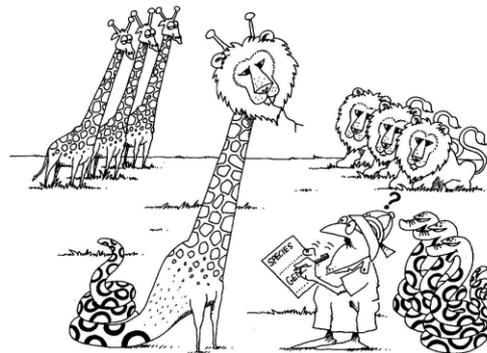




Media Engineering

Objektorientierte Modellierung

Verhaltensmodellierung



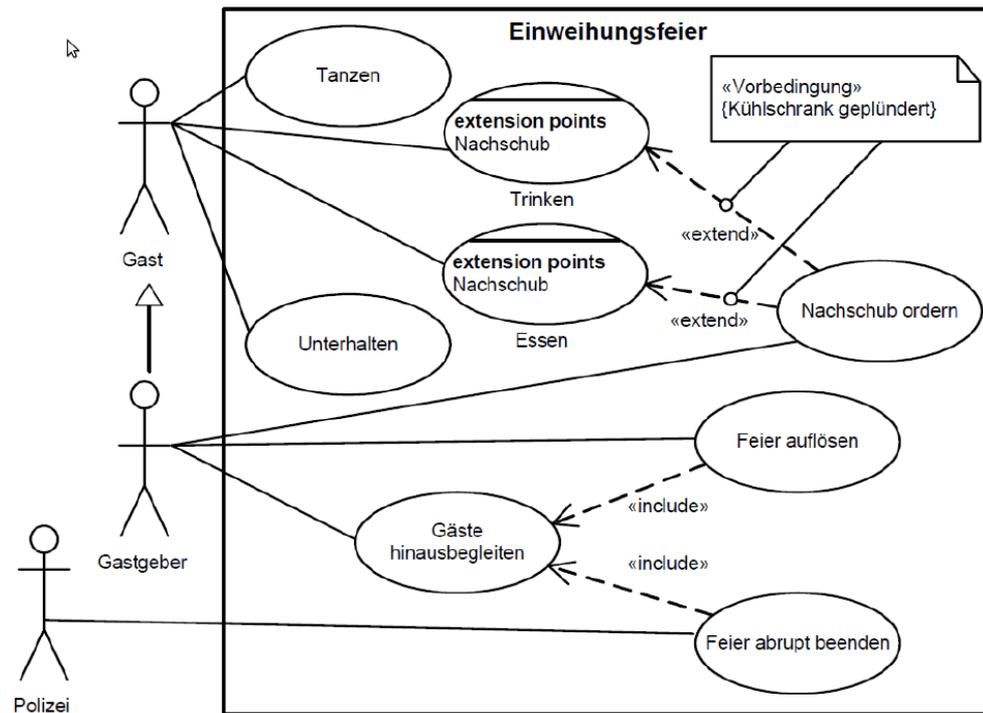
R. Weller

University of Bremen, Germany

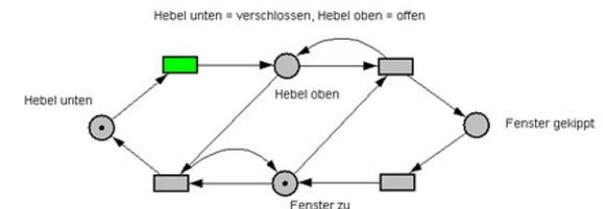
cgvr.cs.uni-bremen.de

- 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)

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

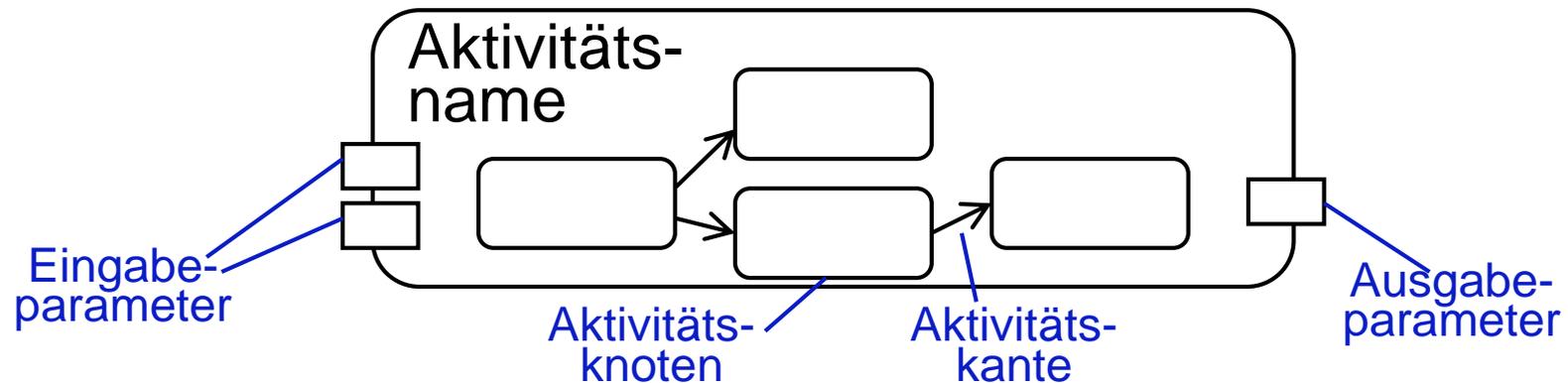


- Spezifizieren **Kontroll- und Datenfluss** zwischen verschiedenen Arbeitsschritten (genannt Aktionen), die zur Realisierung einer Aktivität notwendig sind
 - Beschreiben **Reihenfolge**
 - und **Abhängigkeiten**
- Modellierung klassenübergreifenden Verhaltens (Kontrollfluss)
- Zur detaillierten Beschreibung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

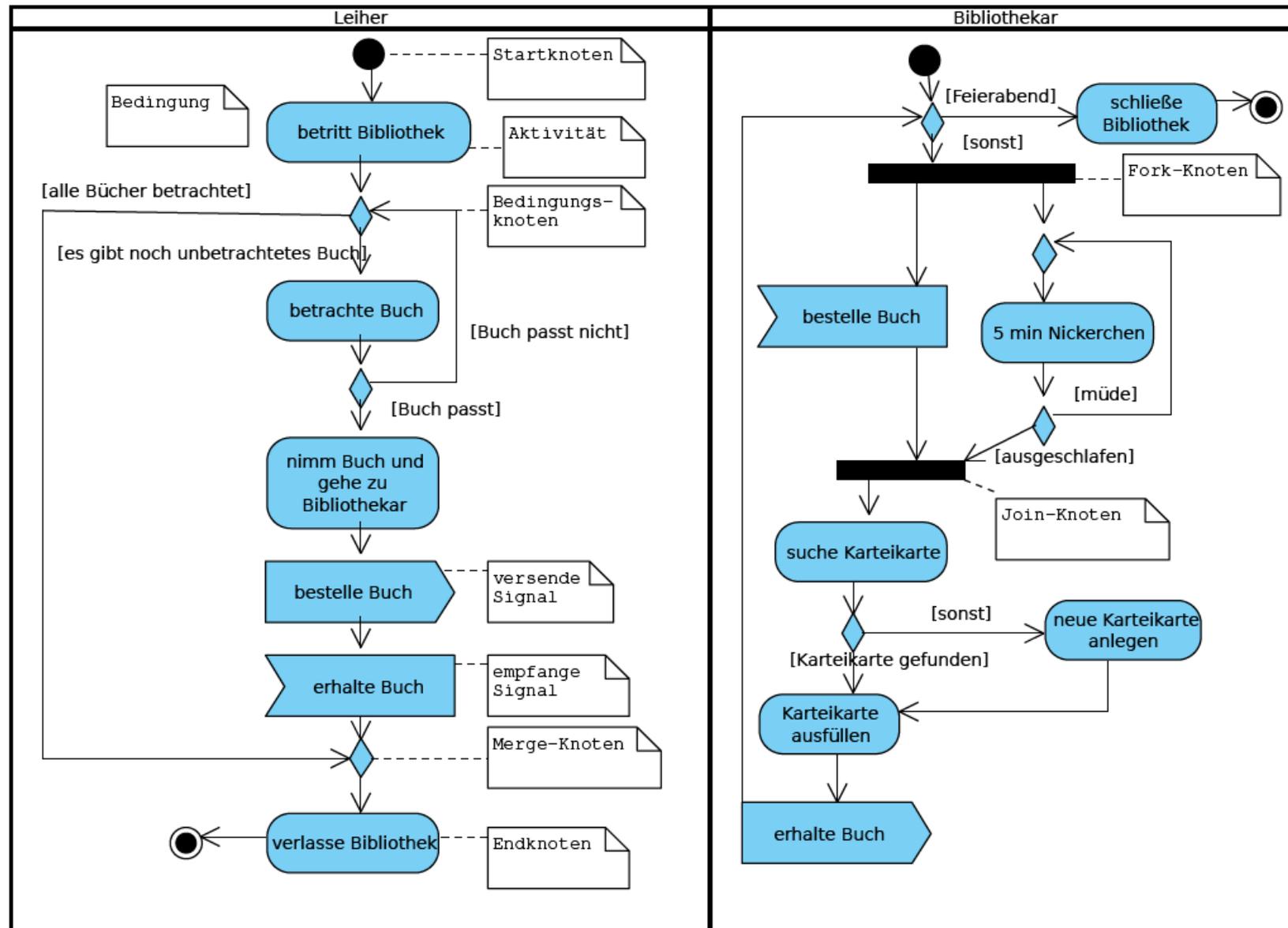


Aktivitäten und Aktionen

- 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



Beispiel Aktivitätsdiagramm



Notation von Aktivitätsdiagrammen in UML

	Aktivität		Synchronisation der Kontrolle (AND) mit Synchronisationsbedingung (Die Bedingung ist optional.)
	Kontrollfluss	[Bedingung]	
	Aktivität2 wird nach Abschluss von Aktivität1 gestartet.		Aufsplitten der Kontrolle (Zulassen von Parallelität)
	Verzweigungsaktivität (kann auch durch normale Aktivität dargestellt werden)		Aktivität wird durchgeführt, wenn über einen der eingehenden Kontrollflüsse die Kontrolle ankommt (OR)
	Kontrollfluss, der unter der angegebenen Bedingung gewählt wird.		
	Start des Ablaufs und Identifikation der betroffenen Klasse		
	Aktion zum Senden eines Signals		
	Aktion zum Empfangen eines Ereignisses		
			Ende des Ablaufs (optional)

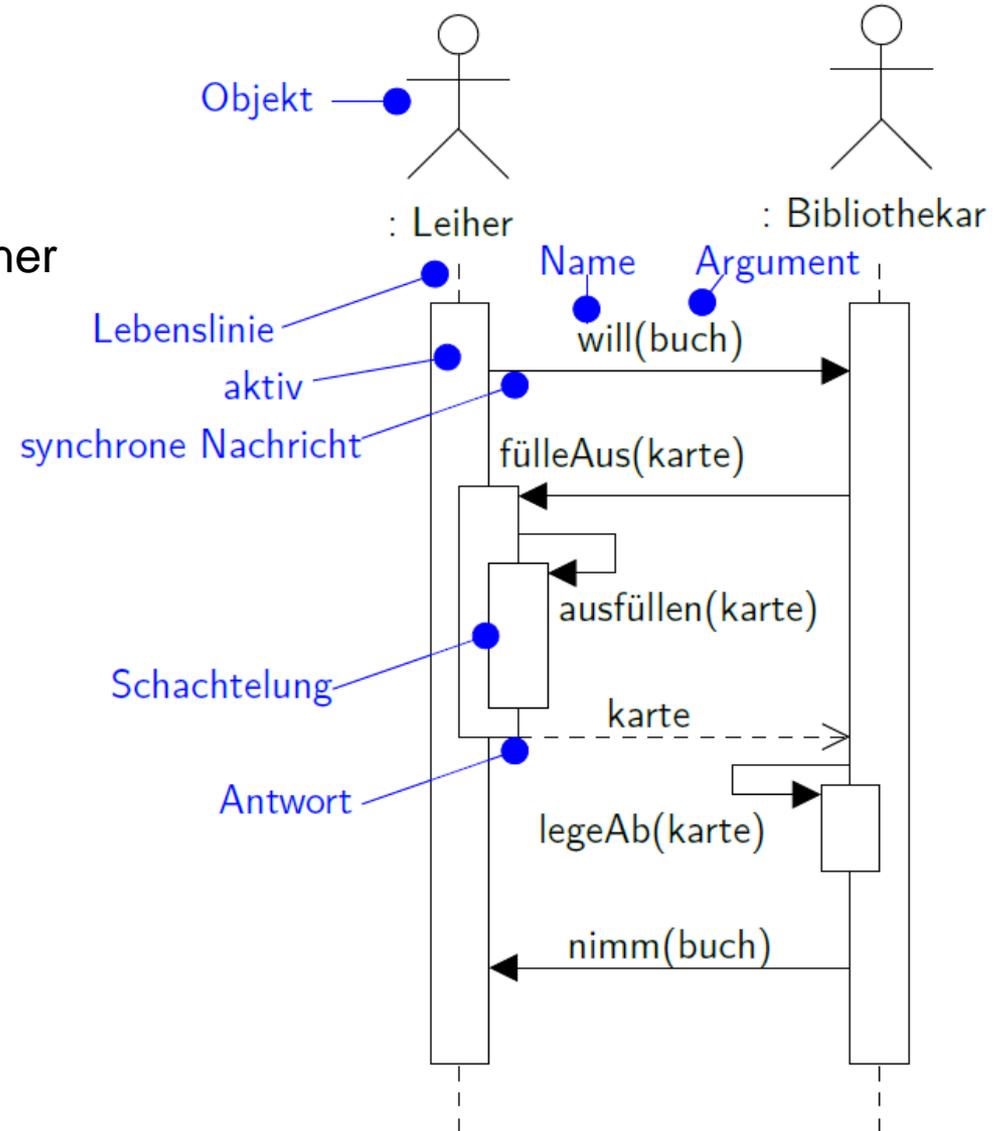
Objektorientierte Analyse und Design im Detail

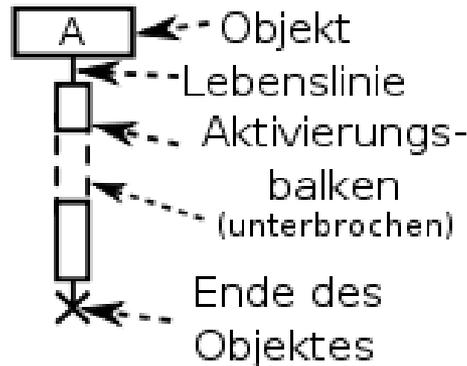
- Identifiziere Akteure
 - Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
 - Bestimme statisches Modell
 - Identifiziere Objekte
 - Identifiziere Eigenschaften der Objekte
 - Bestimme Assoziationen der Objekte => Objektdiagramm
 - Fasse Objekte zu Klassen zusammen
 - Bestimme Funktionen 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)

- Was noch fehlt: Interaktionen zwischen Objekte
- **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





Beschreibung:

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

Notationen:

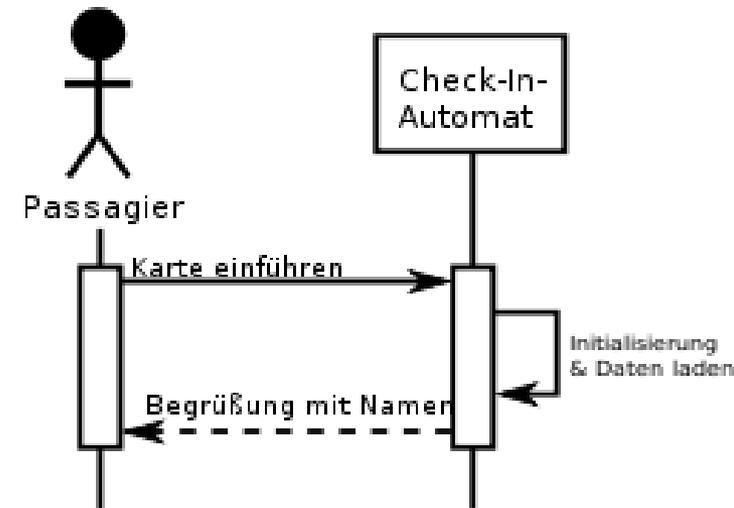
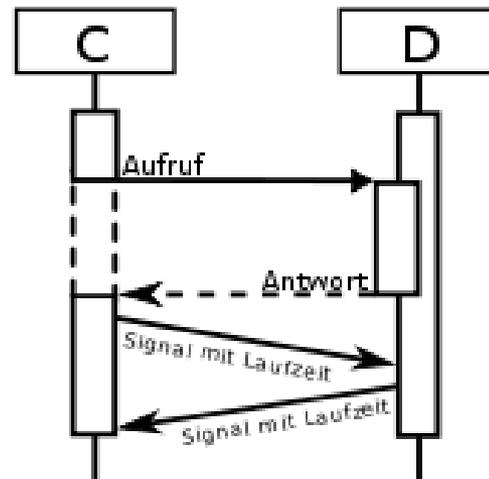
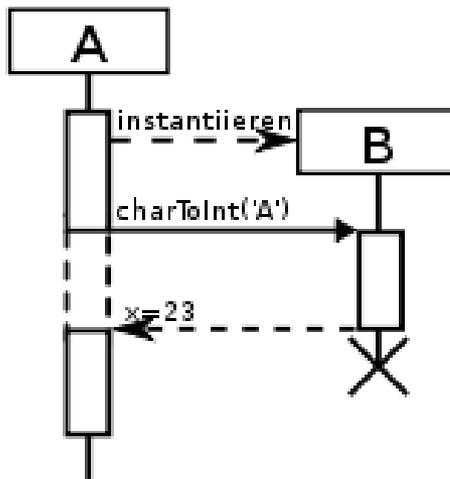
Nachricht ::= Aufruf | Antwort | Signal

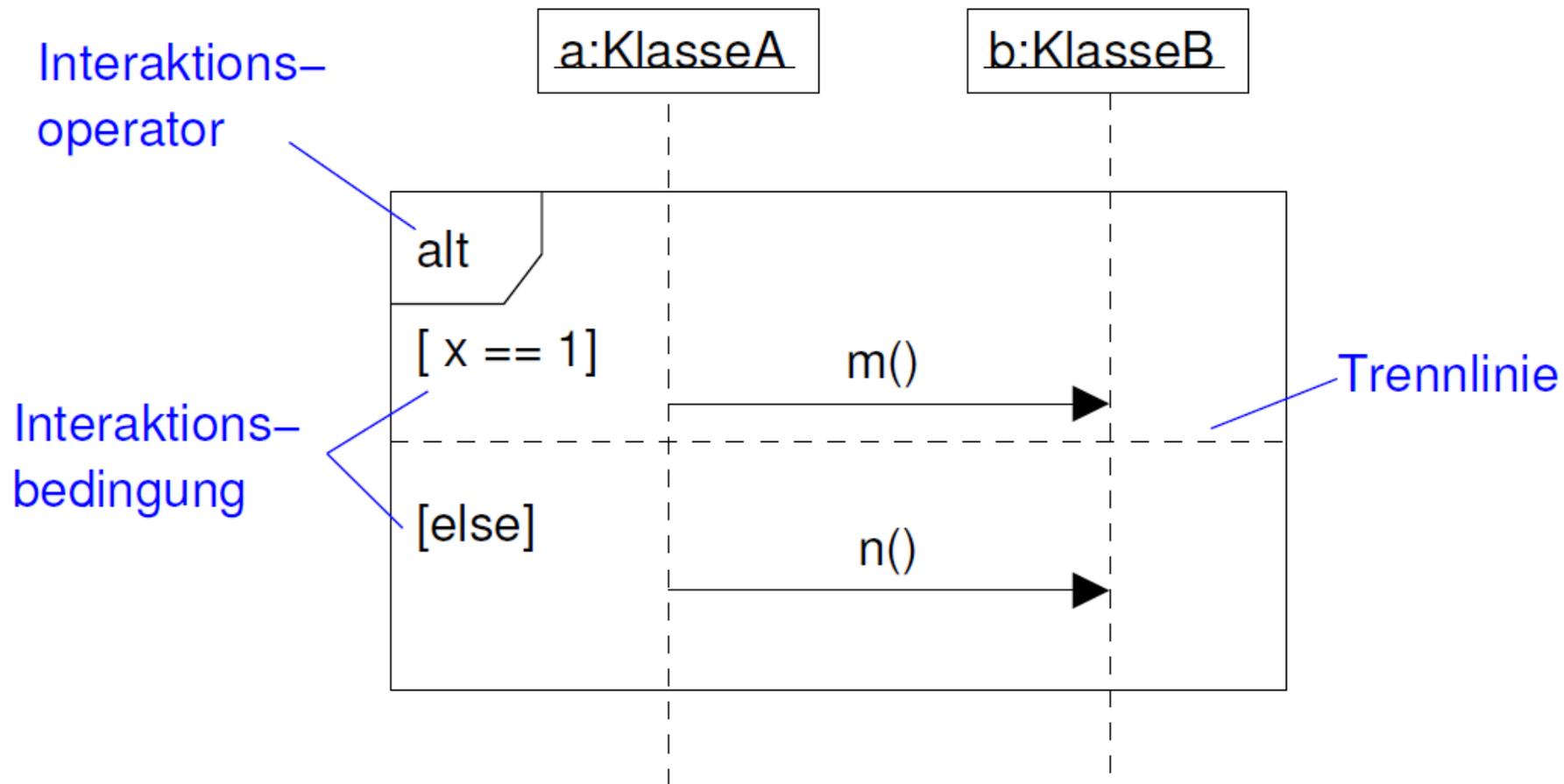
→ Aufruf (synchrone Nachricht)

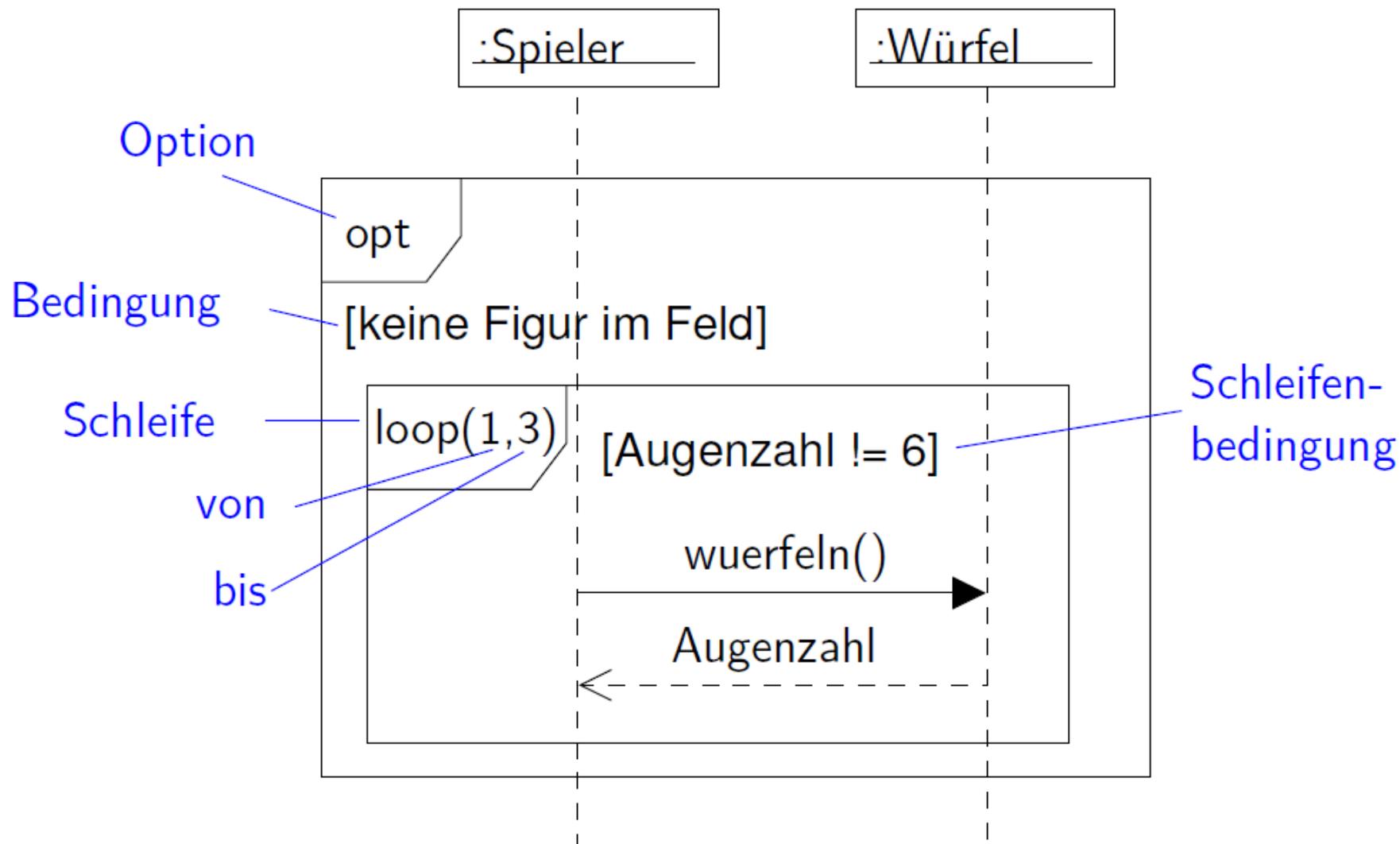
← - - - Antwort (synchrone Nachricht)

→ Signal (asynchrone Nachricht)

← (schräg bei relevanter Laufzeit)

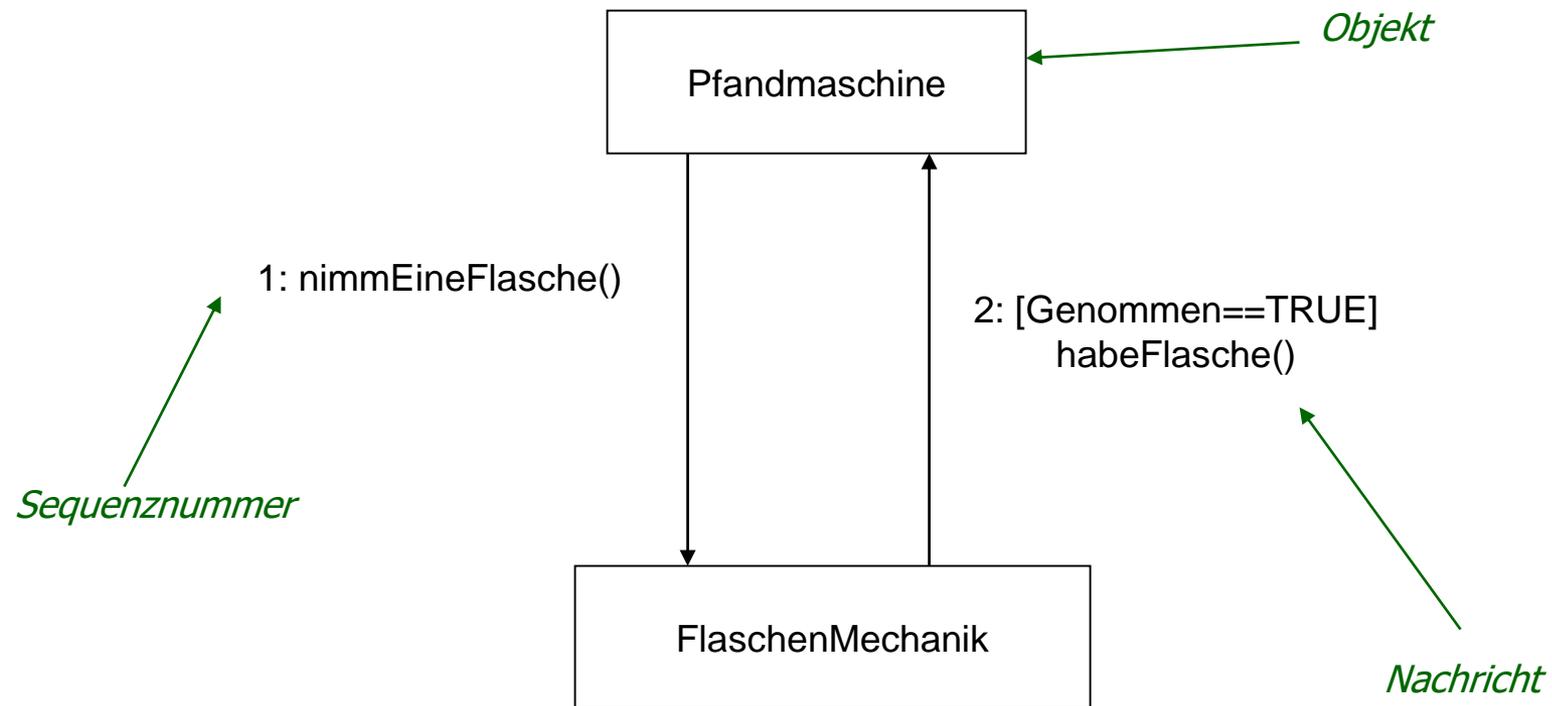


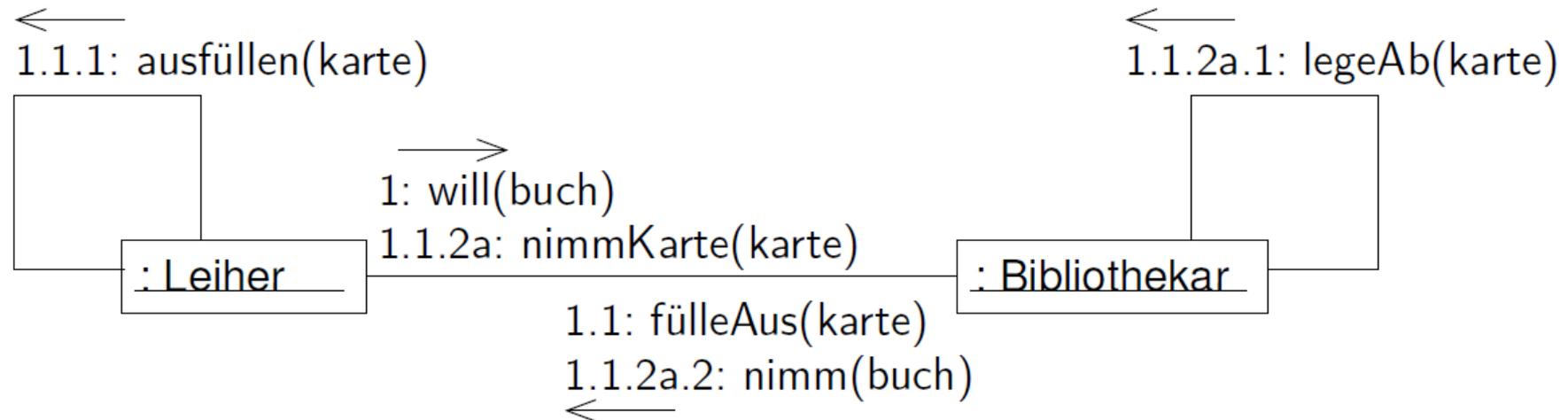


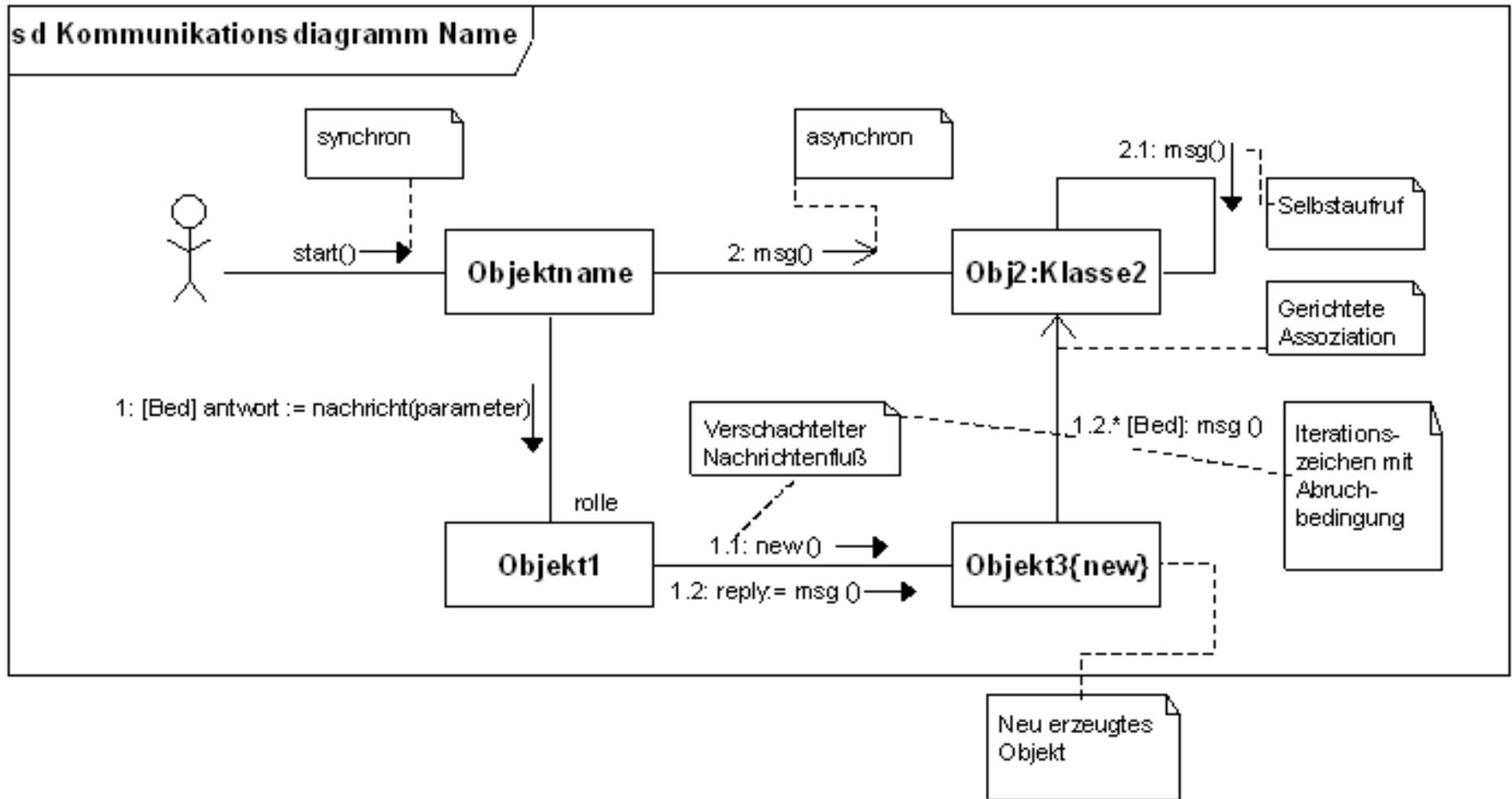


Kommunikationsdiagramme in UML

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

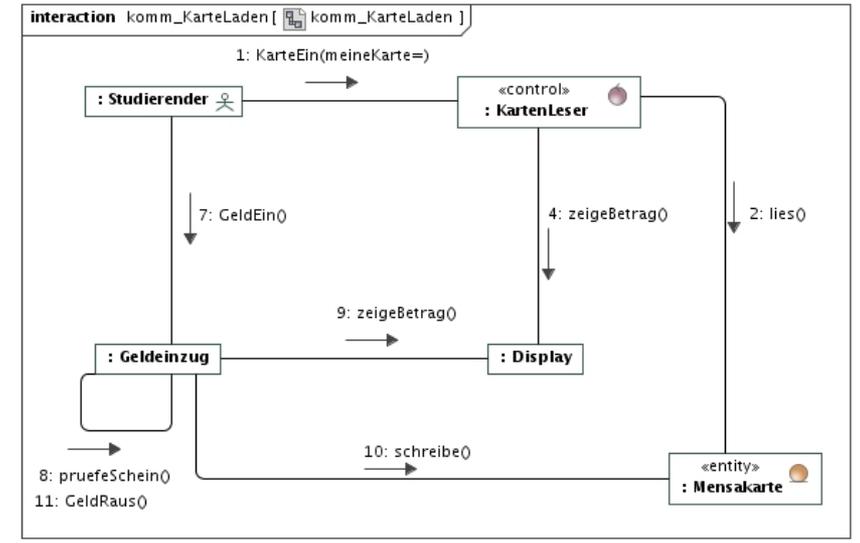
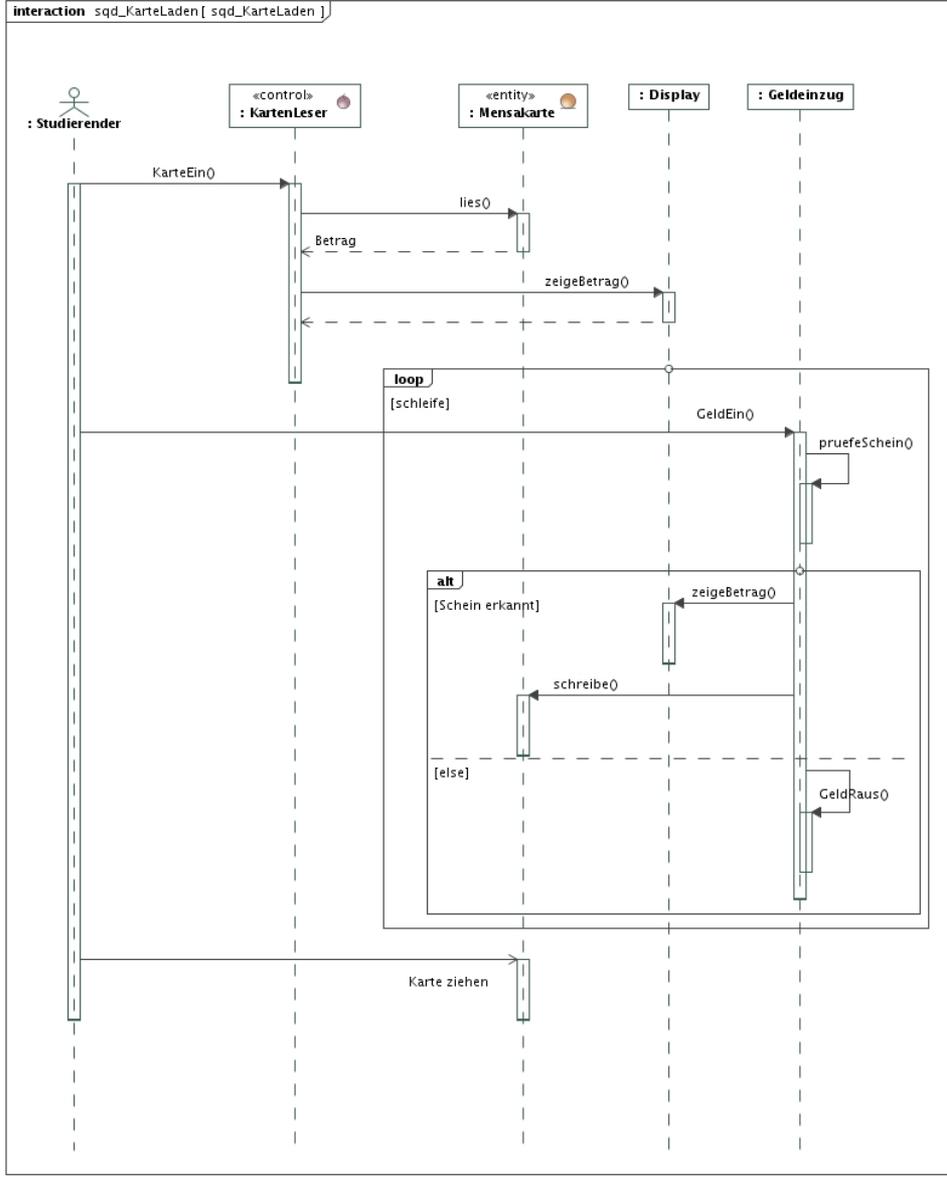








Sequenz- vs Kommunikationsdiagramm



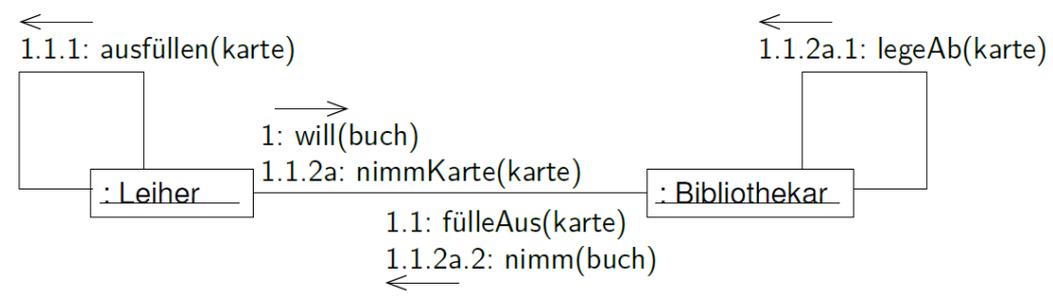
Beide Diagramme modellieren denselben Sachverhalt!

Was haben wir bislang?

- Klassendiagramm:
 - Statische Sicht auf Objekte

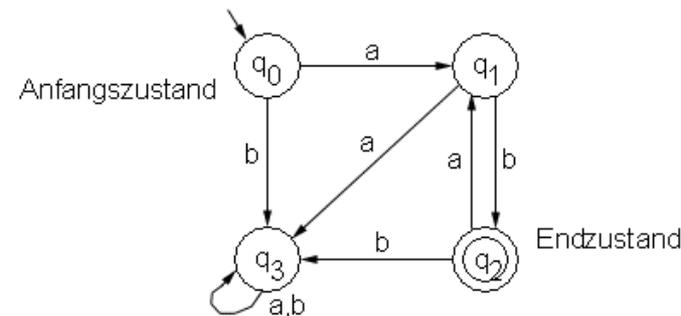


- Interaktionsdiagramme
 - Dynamische Sicht exemplarisch an Beispielszenarien
 - Interobjektinteraktion

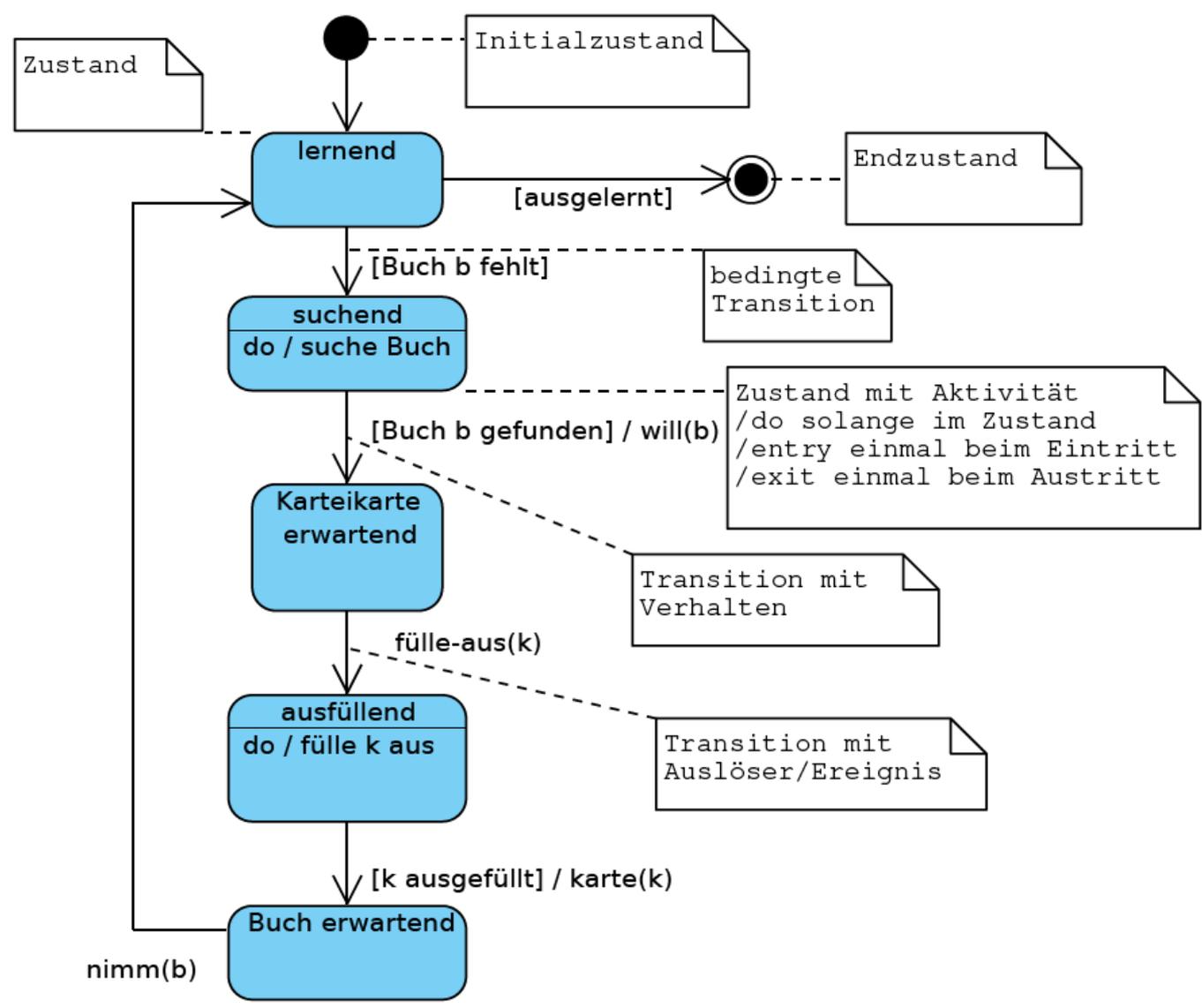


- Es fehlt: Verhalten von Objekten während Ihres Lebenszyklus
 - Intraobjektinteraktion

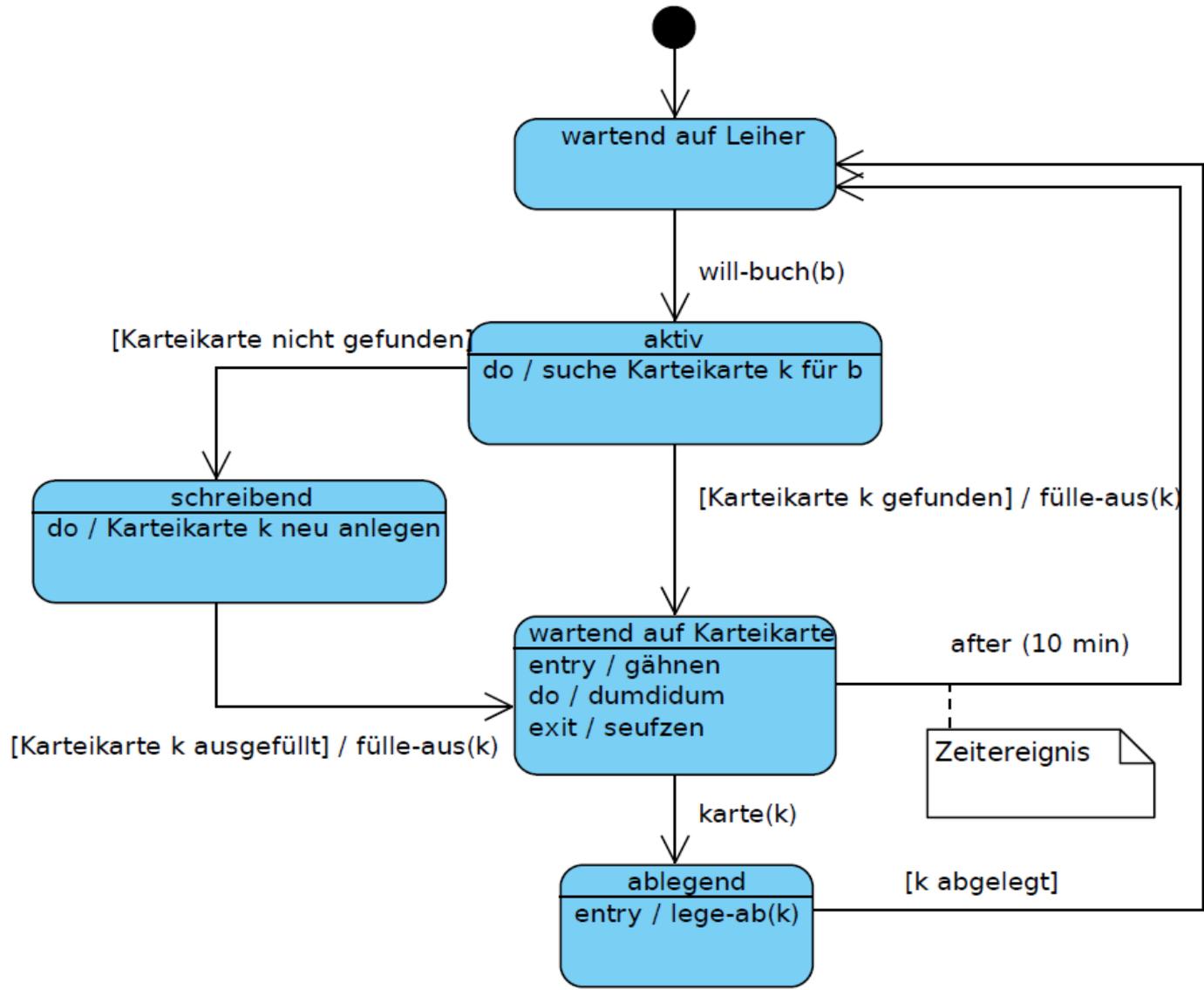
- Modellieren Objektlebenszyklus
 - Verhalten eines Objekts in Bezug auf verfügbare Methoden seiner Klasse
- Bestehen aus
 - Zustand = Menge von Attributwerten eines Objekts
 - Ereignissen = Auslöser für Zustandsänderungen
 - Transition = Übergänge zwischen Zuständen
- Basieren auf Zustandsautomaten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



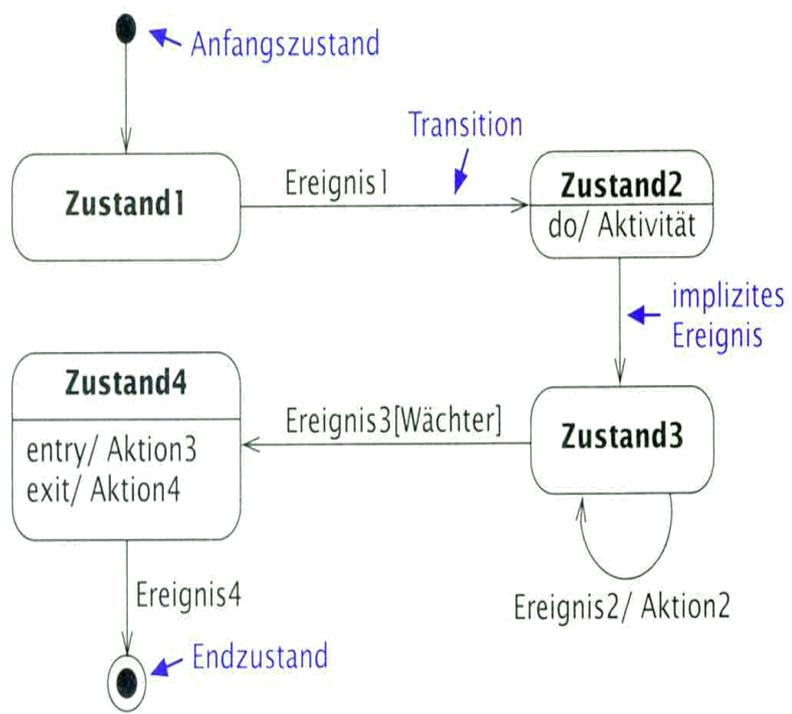
Beispiel: Zustandsdiagramm (Ausleiher)



Beispiel: Zustandsdiagramm (Bibliothekar)



Notation von Zustandsdiagrammen in UML

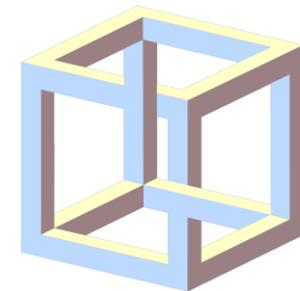


Zustandsname
do / Aktivität wenn System in Zustand
entry / wenn System in Zustand eintritt
exit / sobald Zustand verlassen wird

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)**

- Bisläng:
 - 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. $\forall 1 \leq i < \text{len}(\text{Buchliste}')$:
 $\text{element}(\text{Buchliste}', i) \leq_{\text{Attribut}} \text{element}(\text{Buchliste}', i + 1)$
 - 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

Giesbert's Kindle	
51 Inhalte	Nach Titel
Karl May	
II.1 Scepter und Hammer	Karl May
II.2 Die Juweleninsel	Karl May
II.3 Waldröschen	Karl May
II.4 Die Liebe des Ulanen	Karl May
II.5 Der verlorne Sohn	Karl May
II.6 Deutsche Herzen, deutsch...	Karl May
II.7 Der Weg zum Glück	Karl May
III.1 Der Sohn des Bärenjägers...	Karl May
III.2 Kong-Kheou, das Ehrenw...	Karl May

Seite 1 von 6

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
```

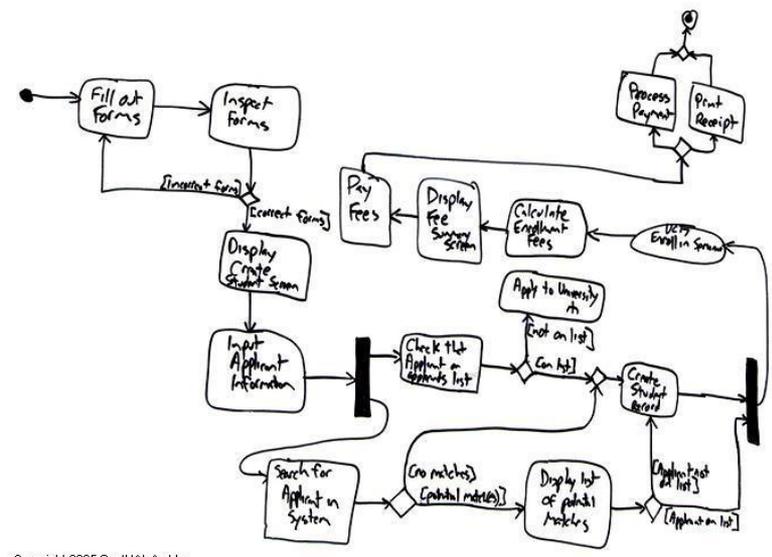
- Mehr Details würden den Rahmen der Vorlesung aber sprengen

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)

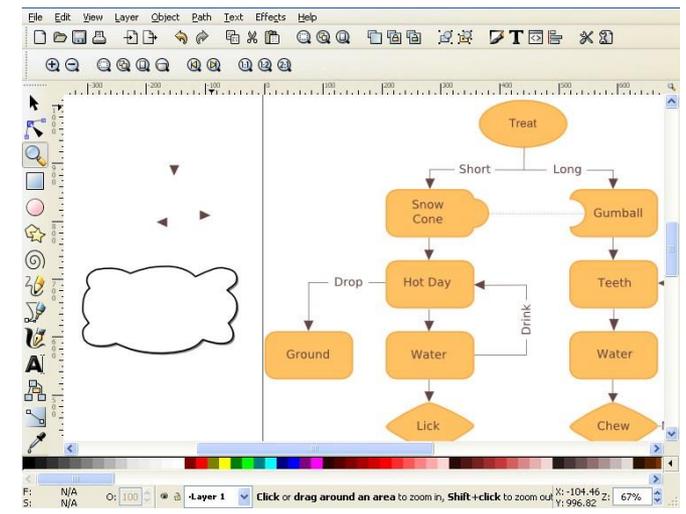
Wie erstellt man UML-Diagramme?

- Stift und Papier
 - Einfach
 - Schnell
 - Änderungen schwierig



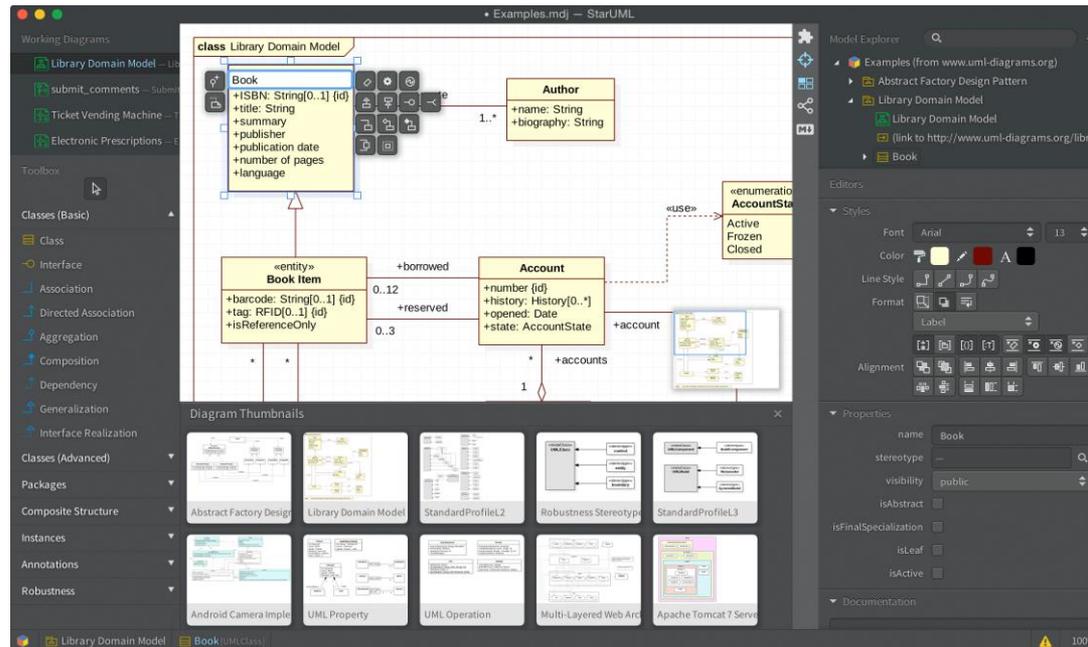
Copyright 2005 Scott W. Ambler

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



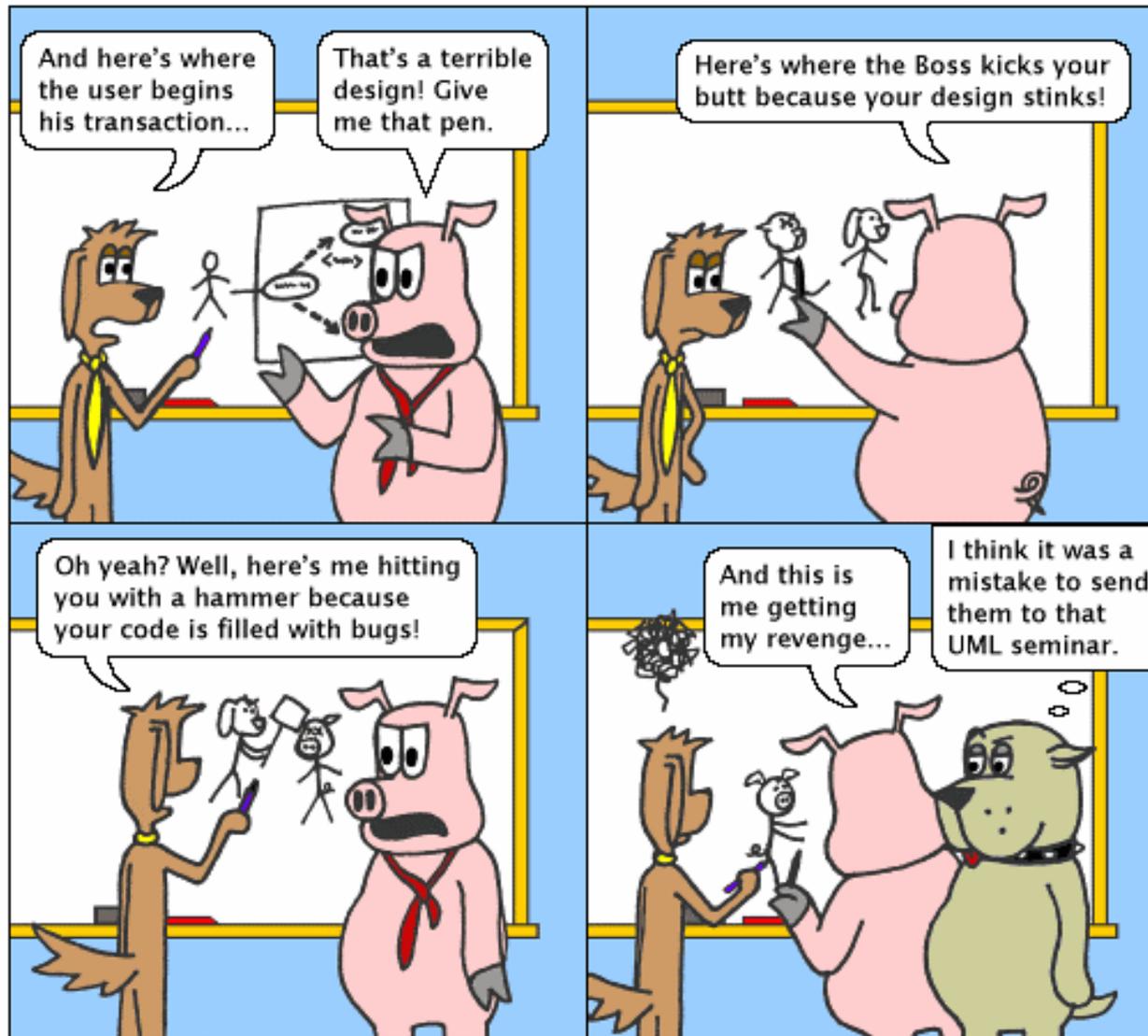
Meine Empfehlung: StarUML

- 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



<http://hackles.org>

Copyright © 2002 Drake Emko & Jen Brodzik