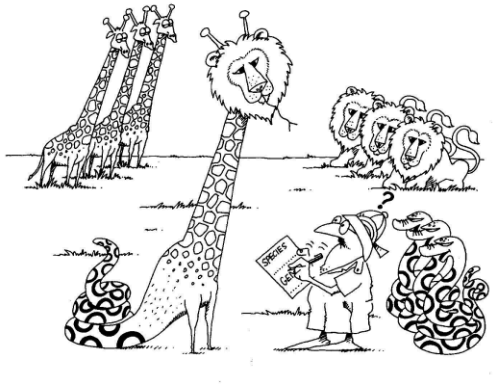
Media Engineering
Objektorientierte Modellierung
Verhaltensmodellierung



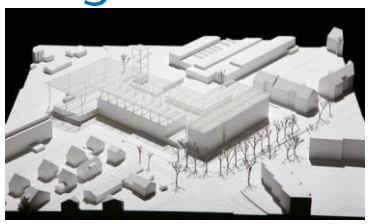
R. Weller

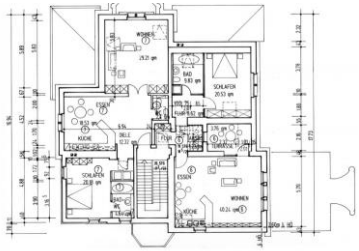
University of Bremen, Germany

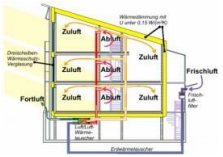
cgvr.cs.uni-bremen.de

Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein => Klassendiagramm
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

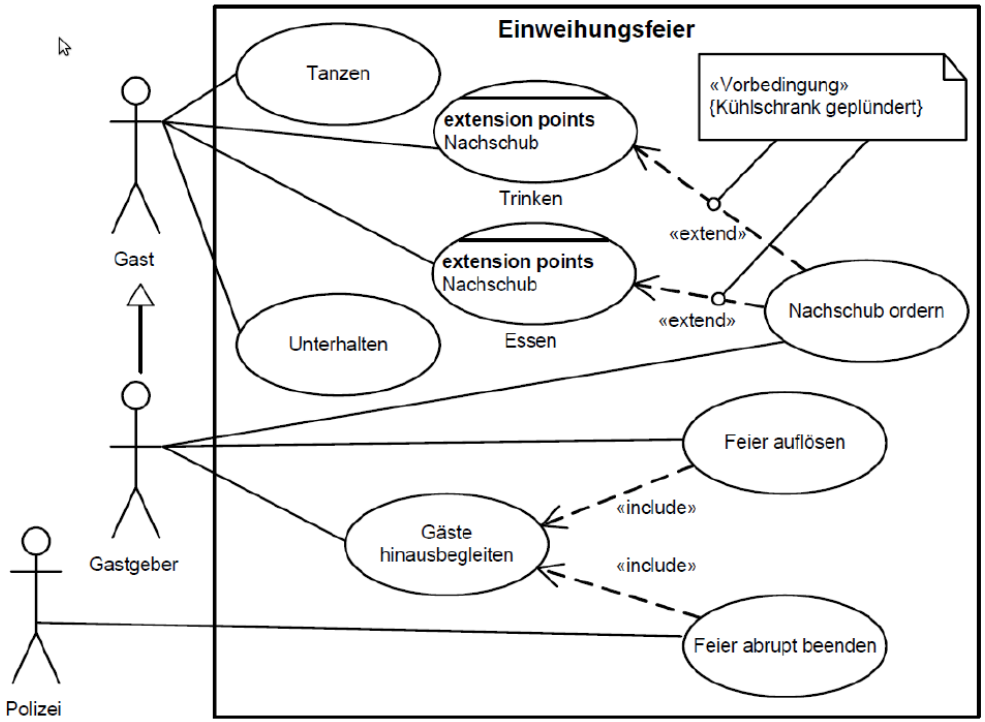






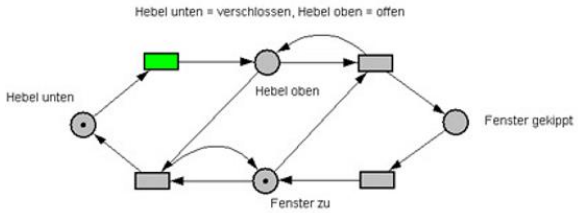
Zur Erinnerung: Use-Case-Diagramme

- Beschreiben Interaktionen zwischen Akteuren und System
- Aber nur strukturelle Darstellung, bilden keine dynamischen Aspekte ab



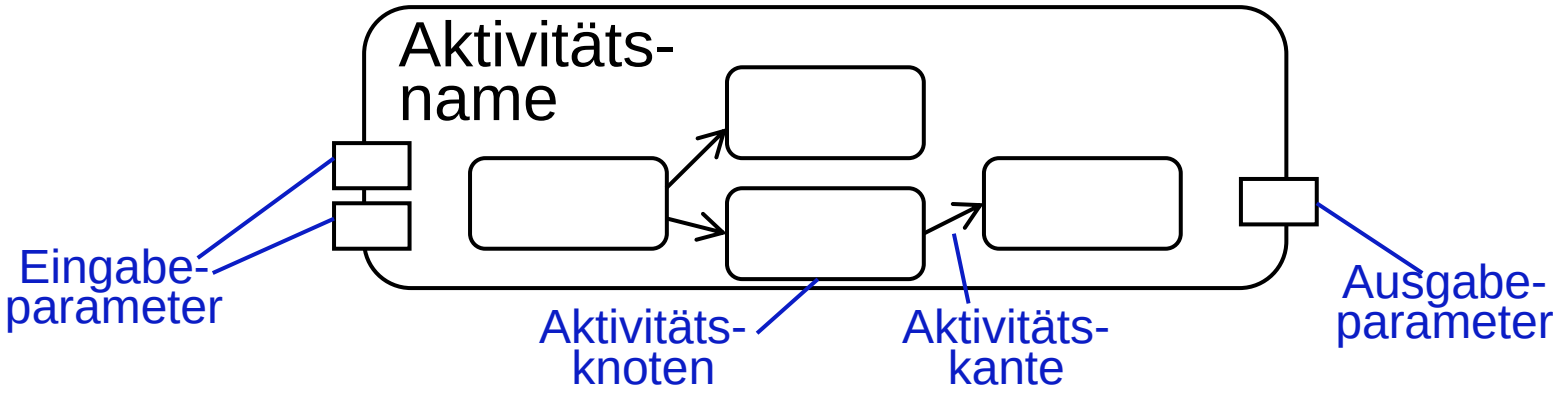
Aktivitätsdiagramme

- Spezifizieren Kontroll- und Datenfluss zwischen verschiedenen Arbeitsschritten (genannt Aktionen), die zur Realisierung einer Aktivität notwendig sind
  - Beschreiben Reihenfolge
  - und Abhängigkeiten
- Modellierung klassenübergreifenden Verhaltens (Kontrollfluss)
- Zur detaillierten Beschreibung von Anwendungsfällen
  - Weniger geeignet für die Beschreibung der Interaktion von verschiedenen Objekten
  - Oder für Zustandsänderungen eines einzelnen Objekts
- Wurzeln: Flussdiagramme und Petrinetze

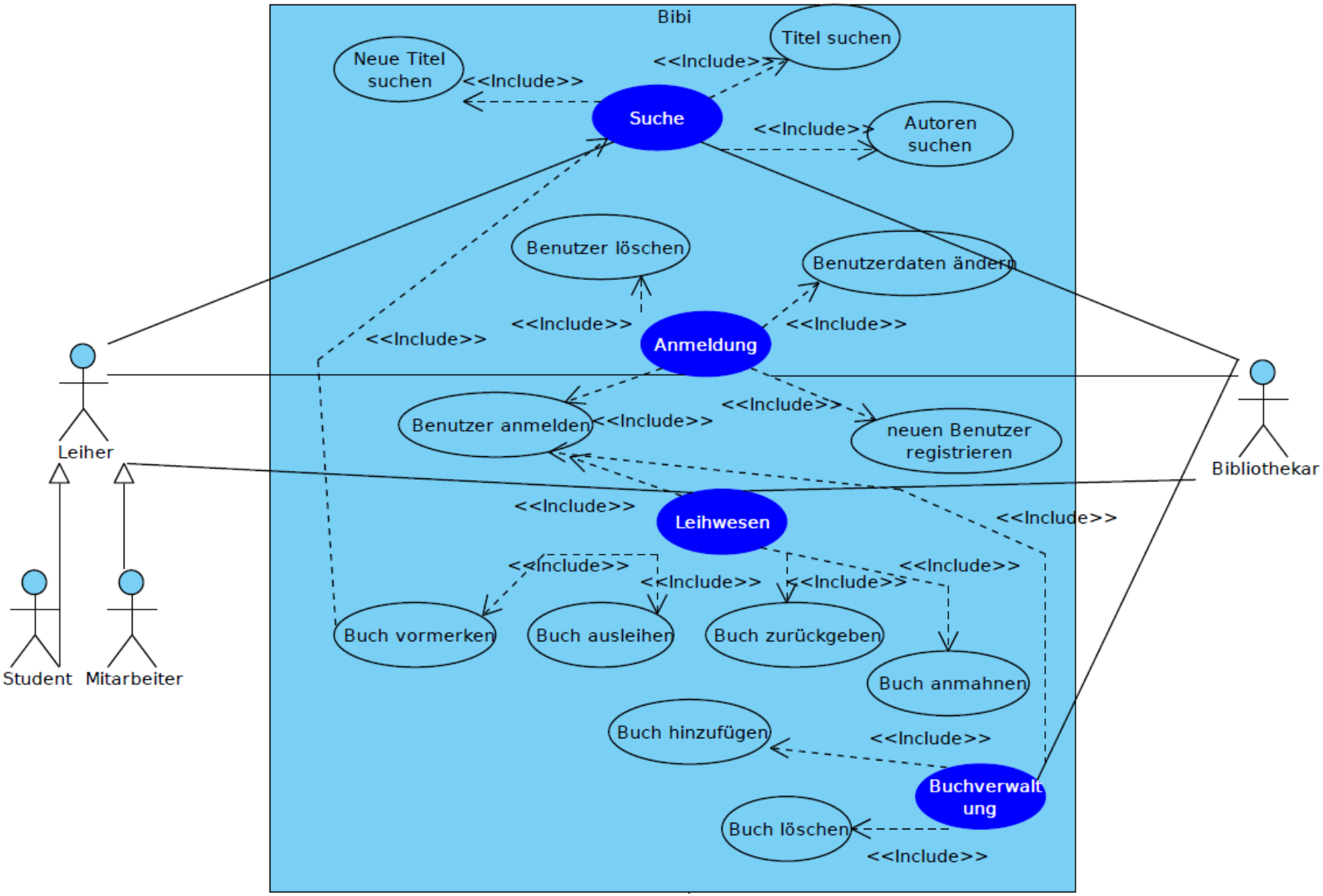


Aktivitäten und Aktionen

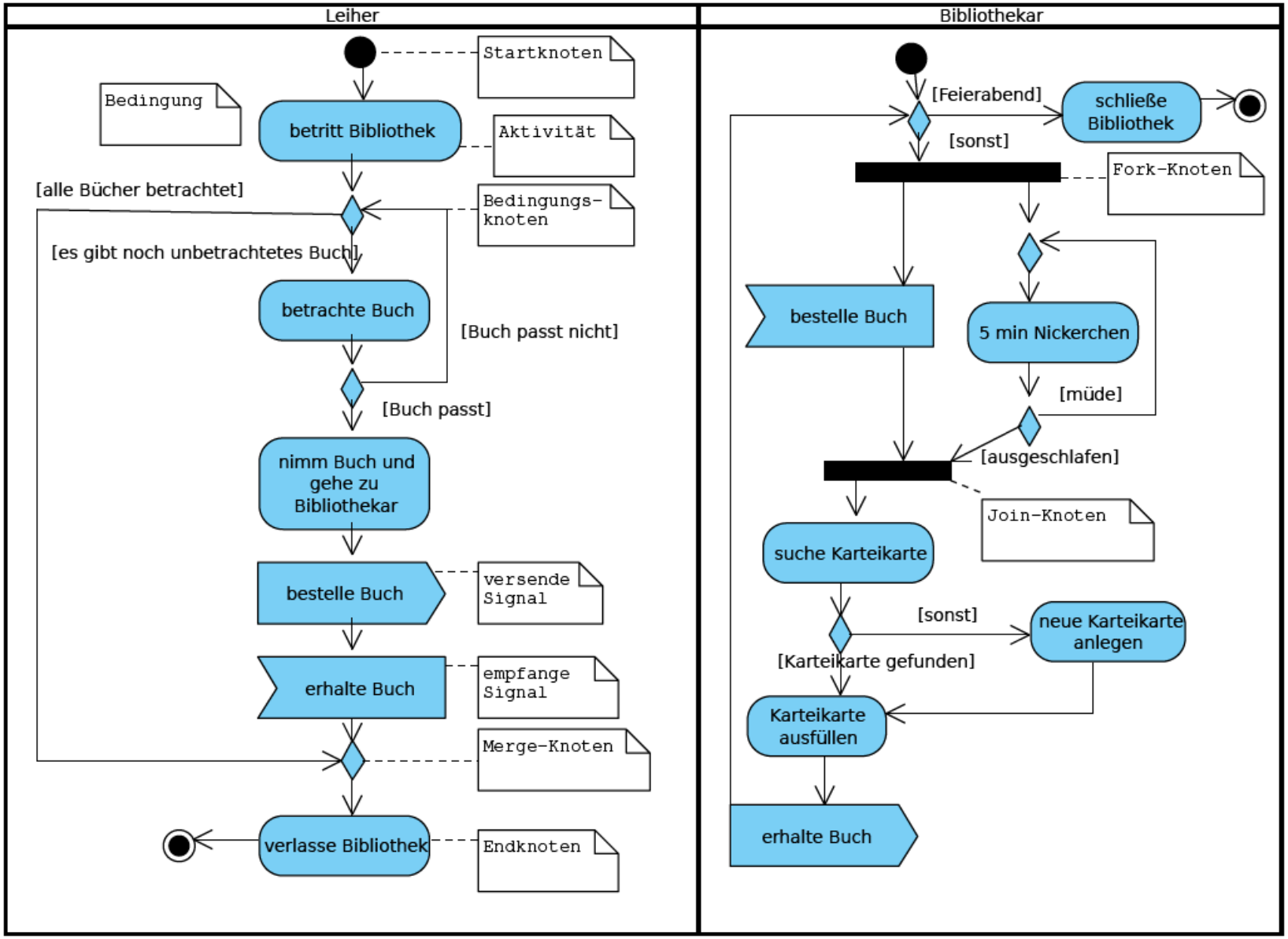
- Aktivität = gesamte Verhaltensbeschreibung im Aktivitätsdiagramm
- Aktion = atomarer Bestandteil einer Aktivität
  - Aktionen leisten die „eigentliche Arbeit“
- Aktivitätsdiagramm = gerichteter Graph mit Aktivitätsknoten und Aktivitätskanten
  - Aktivitätsknoten = Aktionen, Objekte, Kontrollkonstrukte
  - Aktivitätskanten = Abhängigkeiten in Form von Weitergabe von Kontrolle oder Daten



Zur Erinnerung: Beispiel Bibliothek



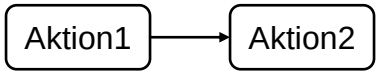
Beispiel Aktivitätsdiagramm



Notation von Aktivitätsdiagrammen in UML

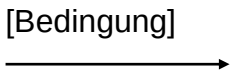
Aktion

Kontrollfluss

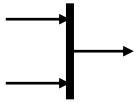


Aktion2 wird nach Abschluss von Aktion1 gestartet.

Verzweigungsaktion (kann auch durch normale Aktion dargestellt werden)

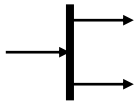


Kontrollfluss, der unter der angegebenen Bedingung gewählt wird.

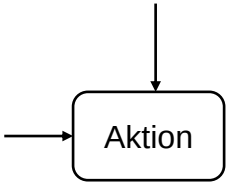


[Bedingung]

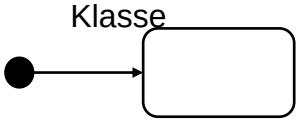
Synchronisation der Kontrolle (AND) mit Synchronisationsbedingung (Die Bedingung ist optional.)



Aufsplitten der Kontrolle (Zulassen von Parallelität)



Aktion wird durchgeführt, wenn über einen der eingehenden Kontrollflüsse die Kontrolle ankommt (OR)



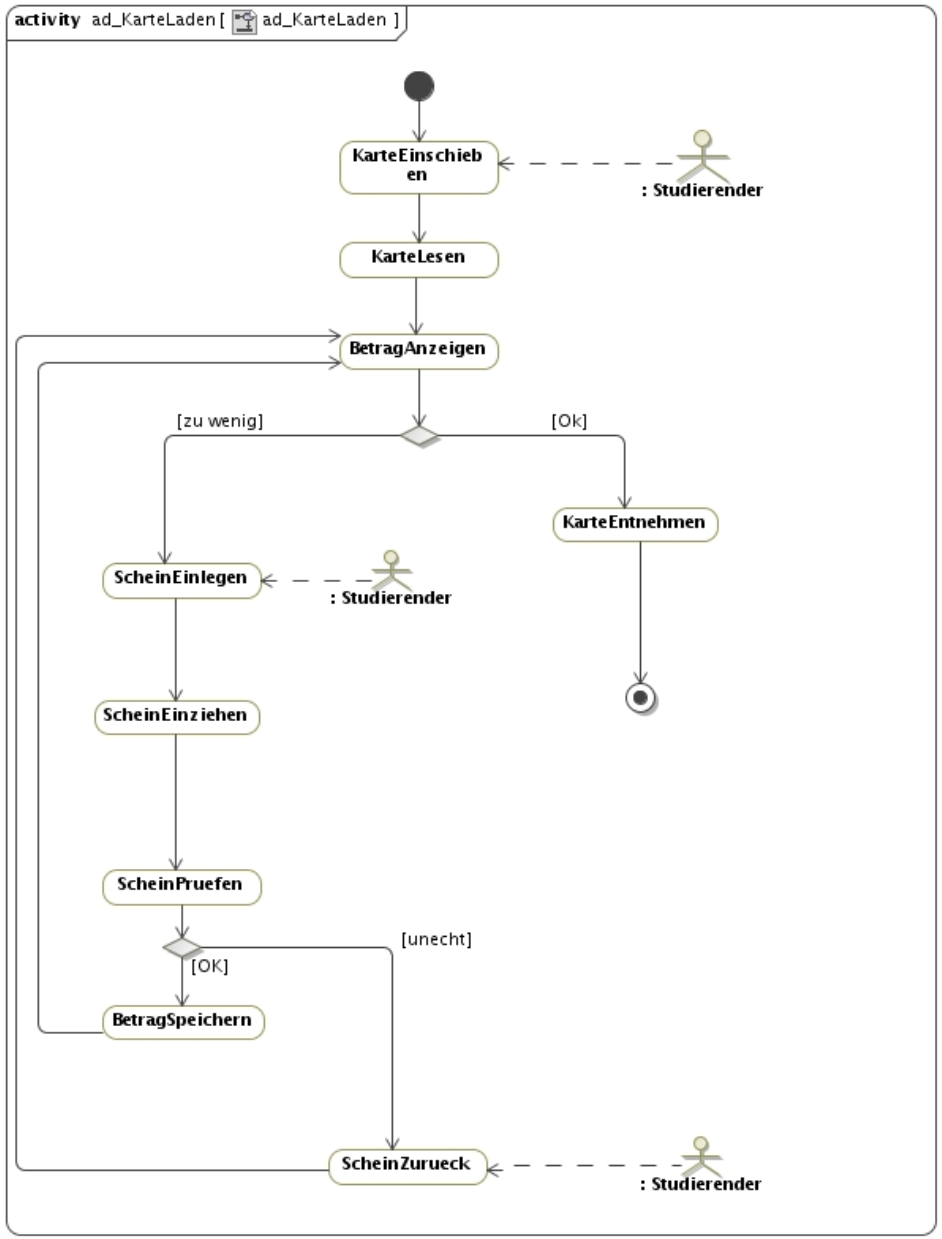
Start des Ablaufs und Identifikation der betroffenen Klasse

Aktion zum Senden eines Signals

Ende des Ablaufs (optional)

Aktion zum Empfangen eines Ereignisses

Ein weiteres Beispiel



Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein => Klassendiagramm
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen

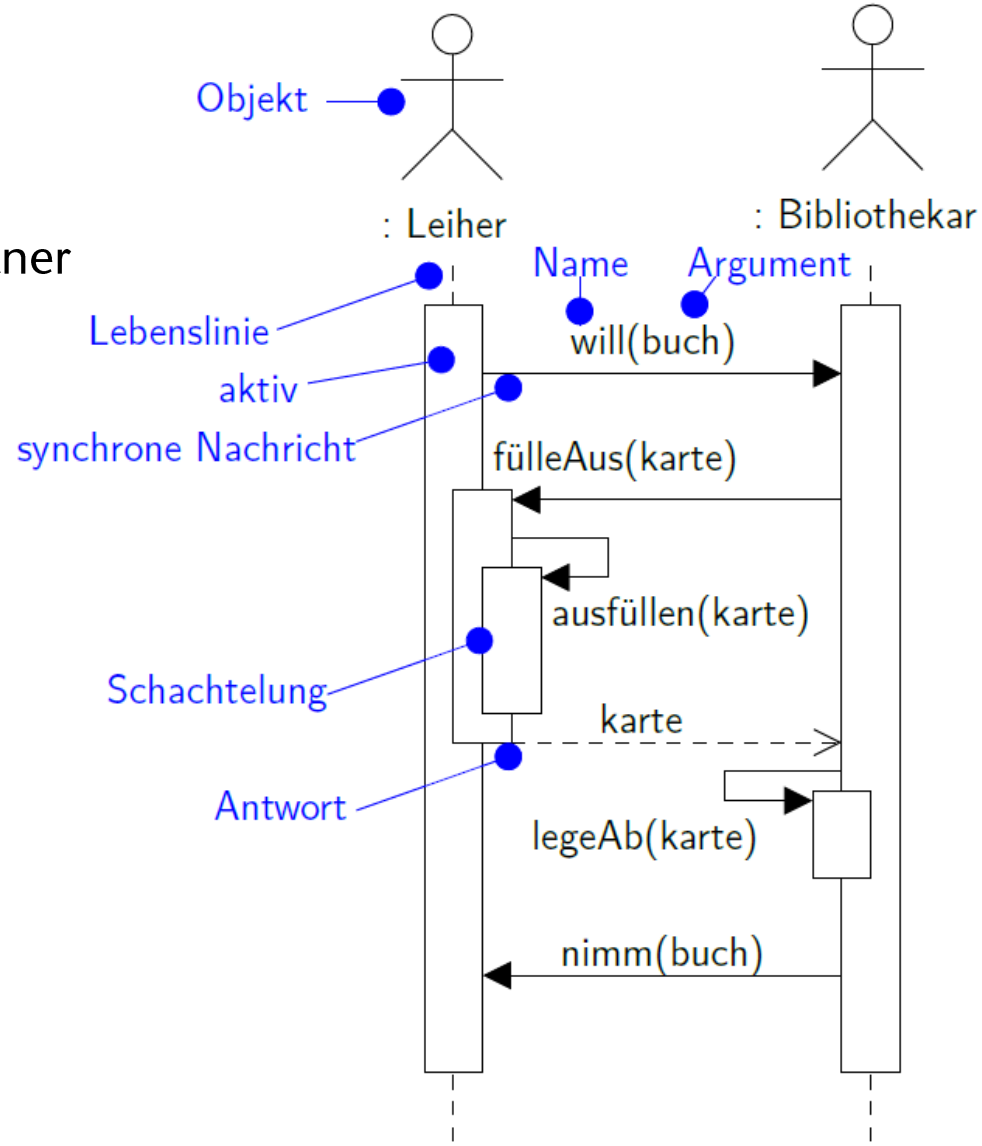
  => Aktivitätsdiagramm
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Interaktionsdiagramme

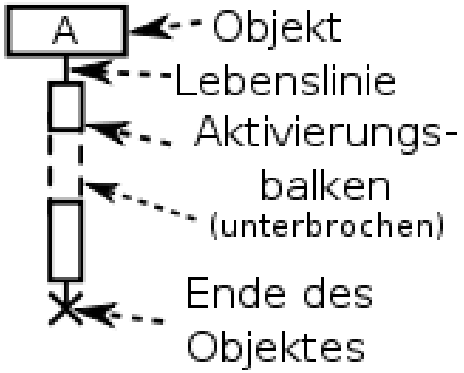
- Was noch fehlt: Interaktionen zwischen Objekten
- Interaktionsdiagramme spezifizieren Interobjektverhalten in Form von Nachrichten zwischen Objekten in bestimmten Rollen
  - Bilden Zeitliche Abläufe (Aufrufsequenzen) ab
  - Protokollieren Nachrichtenaustausch
- Zwei semantisch äquivalente Varianten:
  - Sequenzdiagramme
    - Betonen zeitlichen Ablauf
    - Verwendung bei wenigen Klassen
    - Basieren auf Message Sequence Charts (MSCs) der ITU-T
  - Kommunikationsdiagramme
    - Betonen Objektstruktur
    - Verwendung bei wenigen Nachrichten

Beispiel: Sequenzdiagramm

- Zwei Dimensionen
  - Vertikal: Zeitachse
  - Horizontal: Interaktionspartner
- Pfeile zeigen Nachrichtenaustausch



Notation von Sequenzdiagrammen in UML

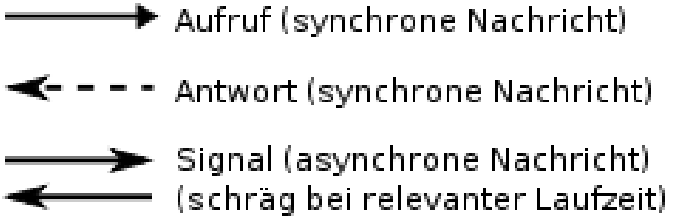


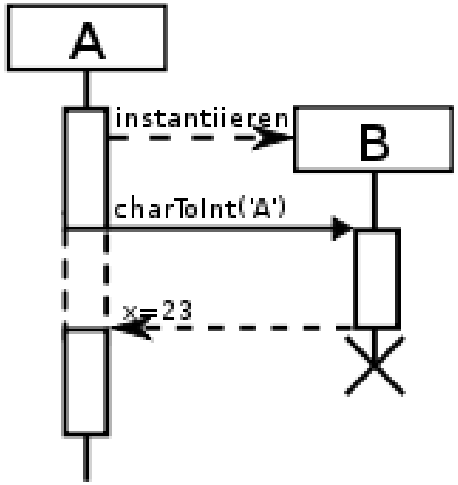
Beschreibung:

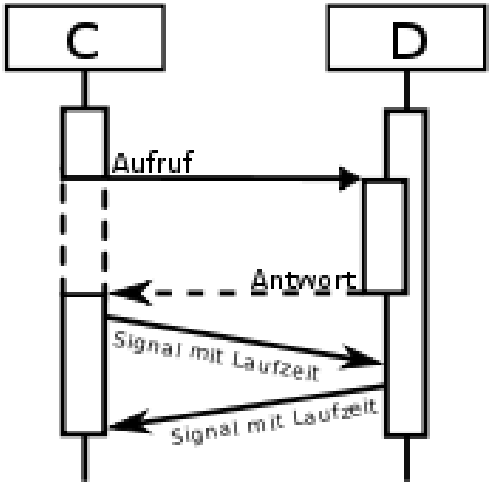
Ein synchroner Aufruf unterbricht den Aktivierungs-balken solange, bis eine synchrone Antwort eintrifft. Eine asynchrone Nachricht verändert nichts am Aktivierungsbalken.

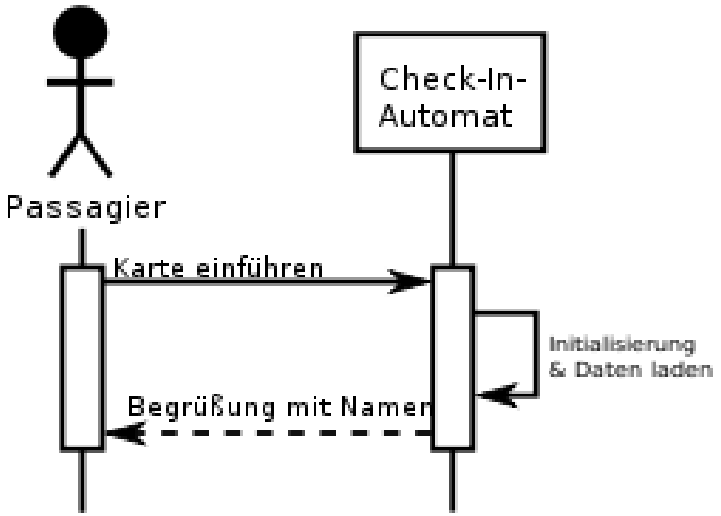
Notationen:

Nachricht ::= Aufruf | Antwort| Signal

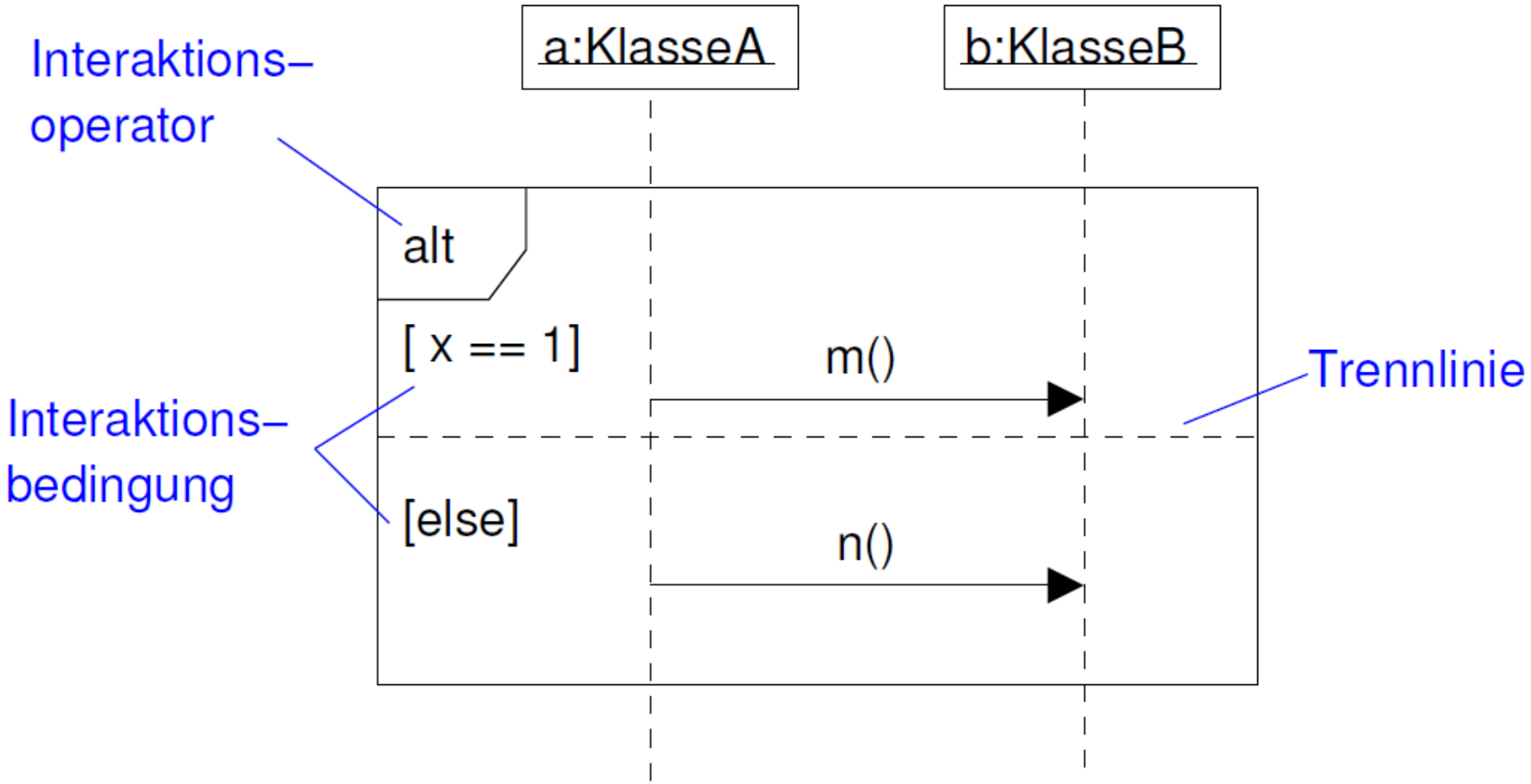




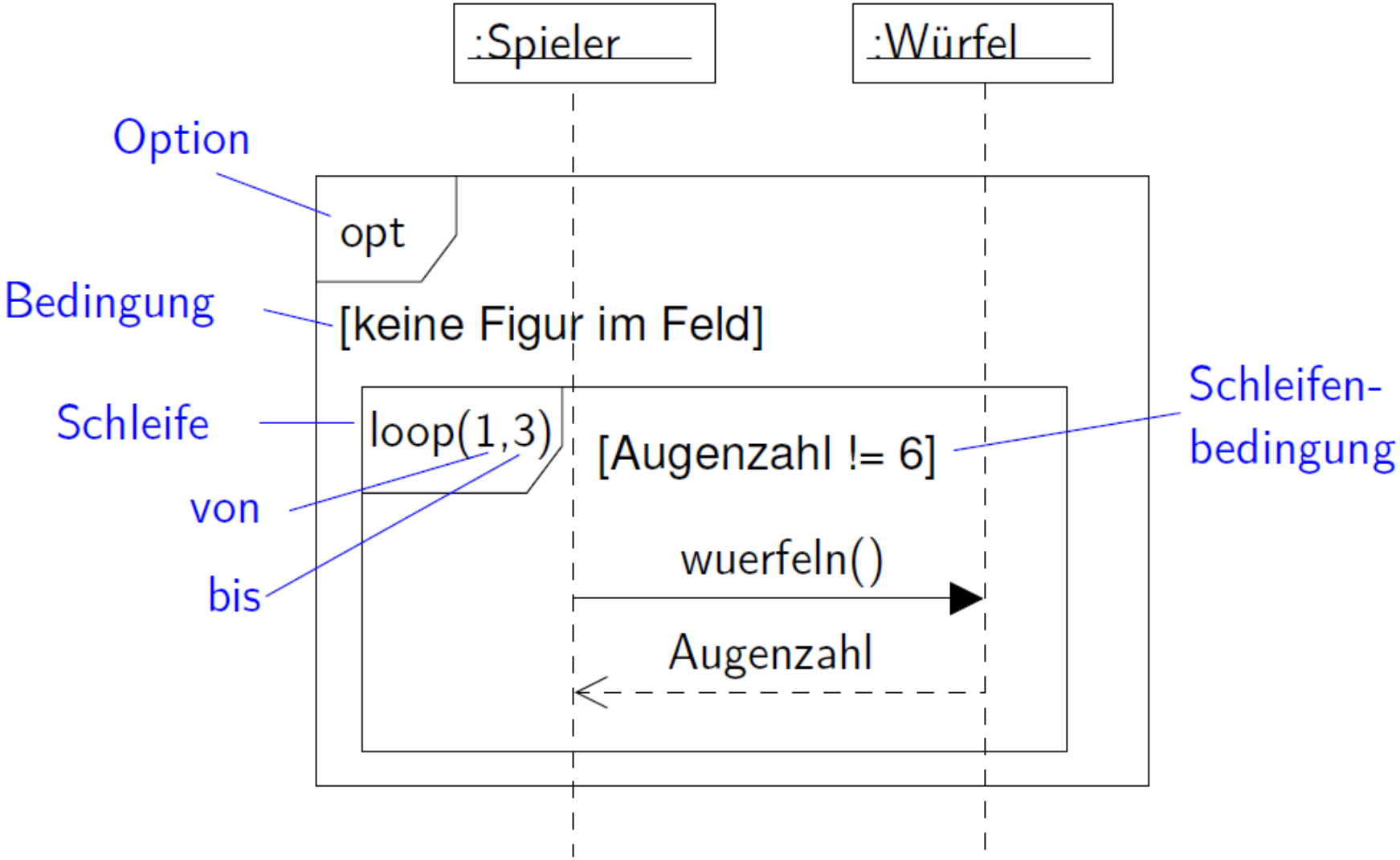




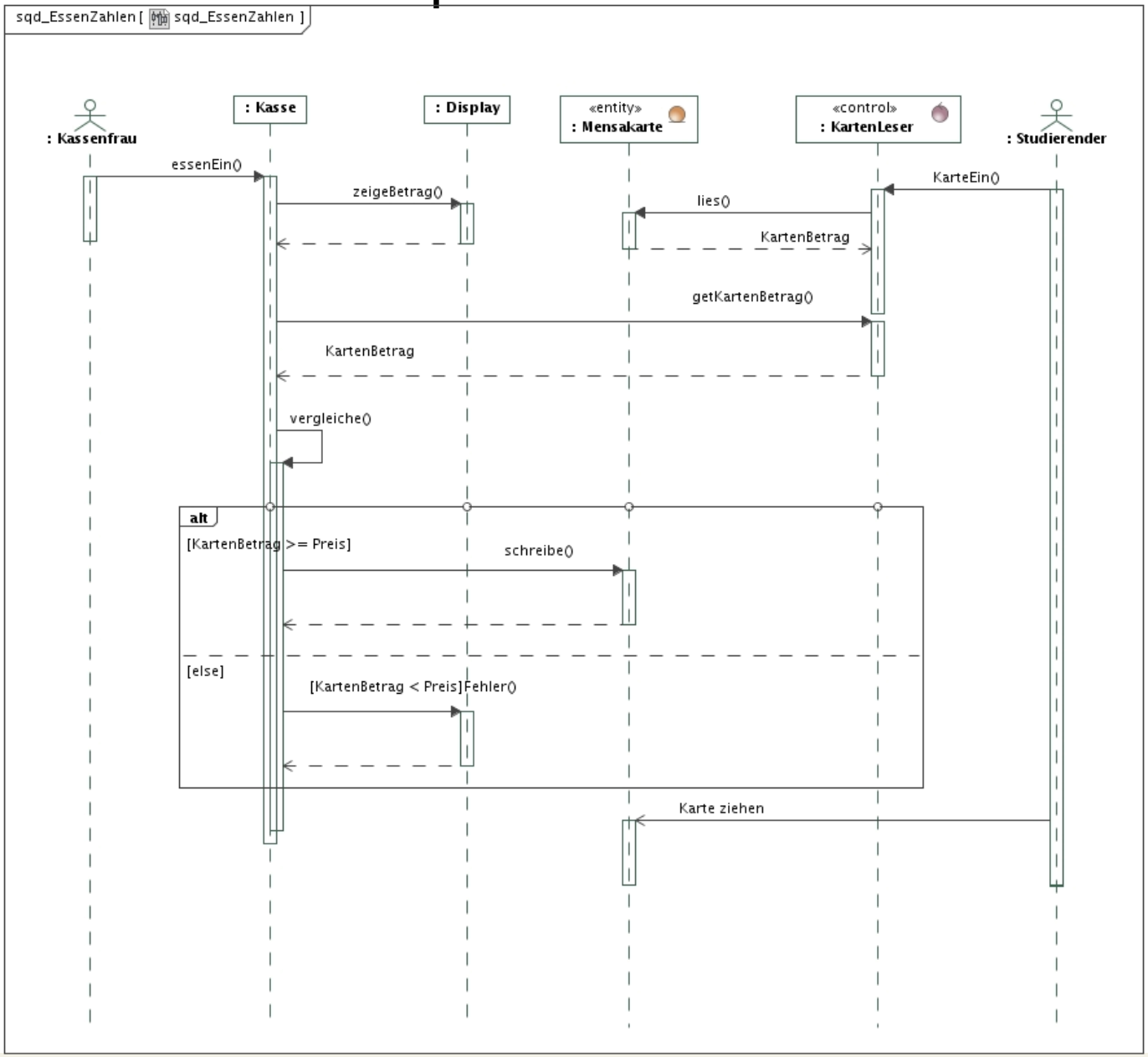
Alternativen in Sequenzdiagrammen



Schleifen und Optionen in Sequenzdiagrammen



Ein weiteres Beispiel

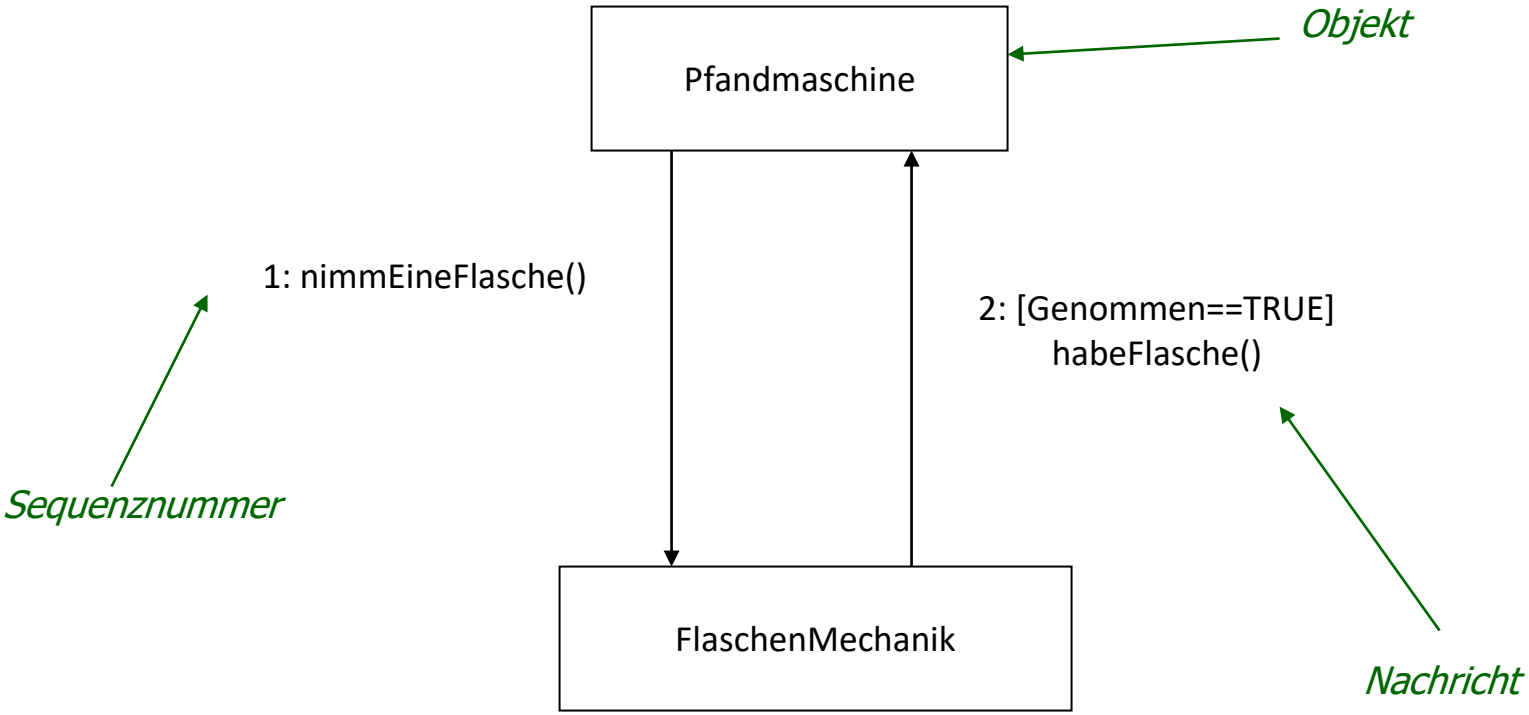


Style-Guide: Sequenzdiagramme

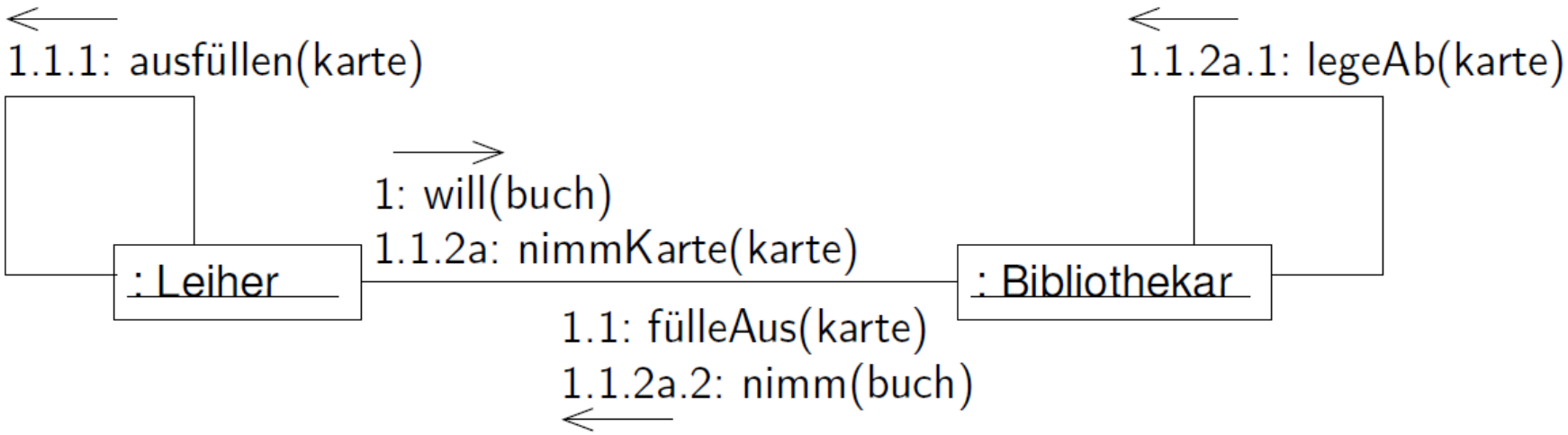
- Objekte möglichst so anordnen, dass möglichst wenige Kreuzungen entstehen
- Nachrichten sollten von links nach rechts zeigen
  - Return entsprechend von rechts nach links
- Fall Akteure (und nicht nur Objekte) beteiligt sind, sollten diese am linken Rand der Diagramme stehen
- Namen (sowohl der Akteure als auch der Objekte) sollte konsistent (zu Use-Case-Diagrammen, bzw Objekt- und Klassendiagrammen) sein

Kommunikationsdiagramme in UML

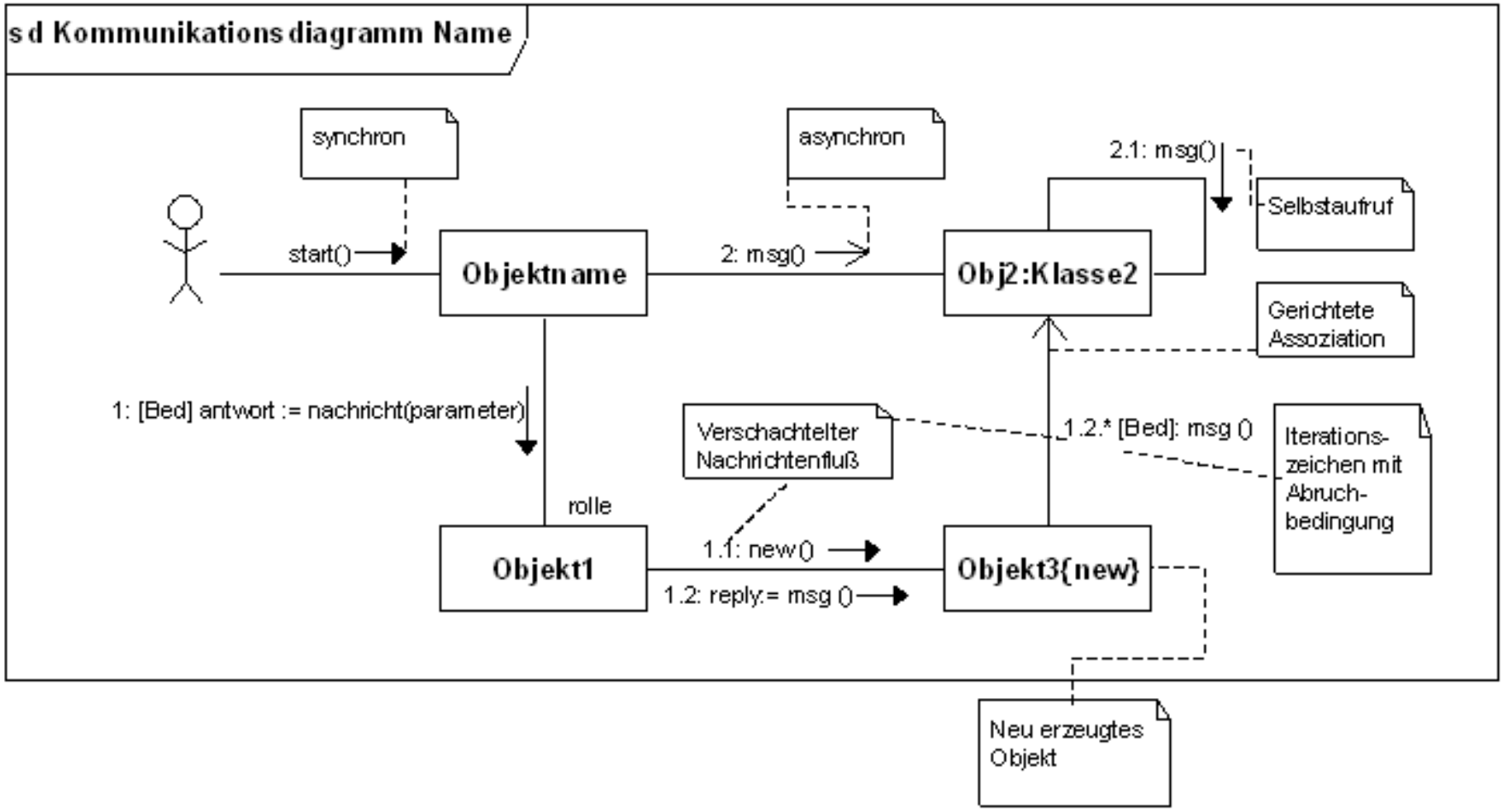
- Keine Zeitachse
- Anordnung der Nachrichten entspricht nicht der Reihenfolge
  - Stattdessen: Explizite Nummerierung durch Sequenznummern
  - => zeitlicher Ablauf nicht direkt ersichtlich



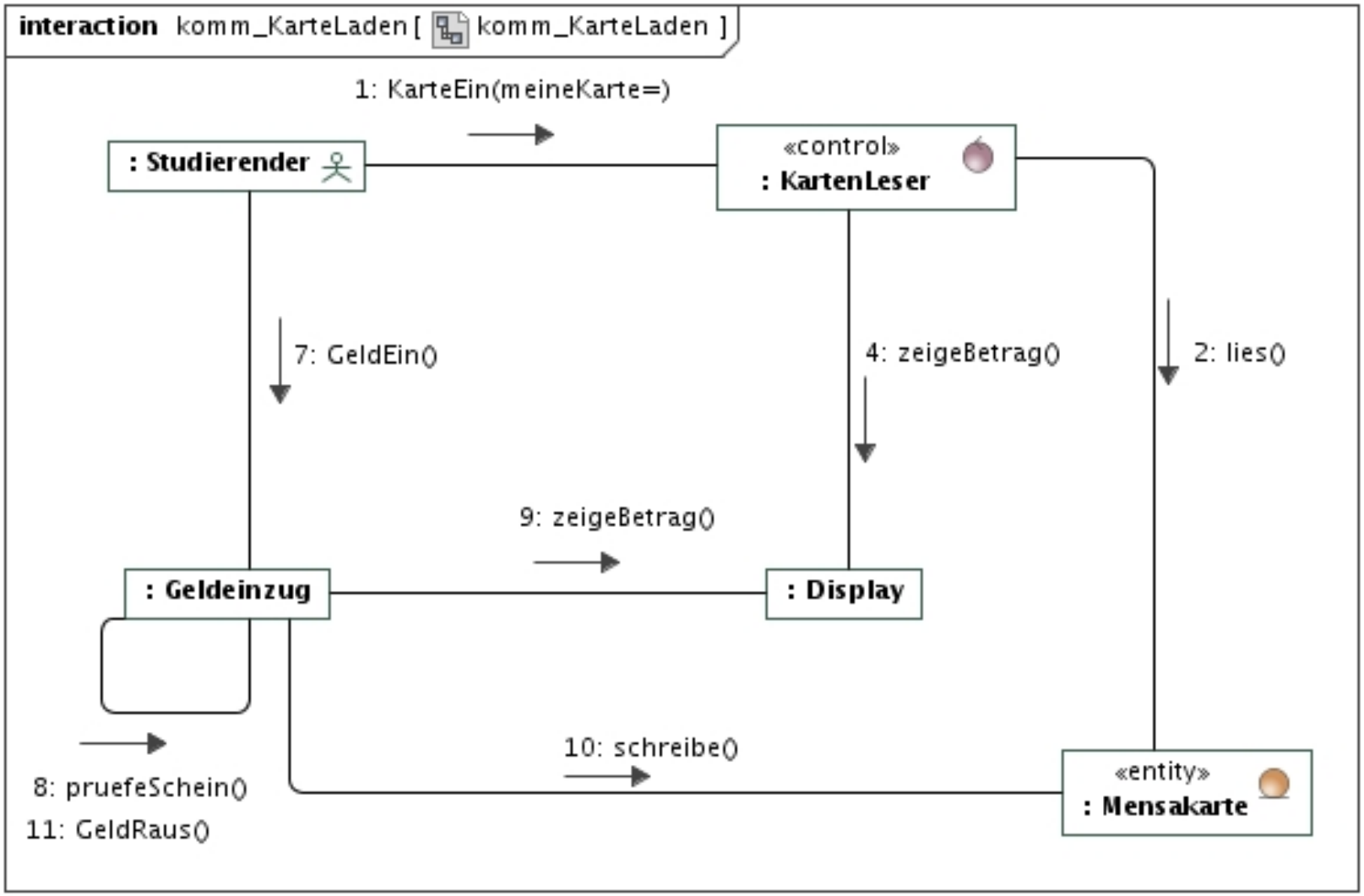
Beispiel Kommunikationsdiagramm



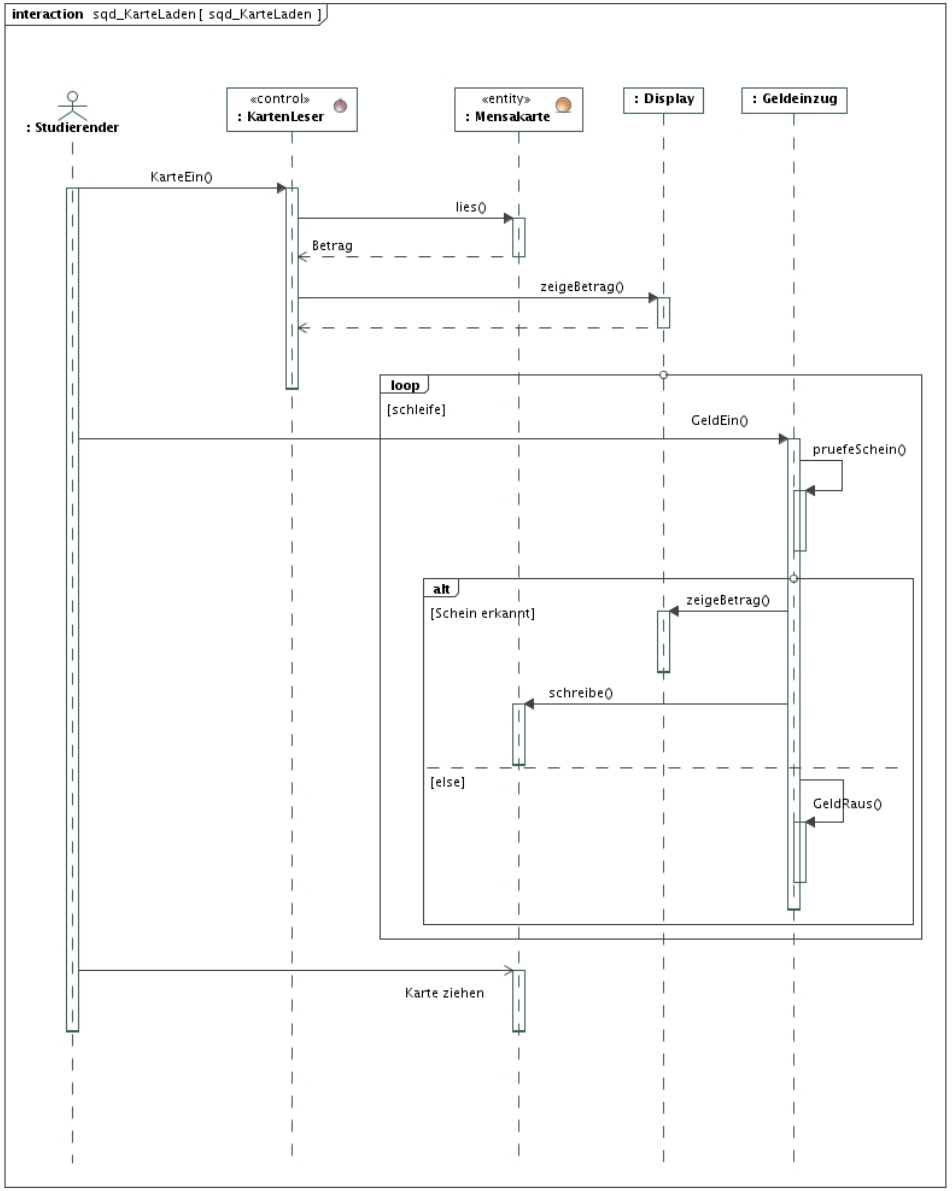
UML

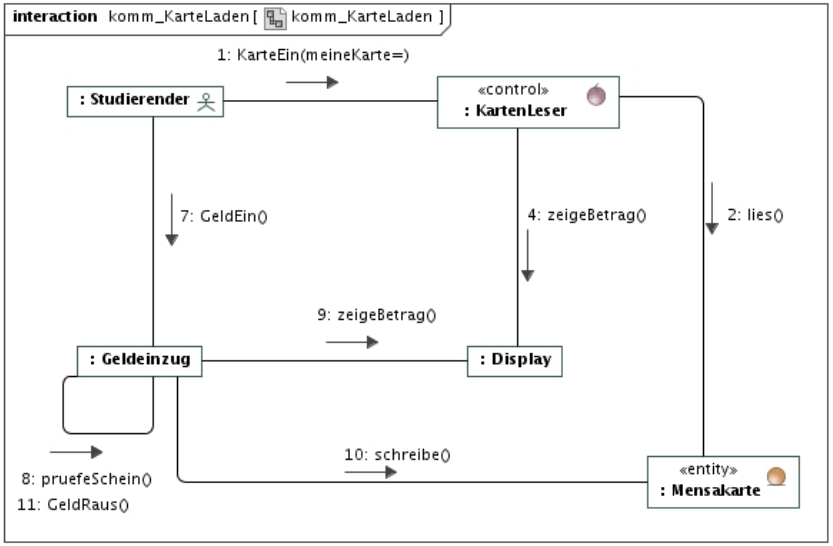

s d Kommunikations diagramm Name
synchron
asynchron
2.1: msg()
Selbstaufruf
start()
Objektname
2: msg()
Obj2:Klasse2
Gerichtete Assoziation
1: [Bed] antwort := nachricht(parameter)
Verschachtelter Nachrichtenfluß
1.2.* [Bed]: msg ()
Iterations-zeichen mit Abruch-bedingung
rolle
Objekt1
1.1: new()
Objekt3{new}
1.2: reply:= msg ()
Neu erzeugtes Objekt

Ein weiteres Beispiel



Sequenz- vs Kommunikationsdiagramm

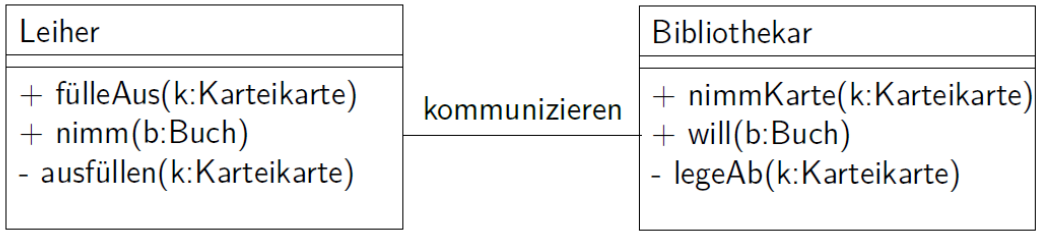




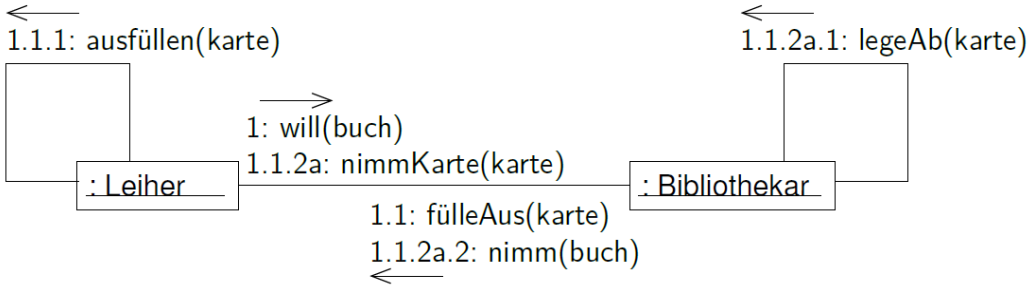
Beide Diagramme modellieren denselben Sachverhalt!

Was haben wir bislang?

- Klassendiagramm:
  - Statische Sicht auf Objekte



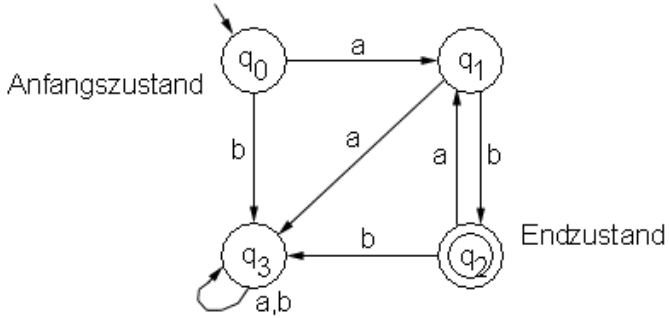
- Interaktionsdiagramme
  - Dynamische Sicht exemplarisch an Beispielszenarien
  - Interobjektinteraktion



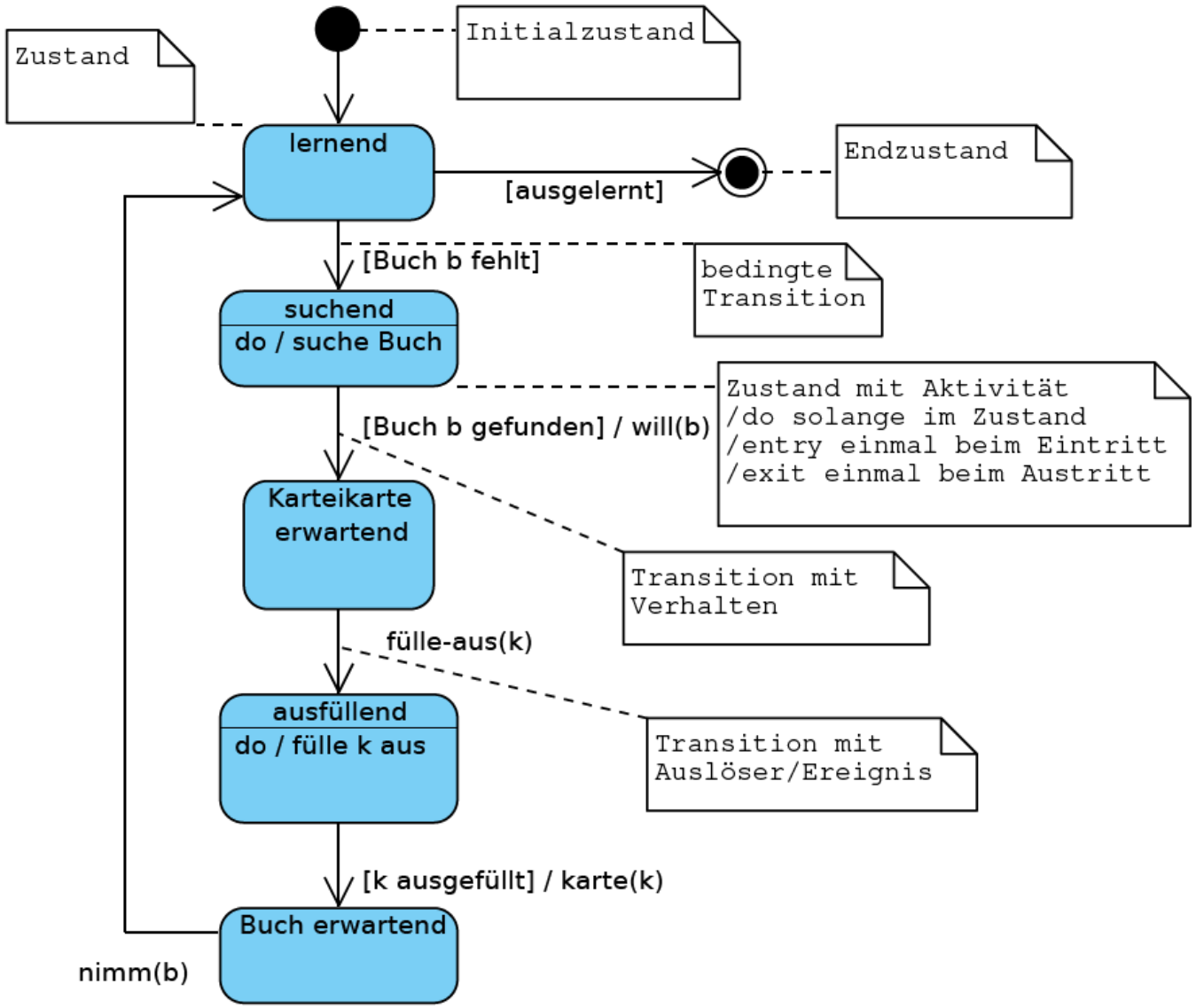
- Es fehlt: Verhalten von Objekten während Ihres Lebenszyklus
  - Intraobjektinteraktion

Zustandsdiagramme

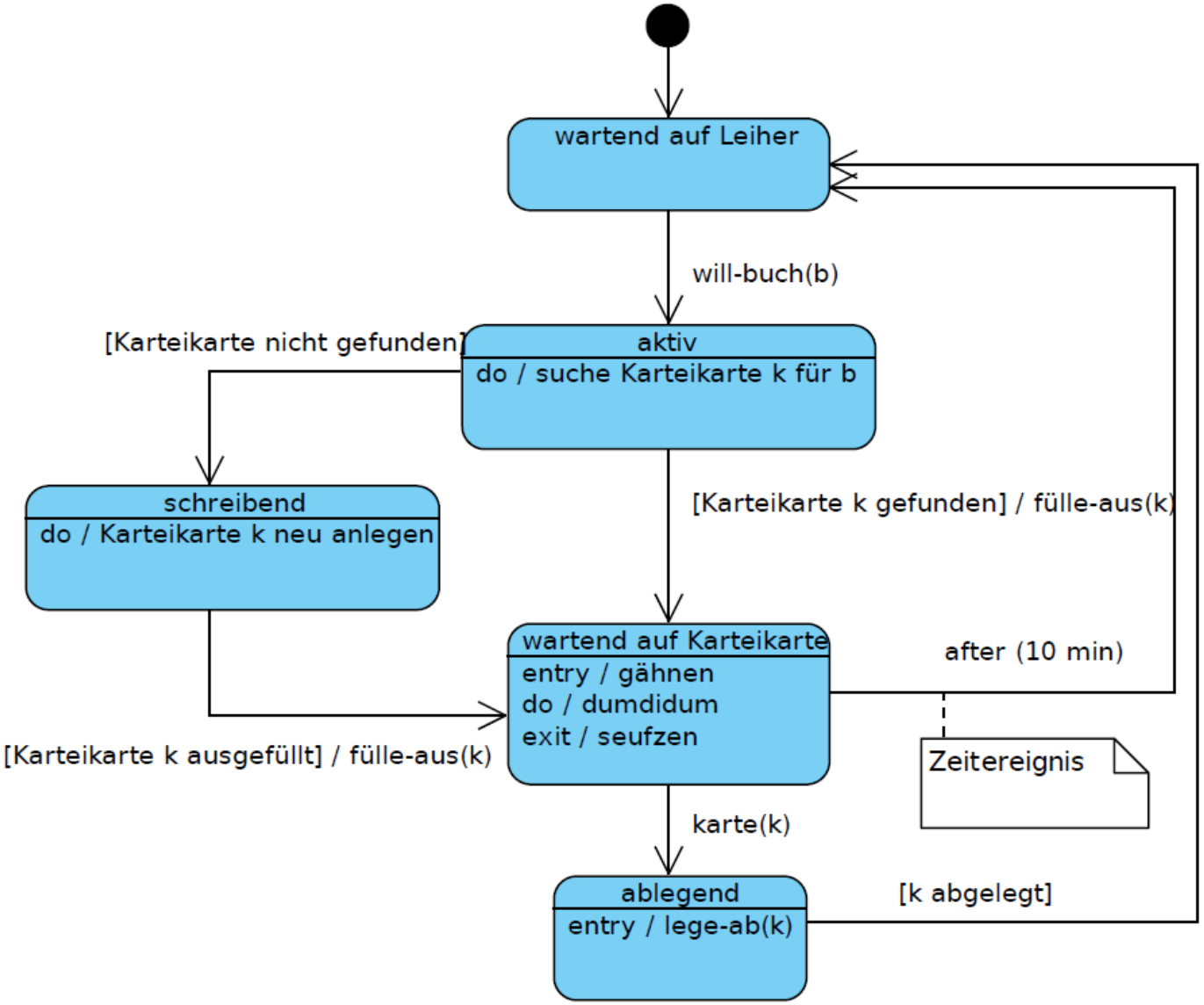
- Modellieren Objektlebenszyklus
  - Verhalten eines Objekts in Bezug auf verfügbare Methoden seiner Klasse
- Bestehen aus
  - Zustand = Menge von Attributwerten eines Objekts
  - Ereignissen = Auslöser für Zustandsänderungen
  - Transition = Übergänge zwischen Zuständen
- Basieren auf Zustandsautomaten (Harel, 1987)
  - Ähnlich zu endlichen Automaten



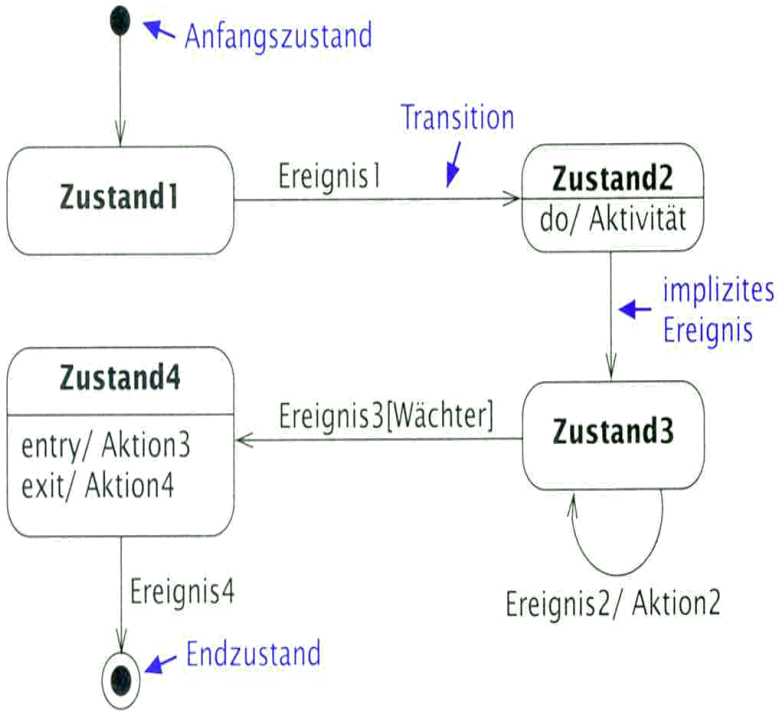
Beispiel: Zustandsdiagramm (Ausleiher)



Beispiel: Zustandsdiagramm (Bibliothekar)



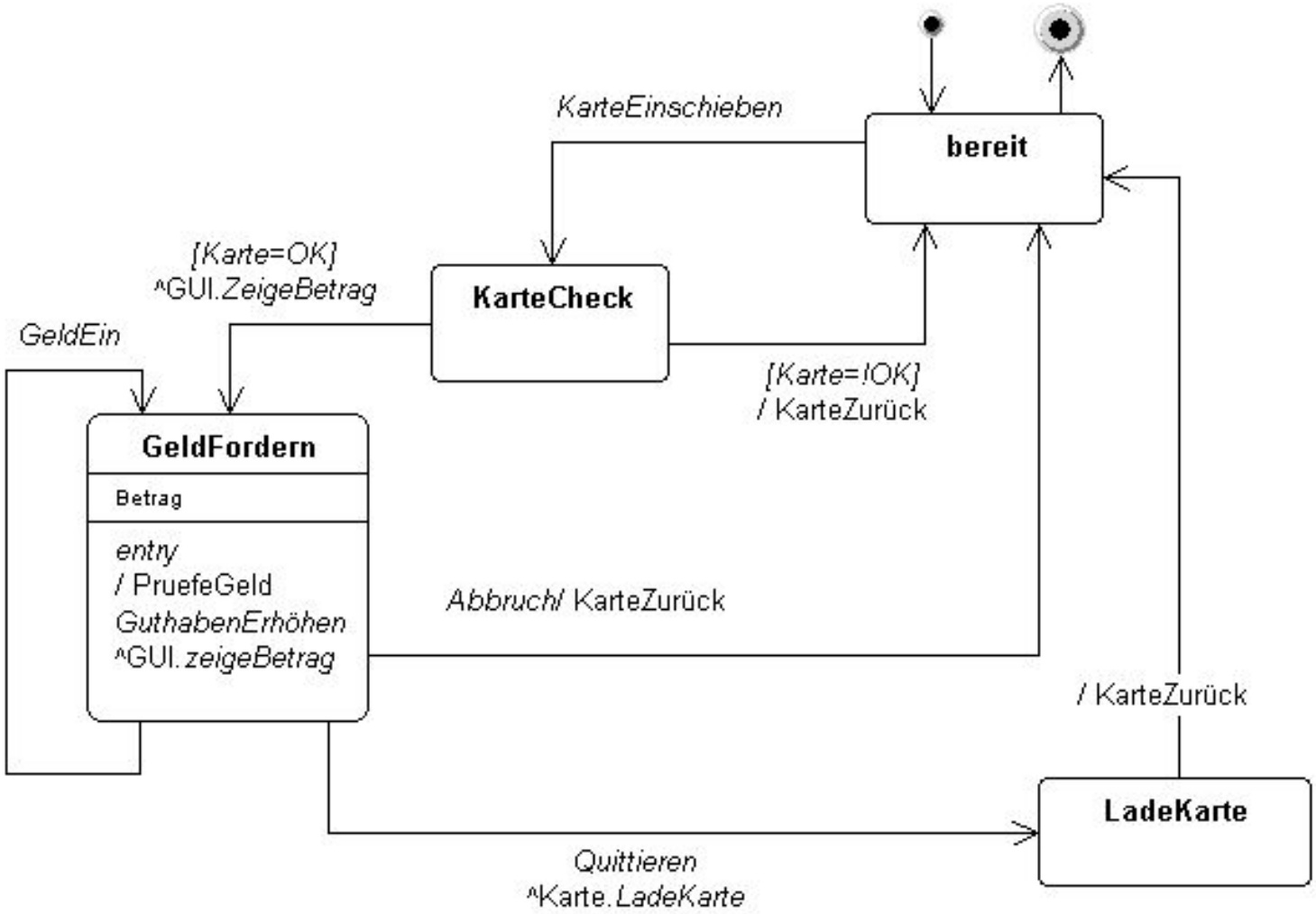
Notation von Zustandsdiagrammen in UML



Zustandsname

do / Aktivität wenn System in Zustand
entry / wenn System in Zustand eintritt
exit / sobald Zustand verlassen wird

Ein letztes Beispiel



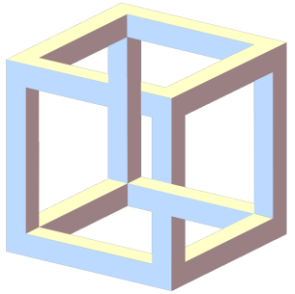
Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein => Klassendiagramm
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen

  => Aktivitätsdiagramm
  - Identifiziere Verhalten der Objekte => Zustandsdiagramm, Interaktionsdiagramm (Sequenz-/Kommunikationsdiagramm)
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Randbedingungen an Methoden (oder das System)

- Bislang:
  - Diagramme um Struktur und dynamisches Verhalten zu definieren
  - Es fehlt: Ausschluss von unerwünschten Systemrealisierungen
  - Idee: Definiere Randbedingungen



- Randbedingungen für Methoden
  - Parameter: Eingabe und Ausgabe
  - Vorbedingung: Annahmen, die gelten müssen, damit die Methode ausgeführt wird
  - Nachbedingungen: Resultat der Methode
  - Fehlerbedingungen: Verletzung der Vorbedingungen und Fehler, die während der Ausführung auftreten können
  - Verhalten in Fehlersituationen: Nachbedingungen für jeden Fehler
  - Reaktionszeit: Maximale Dauer, bis Resultat vorliegt (sowohl im Normal- als auch im Fehlerfall)

Beispiel: Sortierung einer Buchliste

- Parameter:
  - Eingabe: Buchliste, Sortierkriterium (Attribut)
  - Ausgabe: Buchliste‘
- Vorbedingung:
  - Attribut kommt in allen Büchern der Buchliste vor
- Nachbedingungen:
  - Buchliste ist sortiert, d.h. $\forall\ 1 \leq i < len(Buchliste')$:

    $element(Buchliste', i) \leq_{Attribut} element(Buchliste', i+1)$
  - Buchliste‘ ist eine Permutation der Buchliste
- Fehlerbedingungen: keine, außer Vorbedingung nicht erfüllt
- Verhalten in Fehlersituationen:
  - Fehlermeldung
- Reaktionszeit: n log(n) * 0.001 sec, wobei n = Länge der Liste

Randbedingungen in UML?

- Nicht direkt
- Gibt aber die Erweiterung Object Constraint Language (OCL) die zumindest Teile davon unterstützt
  - Angelehnt an Programmiersprache Smalltalk
  - Unterstützt Nebenbedingungen für Klassen, Attribute, Methoden, Komponenten,...
  - Mögliche Nebenbedingungen: Invarianten, Vor- und Nachbedingungen, Definitionen...
  - Beispiel für Invariante:
    - Die Seitenanzahl eines Buchs ist nicht negativ

```
context Buch inv: self.seitenanzahl >= 0
```

- Mehr Details würden den Rahmen der Vorlesung aber sprengen

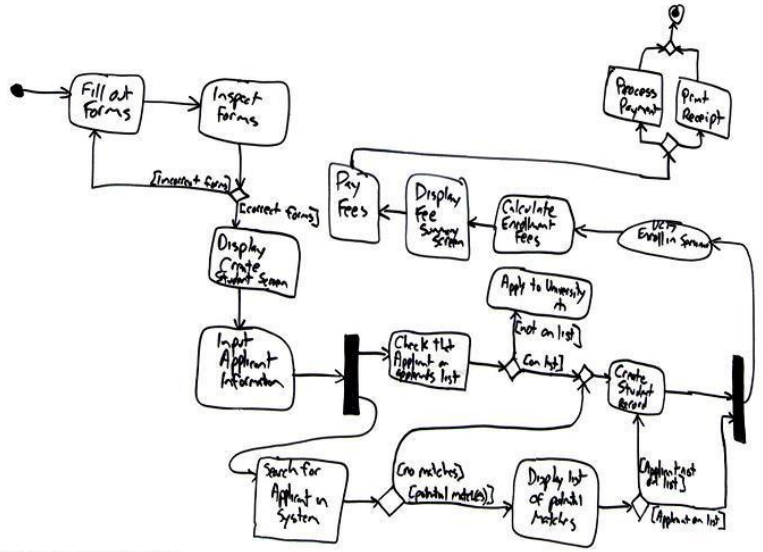
Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein => Klassendiagramm
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen

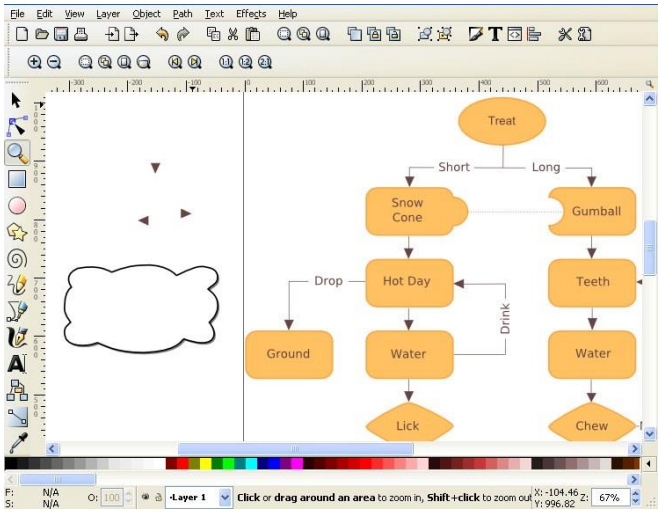
  => Aktivitätsdiagramm, Interaktionsdiagramm (Sequenz-/Kommunikationsdiagramm)
  - Identifiziere Verhalten der Objekte => Zustandsdiagramm
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Wie erstellt man UML-Diagramme?

- Stift und Papier
  - Einfach
  - Schnell
  - Änderungen schwierig

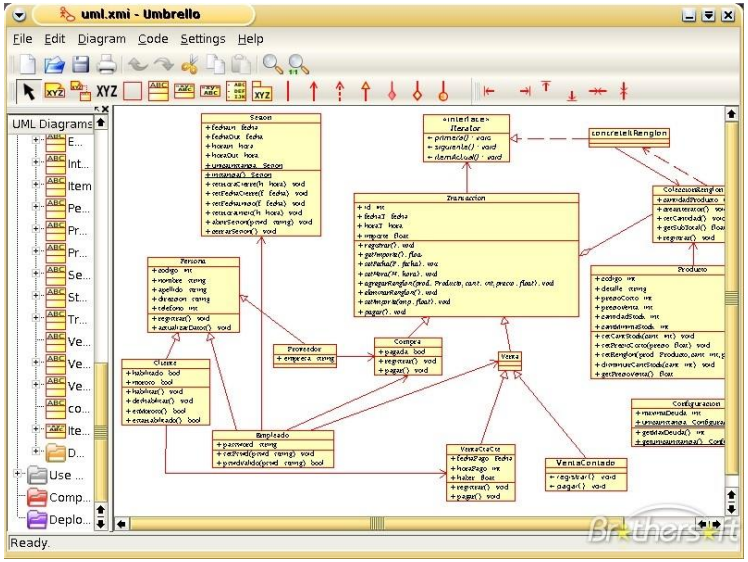


- Zeichenprogramme
  - Inkscape, Powerpoint,...
  - Aufwändiger
  - Hübscher
  - Teils gibt es Vorlagen für UML-Symbole

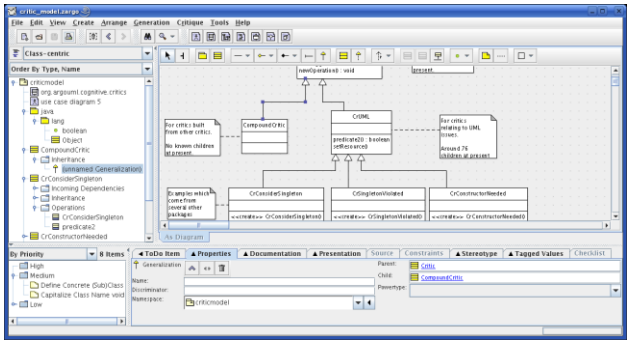


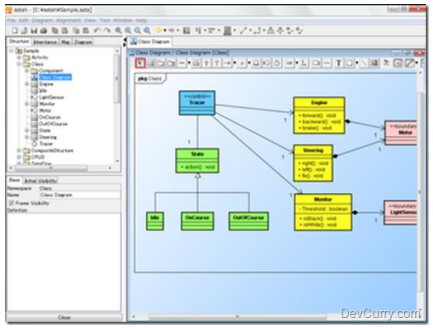
Noch bequemer: Spezielle UML-Editoren

- Es gibt viele
  - Unterschiedlicher Funktionsumfang
    - Unterstützte Sprachen
    - UML-Versionen
    - Diagrammarten
  - Unterschiedliche Lizensierungen
  - Verschiedene Plattformen



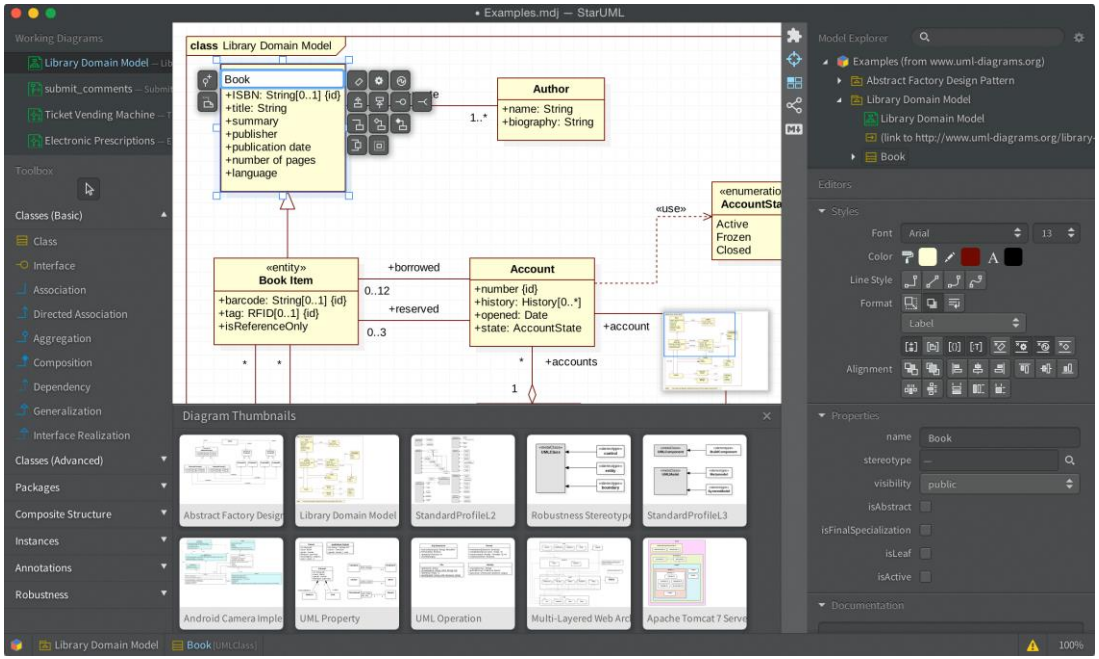
- Teilübersicht:
  - https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_to

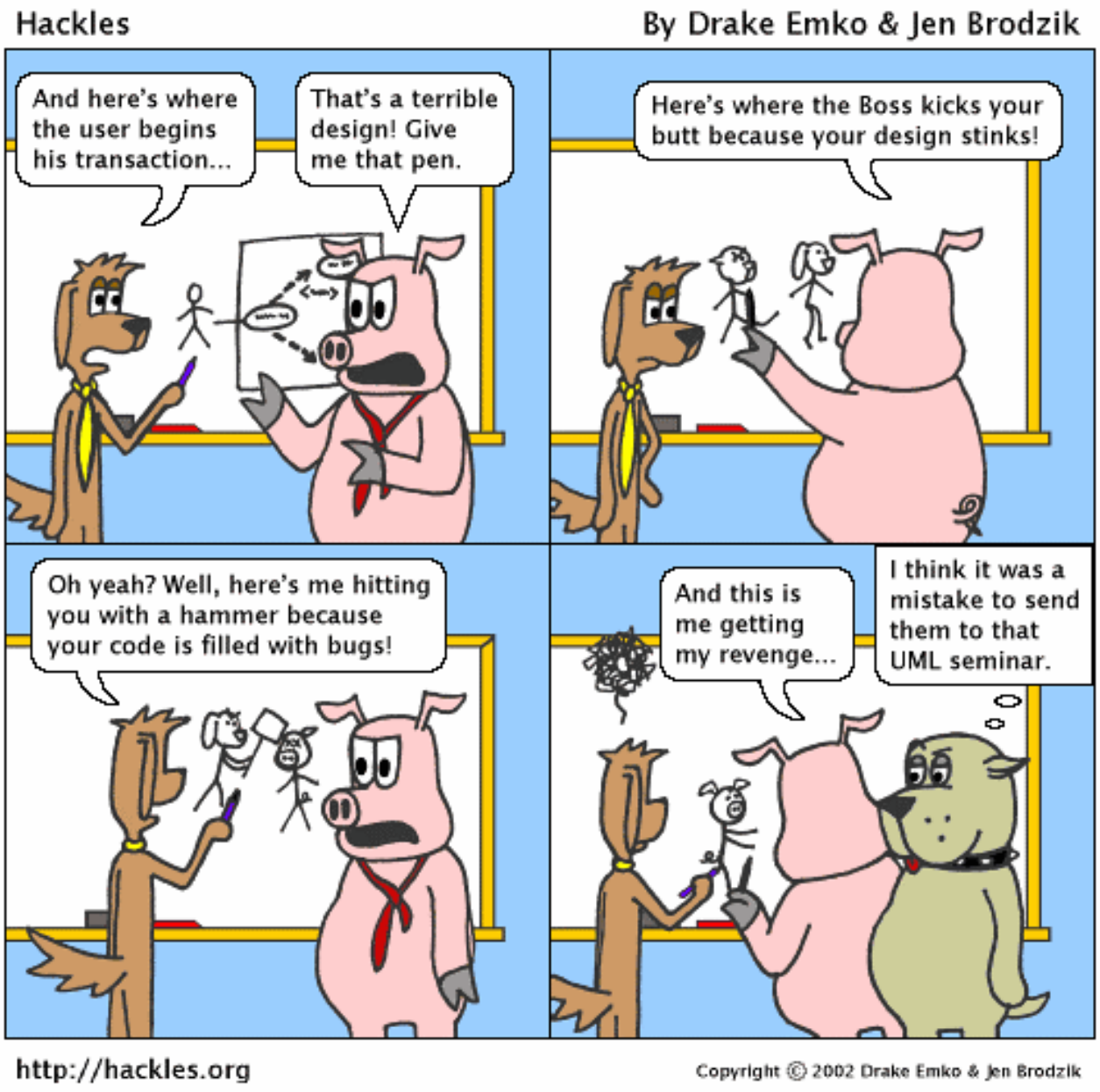




Meine Empfehlung: StarUML

- Kommerziell, aber voll funktionsfähige Evaluationsversion
  - Gibt auch eine Educational-Lizenz
- Für Windows/Linux/Mac
- Unterstützt alle von uns verwendeten Diagramme
- Automatische Code-Generierung für C++ (mit Addon)




Hackles
By Drake Emko & Jen Brodzik
And here's where the user begins his transaction...
That's a terrible design! Give me that pen.
Here's where the Boss kicks your butt because your design stinks!
Oh yeah? Well, here's me hitting you with a hammer because your code is filled with bugs!
And this is me getting my revenge...
I think it was a mistake to send them to that UML seminar.
http://hackles.org
Copyright © 2002 Drake Emko & Jen Brodzik