der Objekte sind leicht zu erkennen**
- Sequenzdiagramme zeigen Kontrollfluss innerhalb eines SW-System



"Stair" "Fork"

- Kommunikationsstruktur gut ersichtlich: welche Objekte kennen sich
  - z.B. durch Assoziationen, Parameter, Rückgabewerte

welche Daten werden ausgetauscht (als Parameter oder Rückgabewerte)

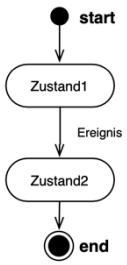
2.3 Zustandsdiagramm

- Methodenaufrufe vom Zustand abhängig
- Problem: **Lebenszyklus eines Objekts** beschreiben
  - Objekte befinden sich in verschieden Zuständen mit unterschiedlichem **Verhalten**

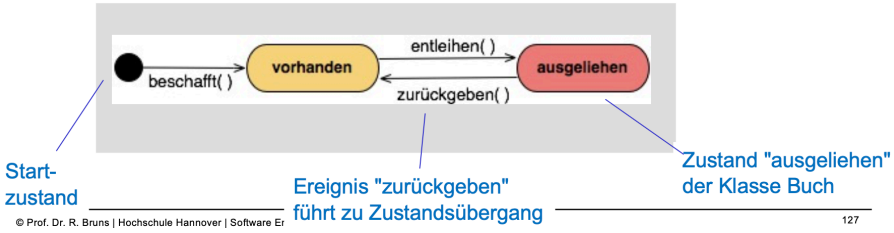
- Zustandsdiagramme sind sinnvoll
  - Wenn sich das **Verhalten** eines Objekt signifikant ändert
  - **Nur für Klassen mit komplexen** (z.B. zeitabhängigen Verhalten, z.B. Fahrkartenautomat
  - **Theoretische Grundlage: endliche Automaten**

Elemente eines UML-Zustandsdiagramms:

- ein **Startzustand** (initial state)
- **Zustand** (repräsentiert durch Gesamtheit der Attributwerte)
- **Übergänge** (**Transitionen**) zwischen Zuständen
- **Ereignisse**, die Zustandsübergänge bewirken (z.B. Erhalt einer Nachricht, Bedingung, Zeitablauf)
- ein (oder mehrere) **Endzustand** (final state) (Mehrere Endzustände aus Übersichtlichkeitsgründen)

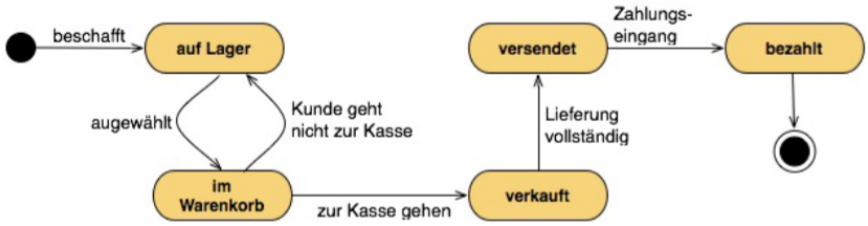


- z.B. Zustände für Klasse Buch in Bibliothek



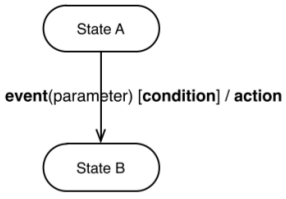
- Zustandsdiagramme verdeutlichen **Zusammenhänge und Abhängigkeiten** von Aktionen
- Verhalten eines Objekts über **mehrere Anwendungsfälle**
- Beliebige Ereignisse möglich, müssen keine Methoden sein

Beispiel: Zustandsmodell für Klasse Buch in Internet-Buchhandlung



Zustandsübergang mit Guard und Aktion

- Ergebnis löst Übergang zwischen zwei Zuständen aus
- Ergebnis kann mit einer Bedingung versehen werden
- Ergebnis kann beim Eintreten eine Aktion auslösen



Ergebnis [Bedingung]/ Aktion

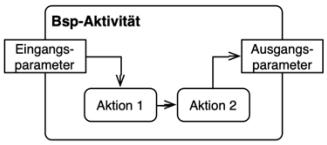
2.4 UML-Aktivitätsdiagramm

- **Problem: Beschreibung des Ablaufs komplexer Prozesse durch Aktivitäten/ Aktionen/ Schritt**
  - Zustandsdiagramme nur für **eine Klasse** (Zustandsraumexplosion)
  - Use Cases nur **exemplarische Abläufe** (typische und alternative Abläufe, aber keine Vollständigkeit)
- Ziele
  - **Verhalten von Objekten in mehreren UC-Szenarien bzw. Use Cases zeigen**
    - alternative Abläufe darstellen
    - iterative Abläufe darstellen
    - potenziell parallele Abläufe identifizieren (z.B. Threads) (Synchronisation von nebenläufigen Aktivitäten)
  - (generell) Spezifikation eines komplexen Ablaufs/ komplexer Funktionalität

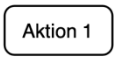
---

UML-Notation

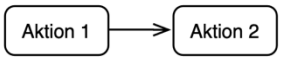
1. **Aktivität (activity)**
   - gesamtes Diagramm beschreibt eine Aktivität



2. **Aktions**knoten
   - Aktion (action): kleinste ausführbare Einheit
     - Aufgabe/Prozessschritt oder Methode
   - Aktion kann ausgeführt werden, wenn Vorgänger-Aktion beendet ist

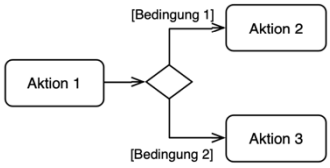


3. **Pfeil/ Kante (→):** Übergang zur folgender Aktion
   - - Aktion (action): kleinste ausführbare Einheit

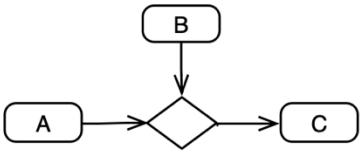


4. **Kontroll**knoten

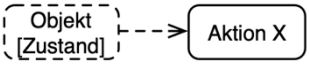
   a) Entscheidung
   - Verzweigung abhängig von Bedingung



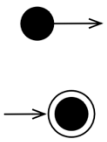
   b) Zusammenführung
   - nach Eintreffen von A oder B wird Zweig C fortgesetzt



5. **Verknüpfung Aktionen mit Objekten**sddd
   - Verknüpfung von Aktionen mit Objekten bzw. deren Zuständen

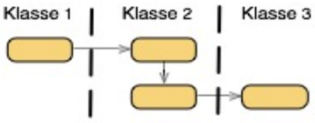


6. **Start-/Endknoten**
   - Aktivität kann einen oder mehrere Startknoten/ besitzen (Aktionen starten parallel)
   - Aktivität kann einen oder mehrere besitzen



7. **Schwimmbahnen ("swimlanes")**

   verdeutlichen die **Verantwortlichkeiten** (für Aktionen, Entscheidungen)
   - Rollen/ Organisationseinheiten (konzeptionell)
   - Klassen (spezifizierendes, konzeptionelles Modell)

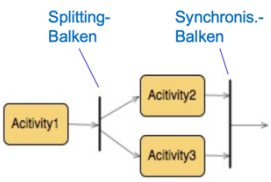


8. **Nebenläufigkeit**

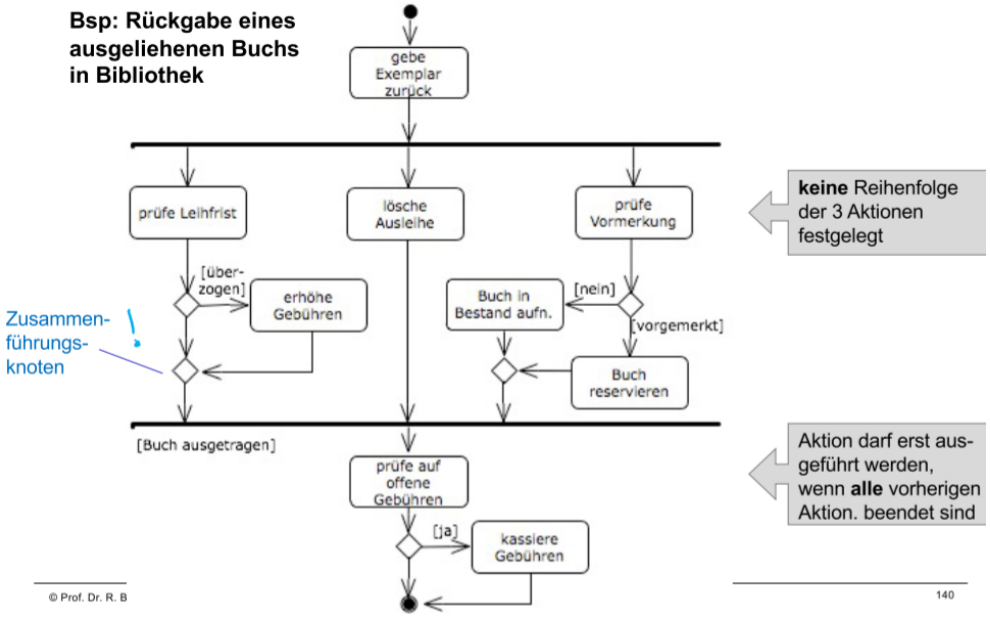
   a) Splitting (fork node, Aufspalten):
   - Kontrollfluss auf mehrere parallel Ströme aufgeteilt

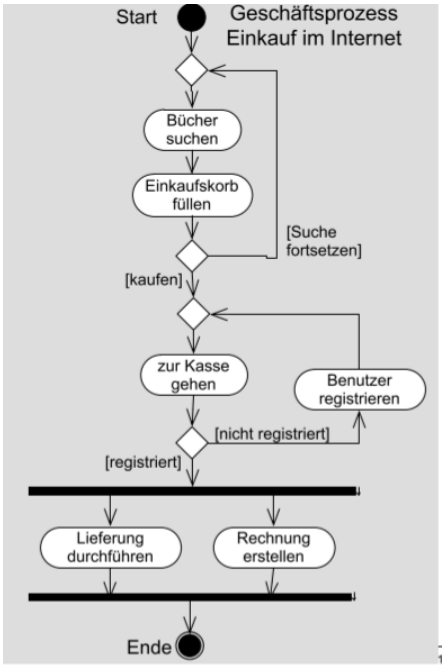
   b) Synchronisation (join node):
   - Zusammenführen mehrerer Aktionen

(alle beteiligten Aktionen müssen beendet sein, bevor nachfolgende Aktion starten kann)



Beispiele





Vor- und Nachteile

- Stärken
  - ermöglichen Modellierung komplexer Abläufe
  - bei der Darstellung nebenläufiger Prozesse
    - fachliche Analyse: Geschäftsprozesse (siehe Kap 3.2)
    - technisch: Nebenläufigkeit durch Threads realisiert
  - zeigen alternative Abläufe und Schleifen (übersichtlicher als in Sequenzdiagrammen)

- Schwächen
  - Beziehung der Aktivitäten zu Objekten schlecht sichtbar
    - Auswege: swimlanes oder Objektnamen in Aktionen aufnehmen
  - wird schnell unübersichtlich
  - für Verhalten eines einzelnen Objektes nicht geeignet
    ⇒ Zustandsdiagramm