3. Objektorientierte Spezifikation

Zunächst: allgemeines zur objektorientierten Spezifikation (auch objektorientierte Analyse (OOA) genannt)

Dann: Die Unified Modeling Language UML

3.1 Objektorientierte Analyse

Objekt = „Gegenstand, mit dem etwas geschieht“

Folgerungen:

- Wann immer gehandelt wird, sind Objekte Behandelte und auch Ausführende.
- Handlungen stellen das Verhalten von Objekten dar.
- Ziel von Handlungen sind Änderungen von Objekten

→ Jedes Objekt (in der Realität) vereint Zustand und Verhalten

Sprachgebrauch

- Der Zustand eines Objektes ist gegeben durch die Wertebelegung seiner Attribute.
- Das Verhalten eines Objektes wird definiert durch seine Methoden.
- Nachrichten reizen ein Objekt zum Handeln und Antworten (= Anwenden von Methoden).
- Objekte werden dynamisch erzeugt und vernichtet.

Beispiele

Objekt	Attribute	Methoden
Mensch	Alter Gewicht ...	Nenne-Alter → Nenne-Gewicht → Verzehre-Nahrung (...) Laufe (...) ...
Personalausweis	Name Nummer Ausstellungsdatum Ablaufdatum ...	Ausstellen (...) Verlängern (...) ...

Beobachtungen aus Beispielen:

- Das Verhalten eines Objektes hängt von seinem Zustand ab.
- Der Zustand wird durch das Verhalten verändert.

Zustand und Verhalten sind eng verbunden.

Folgerungen

- **Einkapselung** = Zustand und Verhalten sind Bestandteile des Objekts.
- **Information Hiding** = Zustand und Verhalten sind im Objekt verborgen.
- **Autonomie** = ein Objekt entscheidet selbst über seine Reaktion auf eine Nachricht.

OOA - Statik und Dynamik

- OOA erfaßt statische und dynamische Aspekte
  - statische Aspekte: alles um die Struktur von Klassen
    - Klassen
    - Attribute, Operationen
    - Subsysteme
  - dynamische Aspekte: alles rund um Objekte und ihre Werte
    - Objekte
    - Ereignisse
    - Prozesse (verstanden als die Ausführung von Operationen)

Ziel der Objektorientierung

Bereitstellung von geeigneten Konzepten für den Umgang mit Objekten.

Zum Beispiel:

- Erzeugen gleichartiger Objekte
- Erzeugen einander ähnlicher Objekte
- Definition gleichartiger Kommunikationsbeziehungen zwischen Objekten.

Grundelemente der Objektorientierung: Klassenbildung

Definition einer Klasse (= Schablone)

- Definition der Attribute
- Definition der Methoden
- bei Bedarf Erzeugung von Objekten als „zu füllende Kopien“

Vorteile:

- Klassenbeschreibung ist unabhängig von der Existenz von Objekten
- Definition und Nutzung sind voneinander getrennt.

Beispiel:

Der Blankovordruck eines Personalausweises wird nach einer Druckschablone erstellt und anschließend nach Anweisung ausgefüllt.

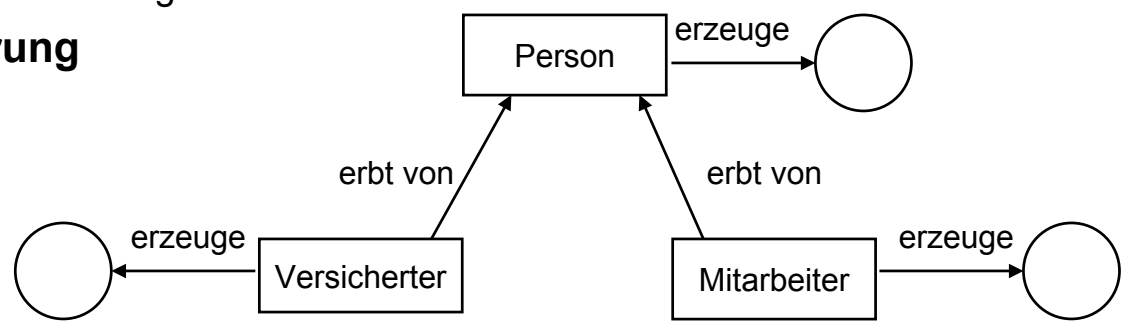
Grundelemente der Objektorientierung: Vererbung

- Ziel: Übernehmen einer Klassenbeschreibung beim Erzeugen einer ähnlichen Klasse
- Konzept: Vererbung

  = Übernahme der Klassendefinition einer Superklasse in die Klassendefinition einer Subklasse

Beispiel für Vererbung:

Versicherung

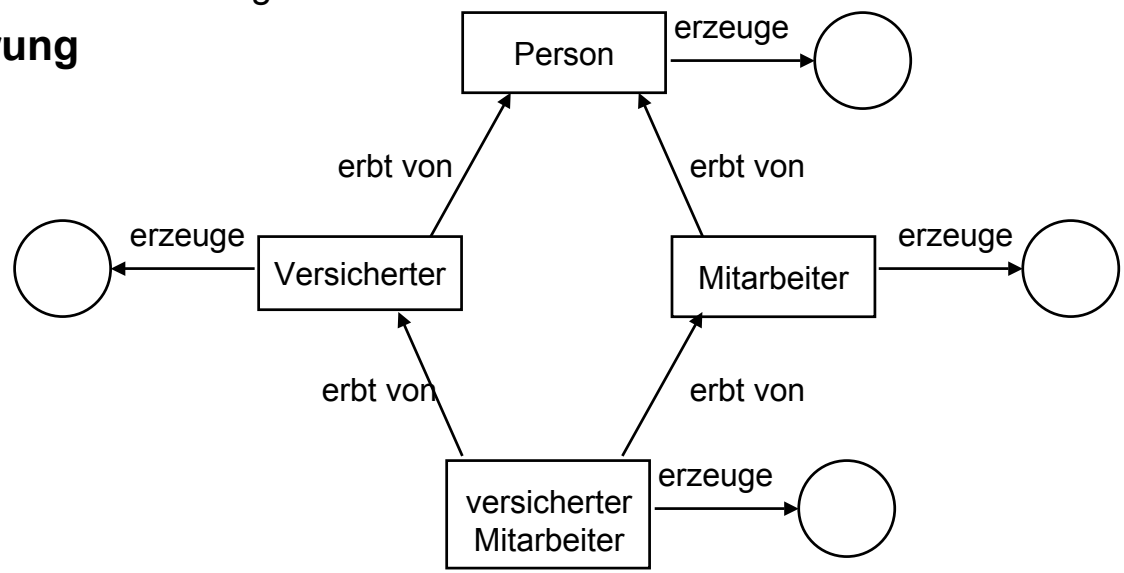


Grundelemente der Objektorientierung: Vererbung

- Konzept: Mehrfacherbung
  = eine Subklasse besitzt mehrere Superklassen

Beispiel für Mehrfacherbung:

Versicherung



Grundlemente der Objektorientierung: Beziehungen zwischen Klassen

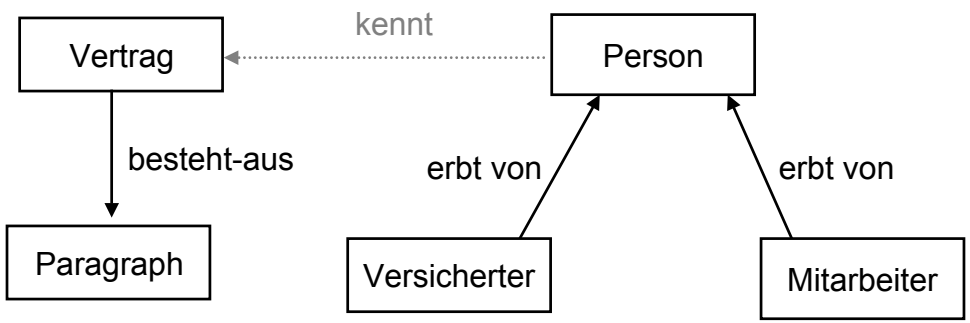
Gleichartige Kommunikationsbeziehungen

zwischen Objekten werden durch die Definition der Beziehungen zwischen Klassen vorgegeben:

- Assoziation („hat-Kenntnis-von“)
- Aggregation („ist-Teil-von“)

Beispiel:

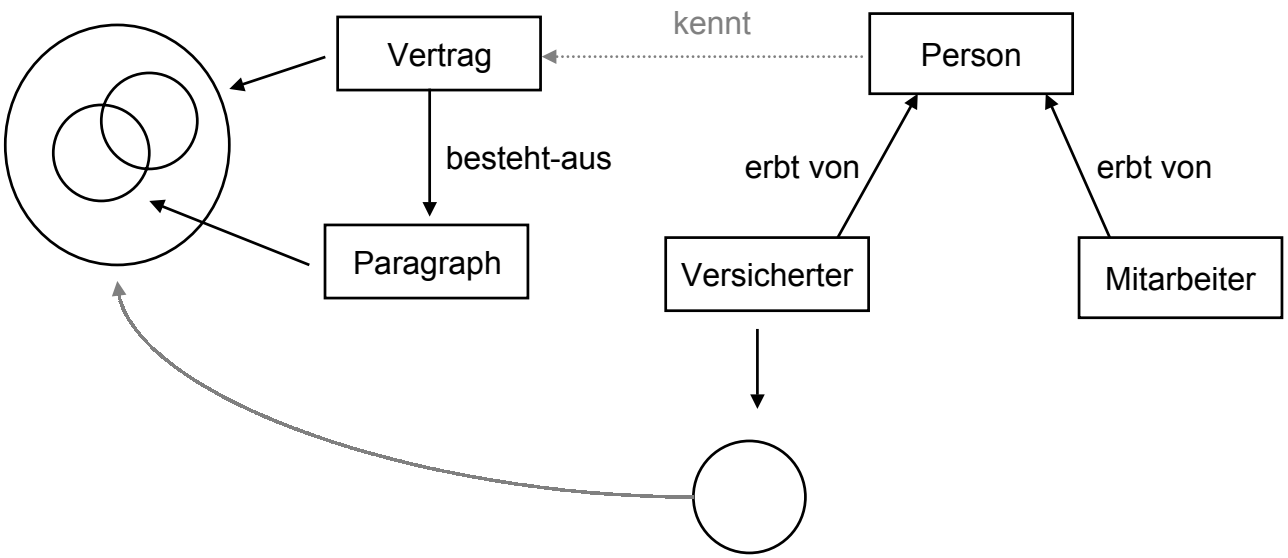
Versicherung



Grundelemente der Objektorientierung: Beziehung zwischen Klassen

Beispiel für erzeugte Objekte:

Versicherung

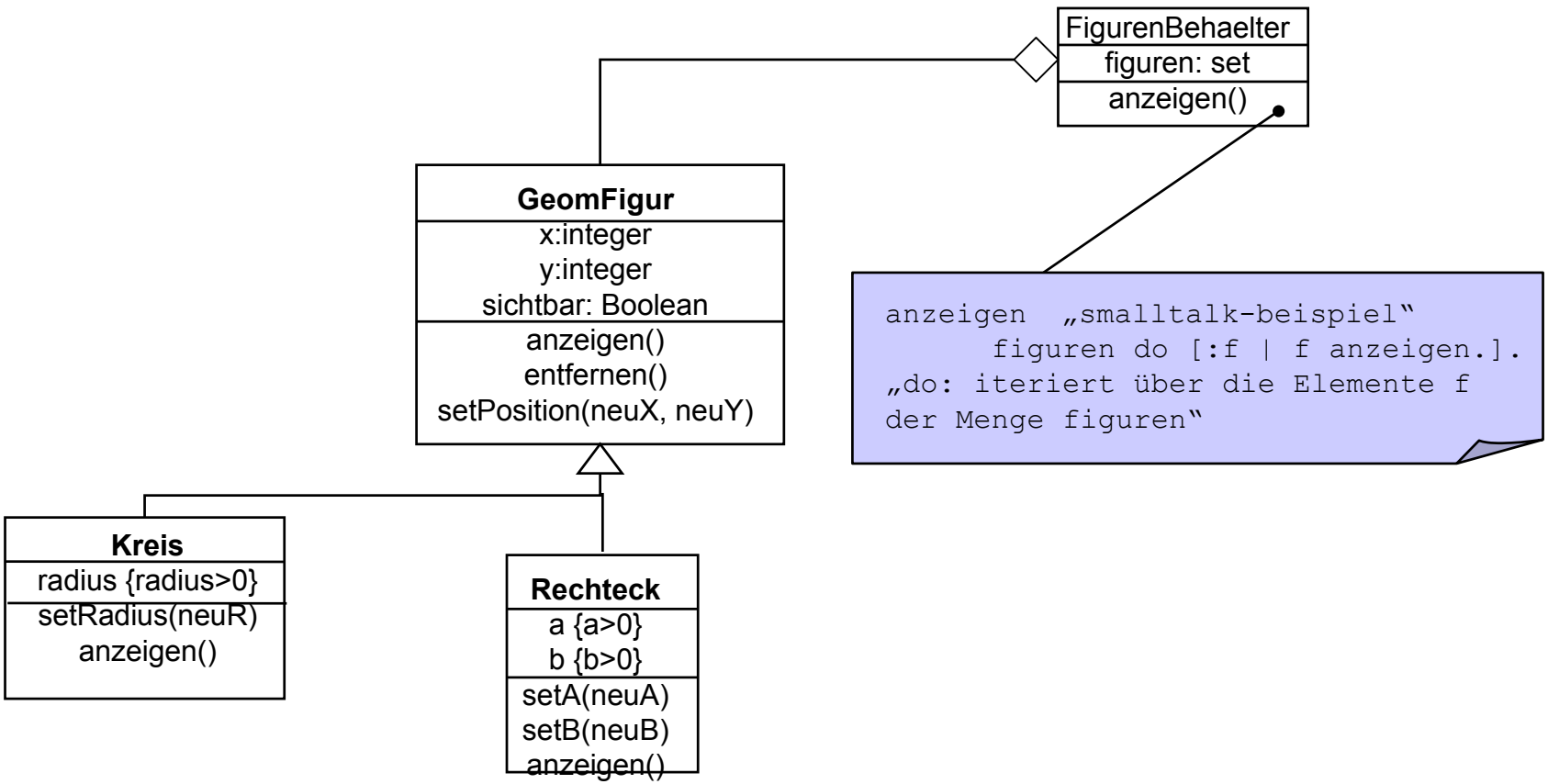


Grundelemente der Objektorientierung: Polymorphie

Polymorphie:

statisch: - in Abhängigkeit von den Parametertypen wird die passende Operation gewählt.
- dies geht auch ohne Objektorientierung
Bsp.: +

dynamisch: die aufzurufende Operation wird erst dann ermittelt (und an das Operationssymbol gebunden), wenn sie aufgerufen wird (und nicht etwa zur Compile-Zeit)


FigurenBehaelter
figuren: set
anzeigen()
GeomFigur
x:integer
y:integer
sichtbar: Boolean
anzeigen()
entfernen()
setPosition(neuX, neuY)
anzeigen „smalltalk-beispiel“
figuren do [:f | f anzeigen.].
„do: iteriert über die Elemente f
der Menge figuren“
Kreis
radius {radius>0}
setRadius(neuR)
anzeigen()
Rechteck
a {a>0}
b {b>0}
setA(neuA)
setB(neuB)
anzeigen()

Grundelemente der Objektorientierung: Objektidentität

Objektidentität

- Objektwertgleichheit ≠ Objektgleichheit
- Es kann objektwertgleiche Objekte geben
- Unterscheidung erfolgt über interne Kennungen (Surrogate)

Zusammenfassung

Objekt-Orientierung

- Ein objekt-orientiertes System besteht aus einer Menge von Objekten, die miteinander Nachrichten austauschen.
- Bei der Gestaltung erleichtern *Klassen*, *(Mehrfach-)Vererbung*, *Aggregation* und *Assoziation* die Definition, die Konfiguration und den Einsatz von Objekten.
- Die Struktur der Klassen ist statisch.
- Die Menge und Struktur der Objekte ändert sich dynamisch.

Grundphilosophie objekt-orientierter Software-Entwicklung

- „reale“ Handlungsabläufe werden ins „Software-Modell“ übertragen
- Realität wird als objektorientiertes System aufgefaßt
- „reale“ Objekte werden durch passende „Modell-Objekte“ ersetzt
- „Modell-Objekte“ übernehmen die „Modell-Handlung“

⟶ Objekte bestimmen und geeignetes Modell schaffen!

Entwicklungsabschnitte

- **OO-Analyse**: Bestimmen von Objekten und Klassen aus der Realität des Anwendungsbereichs
- **OO-Design**: Ergänzen der Strukturen aufgrund technischer Erfordernisse
- **OO-Programmierung**: Umsetzen in Programmiersprache und Anpassen an Sprachspezifika

OO-Analyse besteht aus

der Erforschung des Anwendungsbereichs, d.h.

- Objekte entdecken
- Klassen ableiten
- Klassen strukturieren
- Aufgaben zuordnen
- Zusammenarbeit festlegen

Erforschung: Vorgehen

Entdecken von Objekten des Anwendungsbereichs:

- Ableiten aus textueller Beschreibung
  (Hauptwörter selektieren und normieren)
- Ableiten aus eigenem Wissen
- Ableiten aus Expertenbefragung
- Ableiten aus „Use Case Model“
  (Analyse der anwendungstypischen Handlungsabläufe)

Kandidaten für Objekte sind

- greifbare Dinge
- Konzepte
- Ereignisse

Erforschung: Weiteres Vorgehen

Ableiten von Klassen aus Objekten

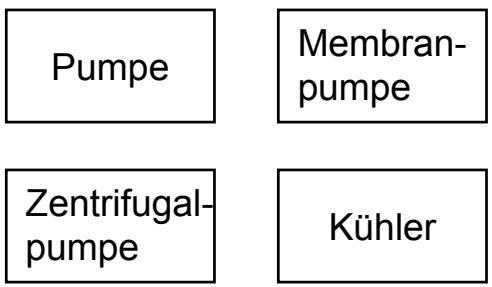
=„natürliche“ Gemeinsamkeiten von Objekten erkennen und diese in Klassen zusammenfassen

Strukturieren der Klassen

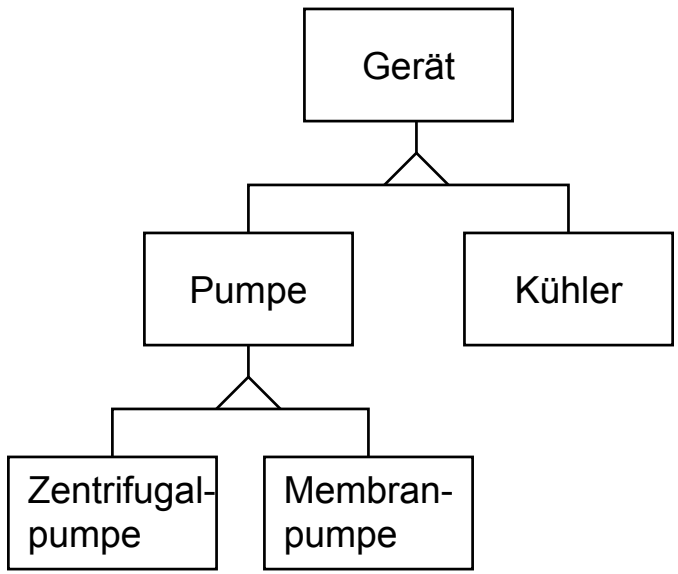
- existierende Vererbungsstrukturen bestimmen
  (mögliche Subklassen suchen)
  (mögliche Superklassen suchen)
- Superklassen ergänzen
  (Gemeinsamkeiten in Superklassen extrahieren)
- Abhängigkeiten bestimmen
  (Aggregationen und Assoziationen charakterisieren)

Erforschung: Beispiel Vererbung

Ohne Beziehungen



Mit Vererbung

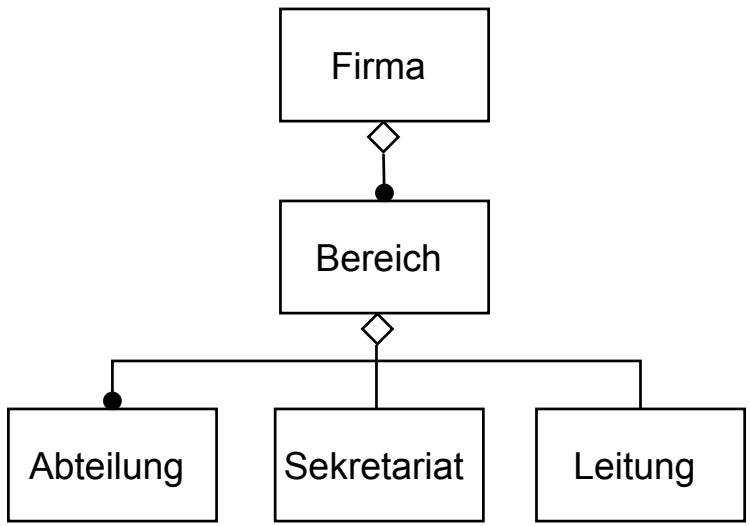


Notation: OMT

Erforschung: Beispiel Aggregation

Ohne Beziehungen

Mit Aggregation



Notation: OMT

Erforschung: Weiteres Vorgehen

Zuordnen von Aufgaben zu Klassen

- Klassen beschreiben
- Attribute bestimmen
  (Zustände von Objekten abgrenzen)
  (Attribute Superklassen zuordnen)
  (Relevanz von Attributen für alle Objekte)
- Methoden bestimmen
  (Verhalten zu Attributen)
  (Leistungen des Systems verteilen)
  (Relevanz von Methoden für alle Objekte)

Erforschung: Weiteres Vorgehen

Zusammenarbeit festlegen

= Nachrichtenaustausch zwischen Objekten verschiedener Klassen charakterisieren

Klassen sind Vertragspartner der Form „Produzent“ und „Konsument“ wie im Wirtschaftsleben:

- Nachfrage nach Verhalten aufstellen
- Angebote an Verhalten vornehmen
- Angebote und Nachfragen einander anpassen

OO-Analyse

Das Ergebnis der vorgestellten OO-Analyse ist eine statische Struktur von Klassen (= Klassenmodell).

Es fehlt eine Beschreibung der Dynamik, z.B. Angaben über

- das Erzeugen und Vernichten von Objekten,
- die zu einem Zeitpunkt konkret agierende Menge von Objekten,
- die Wirkung einer Methodenanwendung auf den internen Zustand eines Objektes
- den Ablauf der Kommunikation zwischen Objekten

OO-Analyse

Zur Beschreibung der Dynamik von objektorientierten Systemen werden „herkömmliche“ Methoden eingesetzt. Zum Beispiel:

- Zustandsdiagramme
- Sequenzdiagramme
- Anwendungsfalldiagramme
- formale, semiformale und informale Texte

Eine statische Struktur ohne Dynamikmodellierung ist Stückwerk !

Vorteile von objektorientierten Methoden gegenüber funktionsorientierten (strukturierten) Methoden

- Ganzheitliche Betrachtung
- unterschiedliche Abstraktionsebenen bei Anwendung des gleichen Paradigmas
- evolutionäre Entwicklung ist möglich
- einfacher Zugang zu Anwendungsgebiet
- bessere Kommunikationsbasis als bei separaten ER-Modellen und Funktionsbäumen (oder ähnlicher Darstellung)
- kein Paradigmenwechsel in der Entwicklung
- bessere Durchschaubarkeit durch Übereinstimmung der Strukturen mit der Realität
- einfache Möglichkeit der Wiederverwendung

Risiken und Gefahren von objektorientierten Methoden

- wenig Erfahrung
- Scaling-Up-Problem noch nicht vollständig bekannt / gelöst
  - Konfigurations-Management
  - integriertes Qualitäts-Management
- Überschätzung von Einzelaspekten, z.B. Vererbung, und deren exzessive Verwendung
- Gefahr der Überspezifikation (der Zweck des Verständnisses gerät außer Sicht)
- unausgereifte Werkzeuge

Rückblick auf Historie

Ursprünge der Vererbung	Programmiersprache SIMULA	1967
Grundlagen für Einkapselung	Information Hiding	1972
	Abstrakte Datentypen	1975
Grundlage für Notationen	Entity-Relationship-Modellierung	1976
Dynamikmodellierung	(traditionelle Methoden)	

Objektorientierung kombiniert etablierte Methoden!

Objektorientierte Modellierungssprachen ermöglichen Vorteile, sie garantieren sie nicht.

Deshalb: objektorientierte Modellierung muß durch geeignete Methoden und Richtlinien ergänzt werden.

Versuch: UML und passende Methode

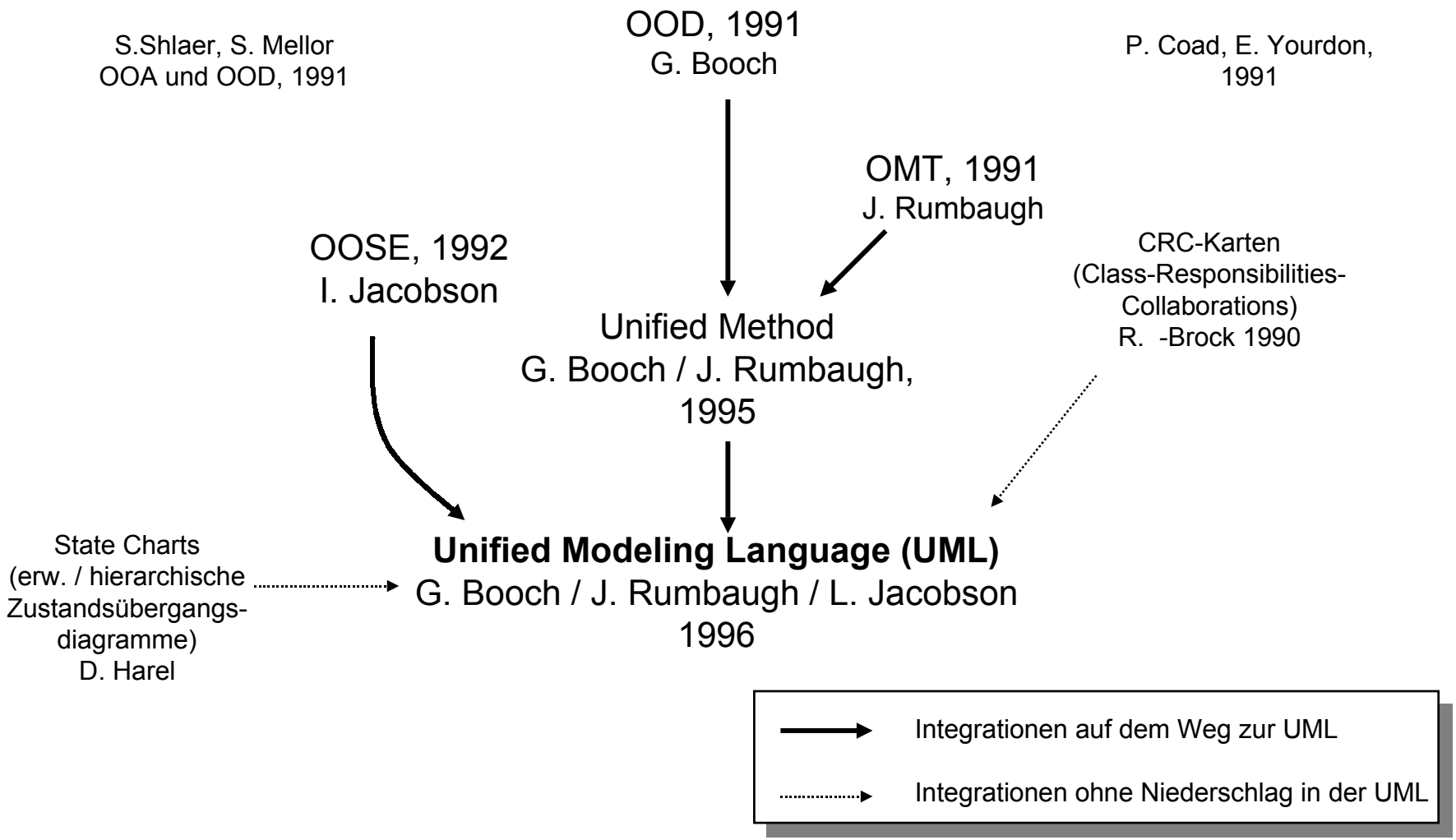
3.2 Die Unified Modeling Language (UML)

Spezifikationssprache: UML

Syntax

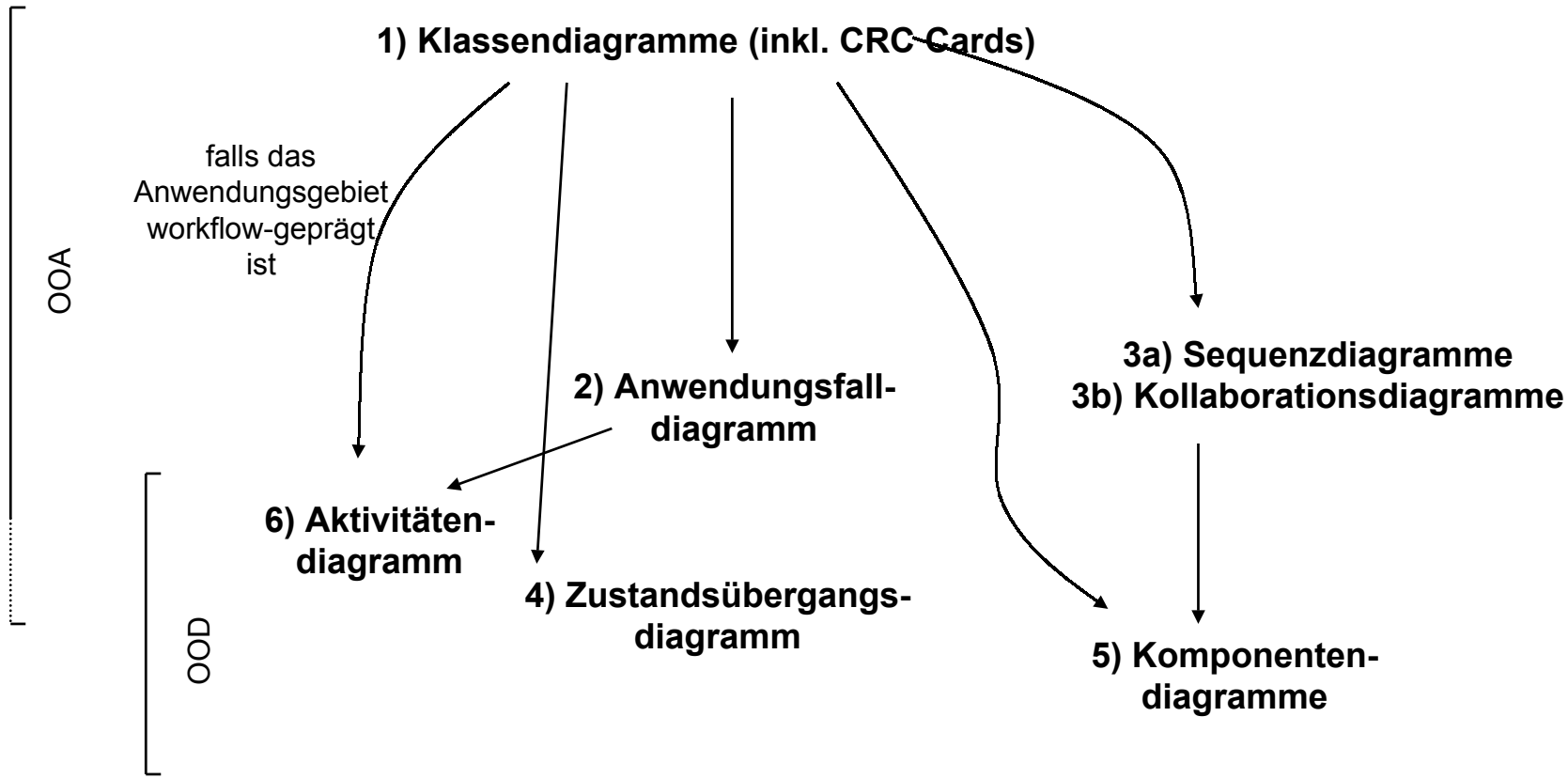
Methodik

Semantik


S.Shlaer, S. Mellor
OOA und OOD, 1991
OOD, 1991
G. Booch
P. Coad, E. Yourdon,
1991
OMT, 1991
J. Rumbaugh
OOSE, 1992
I. Jacobson
CRC-Karten
(Class-Responsibilities-
Collaborations)
R. -Brock 1990
Unified Method
G. Booch / J. Rumbaugh,
1995
State Charts
(erw. / hierarchische
Zustandsübergangs-
diagramme)
D. Harel
Unified Modeling Language (UML)
G. Booch / J. Rumbaugh / L. Jacobson
1996
Integrationen auf dem Weg zur UML
Integrationen ohne Niederschlag in der UML

Beschreibungsmittel in UML

- Anwendungsfalldiagramme (Use Case Diagram)
  - Akteure
  - Anwendungsfälle
  - Beziehungen
- Klassendiagramme (Klassenstrukturdiagramm)
  - Klassen
  - Beziehungen zwischen Klassen
- Verhaltensdiagramme
  - Aktivitätsdiagramme
    - Zustände
    - Zustandsübergänge
    - Ereignisse
    - Aktivitäten
    - Objektzustände
  - Kollaborationsdiagramme
    - Objekte
    - Beziehungen inklusive Nachrichtenaustausch (räumlich geordnet)
  - Sequenzdiagramme (Weg-Zeit-Diagramme)
    - Objekte
    - Beziehungen inklusive Nachrichtenaustausch (zeitlich geordnet)
  - Zustandsdiagramme
    - Zustände
    - Beziehungen
    - Ereignisse
- Implementierungsdiagramme
- Komponentendiagramm
  - Komponenten
  - Beziehungen zwischen Komponenten
- Einsatzdiagramme
  - Komponenten
  - Beziehungen
  - Knoten

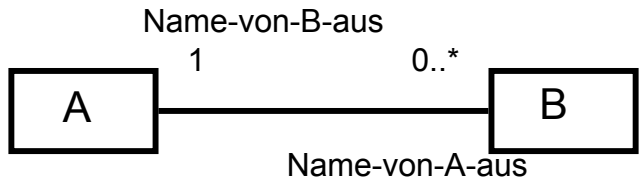

1) Klassendiagramme (inkl. CRC-Cards)
falls das Anwendungsgebiet workflow-geprägt ist
OOA
2) Anwendungsfall-diagramm
3a) Sequenzdiagramme
3b) Kollaborationsdiagramme
6) Aktivitäten-diagramm
4) Zustandsübergangs-diagramm
5) Komponenten-diagramme
OOD

UML-Klassendiagramme

Klassifikation	->	Bildung von Klassen
Generalisierung/	->	Vererbung
Spezialisierung		Bildung von Ober- und Unterklassen
Assoziation	->	Beziehung zwischen Objekten einer oder mehrerer Klassen
Aggregation	->	Spezialfall der Assoziation, Objekte sind Bestandteile eines anderen Objektes

Klassifikation

Assoziation

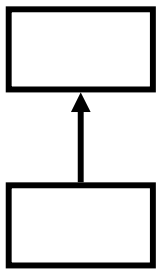


Ein A-Objekt steht zu beliebig vielen B-Objekten in Verbindung.

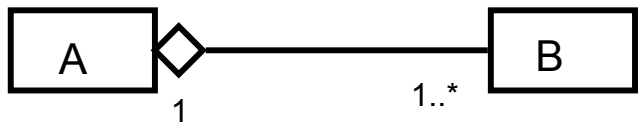
Die Beziehung von A aus gesehen heißt „Name-von-A-aus“.

Ein B-Objekt steht zu einem A-Objekt in Verbindung.

Generalisierung



Aggregation

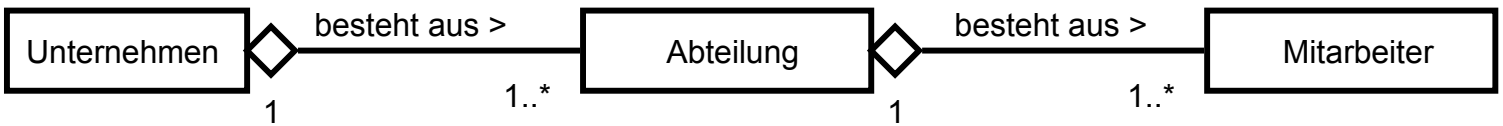


Ein A-Objekt steht zu einem oder mehreren B-Objekten in Verbindung.

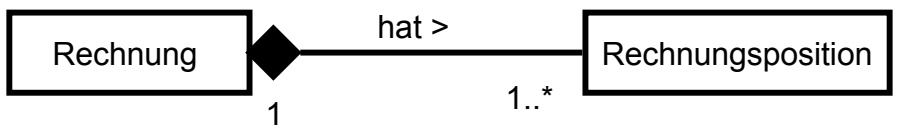
Ein B-Objekt steht zu einem A-Objekt in Verbindung.

Unterscheidungen verschiedener Aggregationen

Normale Aggregation



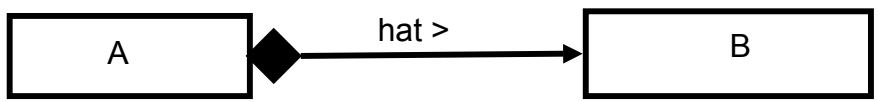
Komposition (Einzelteile sind vom Aggregat existenzabhängig d.h. sie können ohne das Aggregat nicht existieren).



Hinweis: Bei Komposition auf der Aggregatseite 1.

Richtungen von Assoziationen und Aggregationen

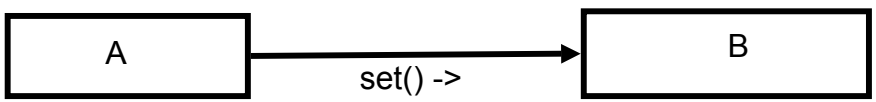
... geben an, welche Objekte von der Beziehung wissen



Ein Objekt der Klasse A kennt das komponierte Objekt der Klasse B,
aber nicht umgekehrt, es kann Nachrichten an Objekte der Klasse B senden,
aber nicht umgekehrt!

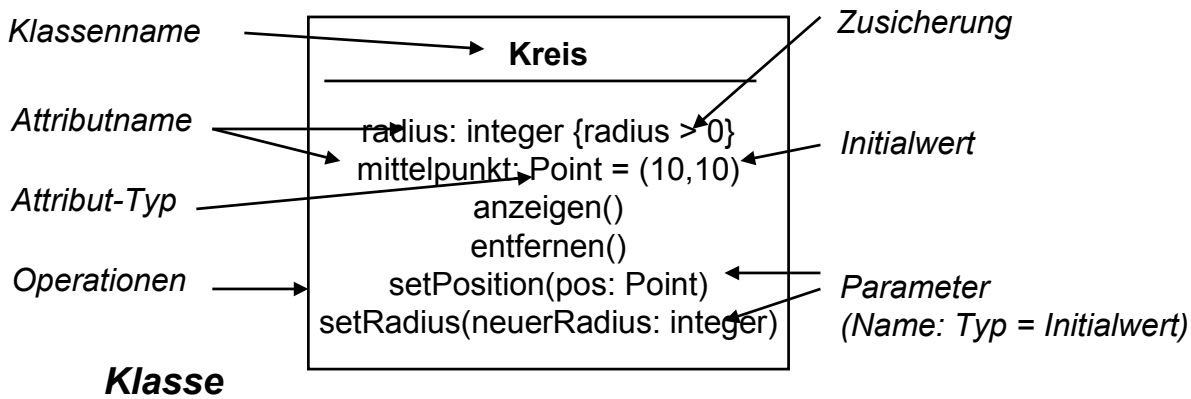
Kommunikation zwischen Objekten

... erfolgt über den Austausch von Nachrichten

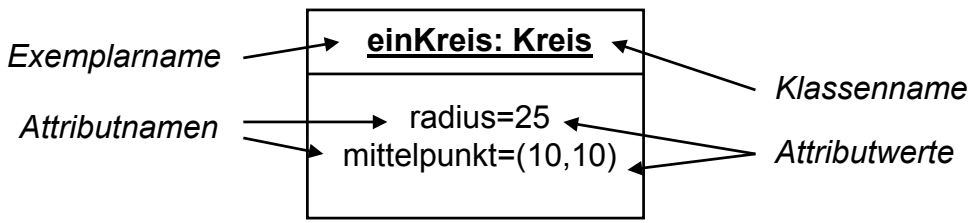


Ein Objekt der Klasse A kann die Nachricht set() an ein Objekt der Klasse B senden.
.

Grundelemente der Objektorientierung: Klassen, ihre Interna und Objekte



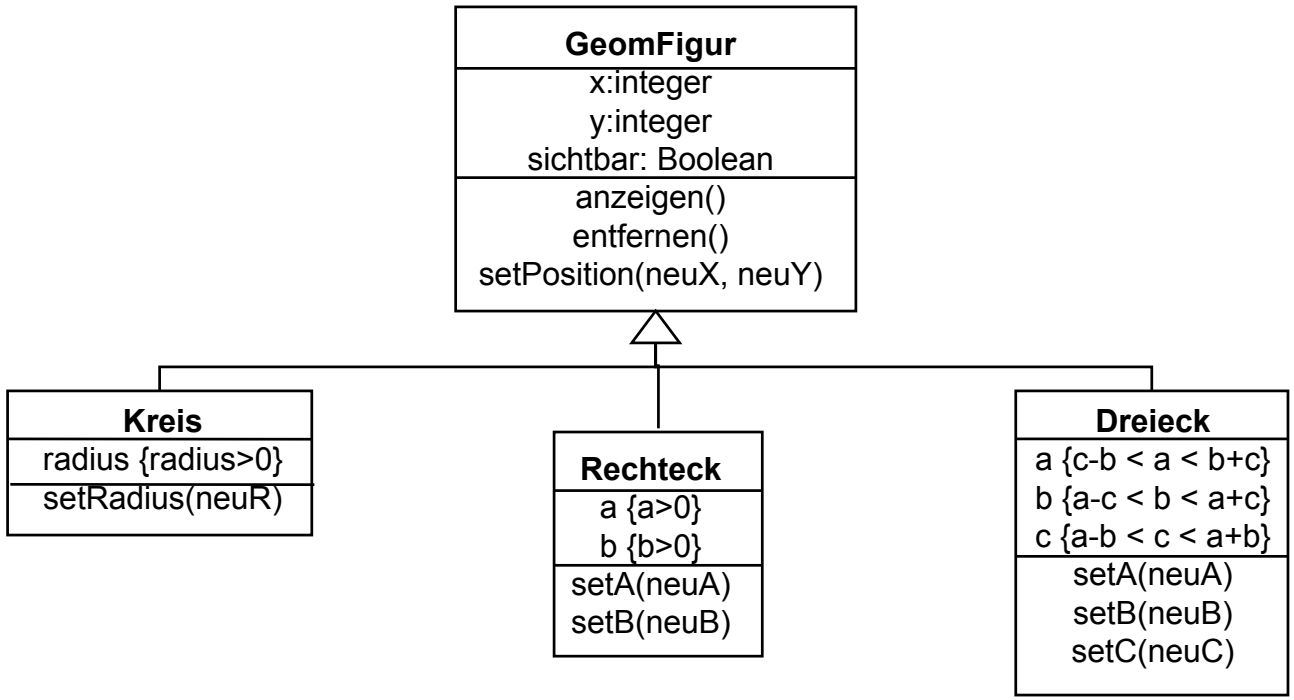
Klasse



Objekt

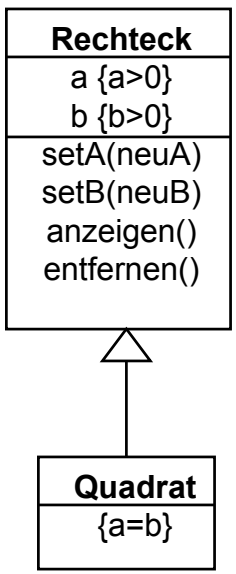
Grundelemente der Objektorientierung: Vererbung

Attribute und Operationen in Vererbungsbeziehungen



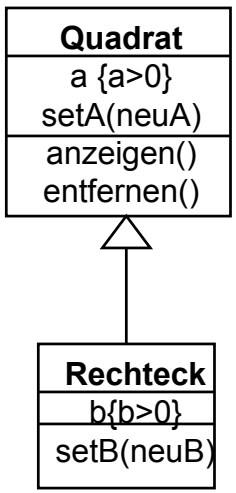
Attribute und Operationen werden in den Klassen angesiedelt, in die sie der Anschauung nach gehören. Redundanz-, Performanz- und sonstige Entwurfs- und Implementierungsaspekte interessieren während der OOA ***NICHT*** !

Vererbung gemäß Anschauung



Liefert für Quadrate nicht die minimale Implementierung, denn es gibt zwei Kantenlängen.

Alternative Vererbung



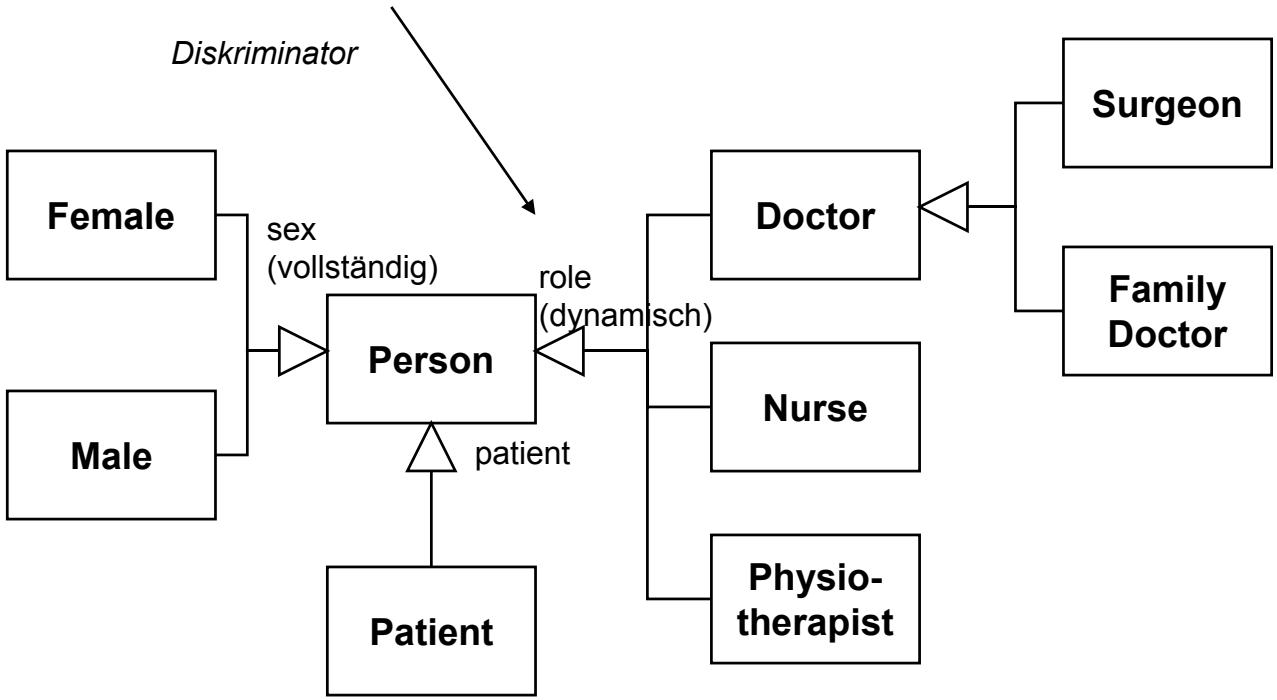
Entspricht nicht der Anschauung,

ist aber redundanzfrei !

Multiple Klassifikation

- Bei multipler Klassifikation kann ein Objekt durch mehrere Klassen beschrieben werden, die nicht über Vererbungsbeziehungen miteinander verbunden sind.
- Zur Angabe der gültigen Kombinationen von Klassen werden Diskriminatoren verwendet.
- Alle Klassen mit dem gleichen Diskriminator sind disjunkt.
- Vollständige Diskriminatoren bedeuten, daß die Vereinigung aller Objekte der „Unterklassen“ die Menge der Objekte der „Oberklasse“ ergibt. Im Sinne des Diskriminators ist die „Oberklasse“ dann eine abstrakte Klasse.
- Dynamische Diskriminatoren erlauben, daß ein Objekt zur Laufzeit den Typ wechselt (schwierige Implementierung!)

Multiple Klassifikation



Legale Kombination von Klassen für ein Objekt:	Female, Patient, Nurse
	Male, Physiotherapist
	Female, Patient
Illegale Kombinationen von Klassen für ein Objekt:	Patient, Doctor
	Male, Doctor, Nurse

(vgl. Entwurfsmuster *Role Model* im Abschnitt zur Rolle Designer)

Abstrakte Klassen

... sind Klassen, von denen keine Objekte erzeugt werden können.

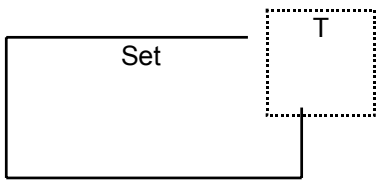
Die Klasse „GeomFigur“ ist eine abstrakte Klasse, denn es werden konkrete Kreise, Dreiecke, Rechtecke, aber keine geometrischen Figur-Objekte erzeugt.

Parametrisierte Klassen

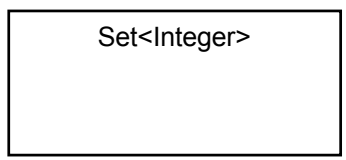
... sind Klassen, die einen Typ als Parameter haben.

Typischerweise: Stacks, Queues, Listen, Mengen

Parametrisierte Klasse

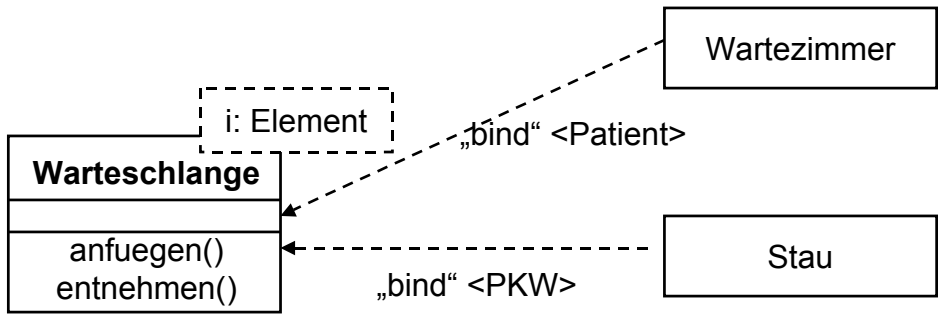


Parametrisierte Klasse mit gebundenem Parameter

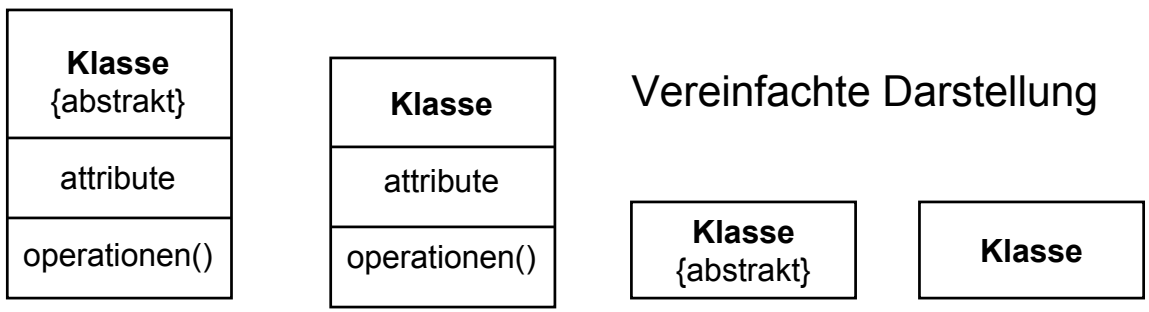


Graphische Darstellung von parametrisierbaren und abstrakten Klassen

parameterisierbare Klassen



abstrakte Klassen



Attribute, Klassenattribute, abgeleitete Attribute

Attribute werden beschrieben durch

- Name
- Datentyp oder Klasse
- Initialwert
- Zusicherungen / constraints (Syntax: {constraint}, keine weiteren Angaben!)

Klassenattribute gehören nicht zu einem Objekt, sondern zur Menge aller Objekte einer Klasse (vereinfacht: zur Klasse).

Beispiel**:** Zähler, der angibt, wieviel Objekte eines Typs erzeugt werden.

Abgeleitete Attribute sind Attribute, deren Werte aus den Werten anderer Attribute abgeleitet werden können. Die namen abgeleiteter Attribute beginnen mit einem Slash /

Beispiel: Das Attribut */Alter* einer Person kann aus ihrem Geburtsdatum abgeleitet werden.

Sichtbarkeit von Attributen

- **public**: für alle sichtbar und benutzbar. UML-Notation +
- **protected**: für die Klasse selbst, für die Unterklassen und für explizit ausgezeichnete Klassen. UML-Notation #
- **private**: für die Klasse selbst und für explizit ausgezeichnete Klassen. UML-Notation -

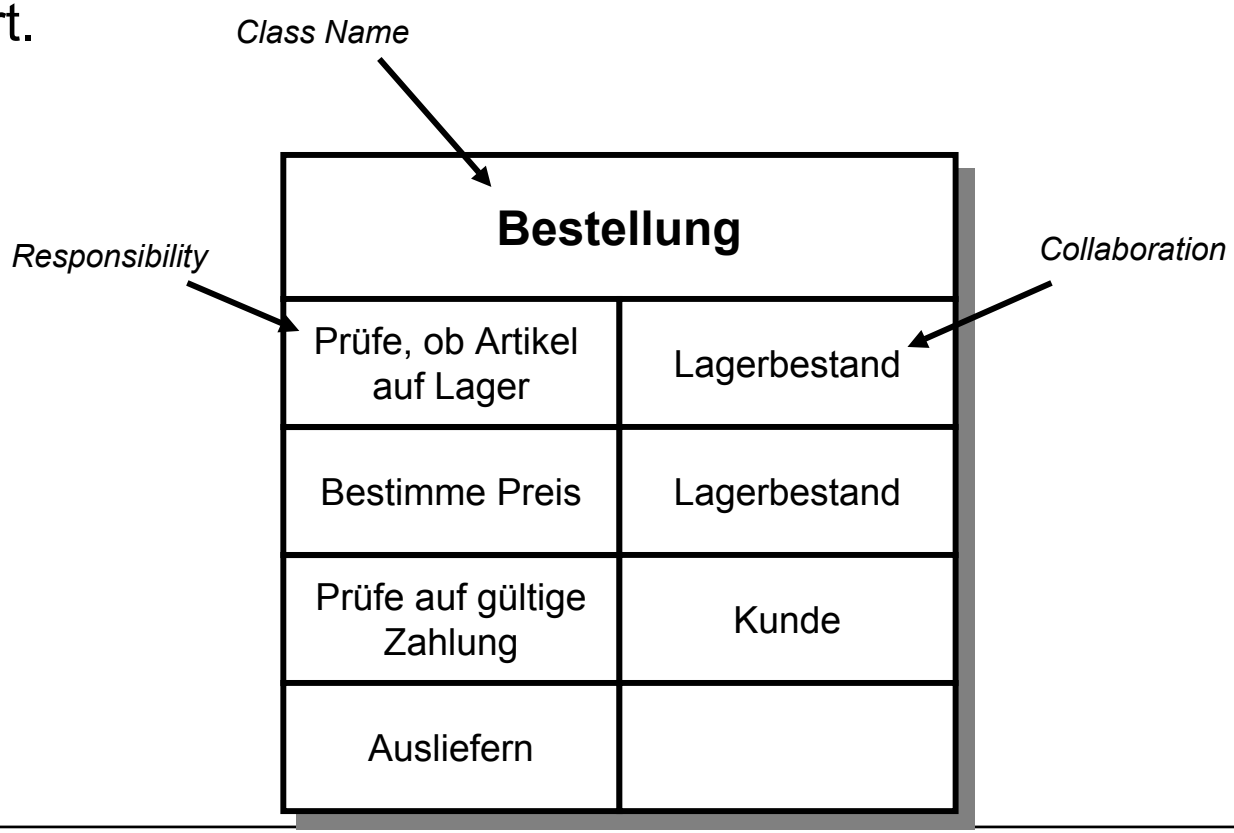
- Syntax der Operationsdefinition:
  visibility name (parameter-list) : return-type-expression {property-string}

  visibility: + (public), # (protected), - (private)
  name: string
  parameter-list: beinhaltet (optional) Argumente, Syntax wie bei Attributen
  return-type-expression: optional Spezifikation des Rückgabewertes (abhängig von der Implementierungssprache)
  property-string: Eigenschaften der Operation

  Hinweis: In der OOA sollten die Operationen nicht im Sinne einer Schnittstellendefinition verwendet werden. Stattdessen sollten sie die prinzipiellen Verantwortlichkeiten und Aufgaben einer Klasse definieren. -> zum Beispiel mit CRC Karten!

CRC Karten (Class-Responsibility-Collaboration)

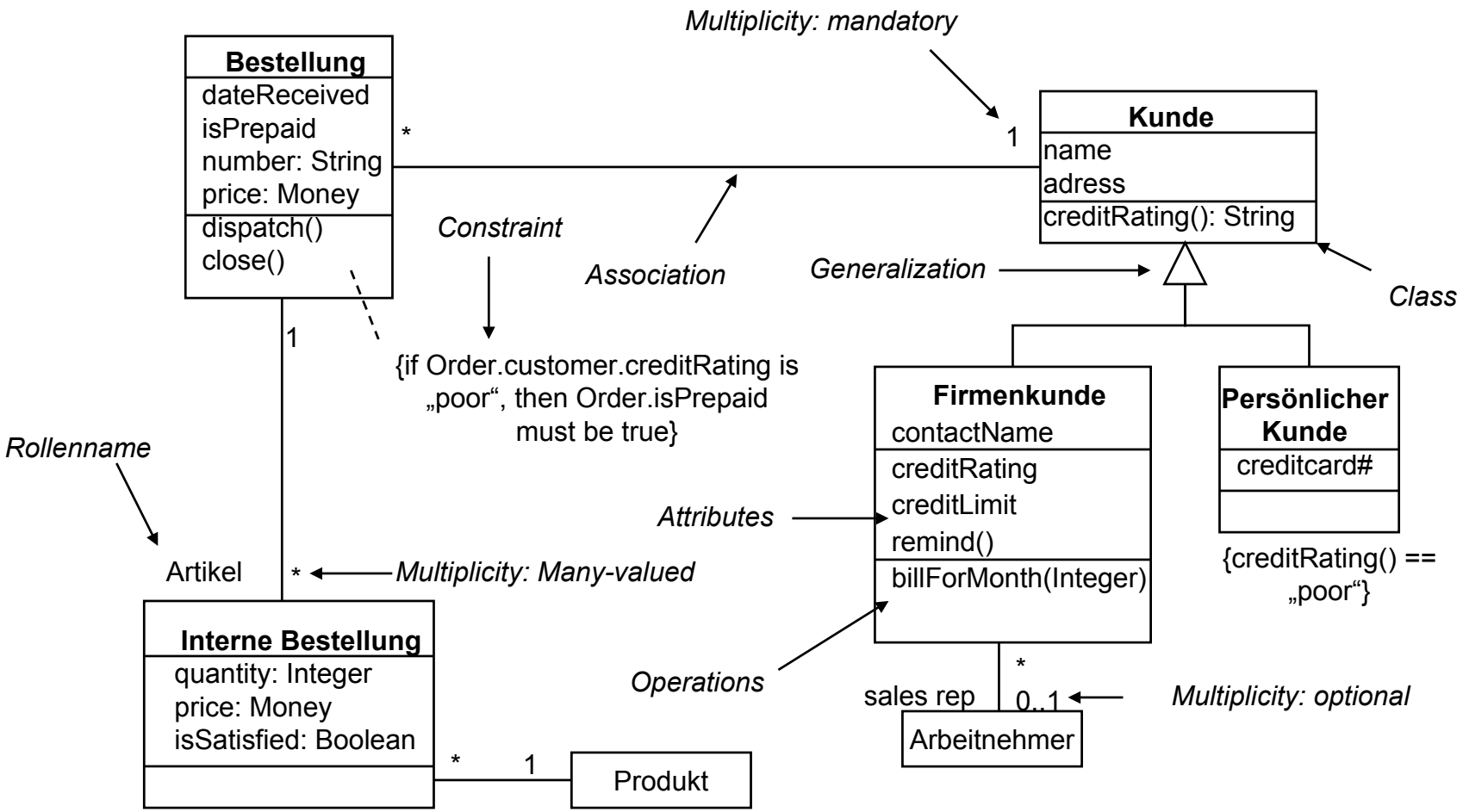
- Mit CRC-Karten werden die abstrakten Aufgaben einer Klasse definiert.



Rollen in Assoziationen

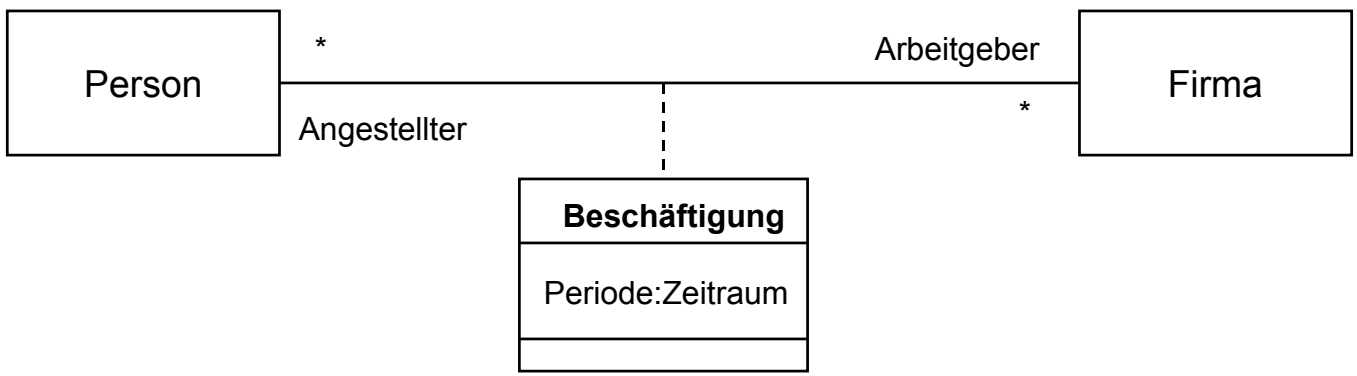
- Jede Assoziation hat zwei Rollen. Jede entspricht einer Richtung.
- Rollen können explizit benannt werden.
- Der Name einer Rolle von Klasse A nach Klasse B steht am B-Ende der Assoziation (vgl. Bestellung - interne Bestellung).

UML: Klassendiagramme



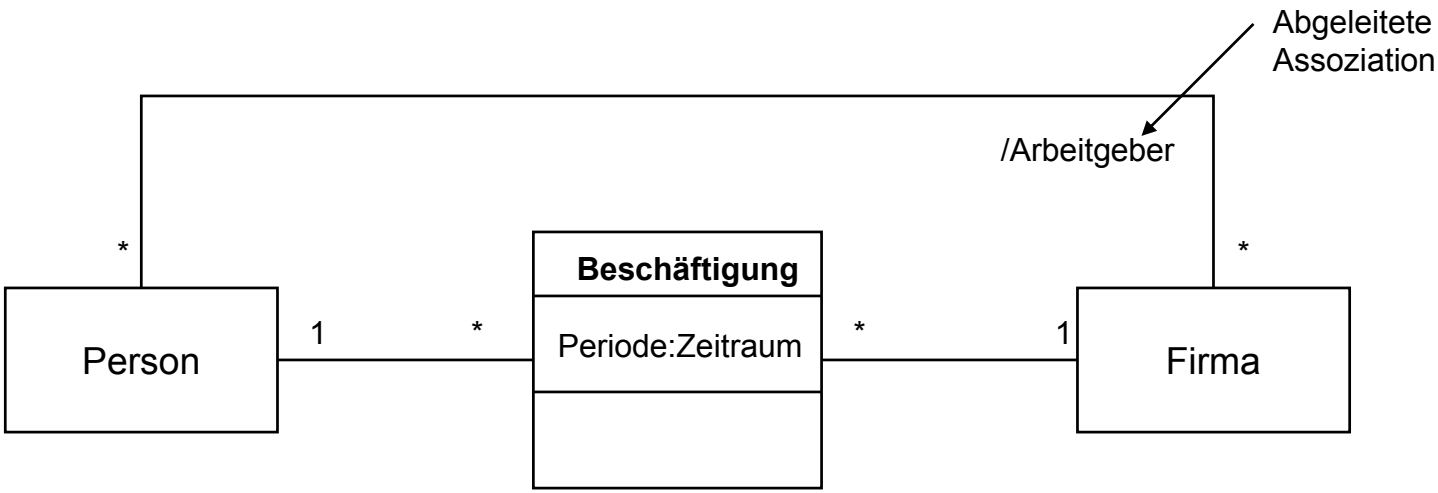
Klassendiagramme: weiterführende Konzepte

Assoziationsklassen werden benutzt, um Assoziationen mit Attributen und Operationen zu verknüpfen.



Klassendiagramme: weiterführende Konzepte

Assoziationsklassen können vermieden werden. Ihre Bedeutung ergibt sich gemäß dem folgenden Beispiel.

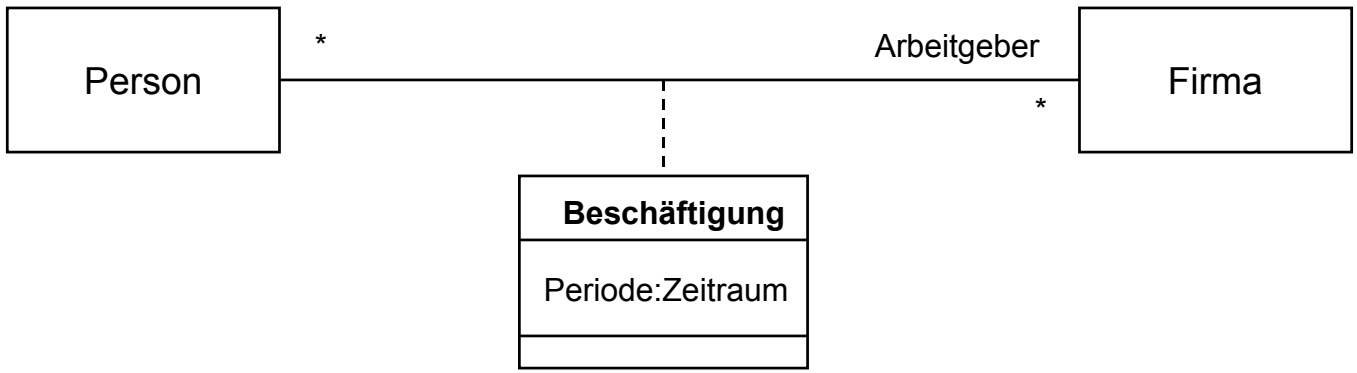


Beförderung einer Assoziationsklasse zu einer vollen Klasse

Bei der Verwendung einer Assoziationsklasse gab es zwischen einer Person und einer Firma nur ein Beschäftigungsverhältnis, diese Restriktion ist durch die Verwendung der normalen Klasse aufgehoben.

Klassendiagramme: weiterführende Konzepte

Frage: Ist die Restriktion sinnvoll?



Annahme: eine Person verläßt eine Firma und kehrt später wieder zurück. Also: zwischen einer Person und einer Firma mehrere Objekte der Klasse Beschäftigung.

Klassendiagramme: weiterführende Konzepte

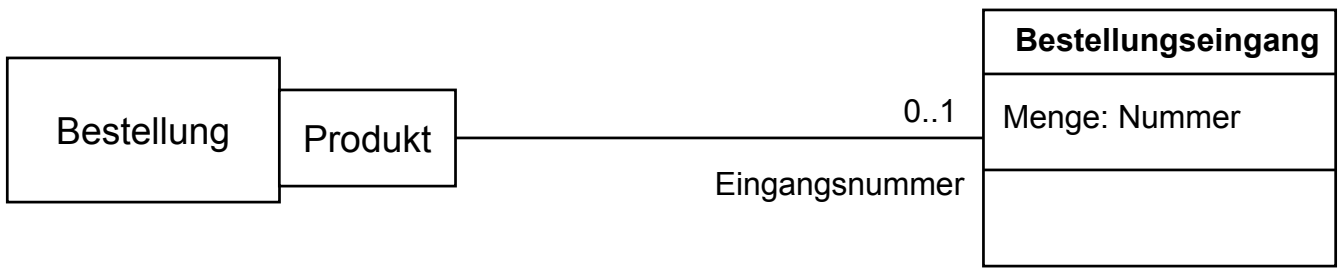
Obwohl es also sinnvolle Kombinationen von mehreren Assoziationsobjekten zwischen zwei assoziierten Objekten gibt, läßt die UML-Semantikdefinition von Assoziationsklassen dies nicht zu.

Dies bedeutet jedoch keine prinzipielle Einschränkung der Ausdruckskraft, da die Beförderung der Assoziationsklasse zu einer vollen Klasse einen Ausweg bietet.

Eine typische Anwendung von Assoziationsklassen findet sich im Entwurfsmuster *Historic Mapping* (vgl. Rolle Designer)

Klassendiagramme: weiterführende Konzepte

Qualifizierte Assoziationen erlauben es, Aussagen über die Assoziationen von Mengen von Objekten zu anderen Objekten zu machen. Das Kriterium, über das die Menge der Objekte bestimmt wird, nennt man Qualifikation.



Pro Produkt gibt es für eine Menge von Bestellungen einen Bestellungseingang. Produkt bündelt eine Menge von Bestellungen. Es qualifiziert alle Bestellungen, die zu einem Bestellungseingang gehören!

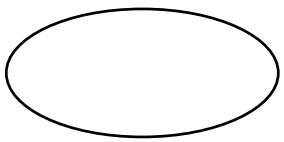
UML - Anwendungsfalldiagramme

- Ein Anwendungsfalldiagram zeigt die Beziehungen zwischen Akteuren und Anwendungsfällen, d.h. es stellt das externe Systemverhalten aus der Sicht des Anwenders dar.
- Anwendungsfälle werden auch als Szenarien oder Geschäftsprozesse bezeichnet.
- Anwendungsfälle beschreiben für den Anwender sichtbare Funktionen.
- Anwendungsfälle beschreiben die Erreichung eines für den Anwender zusammenhängenden Ziels.

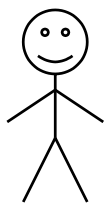
UML - Anwendungsfalldiagramme

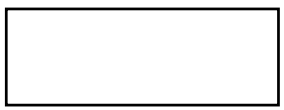
- Wichtig ist die Fokussierung auf Ziele des Anwenders und auf Interna des Systems, die mit diesen Zielen zusammenhängen. Beispiel:
  - *Ziel: Was muß der Benutzer tun, um eine individuelle Dokumentenvorlage zu erstellen?*
  - *Interna: definiere eine Vorlage, kopiere sie in das Verzeichnis für Vorlagen, wende diese Vorlage an.*
- Zielfokussierte Anwendungsfälle eignen sich gut für die Abstimmung mit dem zukünftigen Anwender.
- Auf das interne Verhalten ausgerichtete Anwendungsfälle eignen sich gut für Planungszwecke.

Notation für Anwendungsfälle

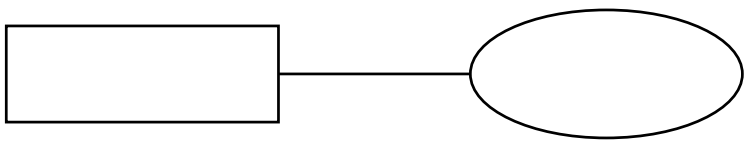


Anwendungsfall

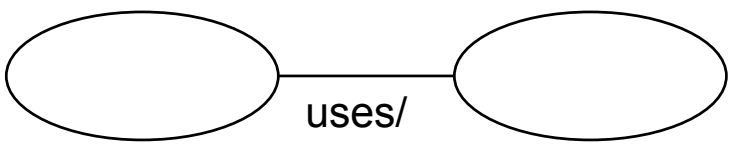




Akteur ein Akteur stellt eine Rolle dar, er gibt nicht an, wieviele Ausprägungen es gibt



Beziehung zwischen Anwendungsfall und Akteur, der Akteur führt den Anwendungsfall aus.

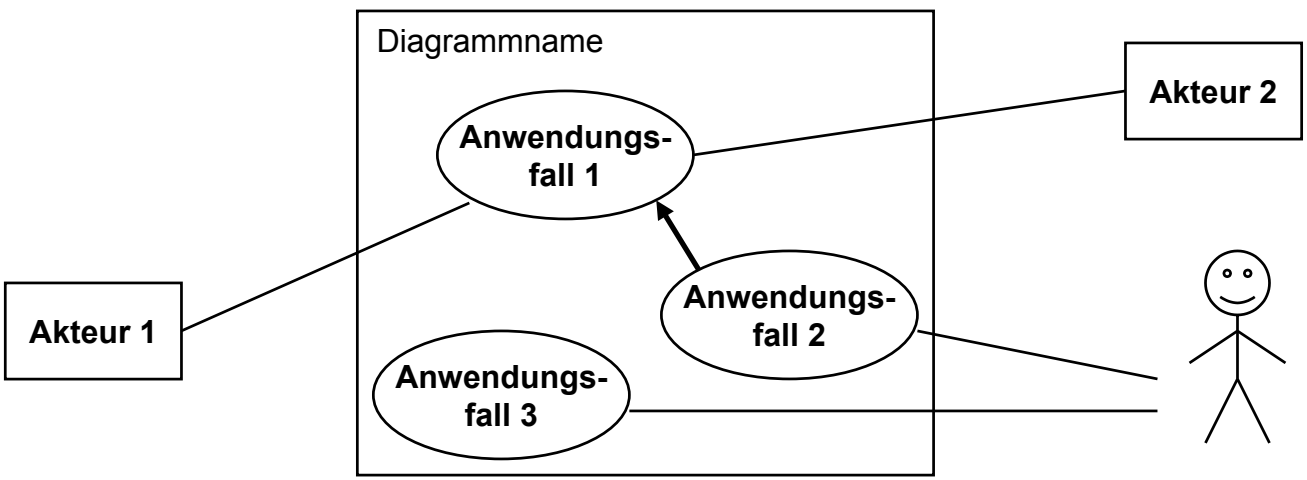


uses/
extends

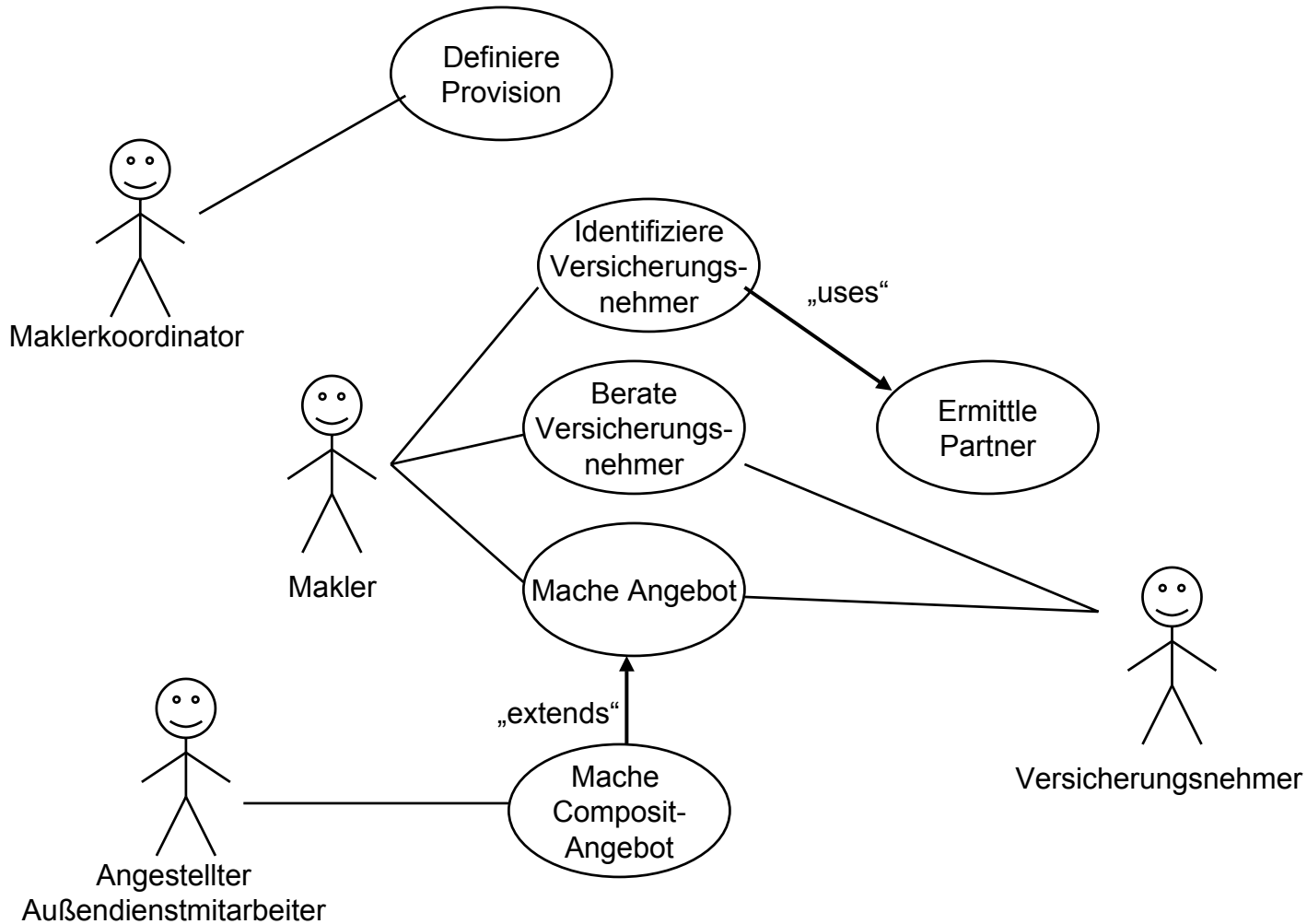
uses- oder extends-Beziehung zwischen zwei Anwendungsfällen
AF1 uses AF2

UML - Anwendungsfalldiagramme

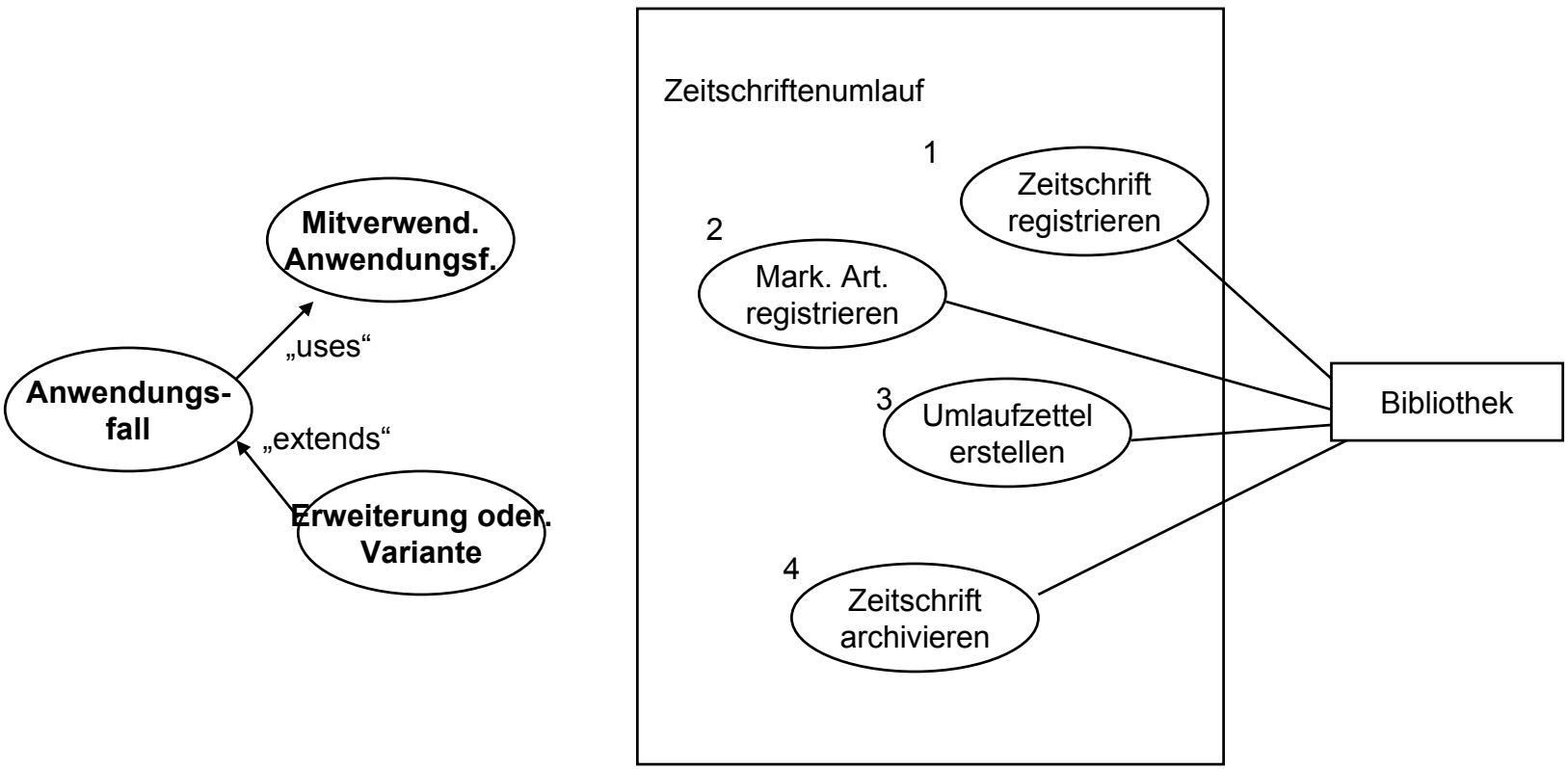
Notation für Anwendungsfälle



UML: Anwendungsfalldiagramme



UML: Anwendungsfalldiagramme



Strukturierte, textuelle Beschreibung eines Anwendungsfalls

Af.-Nr. **Name des Anwendungsfalles**

Vorbedingungen: ...
Nachbedingungen: ...
Nicht-funktionale Anforderungen: ...
Beschreibung: ..
Variationen: ...
Regeln: ...
Services: ...
Ansprechpartner: ...
Anmerkungen / offene Fragen: ...
Dialogbeispiele oder - muster: ...
Diagramme: ...

Quantifizierte Anforderungen	◢ an die Kommunikation zwischen Akteur und Anwendungsfall [Hier klicken und Text eingeben]
Antwortzeit-verhalten	◢ Wie schnell erwartet der Akteur das Ergebnis des Anwendungsfalls? Quantifizieren Sie hier das Antwortzeitverhalten, sofern Sie es nicht bereits unter den Zusatzinformationen zum Anwendungsfall getan haben. [Hier klicken und Text eingeben]
Systemverfügbarkeit	◢ Von wann bis wann erwartet der Akteur die Verfügbarkeit des Systems? [Hier klicken und Text eingeben]
Zuverlässigkeit	◢ Welche Zeit zwischen dem Auftreten von Fehlern ist für den Akteur akzeptabel? [Hier klicken und Text eingeben]
Sicherheit	◢ Welche Backup- und Recovery-Mechanismen sind für die erfolgreiche Arbeit des Akteurs unerläßlich? [Hier klicken und Text eingeben]
Schutz	◢ In welchem Umfang ist Schutz vor unberechtigten Benutzern im Rahmen des Anwendungsfalls vorzusehen? [Hier klicken und Text eingeben]

Ressourcenbelegung	◢ Welche Ressourcen wie Hauptspeicher, Plattenplatz, Leitungen etc. stehen einem Akteur zu bzw. dürfen im Rahmen des Anwendungsfalls in welchem Umfang belegt werden? [Hier klicken und Text eingeben]
Dokumentation	◢ In welchem Umfang benötigt ein Akteur Hilfe zu einem Anwendungsfall und in welcher Form (Online-Hilfe, Referenzkarte, Handbuch etc.) soll sie angeboten werden? [Hier klicken und Text eingeben]
Erweiterbarkeit	◢ Welche zukünftigen Erweiterungen des Anwendungsfalls müssen in Hinsicht auf diese Kommunikationsbeziehung berücksichtigt werden? [Hier klicken und Text eingeben]
Portabilität	◢ Auf welchen Systemen muß der Anwendungsfall ablauffähig sein? [Hier klicken und Text eingeben]
Sonstige	[Hier klicken und Text eingeben]

UML - Sequenzdiagramme (Weg-Zeit-Diagramme) und Kollaborationsdiagramme

- individuelle Objekte (in Sequenzdiagrammen und Kollaborationsdiagrammen)
- Beziehungen zwischen Objekten inklusive Nachrichtenaustausch (zeitlich geordnet) (in Sequenzdiagrammen)
- Beziehungen zwischen Objekten inklusive Nachrichtenaustausch (räumlich geordnet) (in Kollaborationsdiagrammen)

UML - Sequenzdiagramme (Weg-Zeit-Diagramme) und Kollaborationsdiagramme

Sequenzdiagramme und Kollaborationsdiagramme werden zusammenfassend als **Interaktionsdiagramme** bezeichnet.

Beide Arten von Diagrammen werden benutzt, um das Verhalten innerhalb eines Anwendungsfalls detailliert zu beschreiben. Sie liefern keine formale Beschreibung!

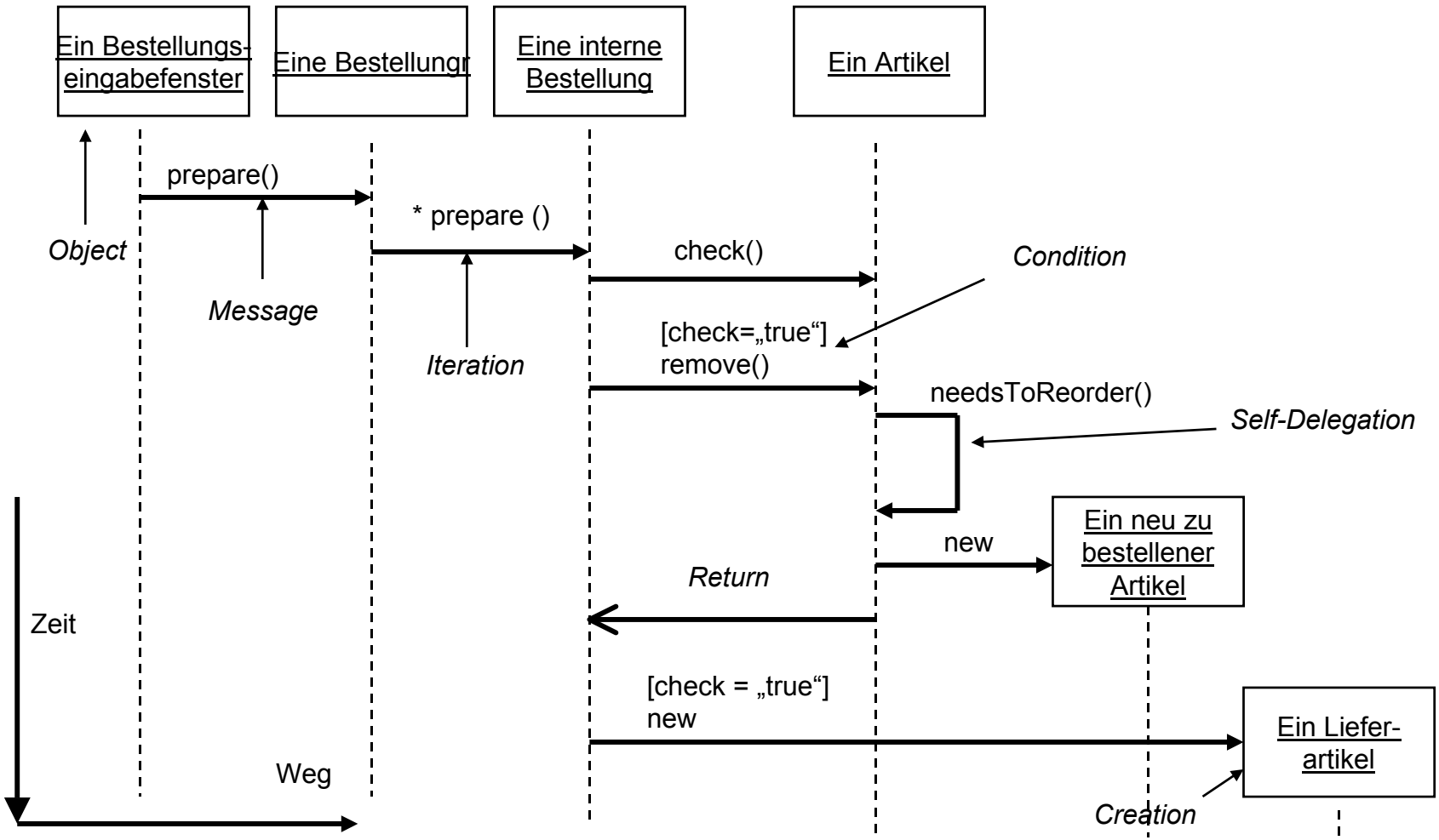
Sie beschreiben, wie Gruppen von Objekten miteinander interagieren.

Es werden Aussagen über die Anzahl der involvierten Objekte und über den Nachrichtenaustausch zwischen den Objekten gemacht.

UML - Sequenzdiagramme (Weg-Zeit-Diagramme)

- Pro Objekt: eine vertikale Lebenslinie (wie lange existiert ein Objekt).
- Nachrichten werden durch horizontale Pfeile zwischen den Lebenslinien der involvierten Objekte angezeigt (optional ergänzt um Parameter und zusätzliche Restriktionen). Zu den Restruktionen gehören:
  - Vorbedingungen für das Versenden der Nachricht
  - Iterationsangaben (wenn mehrere Objekte mit Nachrichten versorgt werden)
- Ein Objekt kann sich selbst eine Nachricht schicken (Selbstdelegation). Dies wird graphisch dargestellt durch einen Pfeil, der als Quelle und Ziel die gleiche Lebenslinie hat.
- Neben Nachrichten gibt es sogenannte *returns*. Ein return gibt an, ob und wann die Kontrolle an den Nachrichtenversender zurückkehrt (Pfeil mit offener Spitze).

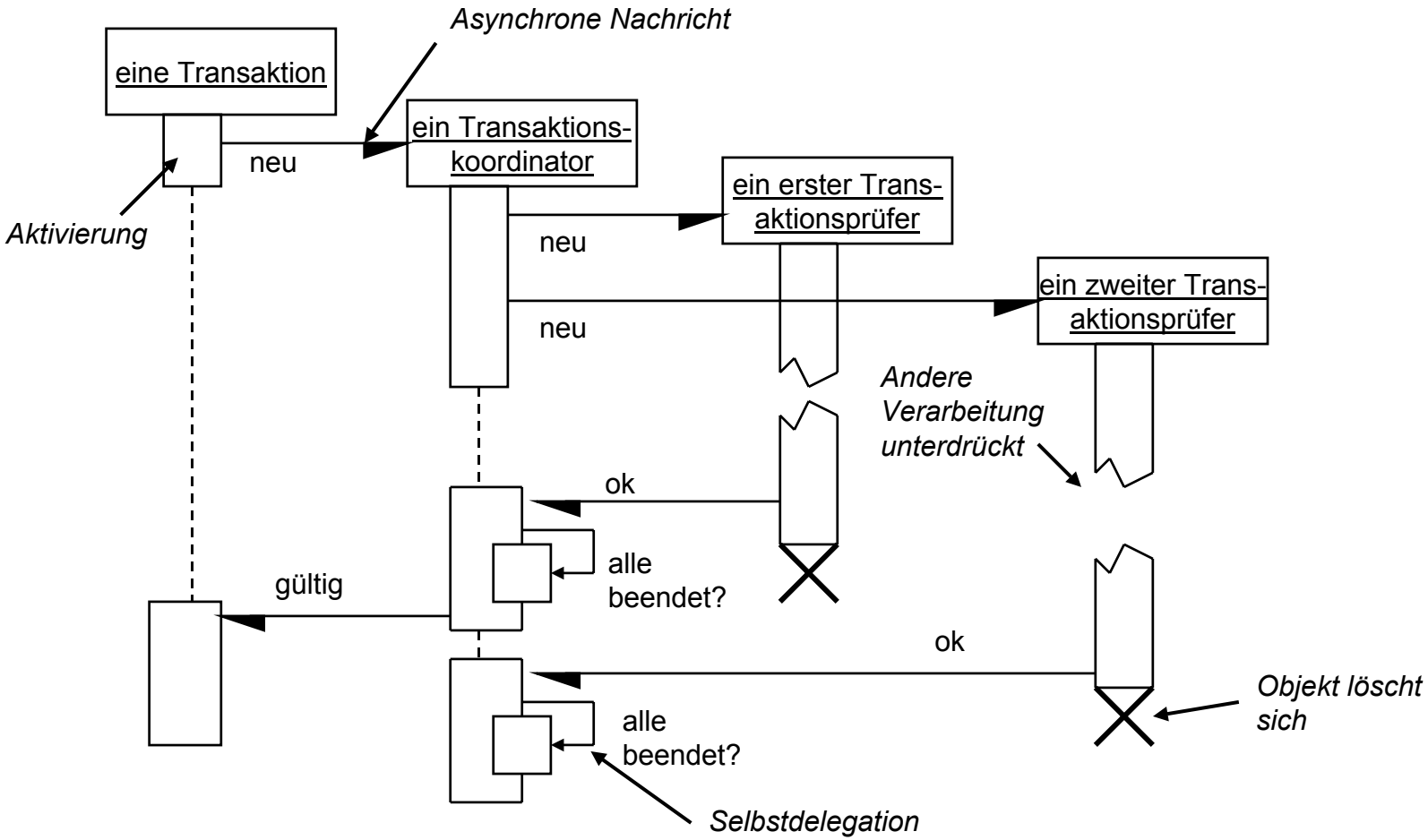
UML - Sequenzdiagramme (Weg-Zeit-Diagramme)



UML - Sequenzdiagramme
Parallele Prozesse und Aktivierungen

- Mit Hilfe von Aktivierungen kann ausgedrückt werden, wann die Operationen eines Objektes ausgeführt werden (wann also Prozesse zu diesen Methoden existieren)
- Aktivierungen werden durch Rechtecke auf den Lebenslinien der Objekte angegeben. Ihre Höhe deutet die Lebenszeit der jeweiligen Prozesse an.
- Selbstdelegation führt zu verschachtelten Aktivierungen.
- Asynchrone Operationsaufrufe werden durch Pfeile mit halber Spitze dargestellt. Bei einem asynchronen Aufruf wartet der Aufrufer nicht auf ein Ergebnis. Typischerweise eingesetzt für:
  - Erzeugen eines unabhängigen Kontrollbereichs.
  - Erzeugen eines neuen Objektes.
  - Kommunikation mit einem bereits existierenden Kontrollbereich.
  - Pro Objekt: eine vertikale Lebenslinie (wie lange existiert ein Objekt).
- Das Löschen eines Objektes wird durch ein X am Ende seiner Lebenslinie dargestellt.

Beispiel Transaktionsbehandlung - Regelfall



Beispiel Transaktionsbehandlung - Fehlerfall

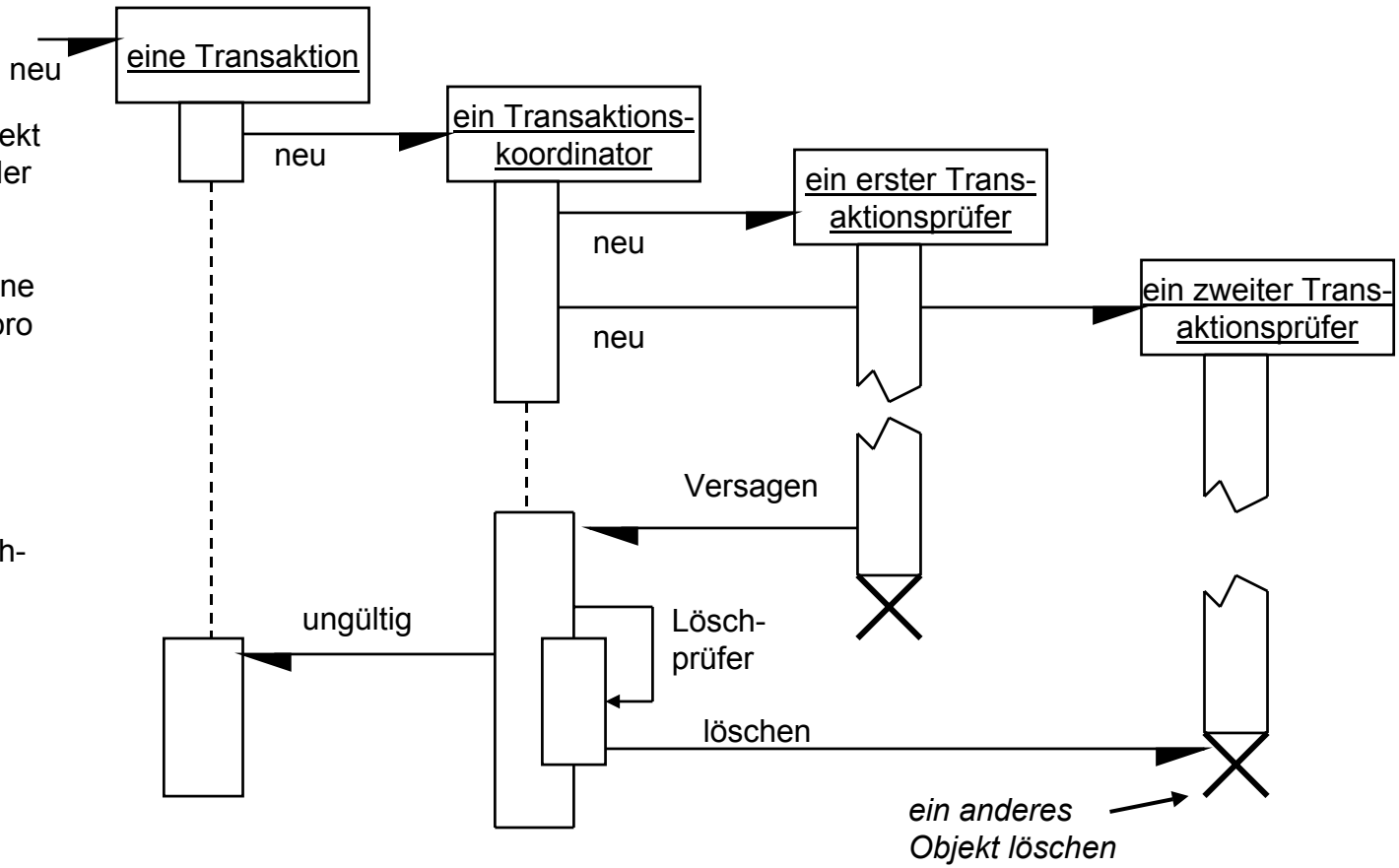
Wenn eine Transaktion erzeugt wird...

...dann erzeugt sie ein Objekt Transaktionskoordinator, der die Prüfungen überwacht.

Der Koordinator erzeugt eine Reihe von Prüfern, einen pro Prüfung. Diese Objekte arbeiten als getrennte Prozesse.

Wenn eine Prüfung nicht erfolgreich ist, dann vernichtet der Koordinator alle noch laufenden Prüfer.

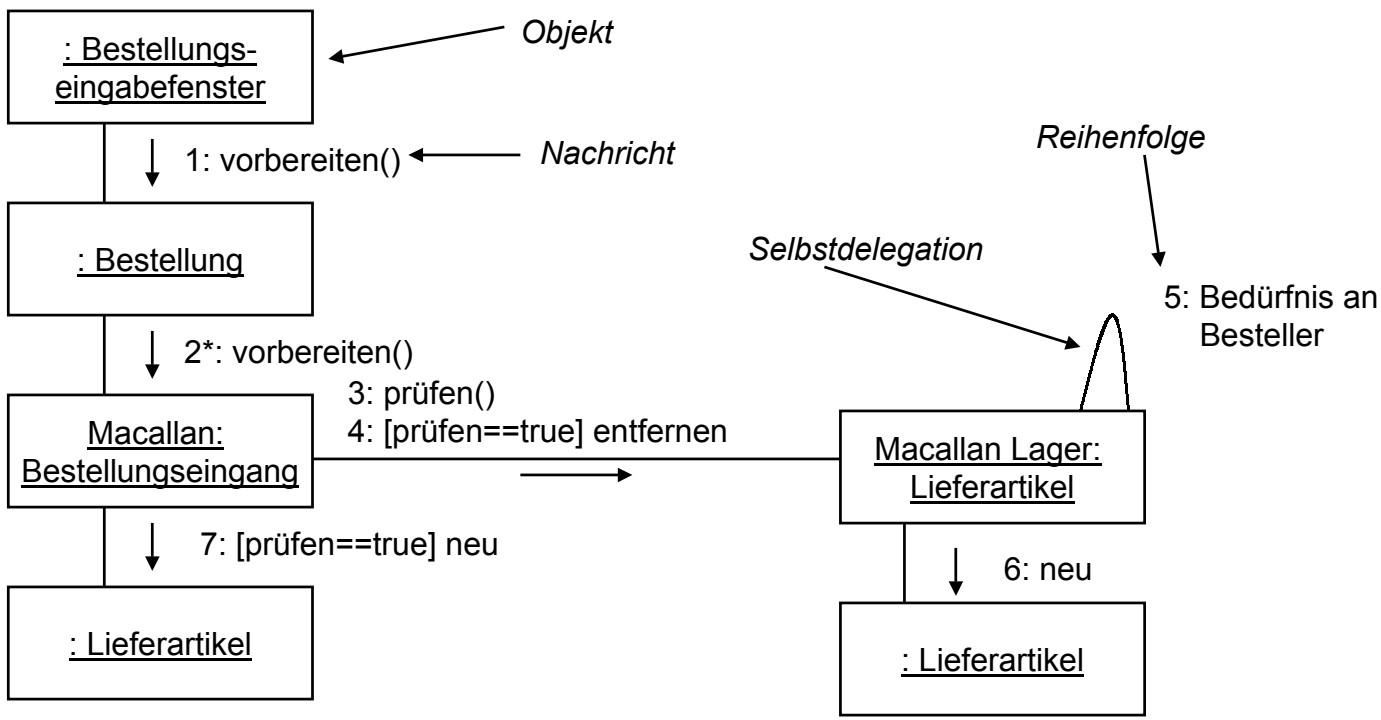
...und teilt mit, daß die Transaktion ungültig ist.



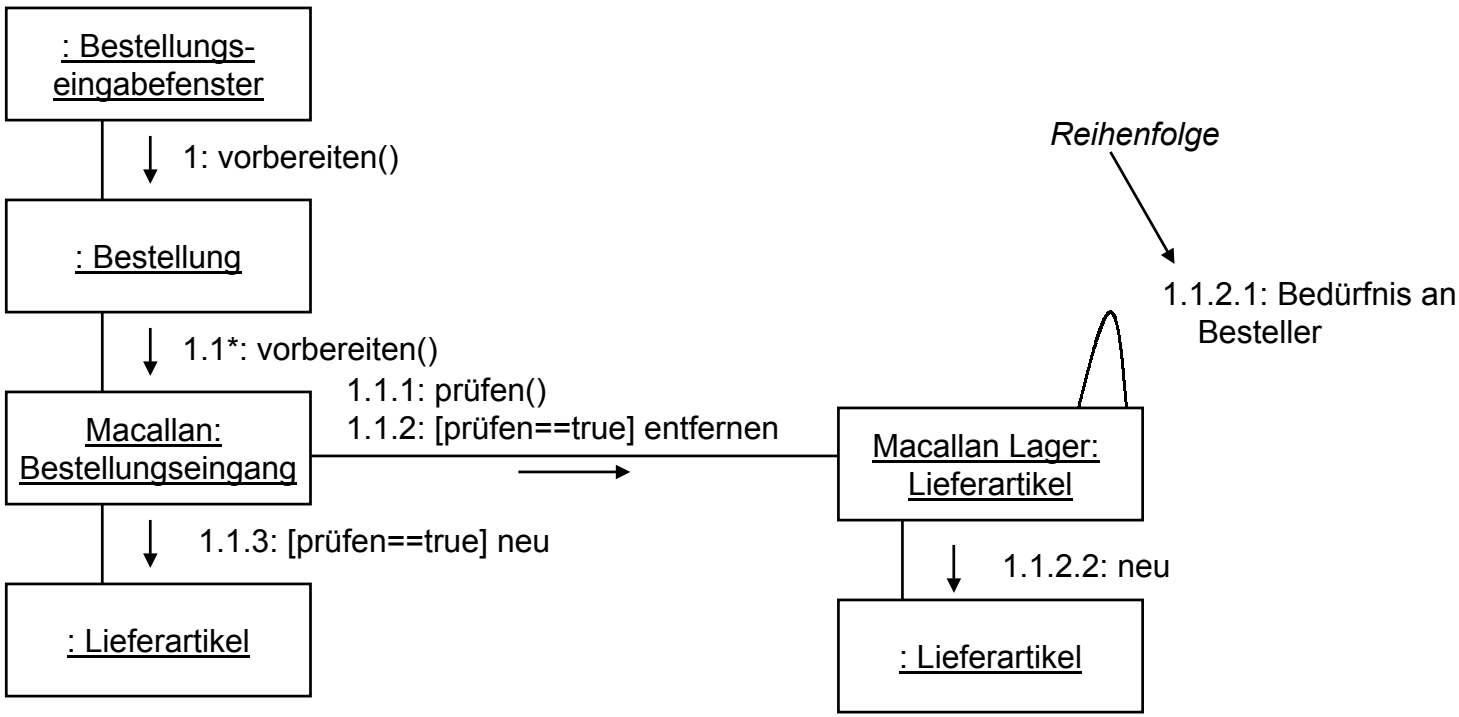
UML - Kollaborationsdiagramme

- Kollaborationsdigramme drücken den gleichen Sachverhalt aus wie Sequenzdiagramme.
- Die Reihenfolge der Nachrichten ergibt sich aus der Numerierung der Nachrichten.
- Die Anordnung der Objekte gibt die Möglichkeit, Zusammenhänge zwischen Objekten (zum Beispiel die Existenz von Beziehungen oder ihren räumlichen / örtlichen Zusammenhang) zu veranschaulichen.
- Namensschema für Objekte: Objektname: Klassenname
- In einem Namen darf entweder der Objektname oder der Doppelpunkt und der Klassenname ausgelassen werden.

UML: Kollaborationsdiagramme



Kollaborationsdiagramm mit hierarchischer Numerierung (UML)



UML
Zusammenfassung der bisherigen Diagramme

- Klassendiagramm (inkl. CRC-Karten) — Statik
- Anwendungsfalldiagramm — Dynamik (Benutzersicht)
- Interaktionsdiagramm — Dynamik (Kooperation von Objekten)
  - Sequenzdiagramm
  - Kollaborationsdiagramm

UML
Zusammenfassung der bisherigen Diagramme (2)

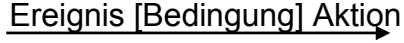
Und nun noch:

• Zustandsübergangsdiagramme einfache parallele	Dynamik einzelner Objekte
• Aktivitätsdiagramm	Abläufe / Anwendungsfälle und ihre Kooperation
• Komponentendiagramm	Systemstrukturierung

UML - Zustandsübergangsdiagramm

- Zustandsübergangsdiagramme beschreiben die erlaubten Zustandsübergänge eines Objektes und die Ereignisse, die die Übergänge auslösen.
- Zustände werden durch abgerundete Rechtecke dargestellt, Übergänge durch Pfeile.
- Syntax für Pfeilanschriften: *Ereignis [Bedingung] Aktion* wobei alle drei Teile optional sind.
- Wenn in einem Zustand interne Operationen ausgeführt werden sollen, so finden sich diese nach dem Schlüsselwort „tue/“ im unteren Teil eines Zustandes.

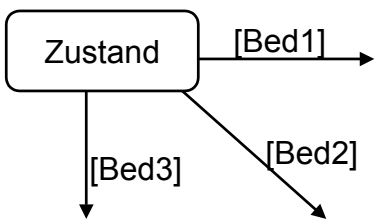
UML-Zustandsübergangsdiagramme (Notation)



Zustandsübergang, der durch das Eintreten des Ereignisses ausgelöst wird (falls die Bedingung gilt!) und dabei die Aktion durchführt

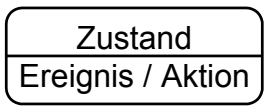
Zustand, dessen Eintreten das Auslösen der Aktivität veranlaßt.

Ein Übergang ohne Ereignis tritt ein, sobald die Aktivität des Quellzustandes beendet ist.

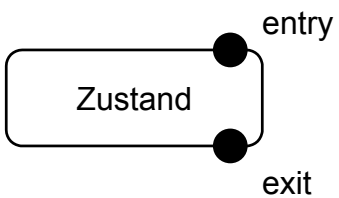


Die Bedingungen werden verwendet, um den gewünschten Zustandsübergang auszuwählen. Die Bedingungen an den Übergängen mit demgleichen Quellzustand sollten sich wechselseitig ausschließen.

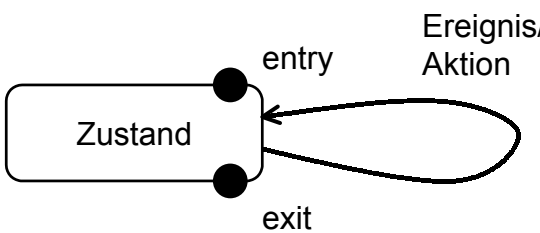
UML-Zustandsübergangsdiagramme (2) (Notation)



Der Zustand reagiert auf das Ereignis mit einer Aktion (die nicht zu einem neuen Zustand führt).

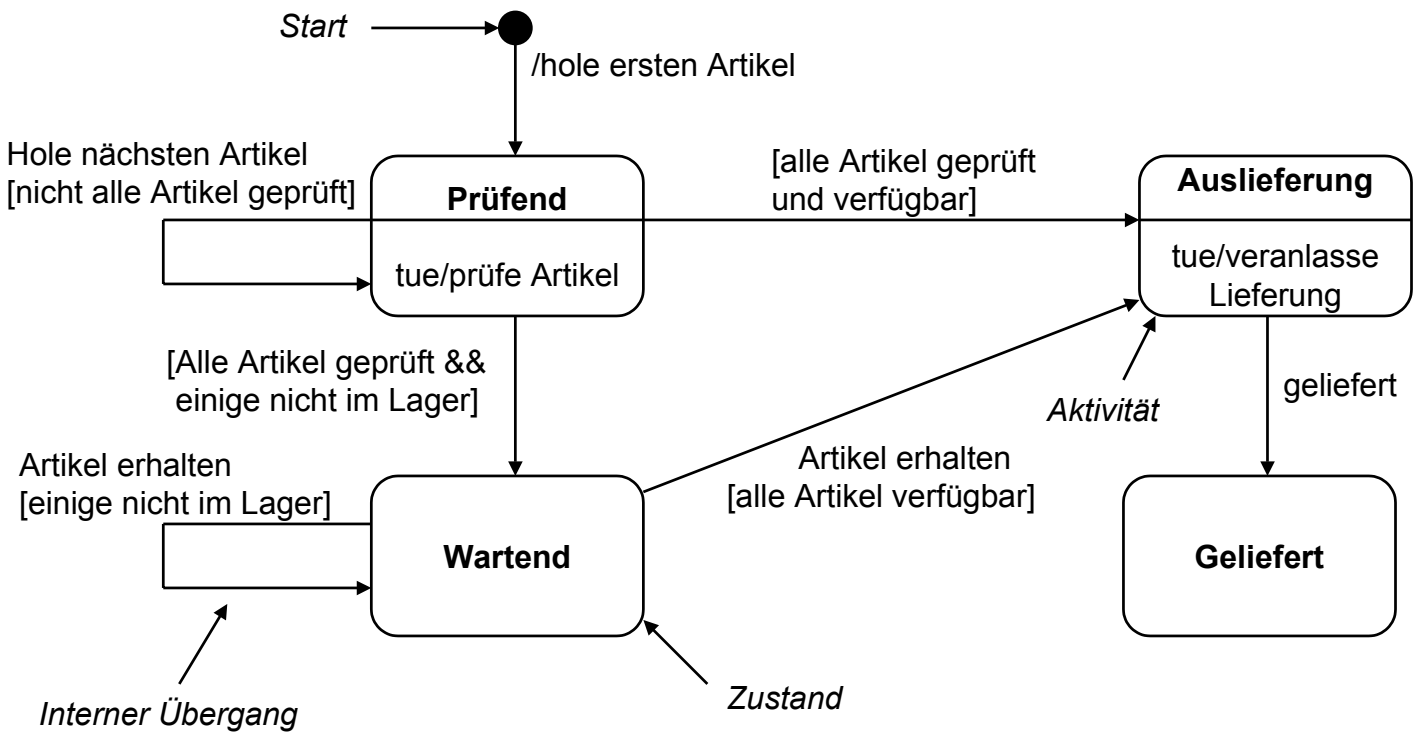


Ein Zustand kann mit einer Eingangsaktion (entry) und einer Ausgangsaktinon (exit) assoziiert sein.
(alternative Darstellung: rein textuell)



Ein Selbstübergang dieser Art führt zur Ausführung von
- exit
- Aktion
- entry

UML - Zustandsübergangsdiagramm (Beispiel Bestellung)



UML - Zustandsübergangsdiagramm

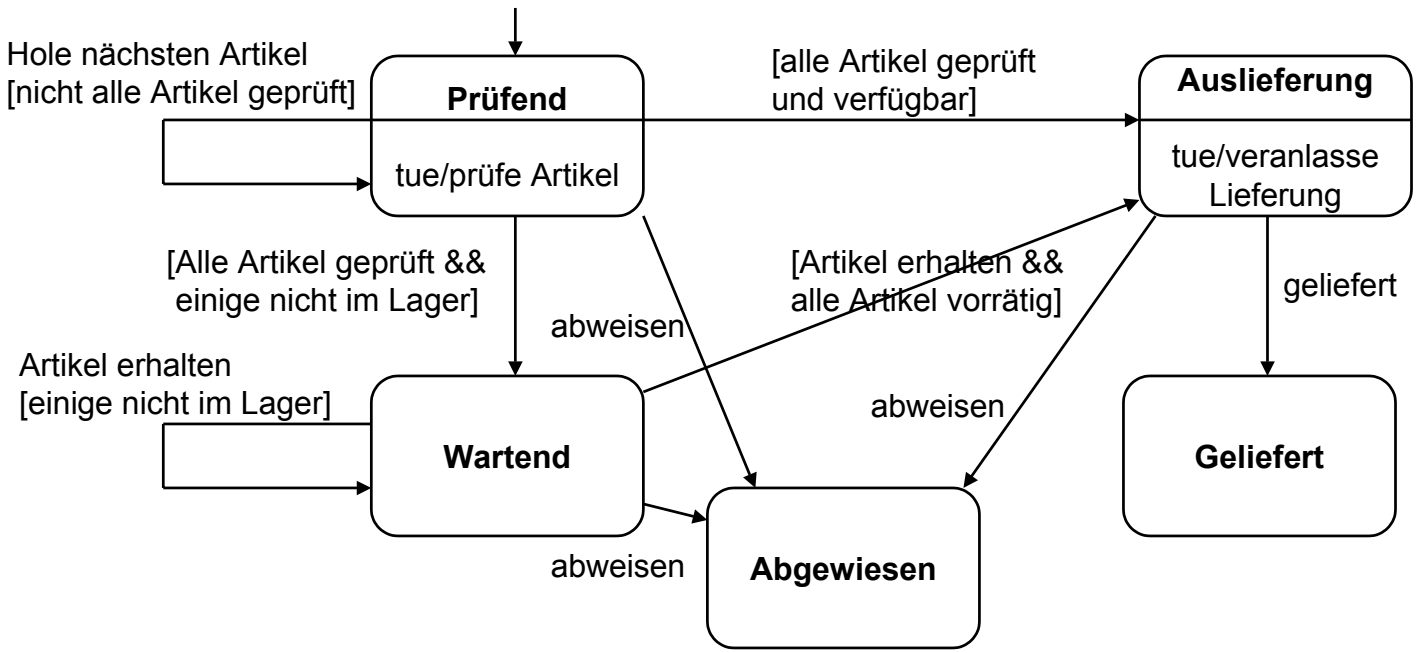
Hinweis:

- Wechselweiser Ausschluß der Bedingungen an den Übergängen, die vom Zustand „prüfend“ ausgehen:
  - nicht alle Artikel geprüft => nächster Artikel wird geprüft
  - alle Artikel geprüft und alle verfügbar => Lieferung veranlassen
  - alle Artikel geprüft, nicht alle verfügbar => wartend
- Zustände ohne Aktivität oder: Warten auf ein Ereignis

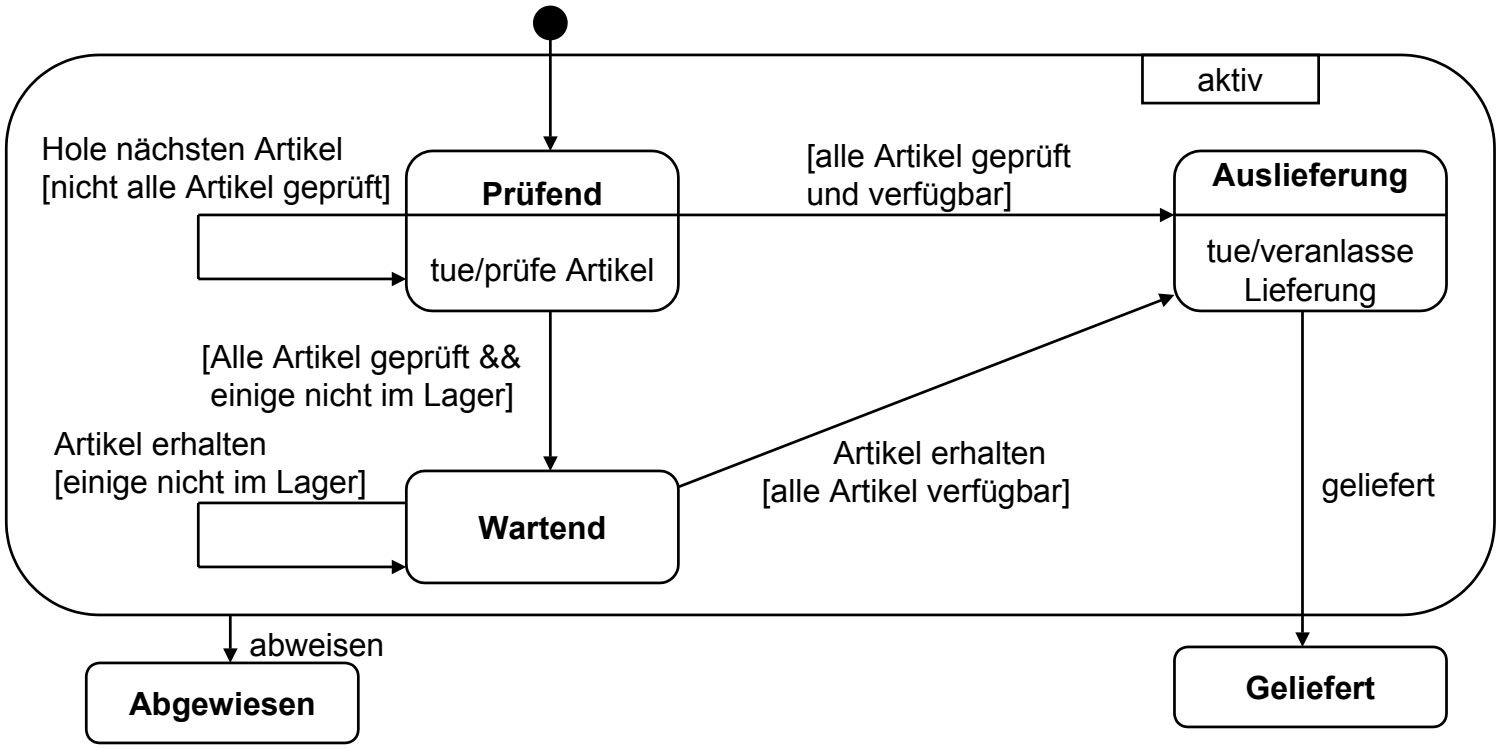
Bsp.: Im Zustand „wartend“ wird auf die Lieferung von Artikeln gewartet, sonst passiert nichts !

UML - Zustandsübergangsdiagramm (Erweiterung um „Abweisen einer Bestellung“)

Anforderung: Abweisen soll jederzeit möglich sein
=> Von allen Zuständen werden Zustandsübergänge zu „Abgewiesen“ ergänzt.

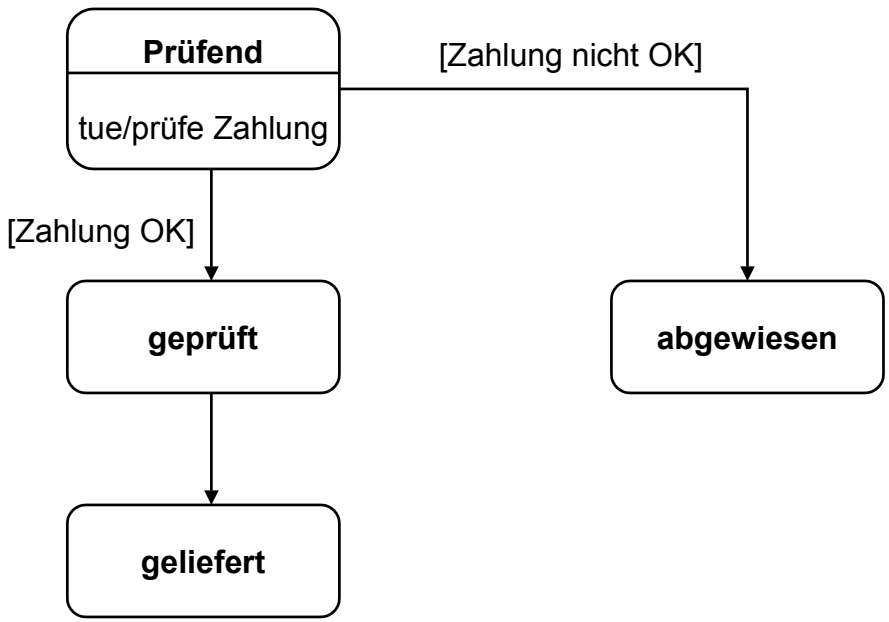


Alternative Strukturierung durch Oberzustände (Superstates)



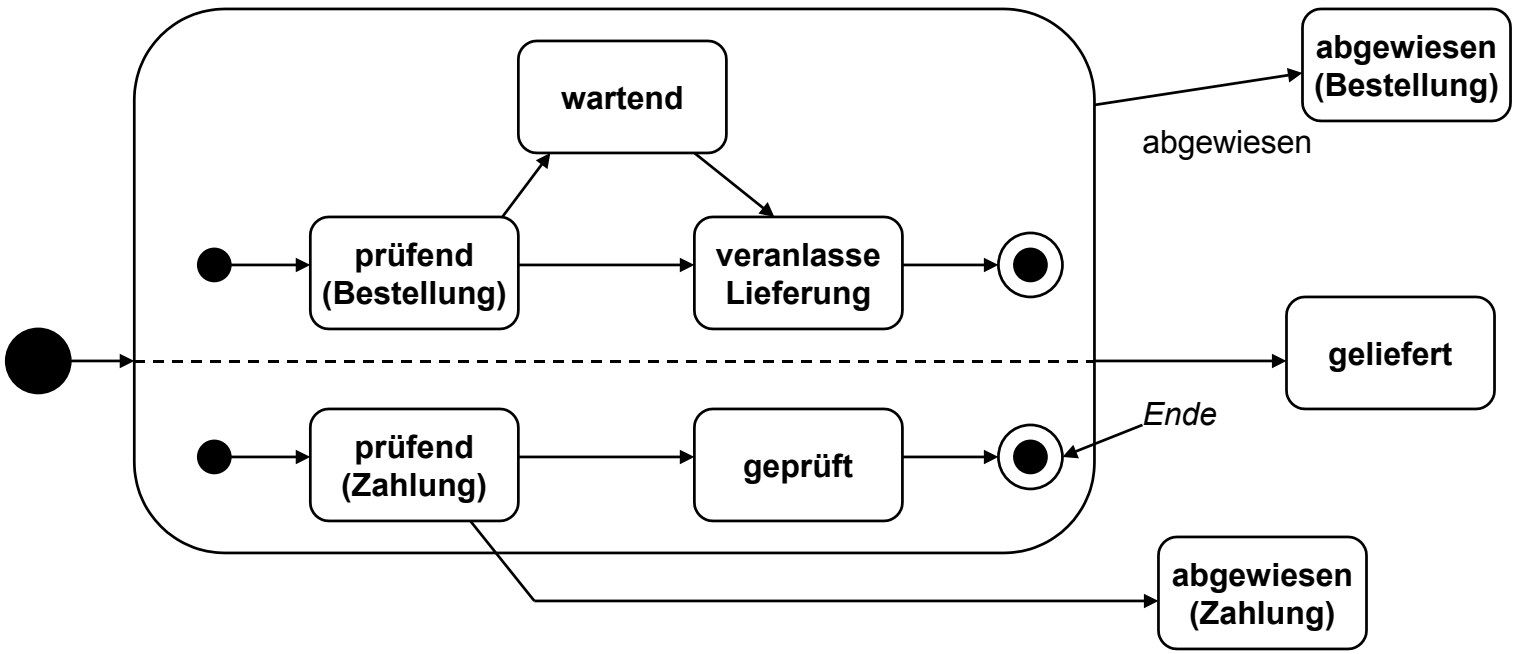
Hinweis: es bleibt auf dieser Ebene unspezifiziert, wann (von welchem Zustand kommend) der Zustand „abgewiesen“ eintritt!

Zustandsübergangsdiagramme (ein zweites Beispiel)



UML-Zustandsübergangsdiagramme

Oft werden externe Ereignisse als Auslöser verschiedener Zustandsübergangsdiagramme betrachtet. In zusammengehörigen Fällen (Bestellung, Zahlung) kommt man auf diese Weise zu parallelen Zustandsübergangsdiagrammen.



UML - Zustandsübergangsdiagramme

- Parallele Zustandsübergangsdiagramme machen Sinn, wenn ein Objekt Mengen von unabhängigen Zuständen hat.

- Komplizierte, parallele Zustandsübergangsdiagramme sind ein Indiz dafür, daß Objekte in mehrere Objekte aufgeteilt werden sollten (oben z.B. in Bestellung und Zahlung)

UML - Zustandsübergangsdiagramme (Bewertung)

- Geeignet für die Beschreibung des Verhaltens eines Objektes über mehrere Anwendungsfälle hinweg
- nicht geeignet für die Beschreibung der Interaktion zwischen Objekten
- geeignet für die Beschreibung des Verhaltens der Objekte der ***wesentlichen Klassen***! Vollständigkeitsfanatismus nützt nicht viel.
- Geeignet für Benutzungsschnittstellen und Kontrollobjekte.

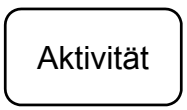
UML - Aktivitätsdiagramme

- Ein Aktivitätsdiagramm beschreibt die Reihenfolge und Abhängigkeiten von logisch zusammengehörenden Aktivitäten.
- Eine Aktivität ist dabei ein einzelner Schritt in einem Verarbeitungsablauf.
- Die Aktivitäten eines Aktivitätsdiagramms sind eindeutig Objekten zugeordnet (wichtiger Unterschied zu Datenflußdiagrammen !)
- Aktivitäten und Aktivitätsdiagramme sind entweder
  - einer Klasse
  - einer Operation (besonders hilfreich für komplexe Operationen)
  - einem Anwendungsfall zugeordnet. (besonders hilfreich bei Anwendungsfällen mit Parallelität)

UML - Aktivitätsdiagramme (Forts.)

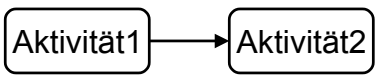
- Aktivitätsdiagramme gehen ***nicht*** auf eine der Ursprungsnotationen (Booch, Rumbaugh, Jacobson) zurück.
- Aktivitätsdiagramme sind nützlich, wenn es um geschäftprozeß-orientierte Softwaresysteme geht.
- Die Ausführungsregeln für Aktivitätsdiagramme erinnern an die Ausführungsregeln von Petrinetzen.
- Zur Übung kann das Aktivitätsdiagramm „Getränke“ in ein Petrinetz überführt werden.

UML - Aktivitätsdiagramm (Notation)



Aktivität

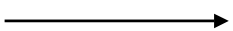
Kontrollfluß



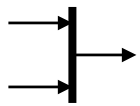
Aktivität2 wird nach Abschluß von Aktivität1 gestartet.

Verzweigunsaktivität (kann auch durch normale Aktivität dargestellt werden)

[Bedingung]

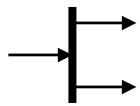


Kontrollfluß, der unter der angegebenen Bedingung gewählt wird.



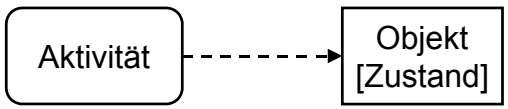
[Bedingung]

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



Aufsplitten der Kontrolle (Zulassen von Parallelität)

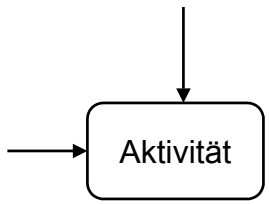
UML-Aktivitätsdiagramm (Forts.) (Notation)



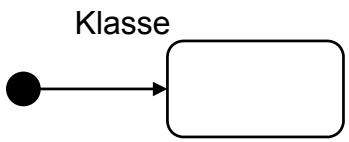
Die Ausführung der Aktivität versetzt das Objekt in den angegebenen Zustand (optionales, selten verwendetes Konstrukt in Aktivitätsdiagrammen).

Hinweis: fragwürdige Überlappung zu Zustands-übergangsdiagrammen !

Beispiel: aus dem Aktivitätsdiagramm wird eine Bestellung in den Zustand „abgewiesen“ versetzt (was bedeutet das,für das Zustandsübergangsdiagramm?)



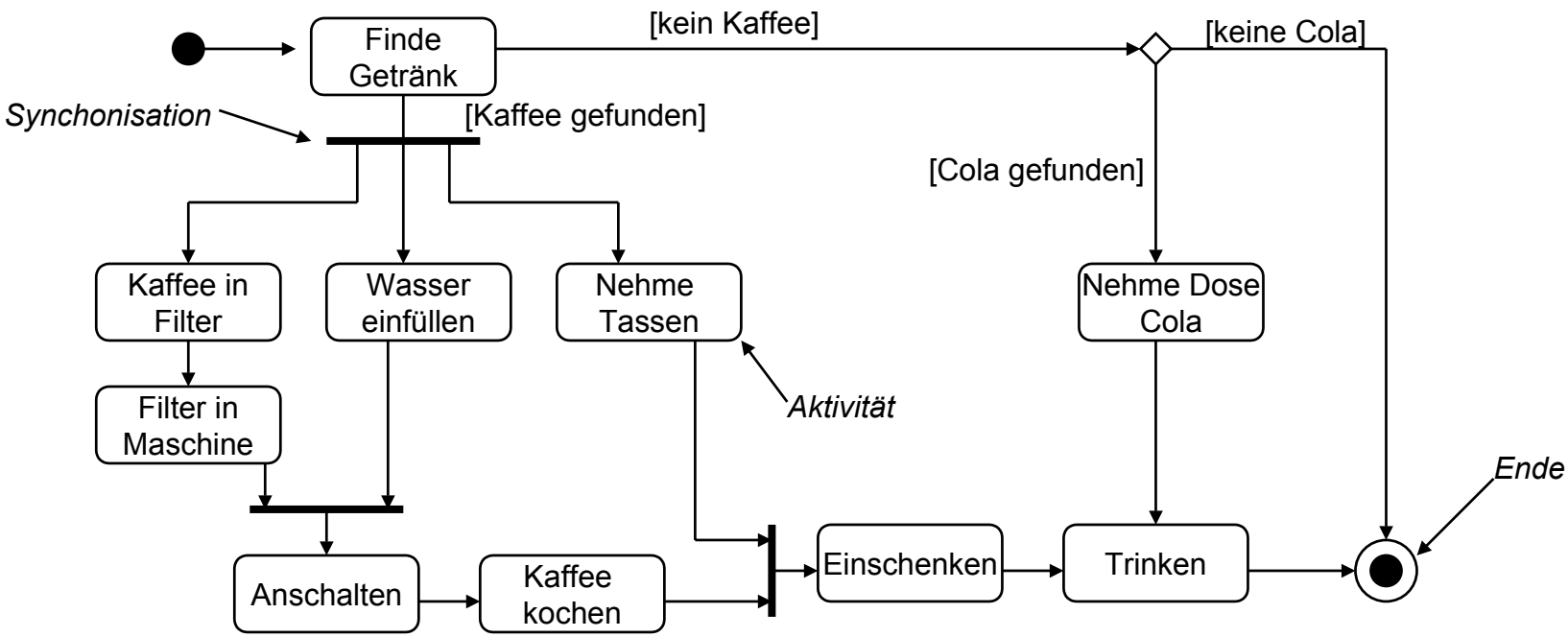
Aktivität wird durchgeführt, wenn über einen der eingehenden Kontrollflüsse die Kontrolle ankommt (OR)



Start des Ablaufs und Identifikation der betroffenen Klasse

Ende des Ablaufs (optional)

UML - Aktivitätsdiagramm (Beispiel zur Spezifikation einer Operation)



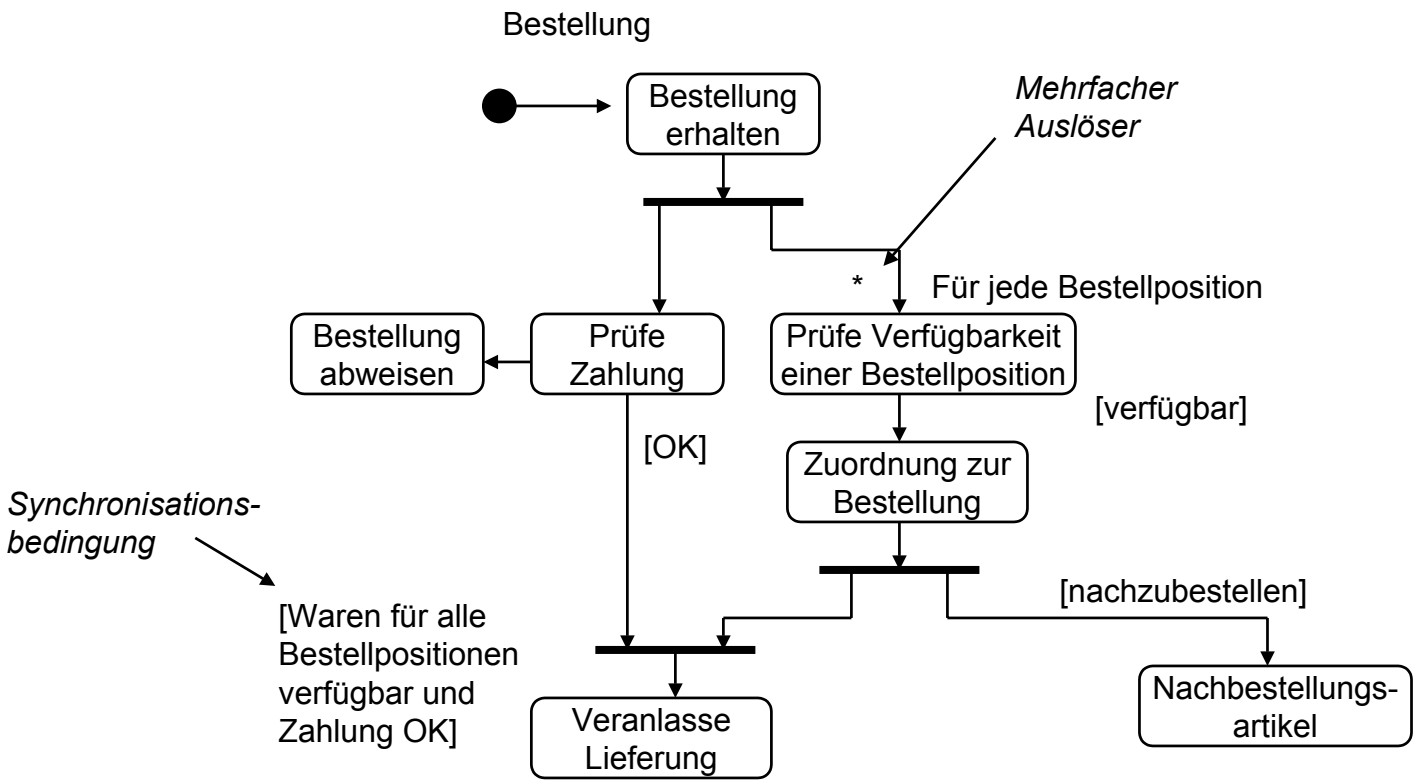
UML - Aktivitätsdiagramm (Beispiel zur Spezifikation eines Anwendungfalls)

Informale Beschreibung:

Wenn eine Bestellung eintrifft, überprüfen wir jede Bestellposition, um zu sehen, ob der Lagerbestand reicht. Falls das so ist, werden die Waren der Bestellung zugeordnet. Wenn durch diese Zuordnung der minimale Lagerbestand unterschritten wird, dann wird eine Nachbestellung veranlaßt. Währenddessen wird die Zahlung überprüft. Wenn die Zahlung OK ist und genügend Waren vorhanden sind, wird geliefert. Wenn wir für die Lieferung auf nachzubestellende Waren warten müssen, bleibt die Bestellung offen. Wenn die Zahlung nicht OK ist, wird die Bestellung abgewiesen.

UML - Aktivitätsdiagramm

(Beispiel zur Spezifikation des Anwendungsfalles „Bestelleingang“)



UML - Aktivitätsdiagramme

Hinweis:

1.) hier wäre ein ausgezeichnetes Ende nicht sinnvoll, da es mehrere Enden gibt

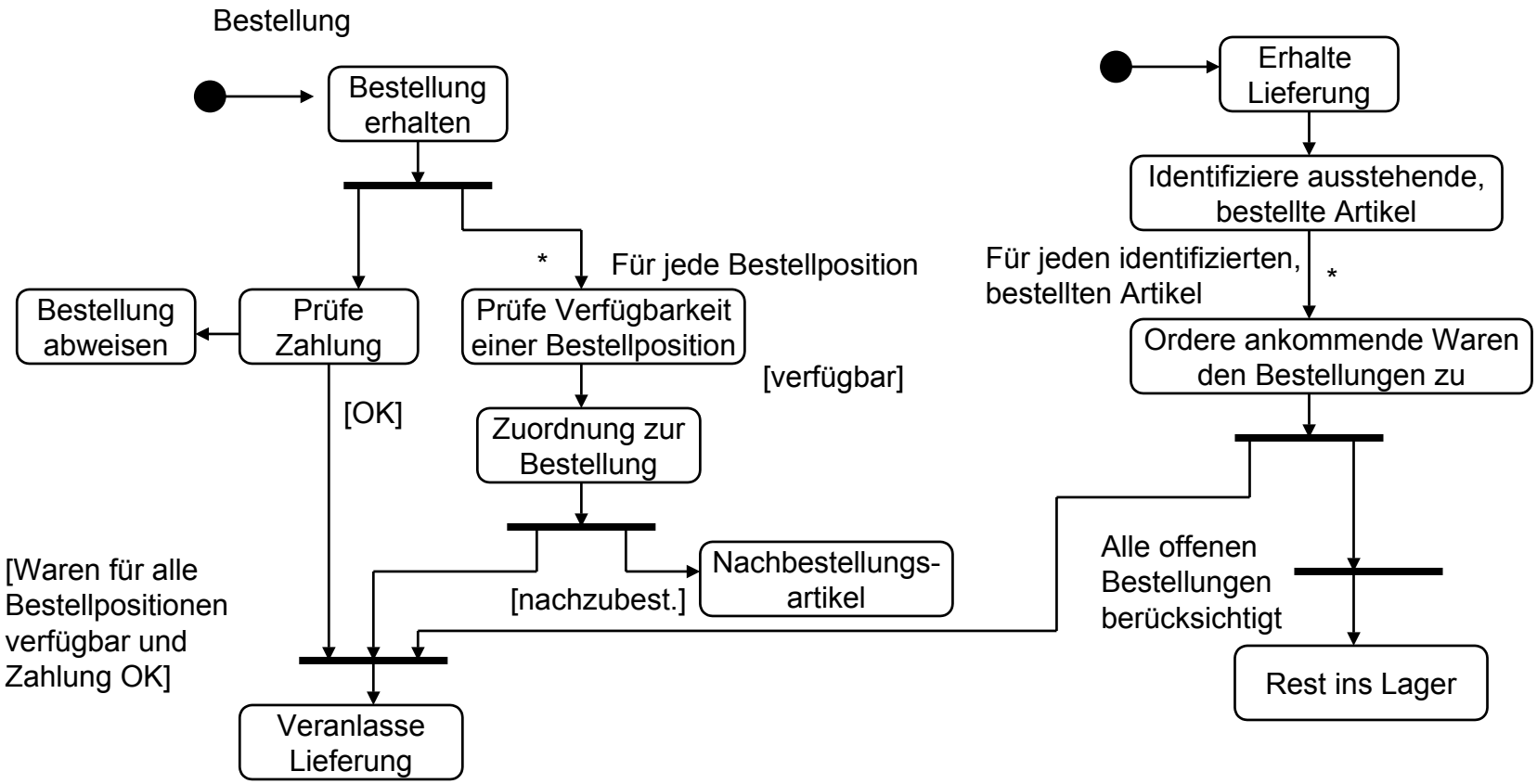
2.) aus dem linken Zweig kann der rechte nicht angehalten werden! (bei strikter Sequentialiseirung ist dieses Problem vermeidbar, allerdings nur auf Kosten der Parallelität)

3.) Wenn die Bedingung am Ende von „prüfe Verfügbarkeit einer Bestellposition“ nicht erfüllt ist, stoppt der Ablauf! Im günstigsten Fall wird eine ankommende Lieferung die weitere Auslieferung veranlassen (Frage der Ausführungssemantik!)

Besser wäre eine Ergänzung zu folgendem Aktivitätsdiagramm:

UML - Aktivitätsdiagramm

(Beispiel zur Spezifikation des Anwendungsfalles „Bestelleingang“)



UML - Aktivitätsdiagramm (Bestelleingang und Nachlieferung)

- Homogene Spezifikation, aber
  - was passiert bei vielen offenen Bestellungen?
  - wie passiert die Kommunikation zwischen einem Ablauf des Typs Nachlieferung und vielen Nachbestellungen?

Allgemein: Wie legt ein Ablaufmodell (ausgedrückt als Aktivitätsdiagramm) die Kommunikation auf der Ebene einzelner Abläufe fest?

Zur Erinnerung: OOA dient der Spezifikation; es müssen keine vollständigen Festlegungen der Implementierungen entstehen. Es ist deshalb durchaus nützlich, mit Beschreibungen zu arbeiten, die nicht alles festlegen, solange man sich der Lücken bewußt ist.

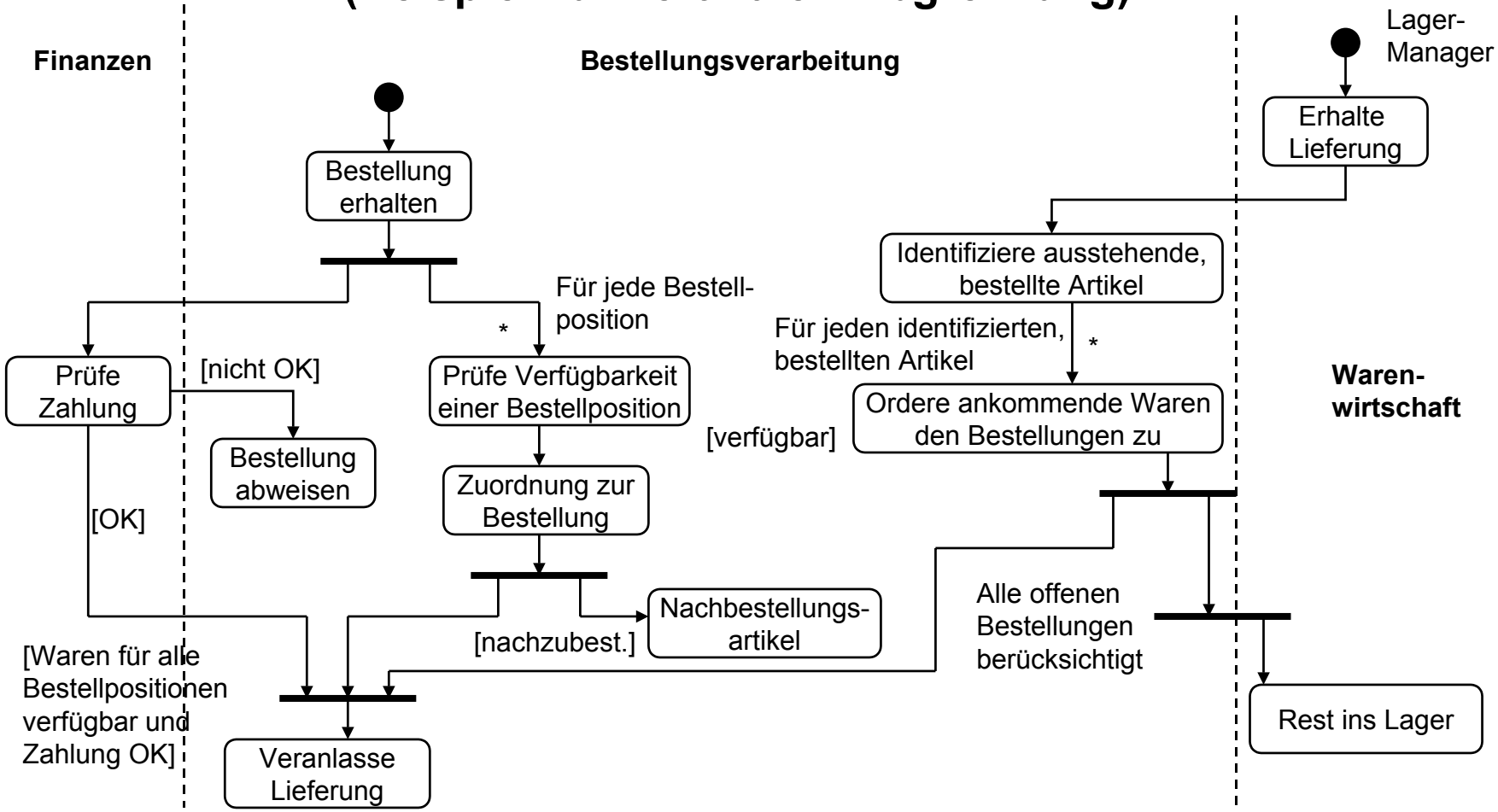
UML - Aktivitätsdiagramm (Strukturierung)

- Das zusammengefügte Beispiel (Bestelleingang und Nachlieferung) ist nicht eindeutig einer Klasse, einer Operation oder einem Anwendungsfall zuzuordnen (weil es durch Komposition zweier getrennter Abläufe entstanden ist).

- Da man einerseits die Wechselwirkungen beschreiben möchte (deshalb die Komposition) und andererseits die klare Zuordnung nicht aufgeben möchte, gibt es das Strukturierungsmittel der vertikalen Abgrenzung.

UML-Aktivitätsdiagramm

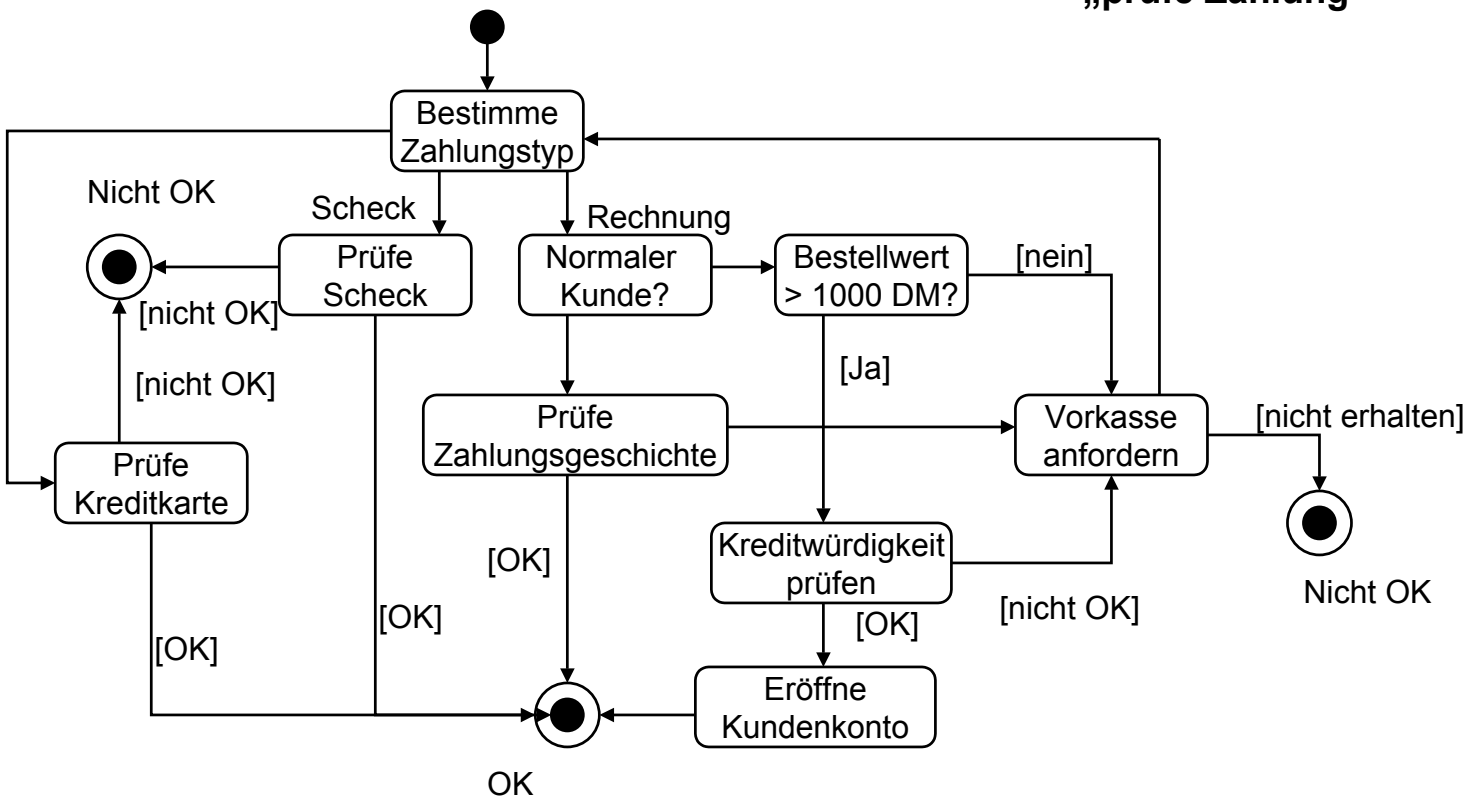
(Beispiel zur vertikalen Abgrenzung)



UML-Aktivitätsdiagramm
(Strukturierung)

Aktivitäten können verfeinert werden (passende Abstraktionsebenen für unterschiedliche Zwecke)

Beispiel: Verfeinerung der Aktivität „prüfe Zahlung“



UML - Aktivitätsdiagramme (Bewertung)

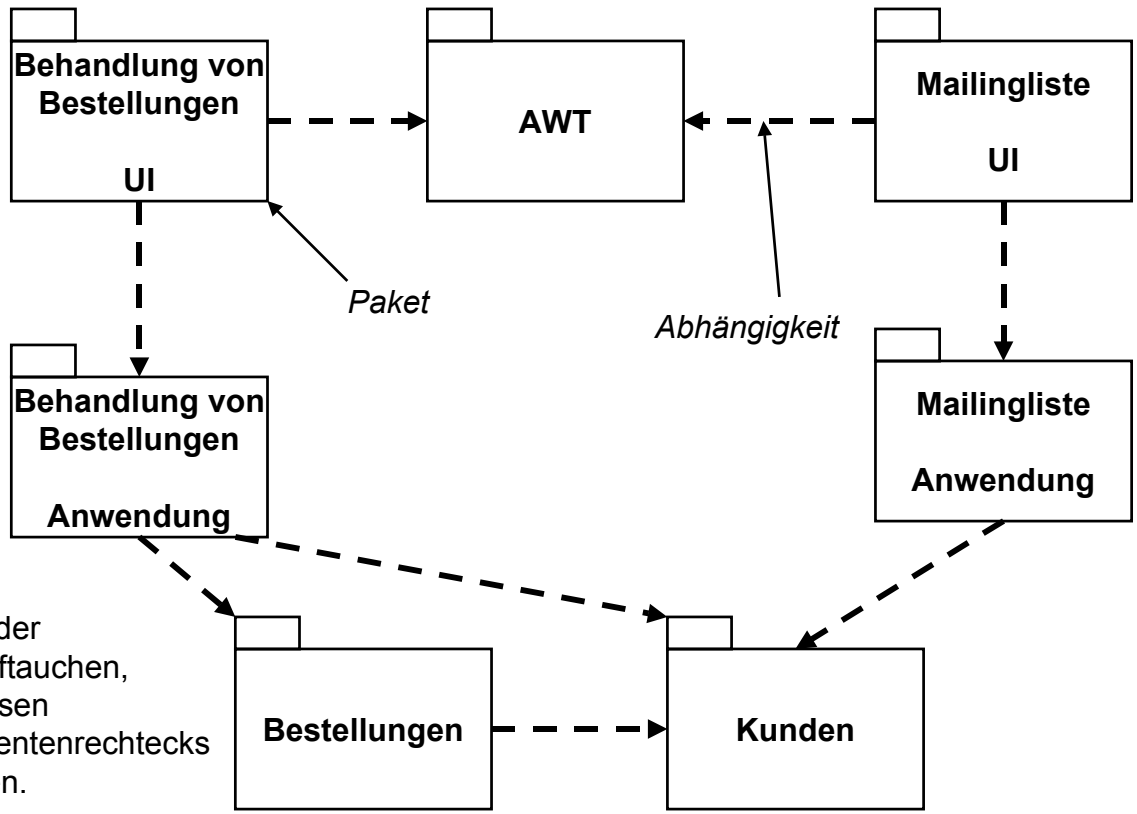
- geeignet für die Modellierung von Geschäftsprozessen über die Grenzen von Anwendungsfällen hinweg.
- geeignet für die detaillierte Analyse von Anwendungsfällen.
- geeignet für die Modellierung von parallelen Software-Systemen

- nicht geeignet für die Beschreibung der Interaktion von Objekten
- nicht geeignet für die Zustandsübergänge eines einzelnen Objektes

UML - Komponentendiagramme - Zwischen OOA und OOD

- In UML heißt eine Gruppe von „zusammengehörenden“ Klassen *package* (Komponente).
- Die Zusammenfassung von Klassen zu Komponenten dient zur Strukturierung von Klassendiagrammen. Oft ist diese Strukturierung ein erster Entwurfsschritt.
- Als Hilfe dient die Ermittlung von Abhängigkeiten.
- Zwei Komponenten sind abhängig, wenn Änderungen an der Schnittstelle der einen der anderen bekannt gemacht werden müssen.
- Zyklische Abhängigkeiten sind in UML nicht verboten, sollten aber dringend aufgelöst werden.

Komponentendiagramme - Zwischen OOA und OO



Im kleinen Rechteck links oben, kann dann der Komponentenname auftauchen, wenn die internen Klassen innerhalb des Komponentenrechtecks angezeigt werden sollen.

UML- Methodik

1 Von den Anforderungen zu Klassendiagrammen

- und dann über Use Case Diagrammen zu Interaktionsdiagrammen zu Zustandsübergangsdiagrammen
- parallel Übergang zu OOD mit Hilfe von Komponentendiagrammen
- Aktivitätsdiagramme nahezu beliebig zur Ablaufveranschaulichung

oder

2 Von den Anforderungen zu Use Case Diagrammen

- und dann Aktivitätsdiagrammen zur Detailbeschreibung von Use Cases
- und dann über Klassendiagramme zu Interaktionsdigrammen
- und dann zu Zustandsübergangsdiagrammen
- parallel Übergang zu OOD mit Hilfe von Komponentendiagrammen

Beide Ansätze nur als grobe Richtlinien, nicht als Dogmen!