Einführung in die Unified Modeling Language

SS 2017

Rainer Schmidberger

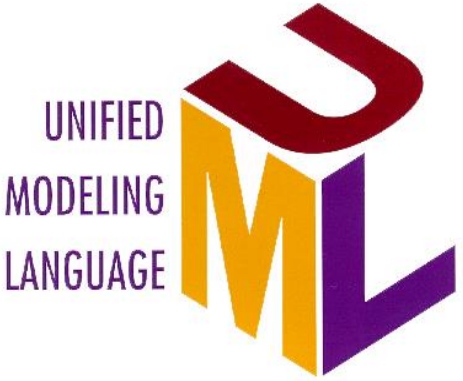
rainer.schmidberger@HFT-Stuttgart.de

Überblick

- Was ist die UML?
- UML und OMG
- Motivation
- Historie
- Übersicht über die Diagrammtypen
- Die Diagramme im Einzelnen

Was ist UML?

- UML - Unified Modeling Language
- Standardisierte Modellierungssprache
  - ⇨ Grafische Sprache und Notation zur Beschreibung von Softwaresystemen
- Einsatz für…
  - ⇨ Spezifikation
  - ⇨ Visualisierung
  - ⇨ Konstruktion
  - ⇨ Dokumentation
- Releasestand
  - ⇨ 1.x: 1.5
  - ⇨ 2.x: 2.3 (5/2010). Version 2.4 in Arbeit
- Keine Methode
  - ⇨ Beinhaltet keine Vorgehensweise für Softwareentwicklungsprozess
  - ⇨ Kann Basis für verschiedenste Methoden/Vorgehensmodelle sein.



Quelle: Gerhard Wanner

UML und OMG

- UML lag und liegt der OMG zur Standardisierung vor
- OMG – Object Management Group
  - Gegründet 1989 in Framingham USA
  - http://www.omg.org
  - Aktuell ca. 350 Mitgliedsfirmen
  - Ziele:
    - Erstellung von Industrienormen und Standardisierung
    - Wiederverwendbarkeit von Softwaremodulen
    - Anwendungen in heterogenen Rechnerumgebungen
  - Arbeitsergebnisse
    - XML
    - CORBA
    - IIOP
    - UML
    - MDA.

Quelle: Gerhard Wanner

Motivation

- Warum mit UML modellieren?
  - ⇨ Unabdingbar für Entwicklung komplexer Softwaresysteme
  - ⇨ Modell ermöglicht Vorschau/Visualisierung des realen System
  - ⇨ Vorlage zur Konstruktion der Software
  - ⇨ Besseres Verständnis eines komplexen Systems
  - ⇨ Beschreibung des Verhaltens und der Struktur des Systems
  - ⇨ Konzentration zu einem Zeitpunkt auf bestimmten Aspekt möglich
  - ⇨ Gute Kommunikationsebene zwischen Kunde und Hersteller durch einheitliche Sprache
  - ⇨ Frühzeitige Qualitätssicherung und Problemerkennung
  - ⇨ Dokumentation der getroffenen Entscheidungen.

➔UML stellt einheitliche und umfassende Notation zur Verfügung

Quelle: Gerhard Wanner

UML (1)

- The Unified Modeling Language is **a visual language for specifying, constructing, and documenting the artifacts of systems**. It is a general-purpose modeling language [...] that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms [www.uml.org].
- Die Unified Modeling Language (UML) ist eine von der Object Management Group (OMG) entwickelte und standardisierte Sprache für die Modellierung von Software und anderen Systemen [de.wikipedia.org].
- Aktuelle Version: 2.3, faktisch wird aber 2.0 genutzt

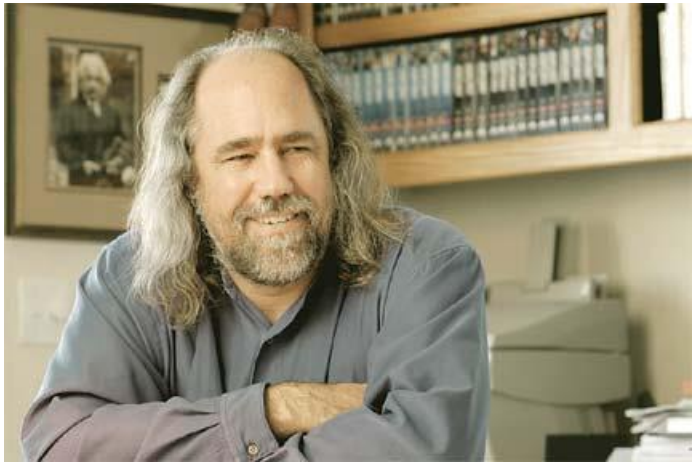
UML (2)

- Wurde entscheidend durch **Grady Booch, Ivar Jacobson und James Rumbaugh** geprägt.
- Hat die vor 1996 existierenden verschiedenen Modellierungsmethoden (der genannten drei, sowie anderer) abgelöst
- Wird zur **Prozessmodellierung, Analyse, Spezifikation und zum Systementwurf** verwendet
- Es sind sehr viele gute Werkzeuge zur UML auf dem Markt
- Bietet sehr konkreten Übergang zur Implementierung. Code-Generierung ist aus den Modellen heraus in vielen Teilen möglich
- In Zukunft: MDA, MDD und Executable UML?

Die drei „Amigos“

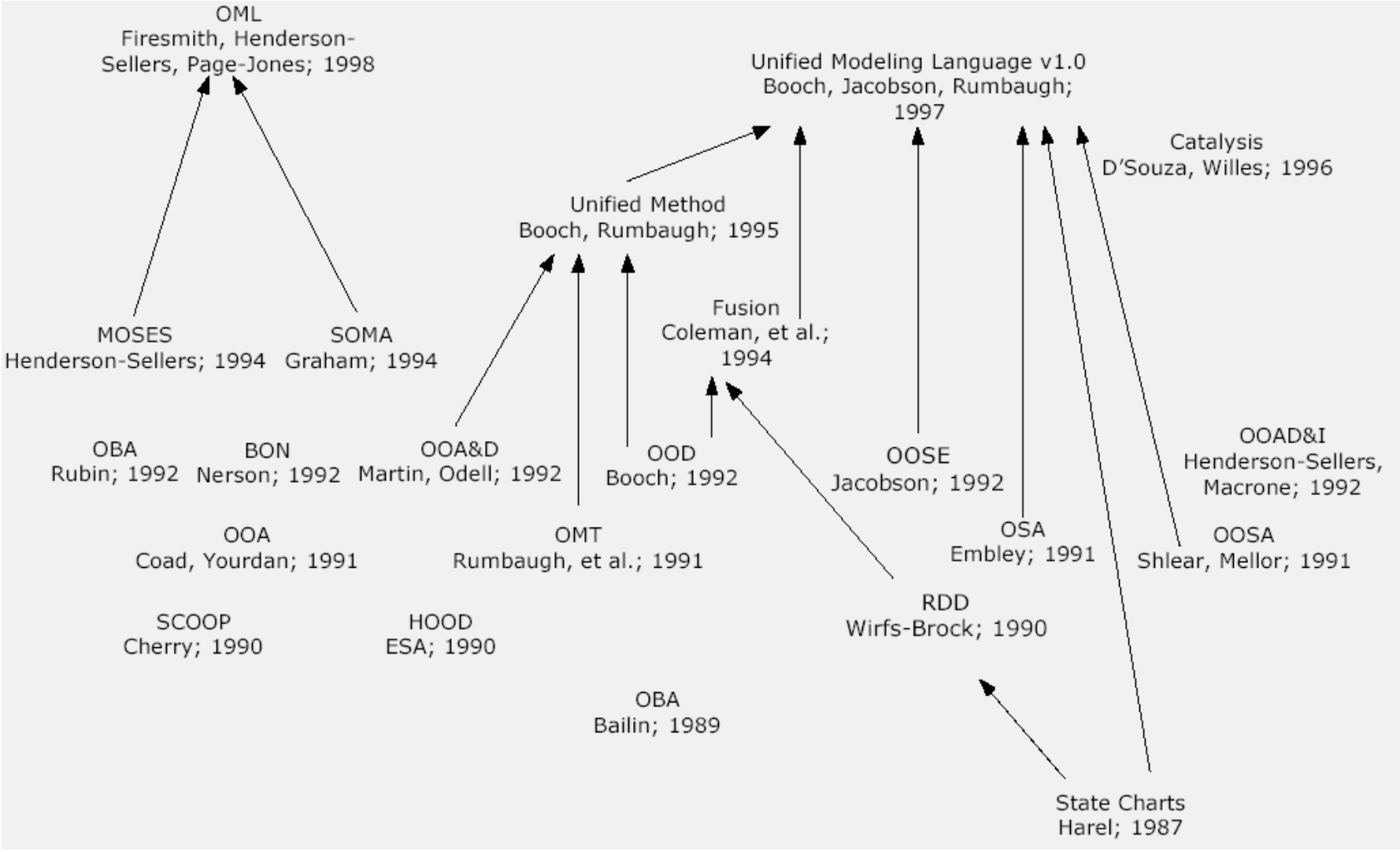
Ivar Jacobson

Grady Booch



James Rumbaugh

Geschichte der UML



Quelle: Mario Jeckle, 2003

Folie 9

„Entstehung“ der UML

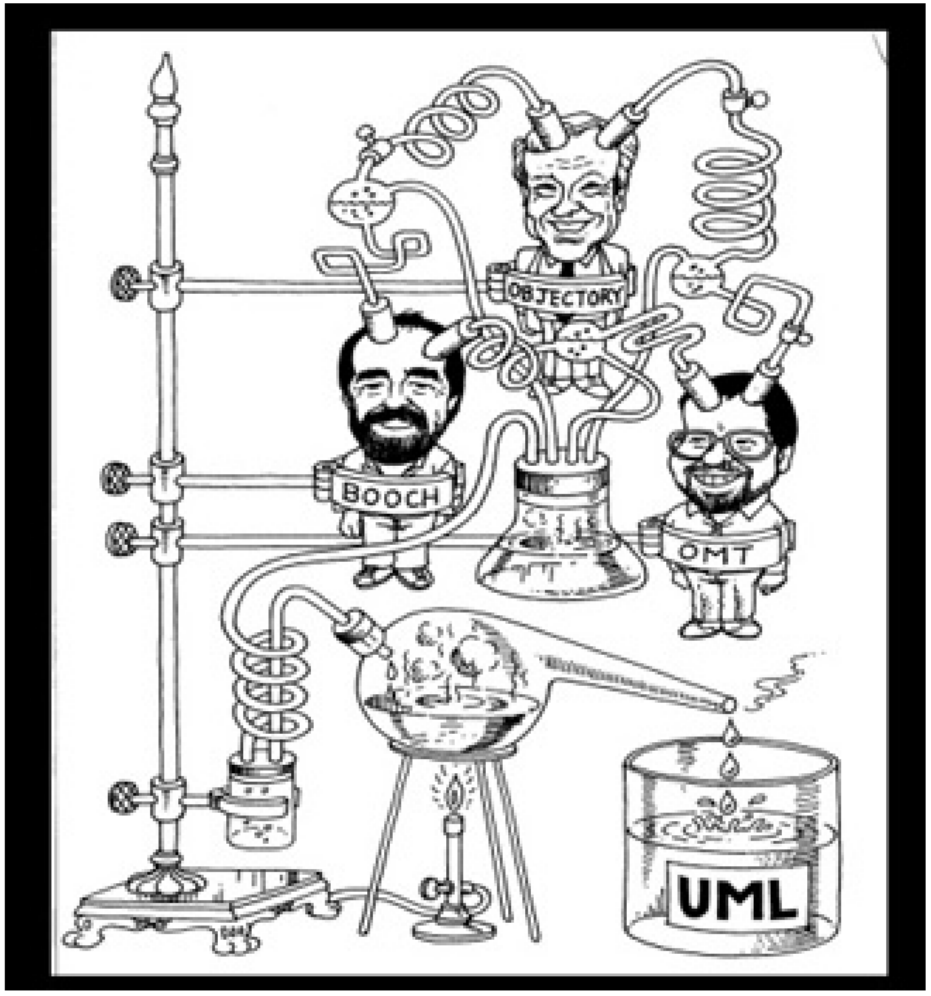


Figure 1: How it all began

Quelle: www.ibm.com

Folie 10

Kurzübersicht über die UML-Diagramme

Diagramm Übersicht

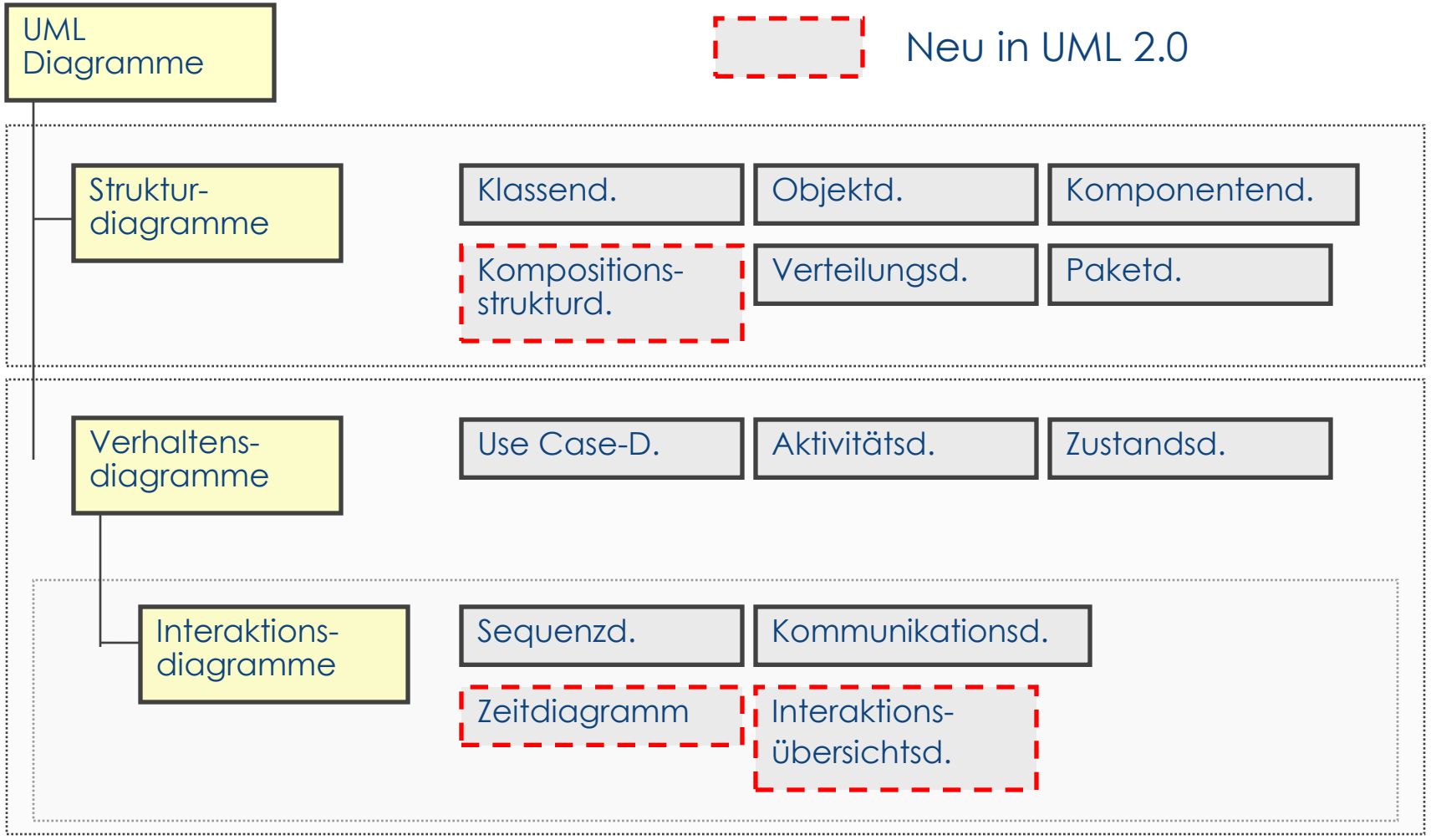
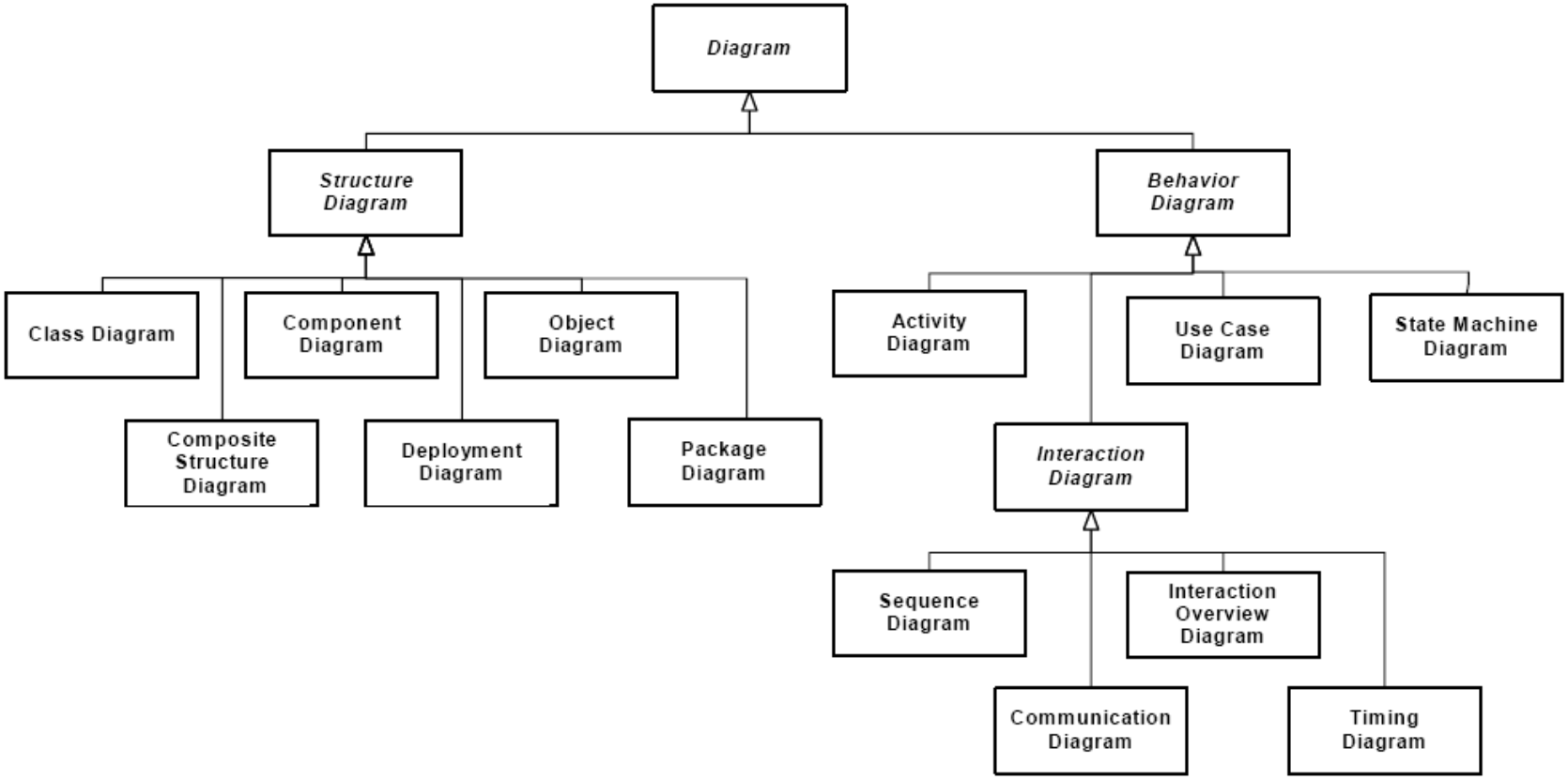
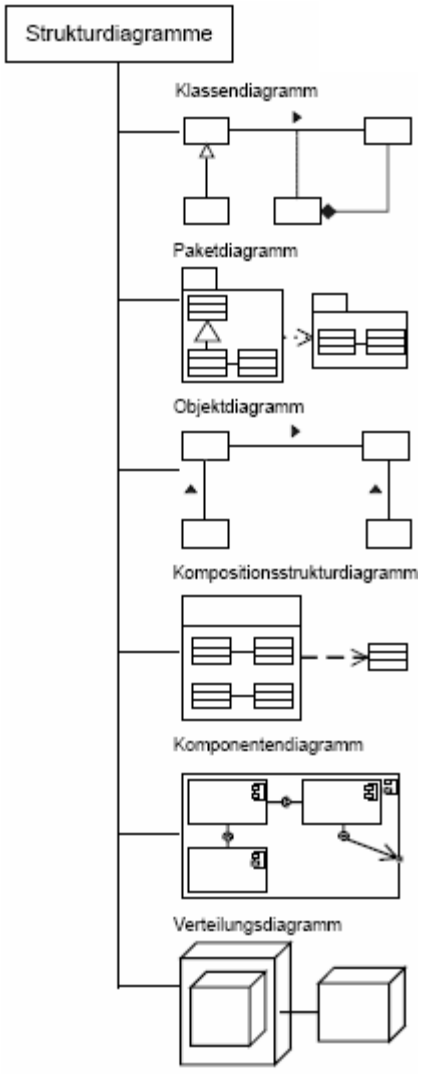


Diagramm Übersicht (engl.)



Quelle: www.uml.org, 2006

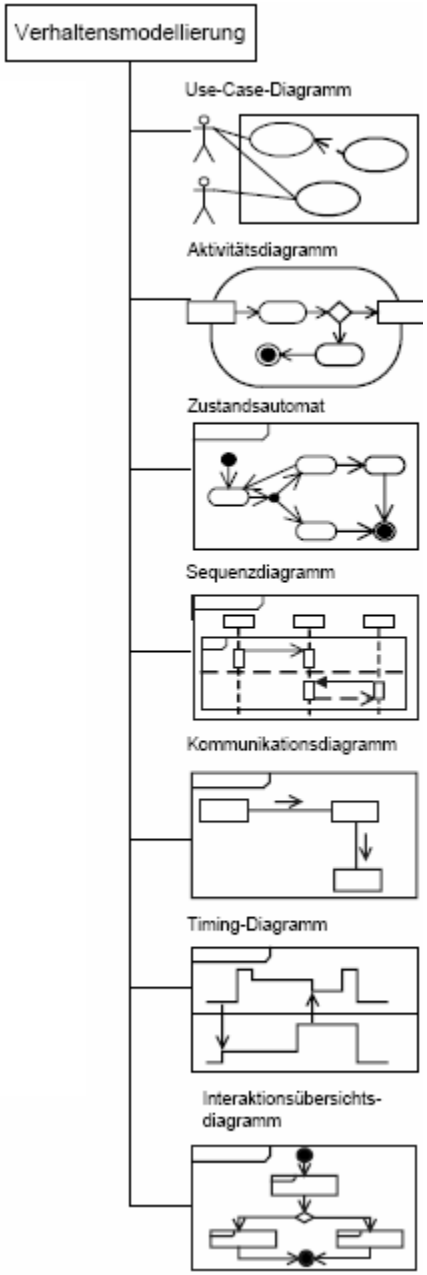
Strukturdiagramme



Quelle: Mario Jeckle, 2003

- Klassendiagramm (class diagram)
  - ⇨ Modellierung der Klassen mit Methoden, Attributen und Beziehungen
- Paketdiagramm (paket diagram)
  - ⇨ Modellierung der logische Struktur der Modellelemente mit deren (z.T. abstrakten) Abhängigkeiten
- Objektdiagramm (object diagram)
  - ⇨ Modellierung von „Momentaufnahmen“ konkreter Objekte mit Werten und Beziehungsausprägungen
- Kompositionsstrukturdiagramm (composite structure diagram)
  - ⇨ Modellierung der internen Struktur eines Modell-elements (Classifiers) sowie dessen Möglichkeiten zu Interaktion mit anderen Systemkomponenten
- Komponentendiagramm (component diagram)
  - ⇨ Modellierung der Komponenten (=Module) und deren Abhängigkeiten
- Verteilungsdiagramm (deployment diagram)
  - ⇨ Zeigt die (physischen) Geräte des Systems im Betrieb

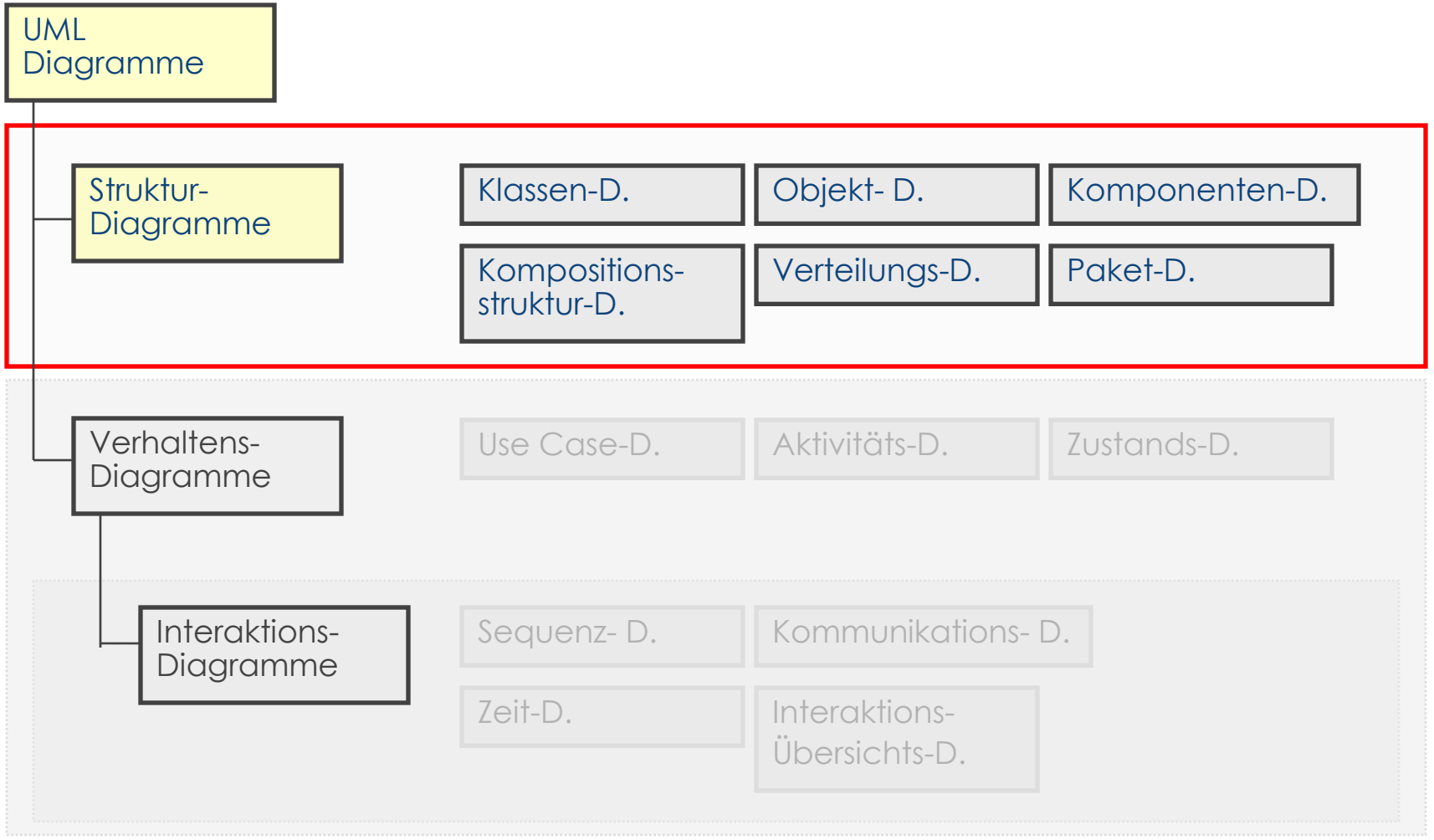
Verhaltensdiagramme



- Use-Case Diagramm (use case diagram)
  - ⇨ Modellierung der Anwendungsfunktionalität aus der Sicht der Anwender
- Aktivitätsdiagramm (activity diagram)
  - ⇨ Modellierung des dynamischen Ablaufs (i.d.R. eines Use Case)
- Zustandsdiagramm (state transition diagram)
  - ⇨ Modelliert die Zustände und Zustandswechsel (i.d.R. zu einer Klasse)
- Sequenzdiagramm (sequence diagram)
  - ⇨ Modelliert ein Szenario: den Botschaftsfluss zwischen Objekten
- Kommunikationsdiagramm (comunication diagram)
  - ⇨ Dynamik wie Sequenzdiagramm; zusätzlich Objekteigenschaften
- Timing-Diagramm
  - ⇨ Präziser zeitlicher Verlauf
- Interaktionsübersichtsdiagramm
  - ⇨ Modellierung des Zusammenspiels einzelner Interaktionsdiagramme

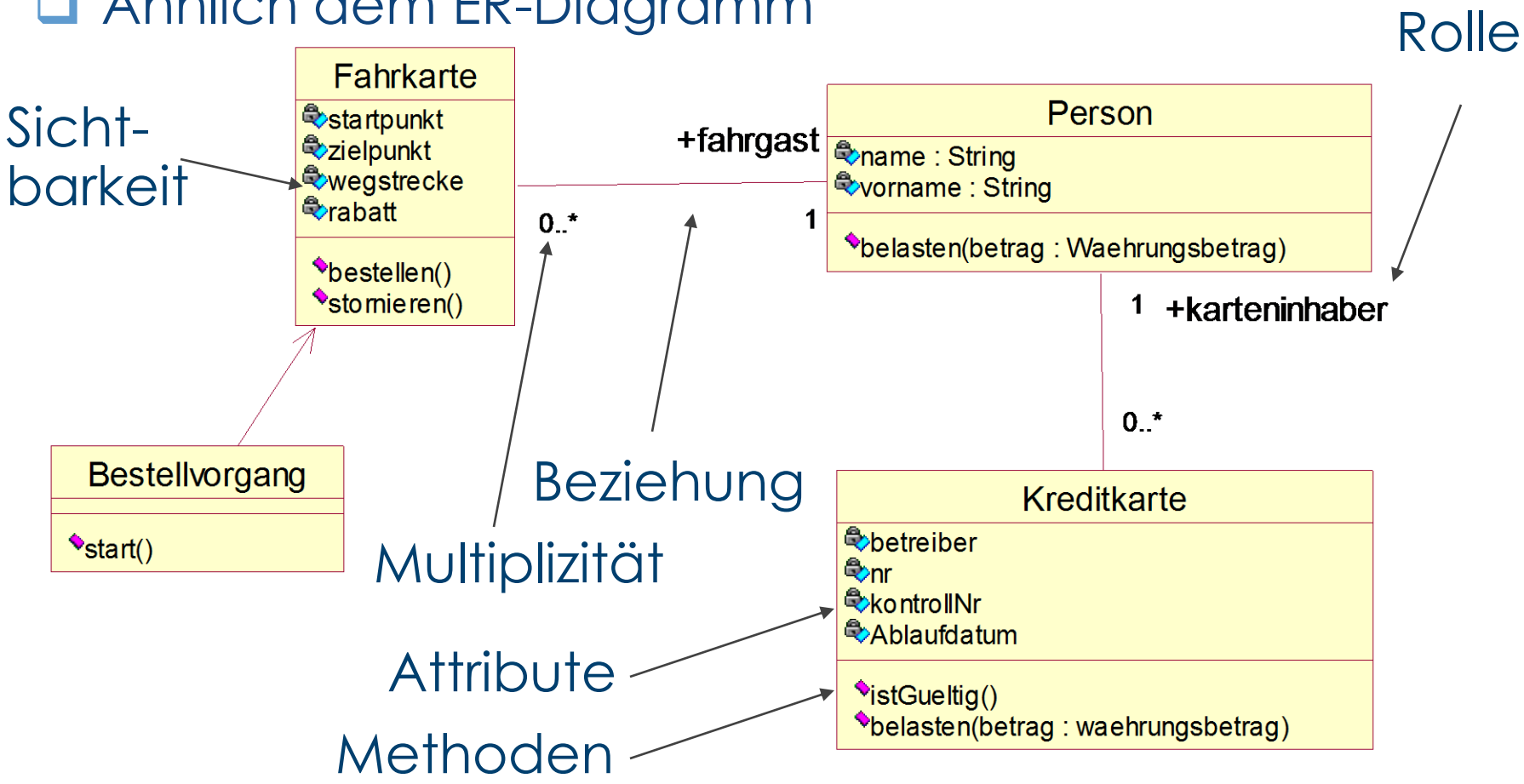
Quelle: Mario Jeckle, 2003

Diagramm Übersicht



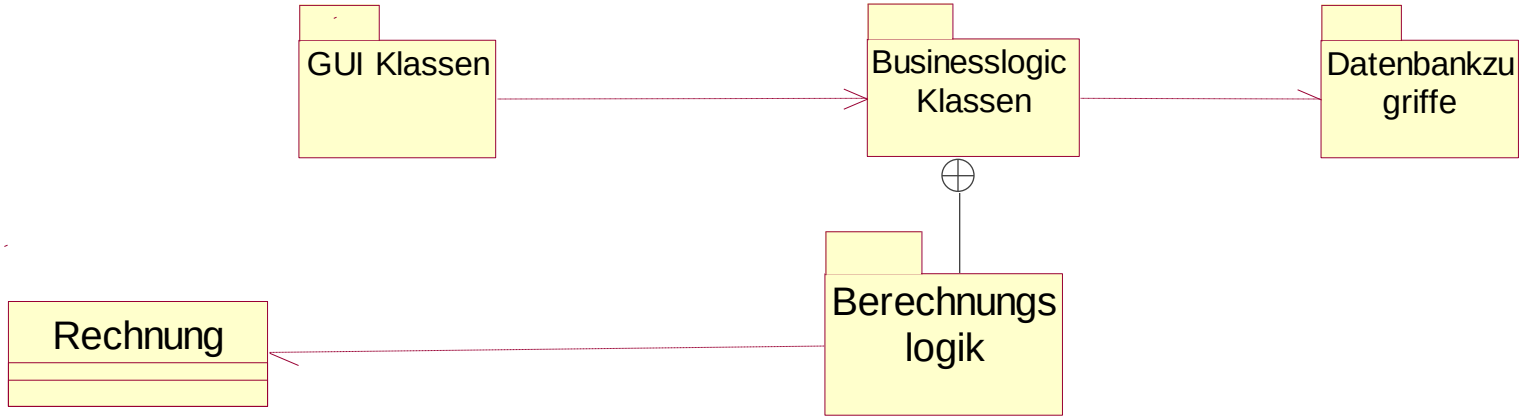
Klassendiagramm

- Modellierung der **Klassen mit Methoden, Attributen und Beziehungen**
- Ähnlich dem ER-Diagramm



Paketdiagramm

- Pakete dienen als „Container“ für beliebige UML Modellelemente
- Das Paketdiagramm zeigt (abstrakte) Beziehungen zwischen einzelnen Paketen

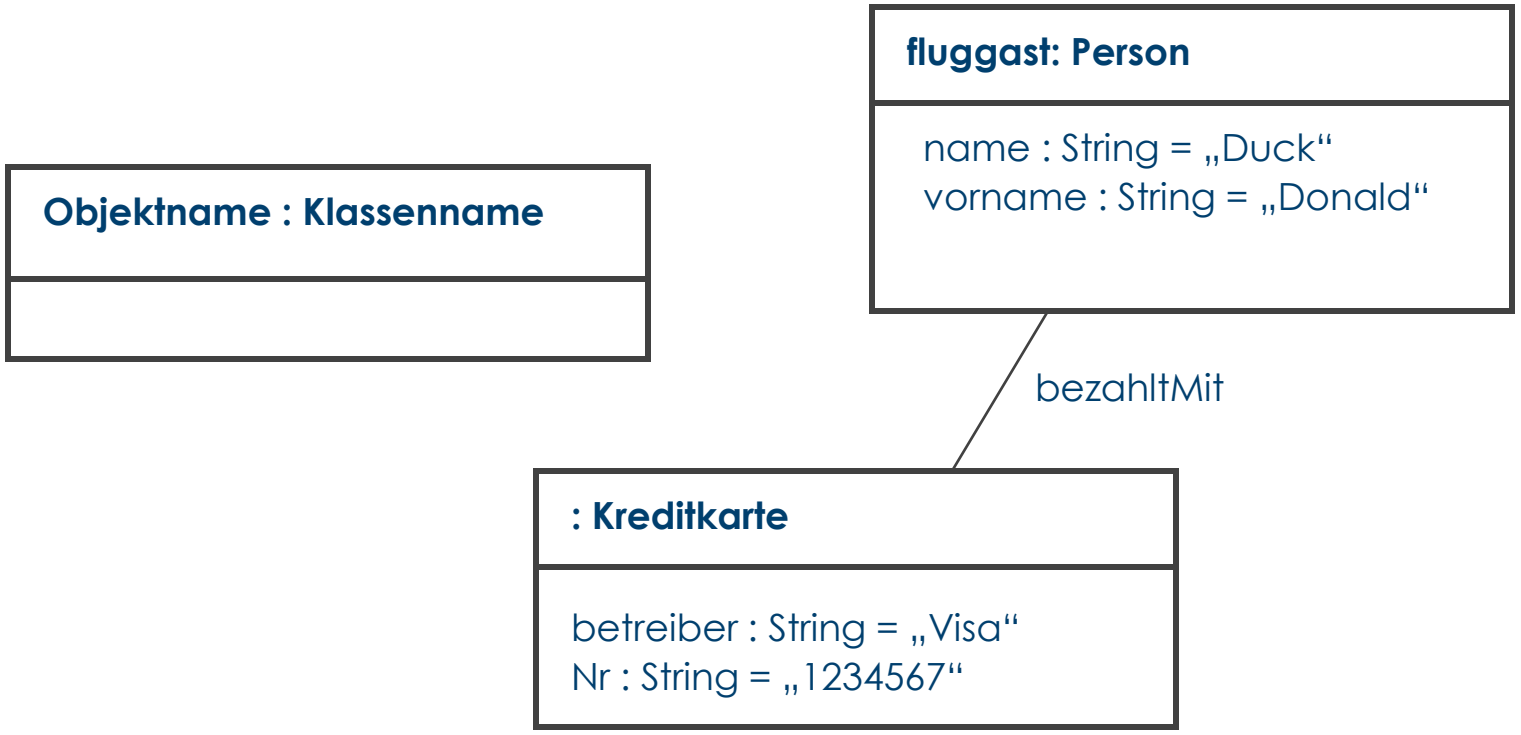


Modellsicht:

- Ein Paket kann selber wieder Pakete oder andere Modellelemente wie z.B. Klassen enthalten

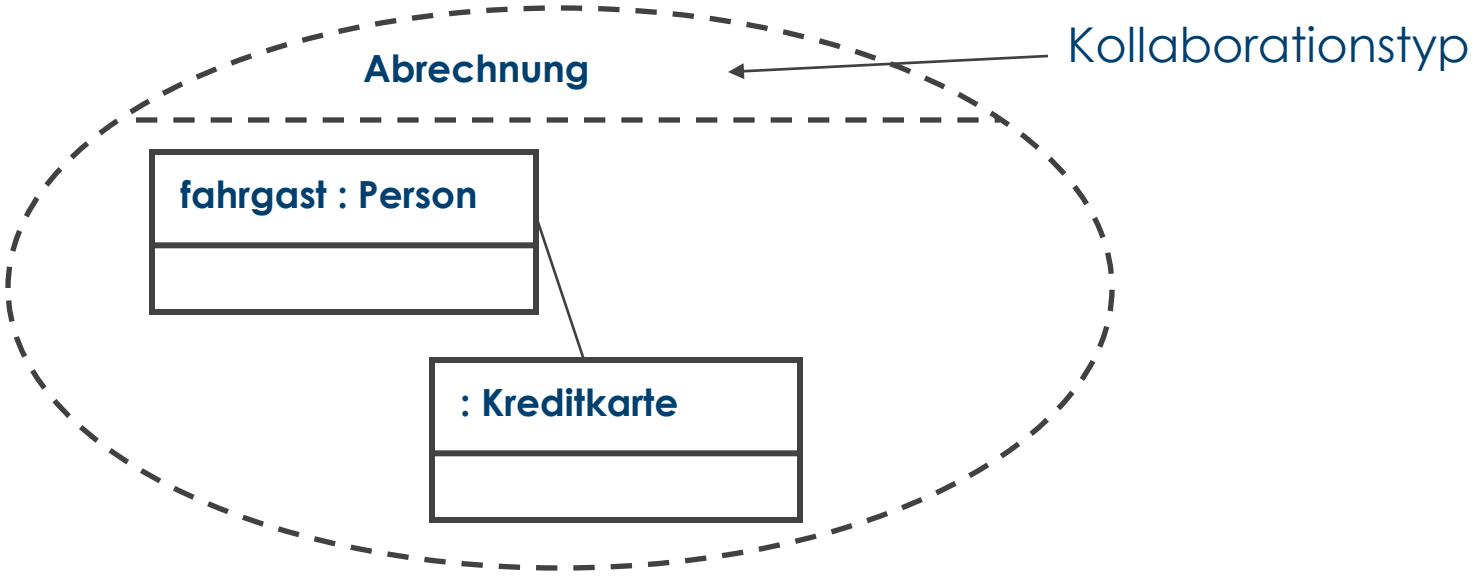
Objektdiagramm

- Zeigt eine Momentaufnahme des Systems
- Dem Klassendiagramm sehr ähnlich: statt der (abstrakten) Klassen werden (konkrete) Objekte dargestellt
- Objektname ist in der Regel eine Rolle, in der ein Objekt agiert
- Attributwerte können auch angezeigt werden



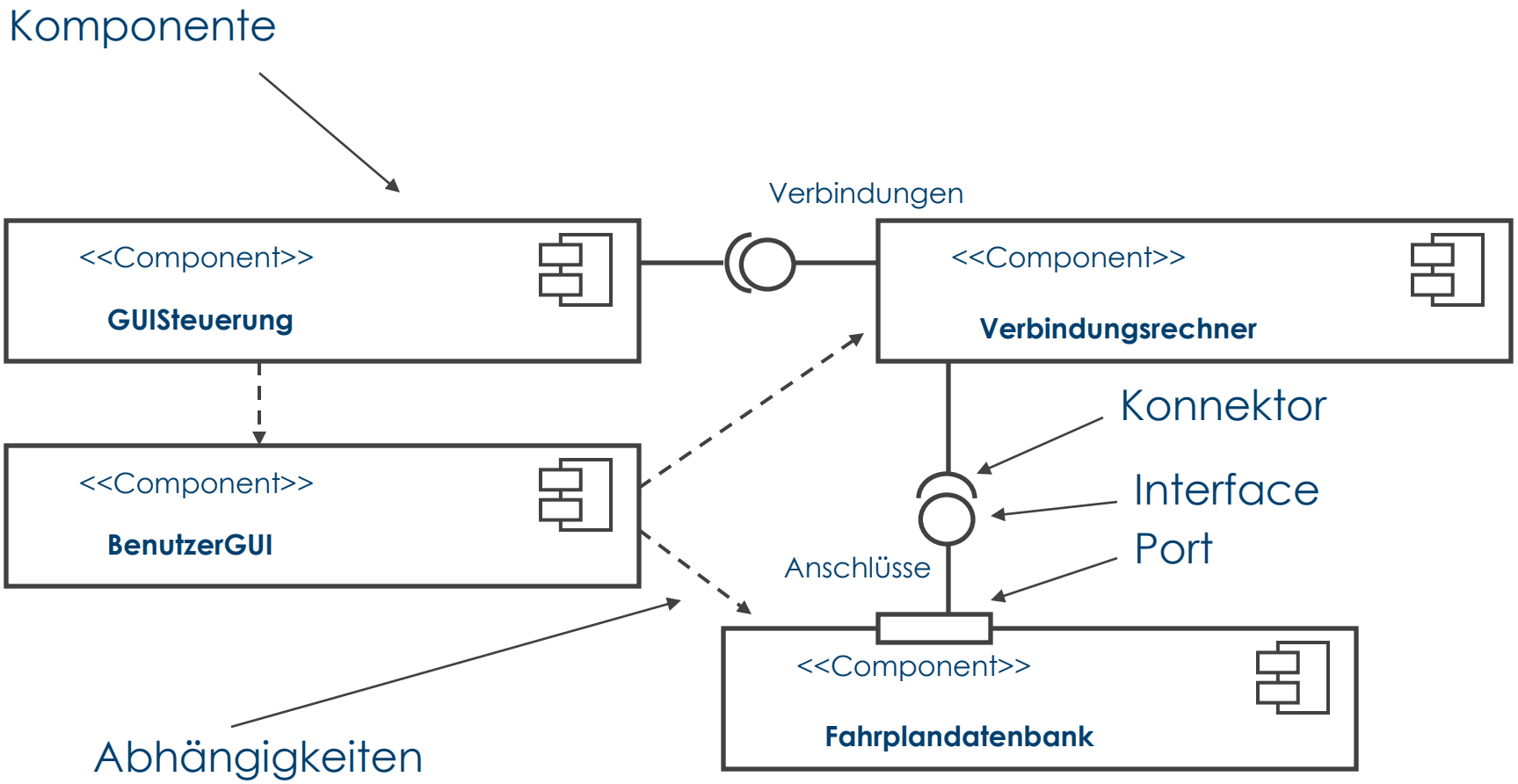
Kompositions-Struktur-Diagramm

- Ab UML 2.0, Composite Structure
- „Kontext“-bezogenes Klassen-, Objekt oder Komponentendiagramm
- Zeigt eine „Ausprägung“ oder eine bestimmte Konfiguration



Komponenten Diagramm

- Modelliert die Software-Architektur mit Bibliotheken, Komponenten, Datenbanken, usw.



Verteilungsdiagramm

- Deployment diagram
- Modelliert die Abbildung auf das physische Ziel-System (die Geräte) wie Server, Clients, externe Geräte, usw.
- Beschreibende Artefakte können ebenso mit aufgenommen werden

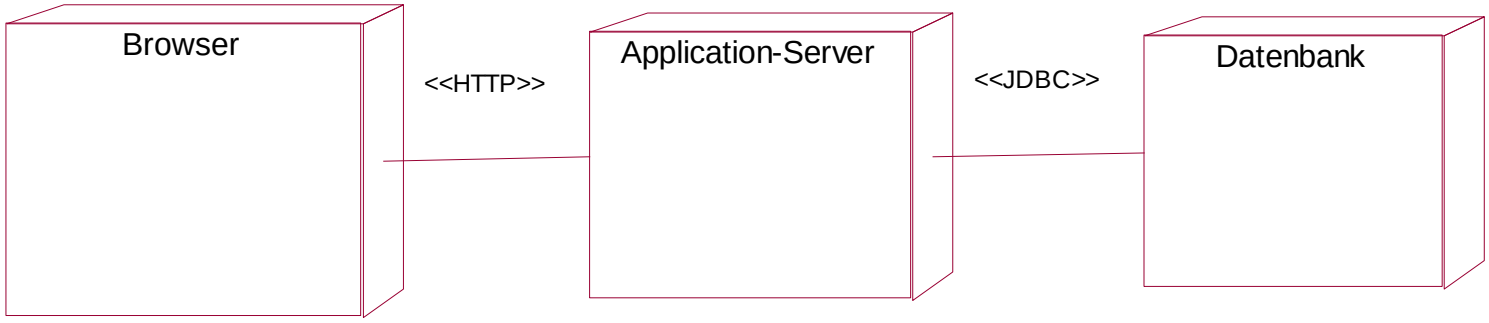
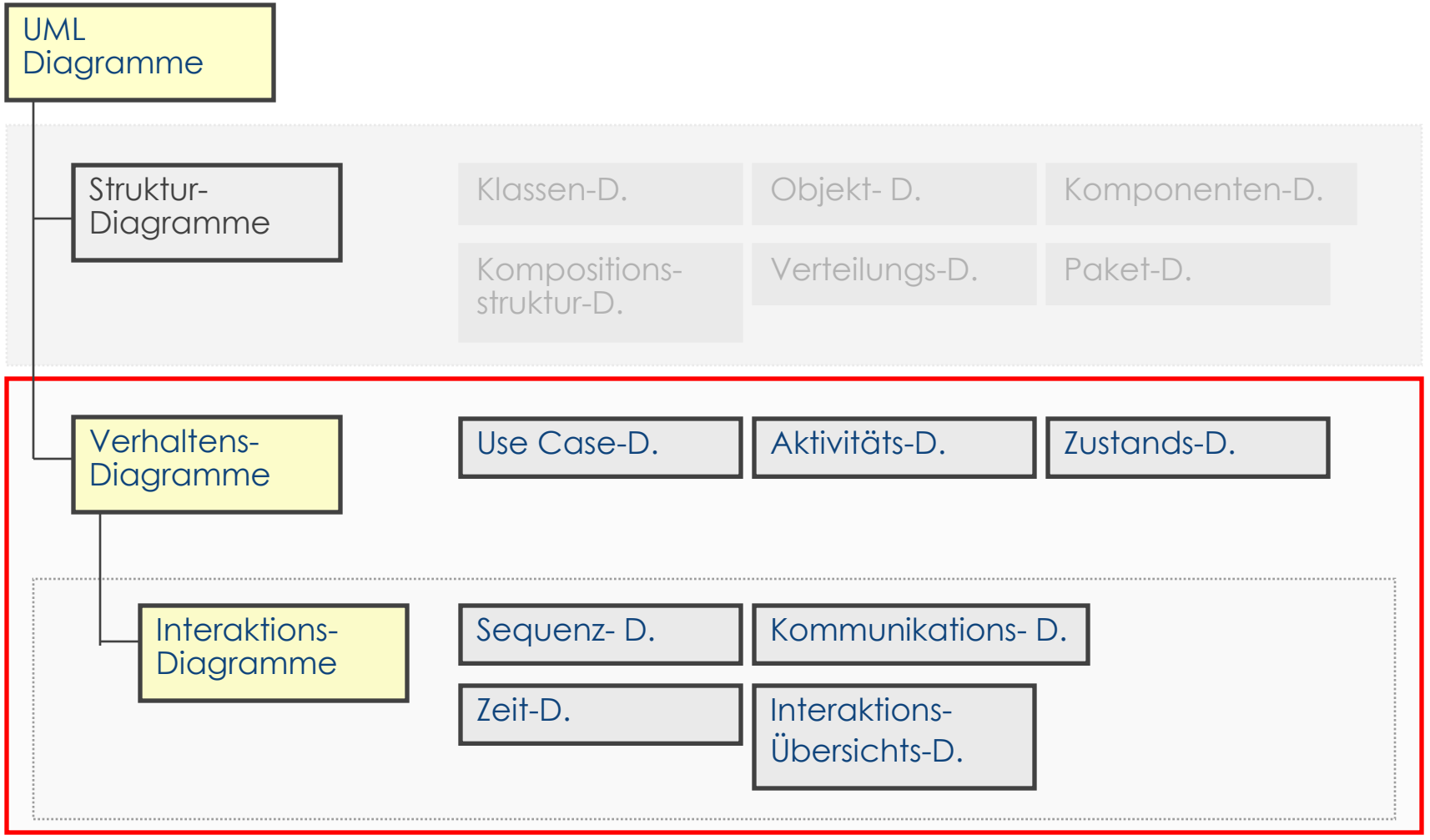
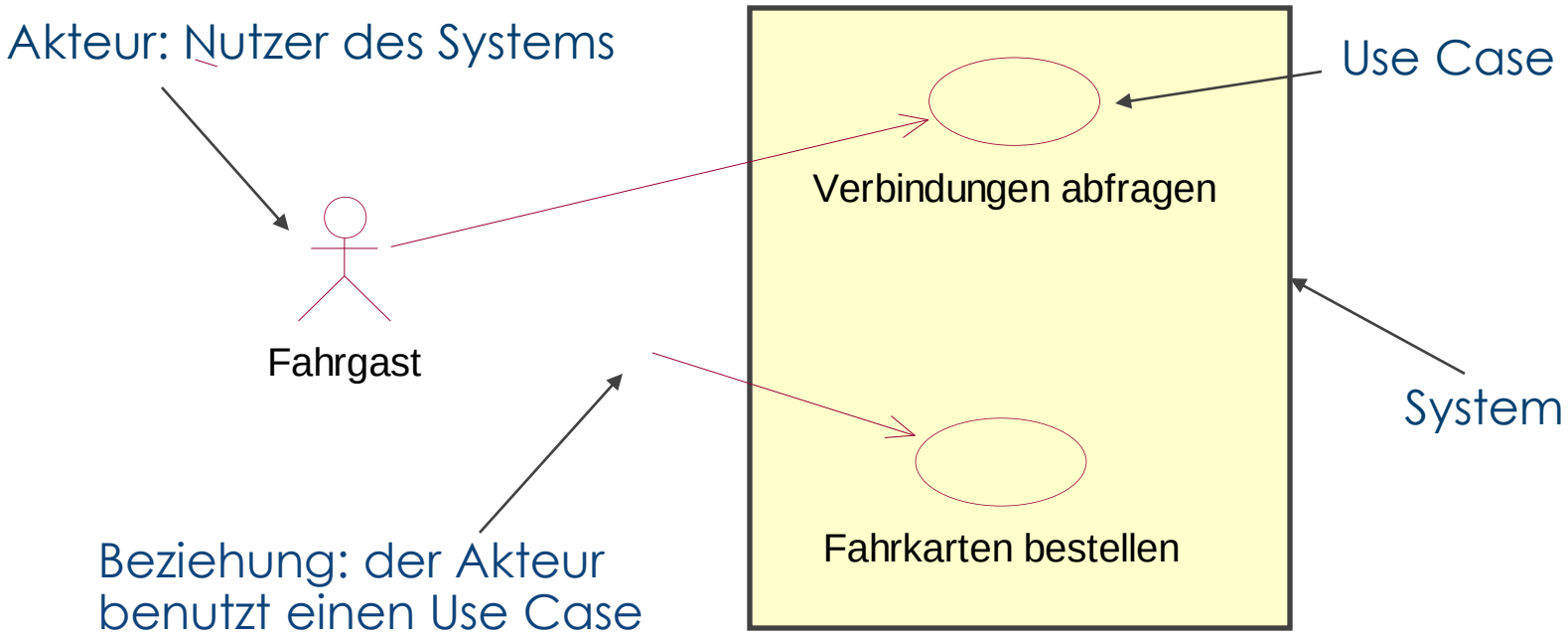


Diagramm Übersicht



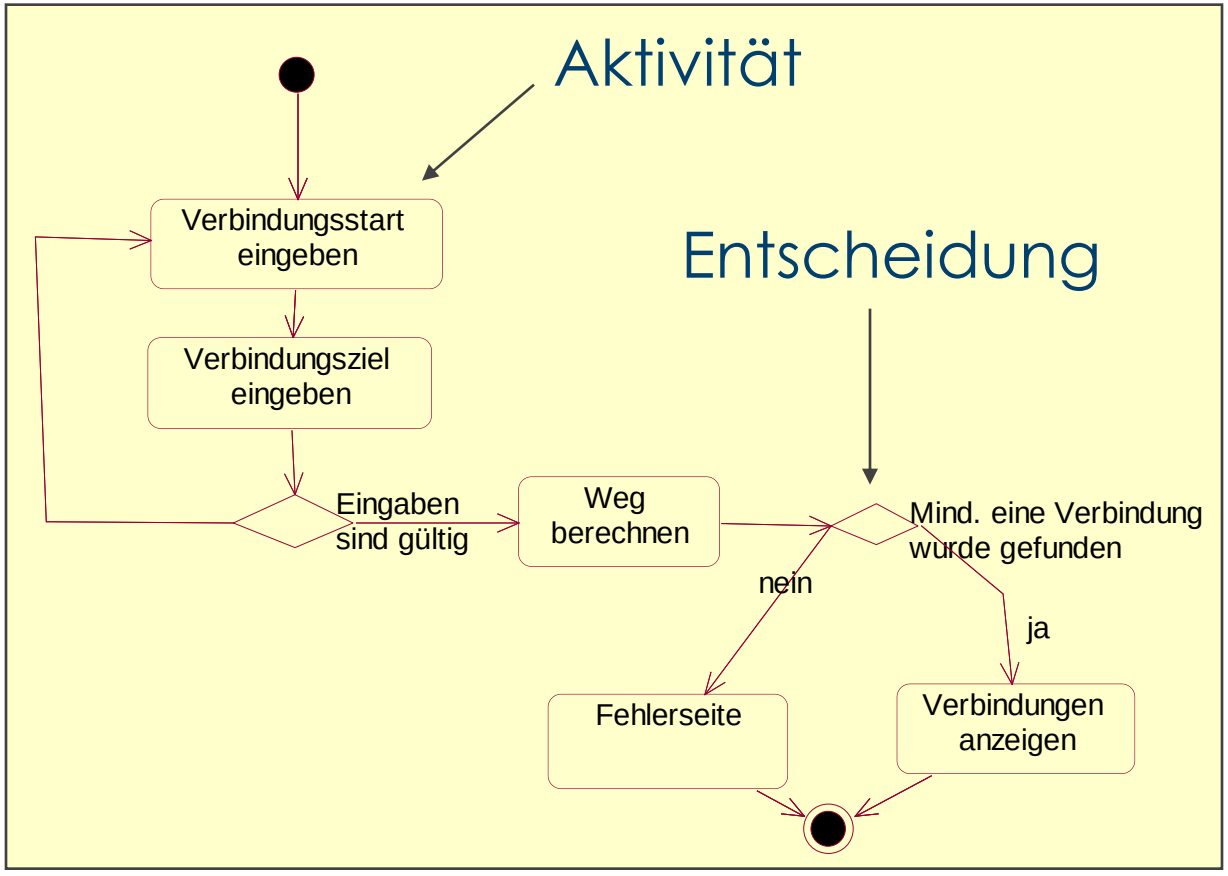
Use Case

- Modellierung der **Anwendungsfunktionalität** aus der **Sicht der Anwender**
- **Geschäftsprozesse**
- Sehr einfache Diagramme



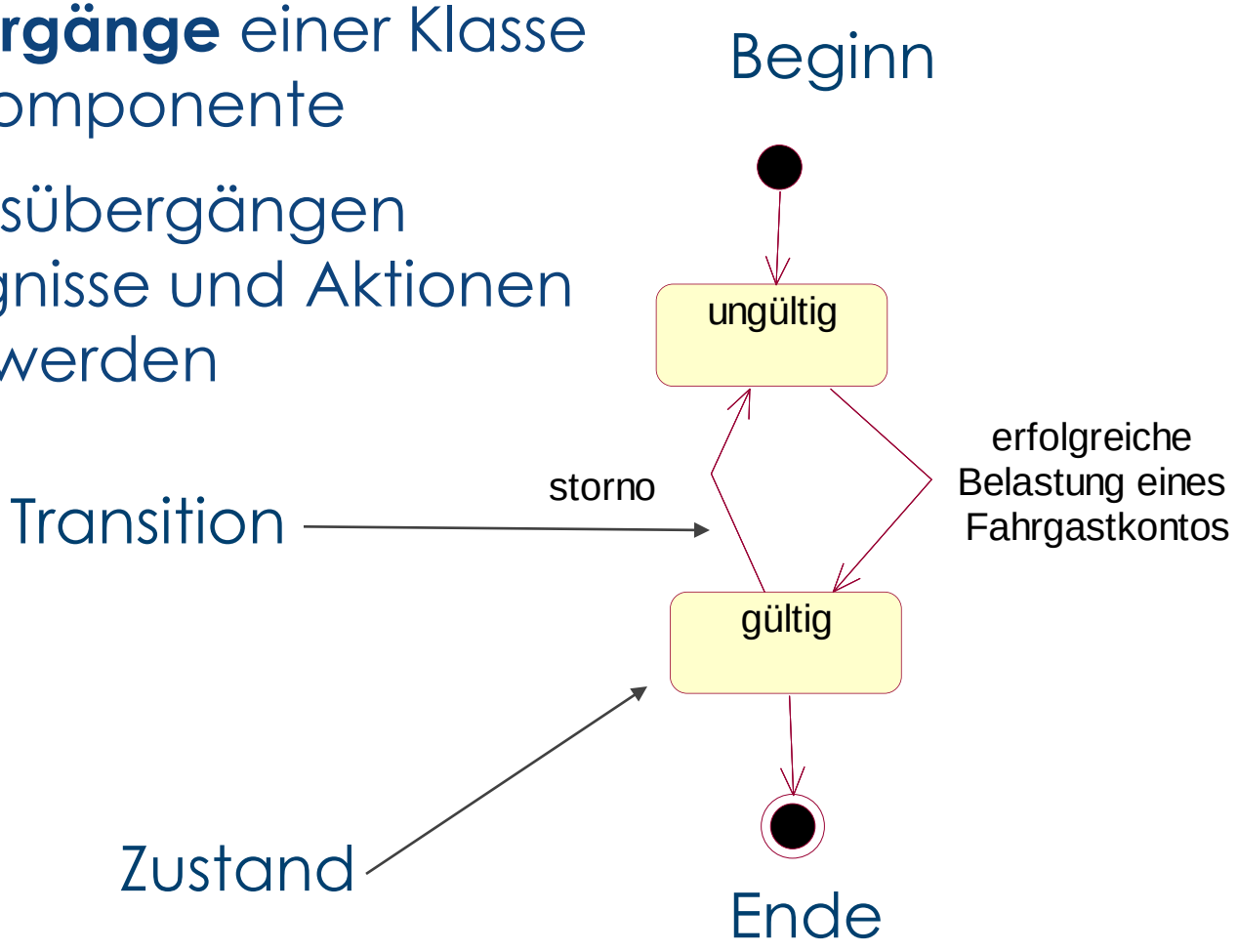
Aktivitätsdiagramm

- Beschreibt den Ablauf eines Use Case
- Ist auch für parallele Abläufe geeignet



Zustandsdiagramm

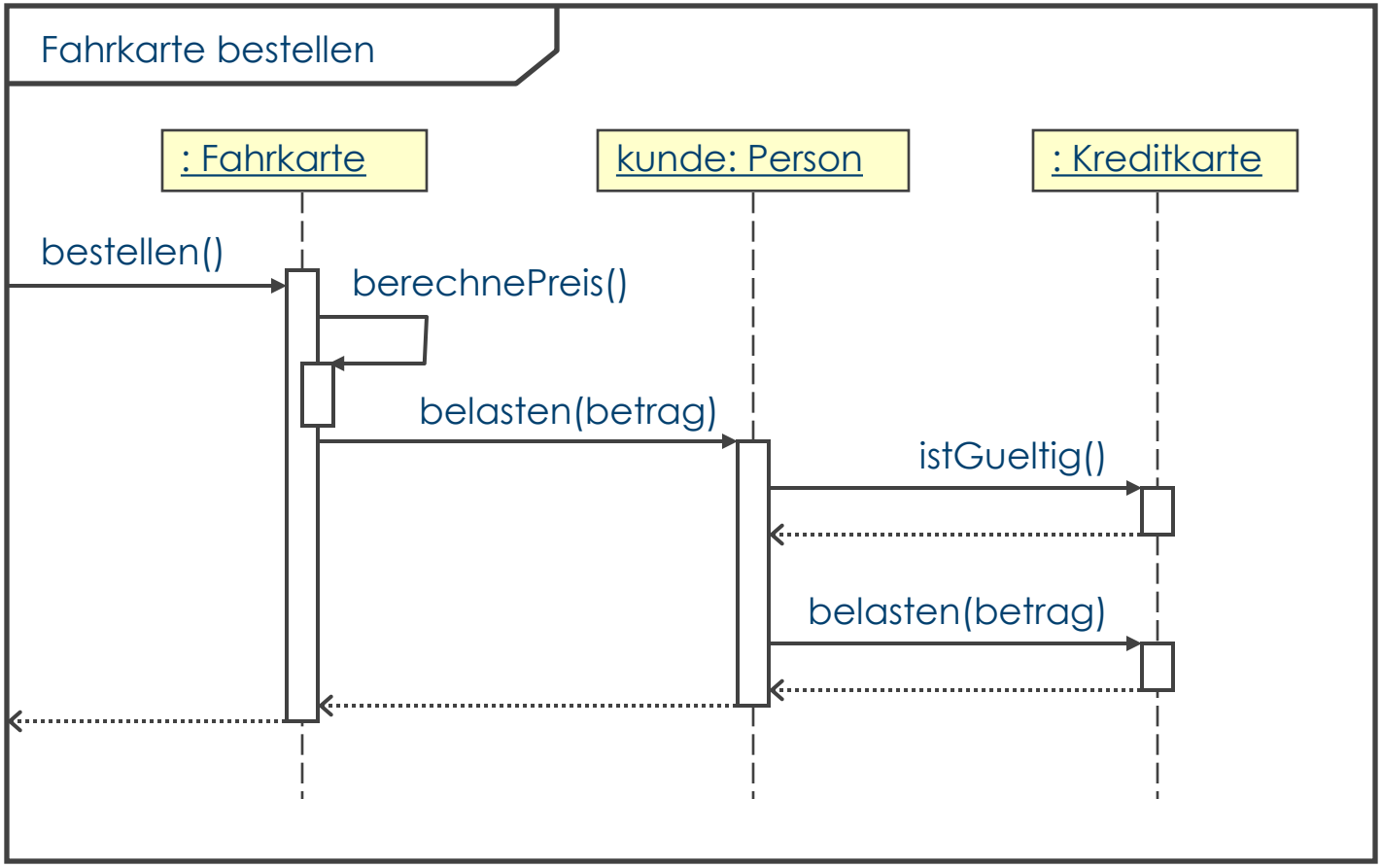
- Modelliert die **Zustände** und **Zustandsübergänge** einer Klasse oder einer Komponente
- Den Zustandsübergängen können Ereignisse und Aktionen zugeordnet werden



Folie 26

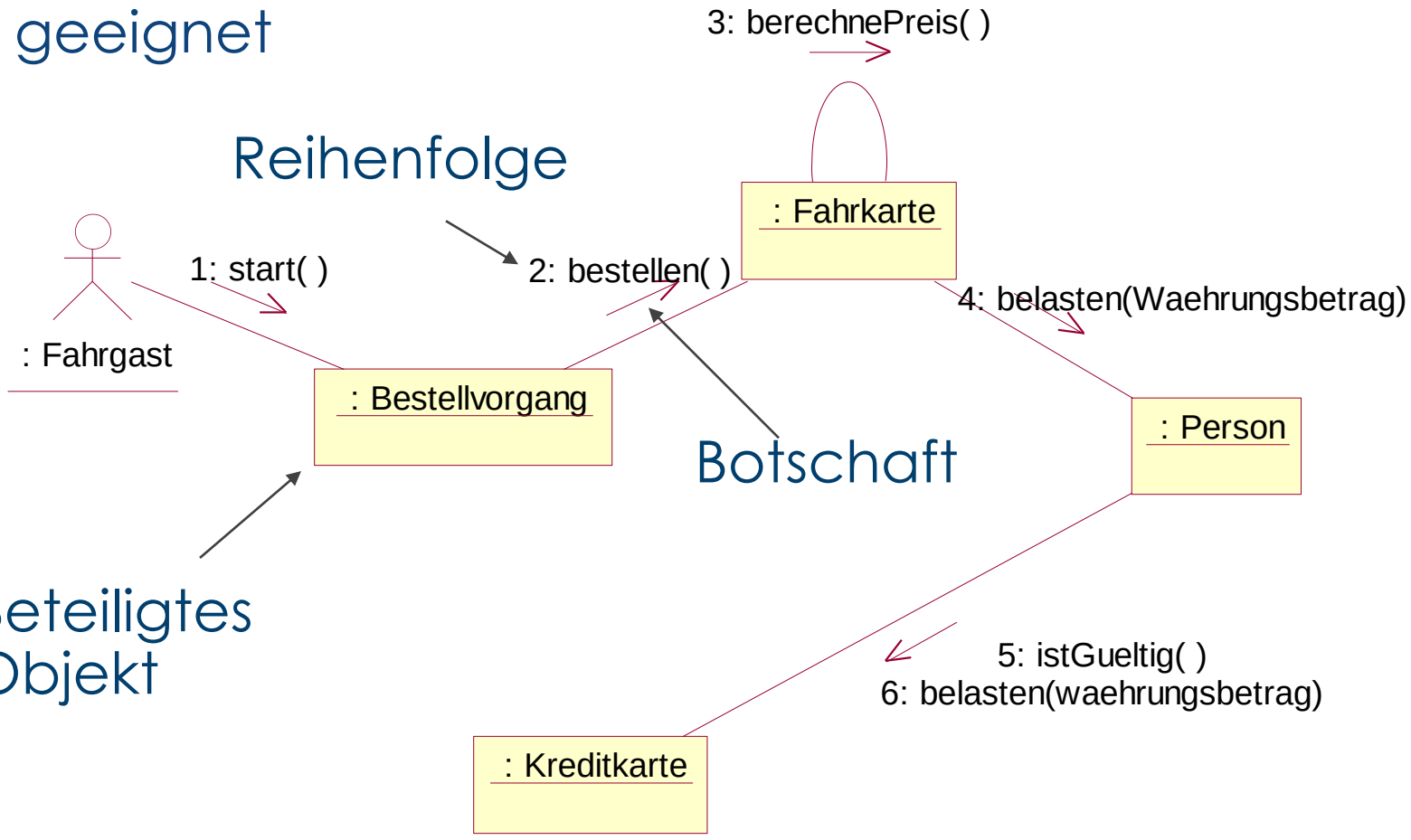
Sequenzdiagramm

- Botschaftsfluss zwischen Objekten



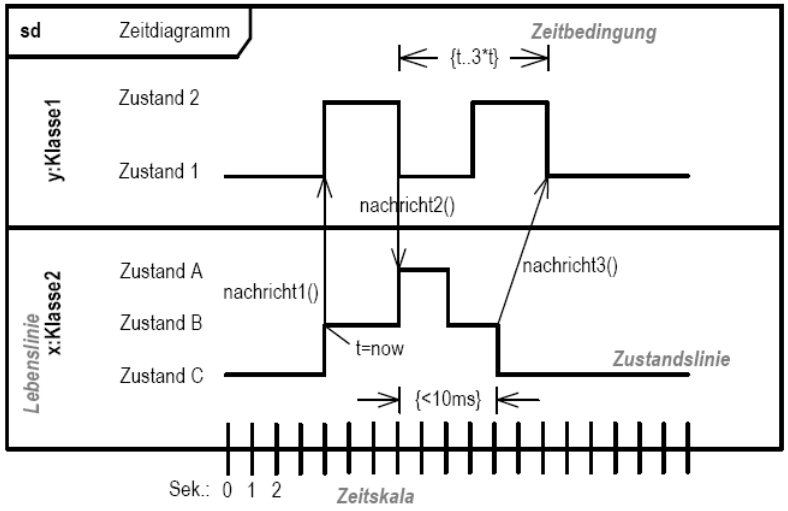
Kommunikationsdiagramm

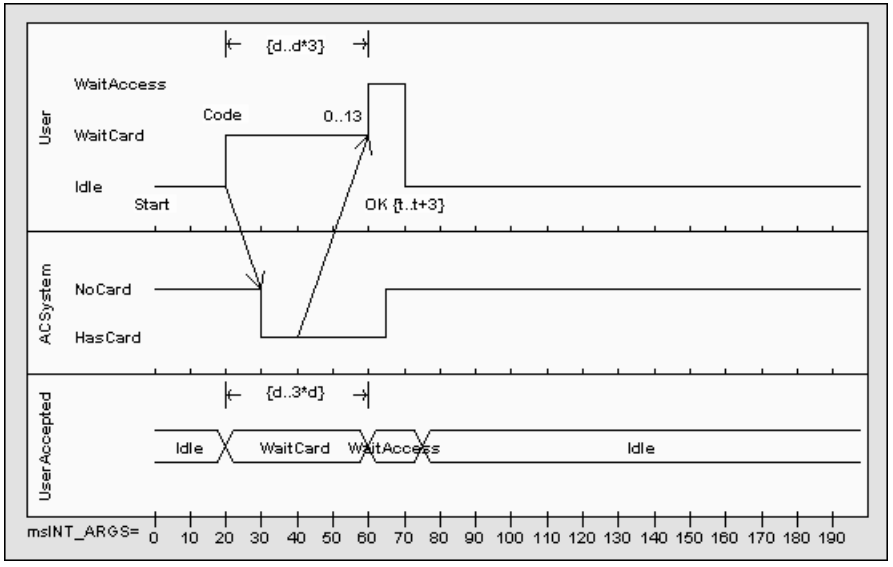
- Ähnlicher Inhalt wie beim Sequenzdiagramm
- Bei vielen Objekten mit wenig Botschaftsfluss geeignet



Timing diagram

- Ab UML 2.0
- Zeigt das Verhalten von Objekten mit genauer Zeitachse.
- Speziell für Hardware dominierte oder Realtime Software.

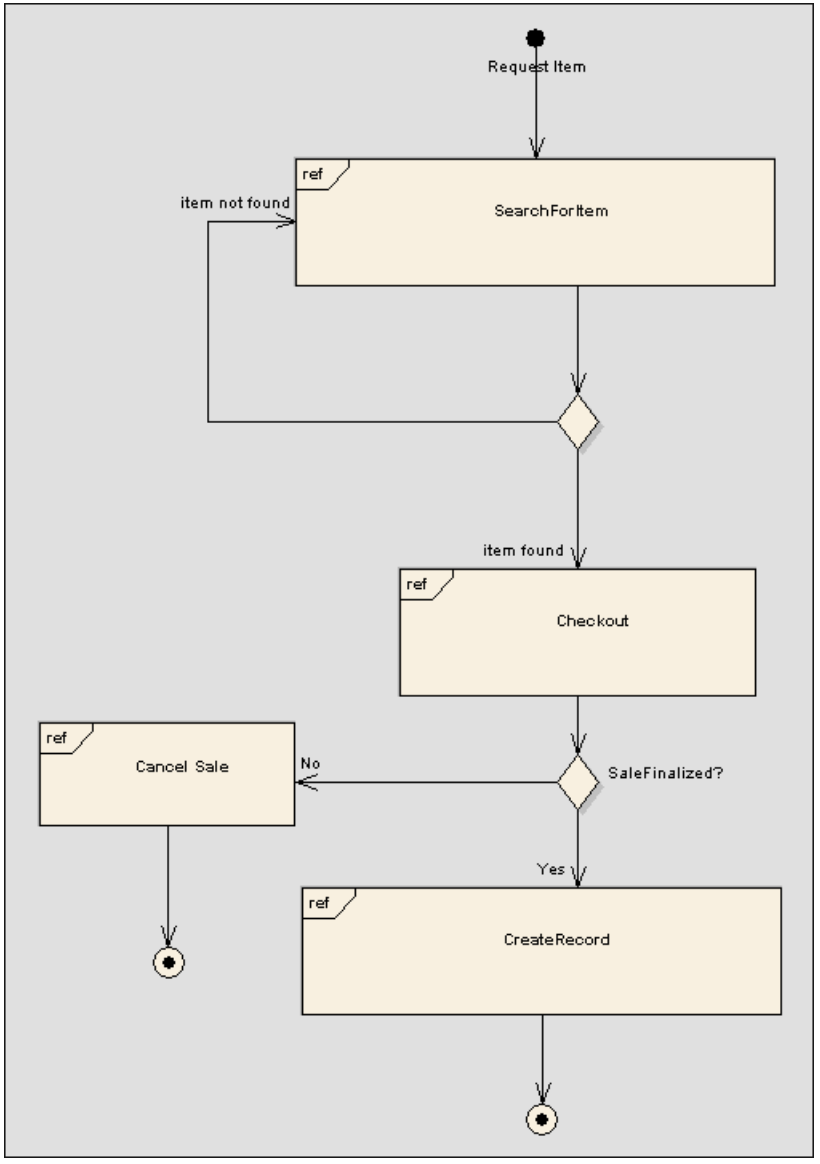




Quelle: www.oose.de, www.sparxsystems.com.au

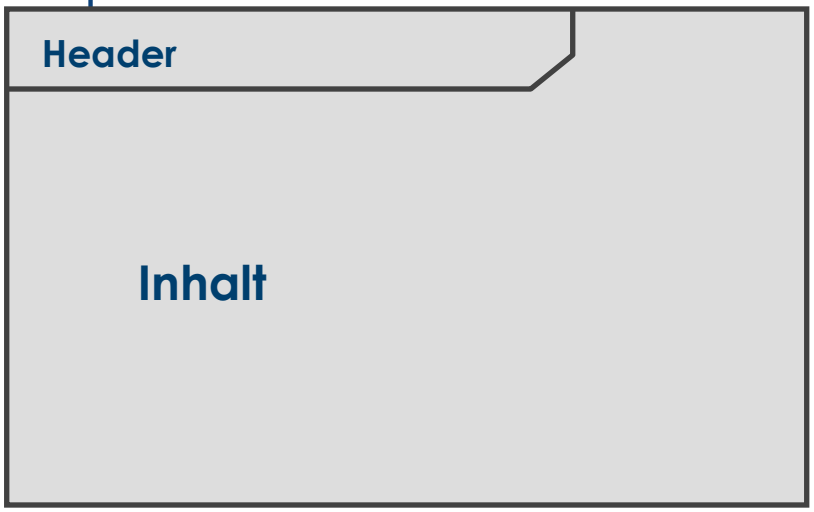
Interaktionsübersicht

- Ab UML 2.0
- Zeigt den Zusammenhang zwischen einzelnen Interaktionsdiagrammen auf
- Elemente wie im Aktivitätendiagramm



UML – Diagramm Layout

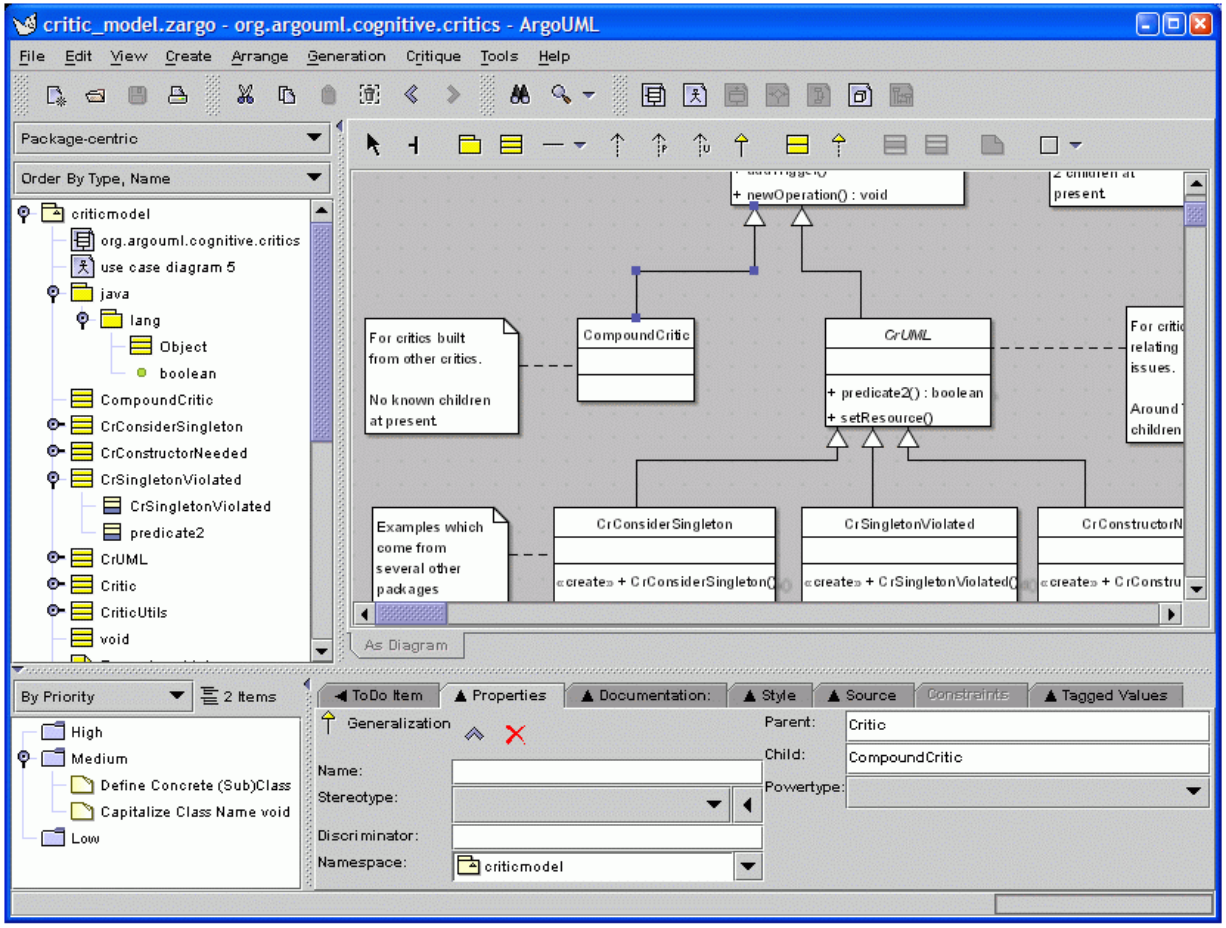
- Header
  - ⇨ Syntax:
    - [<DiagrammTyp>]<DiagrammName>[<Parameter>]
  - ⇨ DiagrammTyp (Kürzel) - optional
    - cd, sd, uc, ...
  - ⇨ DiagrammName
  - ⇨ Parameter – optional
    - wird nur selten benutzt



- Inhalt
  - ⇨ Graphisches Diagramm des angegebenen Typs
- Anmerkung
  - ⇨ Die Diagramm-Darstellung wird letztendlich vom Werkzeug festgelegt

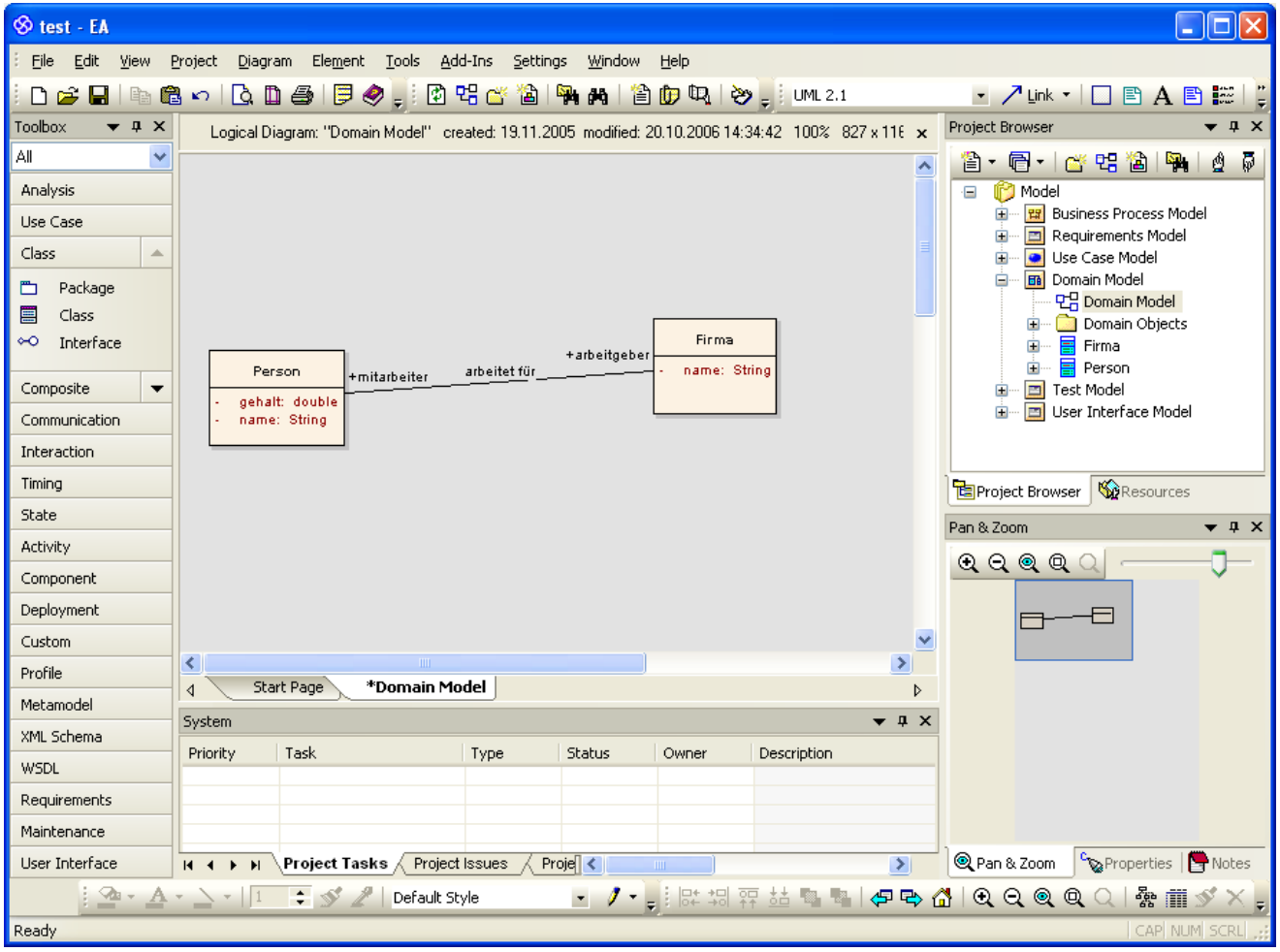
UML Werkzeuge: Argo UML

- www.argouml.org



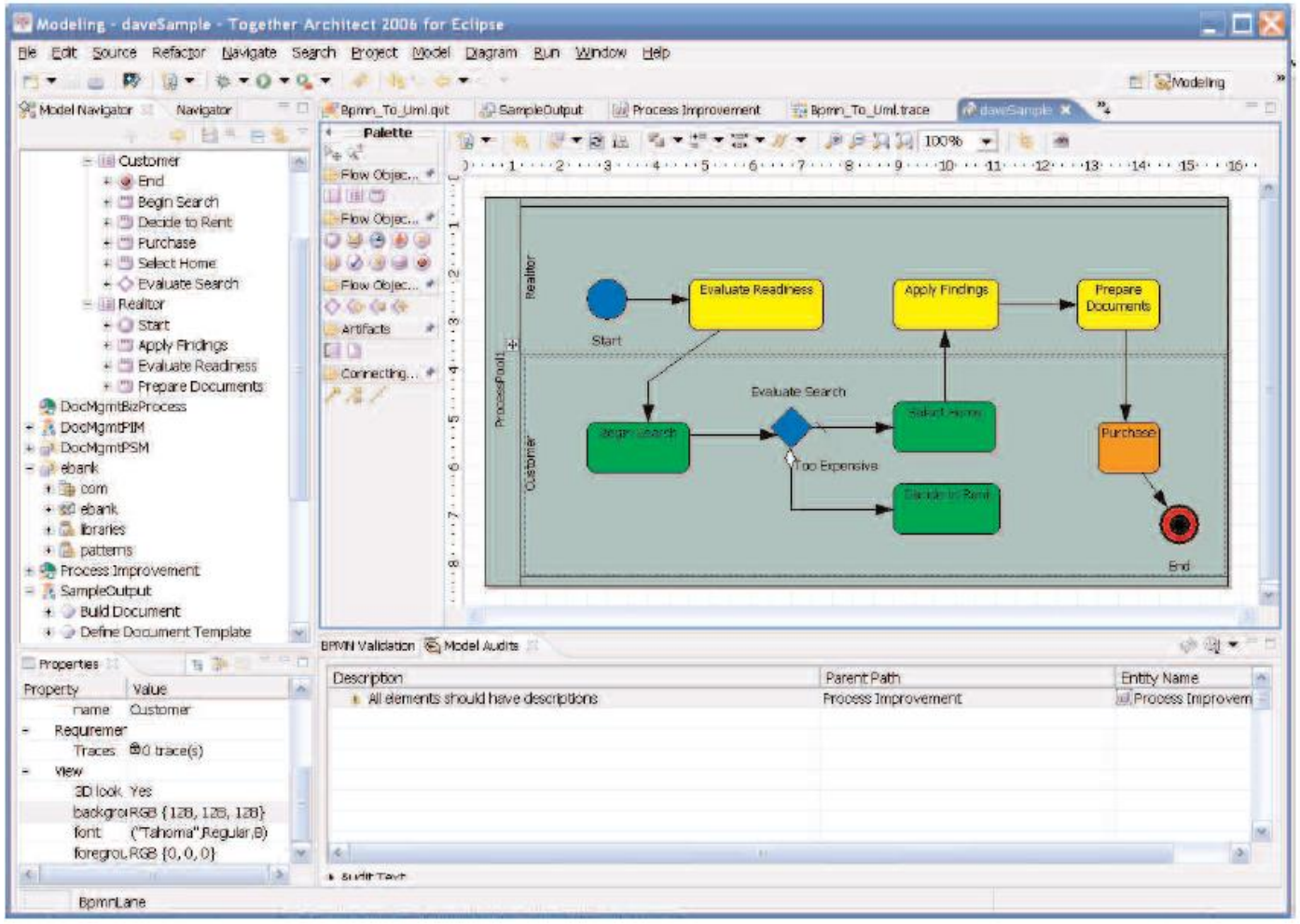
UML Werkzeuge: Enterprise Architect

- http://www.sparxsystems.com/



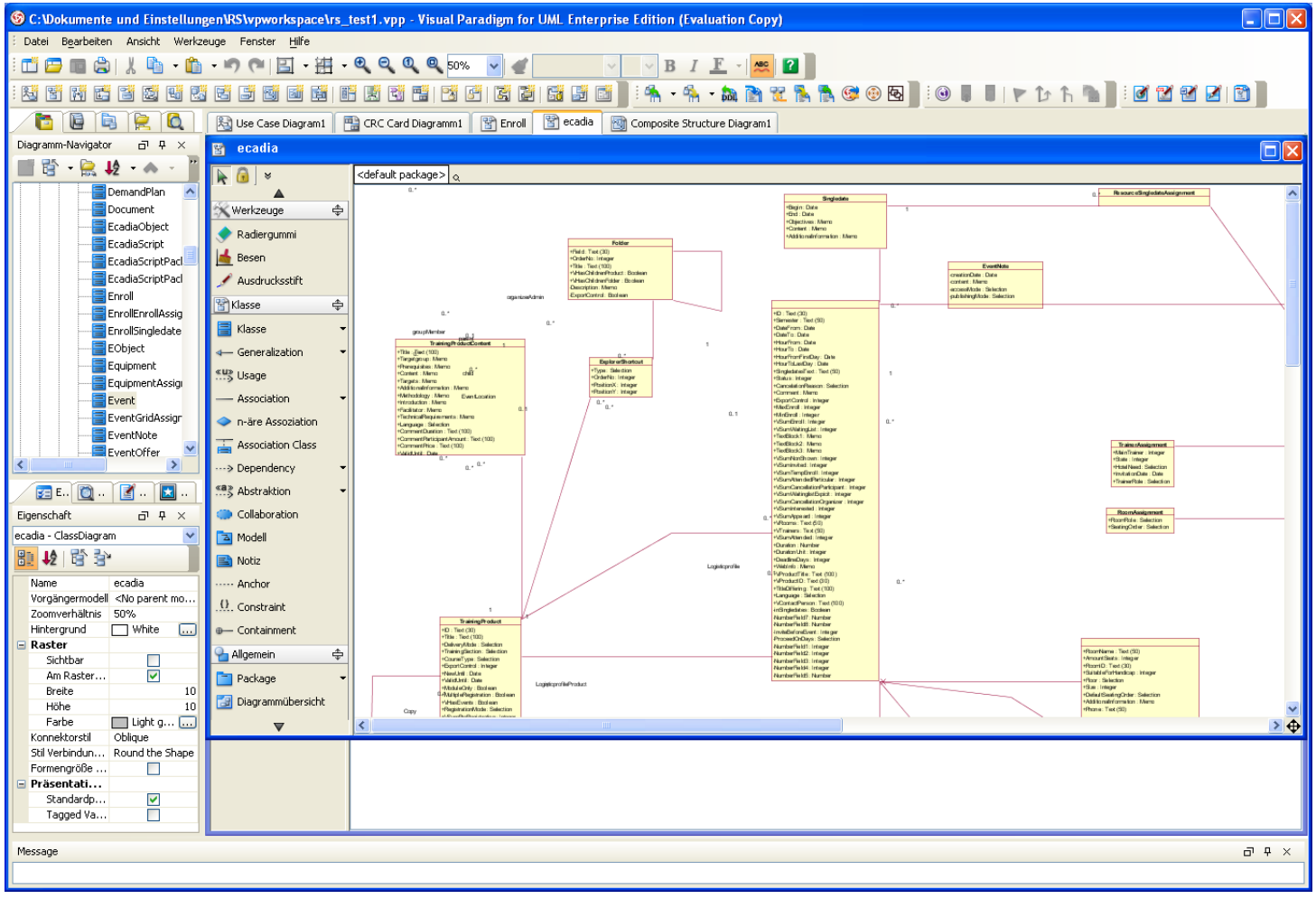
UML Werkzeuge: Borland Together

- www.borland.com/de/products/together/index.html



UML Werkzeuge: Visual Paradigm

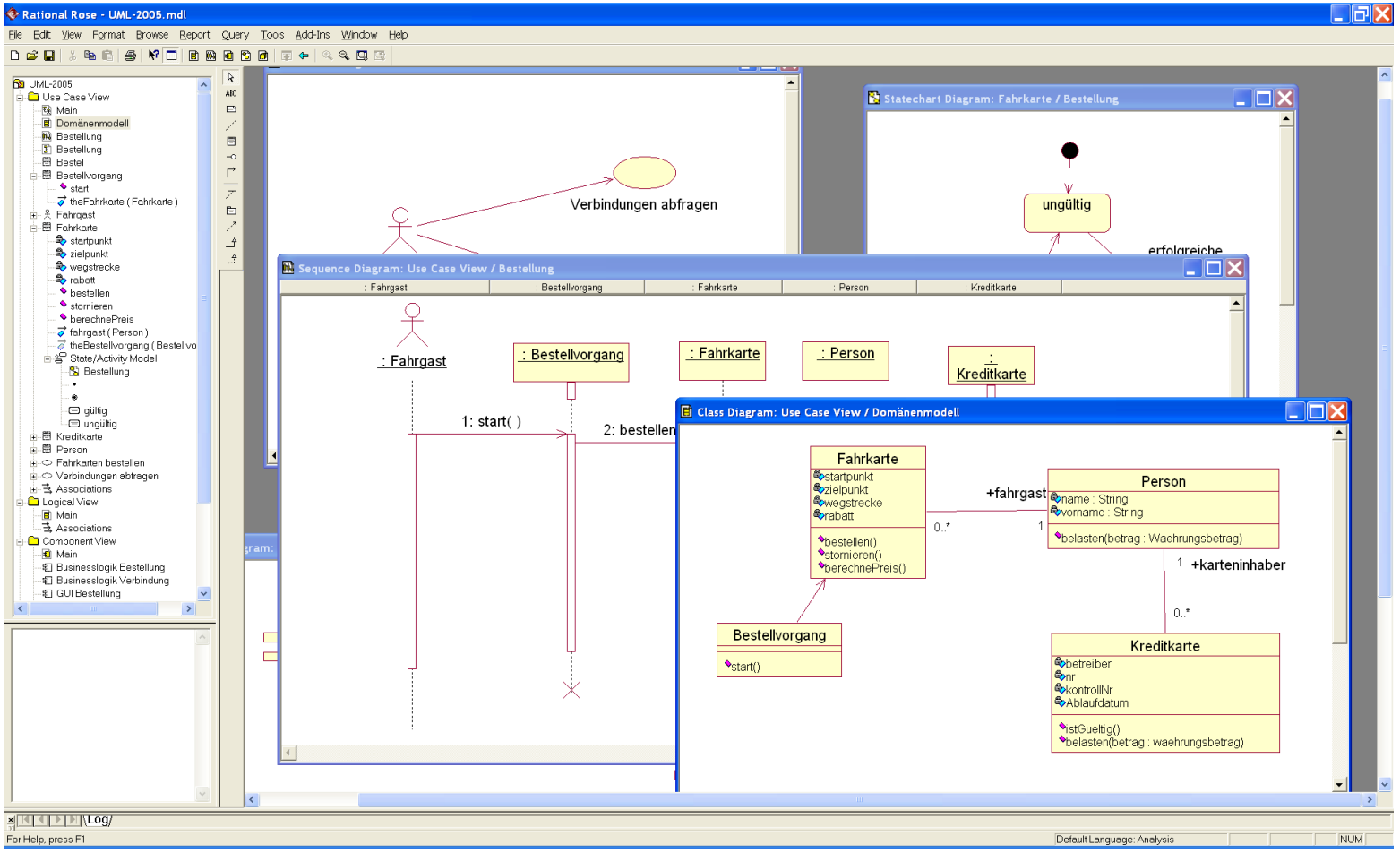
- http://www.visual-paradigm.com/



UML Werkzeuge: Rational Rose

Veraltet

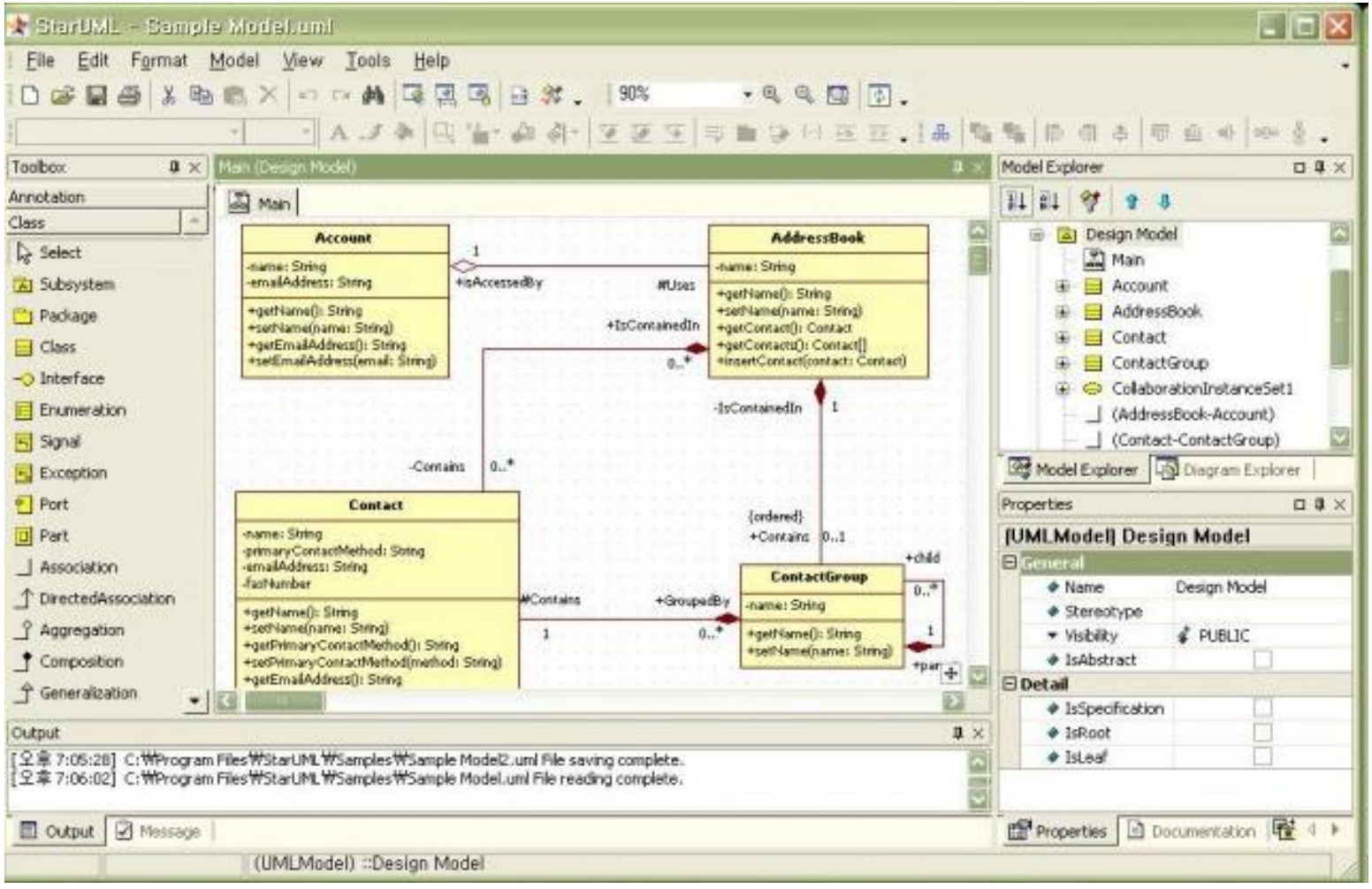
- www.ibm.com/software/rational



Folie 36

UML Werkzeuge: StarUML

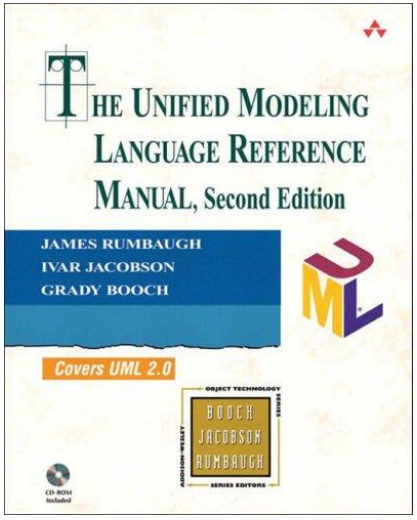
- http://staruml.sourceforge.net/



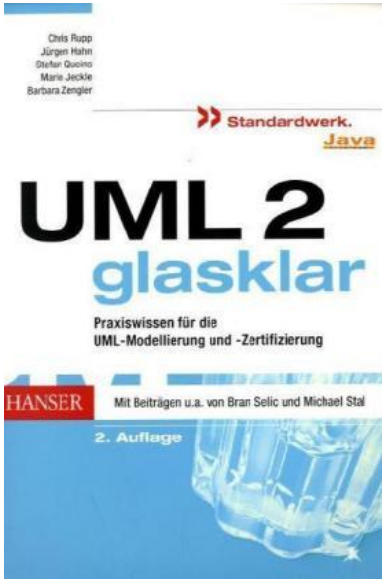
UML im Projektverlauf

	Analyse/ Spezifikation	Entwurf	Implement-ierung	Test	Auslieferung/ Betrieb
Use Case Diagramm	x			x	
Aktivitätendiagramm	x			x	
Zustandsdiagramm	x		x	x	
Klassendiagramm	x	x	x		
Sequenzdiagramm	x	x	x		
Komponenten-diagramm		x			
Paketdiagramm		x			
Verteilungsdiagramm					x

UML Literatur

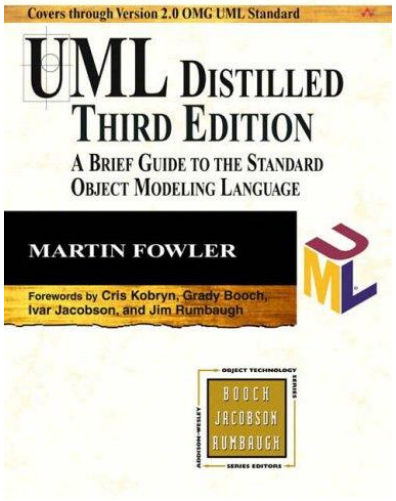


Rumbaugh, James; Jacobson, Ivar; Booch, Grady : „The unified modeling language reference manual“, Second Edition, Addison-Wesley, 2004

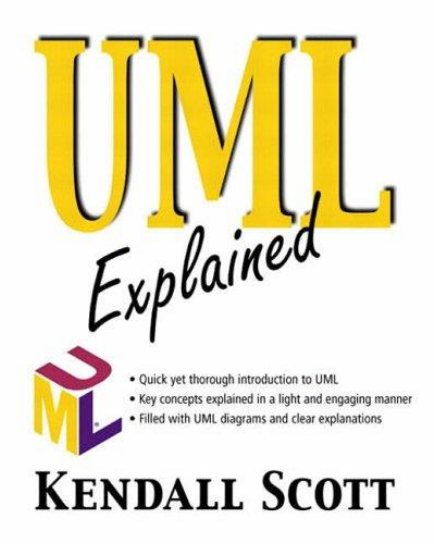


Jeckle, Mario; et al. : „UML2 glasklar“, Hanser Verlag, 2004

UML Literatur



Martin Fowler: „UML Distilled. A Brief Guide to the Standard Object Modeling Languange“, Addison-Wesley, 2003

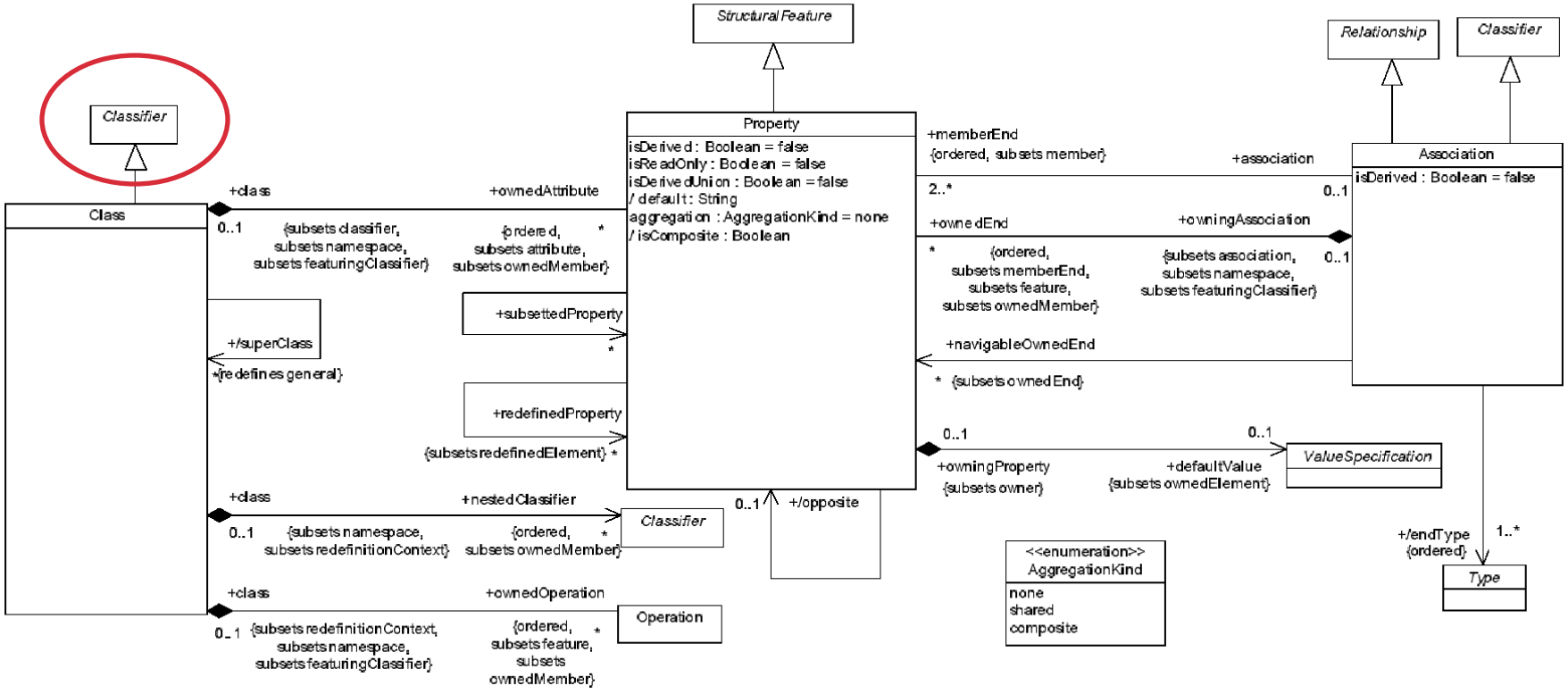


Kendall Scott : „UML Explained “, Addison-Wesley, 2001

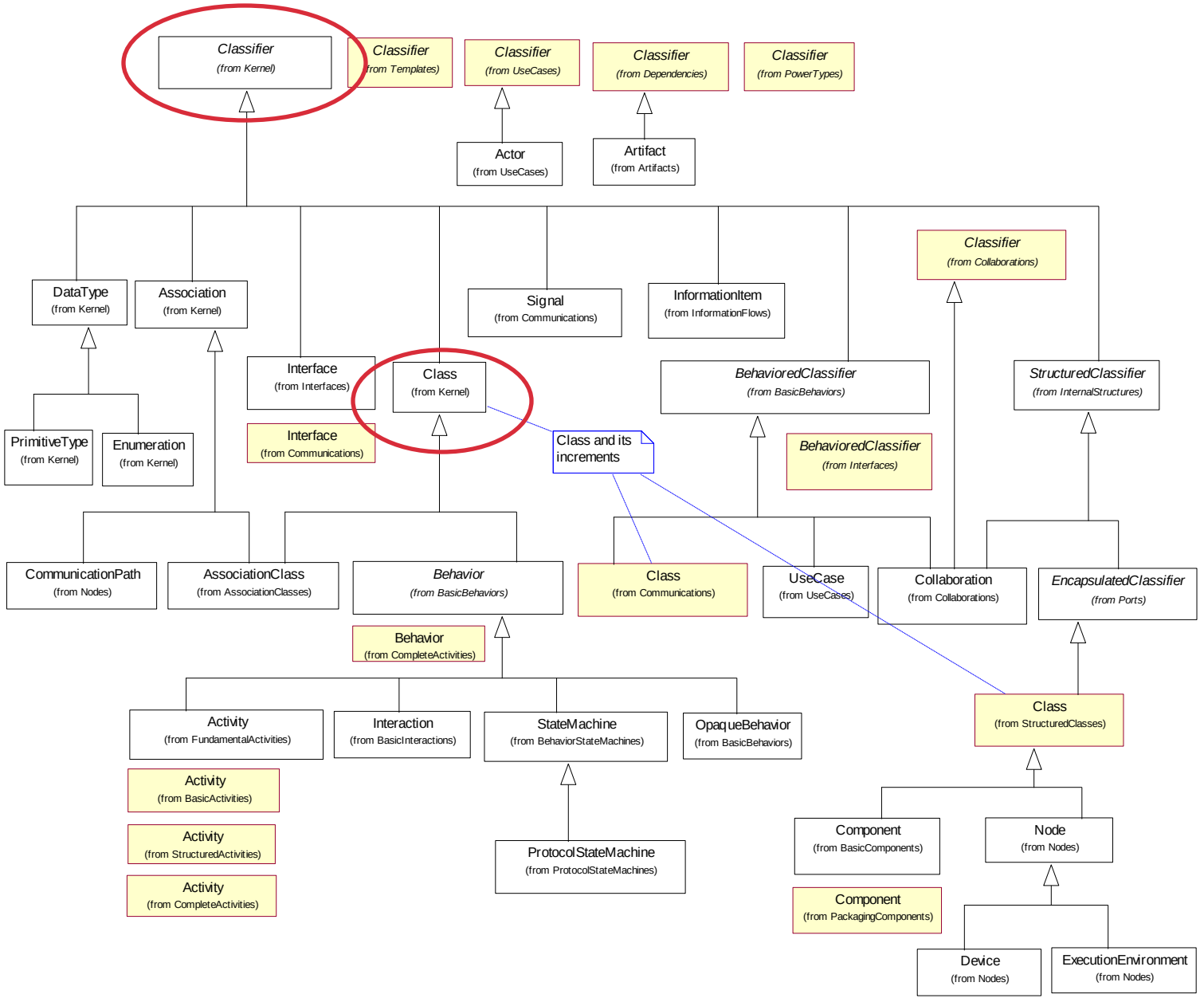
UML Spezifikation (1)

- Erhältlich über www.uml.org
- Gliedert sich in zwei Dokumente:
  - ⇨ Infrastucture (http://www.omg.org/docs/formal/05-07-05.pdf)
    Definition der grundlegenden Sprachelemente.
  - ⇨ Superstructure (http://www.omg.org/docs/formal/05-07-04.pdf)
    Definition der Diagrammtypen
- Überwiegend in UML Klassendiagrammen und OCL (Object Constraint Language) spezifiziert.

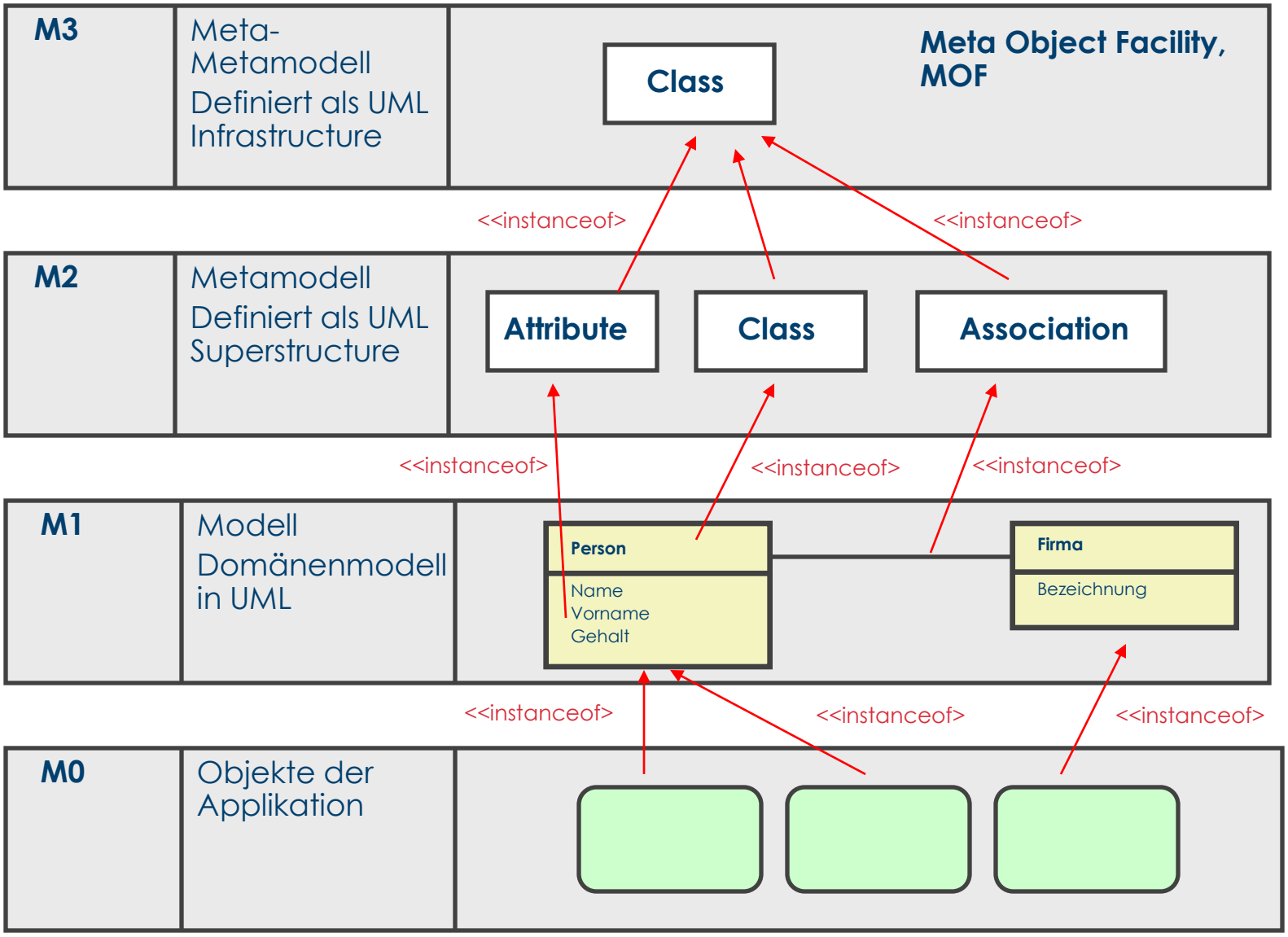
UML Superstructure: Class



UML Spezifikation: Classifier



UML Spracharchitektur



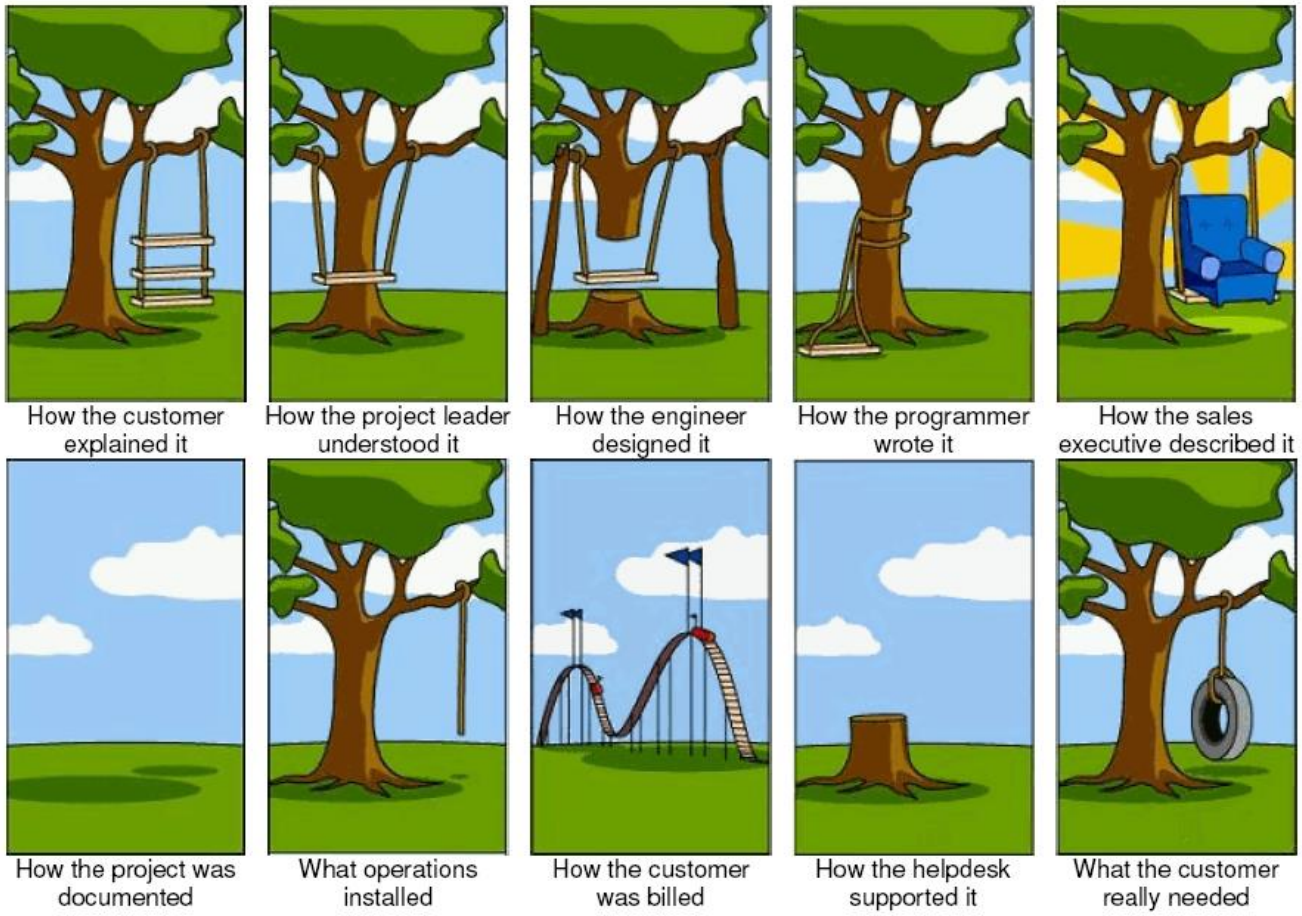
Anmerkungen zur UML

- In der Praxis werden die verschiedenen Diagramme unterschiedlich stark eingesetzt
- Werkzeuge spielen eine wichtige Rolle
- Nicht alle Werkzeuge unterstützen die UML 2.0
- Praktisch niemand kennt und nutzt die UML vollständig

Use Cases

Nutzen von Use Cases

- Formalisierung und Beschreibung der funktionalen Anforderungen



Quelle: Alan Chapman

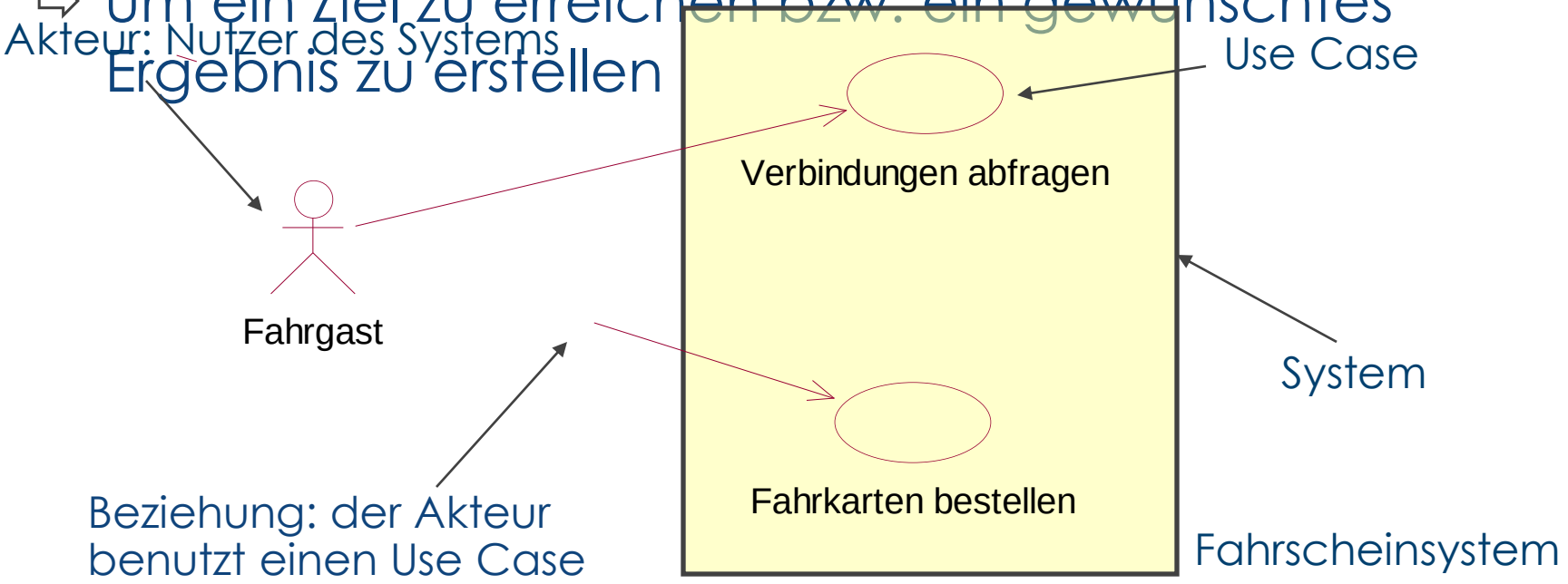
Definition

- "A use case is a piece of functionality in the system that gives a user a result of value"
- "All the use cases together make up the use case model, which describes the complete functionality of the system"
- "A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor"

Aus "The Unified Software Development Process", Ivar Jacobson u.a., Addison-Wesley, 1999

Use Case (1)

- Ein Use Case
  - ⇨ beschreibt mehrere zusammenhängendende Aktivitäten,
  - ⇨ die von einem oder mehreren Akteuren durchgeführt (genutzt) werden,
  - ⇨ um ein Ziel zu erreichen bzw. ein gewünschtes Ergebnis zu erstellen



Use Case (2)

- Use Cases beschreiben aus Anwendersicht den **Systemnutzen**
- Einfaches **Kommunikationsmedium** mit "Stakeholdern" (Anwendern, Kunden, Management, ...)
- Die Gesamtmenge der Use Cases beschreibt das **System**, das an der **Systemschnittstelle beobachtet** werden kann.
- Abgrenzung zur **Aktivität**: ein Use Case umfasst viele Aktivitäten

Use Case (3)

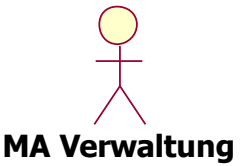
- Bezeichnung durch **Substantiv + Verb (Infinitiv)** z.B.
  - ⇨Einstellung durchführen
  - ⇨Prämie berechnen, usw.
- (Geschäfts-)**Prozesse** können als Use Case modelliert werden
- Systemgrenzen werden festgelegt
- Systemnutzer (=Akteure) werden festgelegt
- **Einsatzmöglichkeiten**: Ist-Analyse, Soll-Analyse und Spezifikation
- Achtung: ein Use Case ist kein Softwaremodul!
- Strukturierung der funktionalen Anforderungen

Use-Case-Diagramm

- dient der **Visualisierung** und als Übersicht über die vorhandenen Use Cases und Akteure
- enthält die **Beziehungen** zwischen Akteuren und Use Cases
- Tipp: mehrere Use-Case-Diagramme erstellen!

  ⇨ Jedes Use-Case-Diagramm sollte einen wichtigen Aspekt des Systems wiederspiegeln!

Akteure

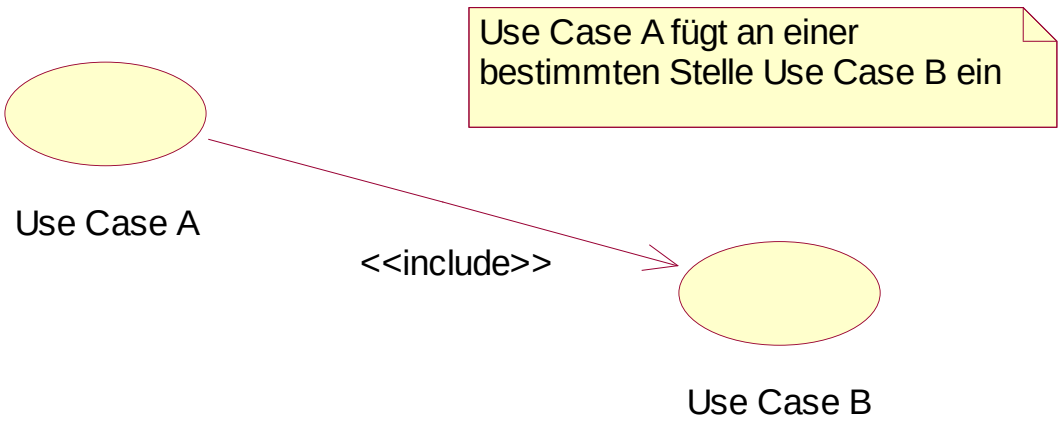


Akteure sind

- **Rollen** (Personen oder Systeme, z.B. Benutzer, Rechner, externe Datenbanken, auch Ereignisse oder Termine)
  - ⇨Akteure werden im Begriffslexikon beschrieben
  - ⇨Akteure werden im Use-Case-Diagramm eingezeichnet
  - ⇨Akteure werden im Diagramm durch Assoziationen mit den Use Cases verbunden, an denen sie beteiligt sind
- **selbst nicht Teil** des zu beschreibenden Systems,
- direkt mit einem oder mehreren Use Cases verbunden und **kommunizieren** mit diesem
- in der Regel **aktiv**, das heißt sie können Vorgänge anstoßen

Achtung: es werden nur die Akteure mit einem Use Case verbunden, die an der Interaktion mit dem System selbst teilnehmen!

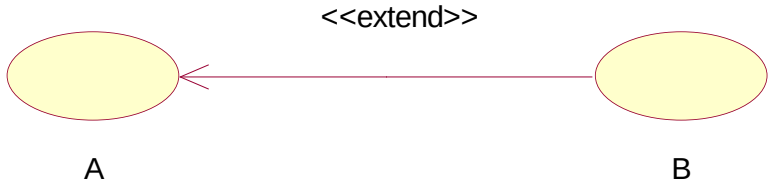
Include-Beziehung



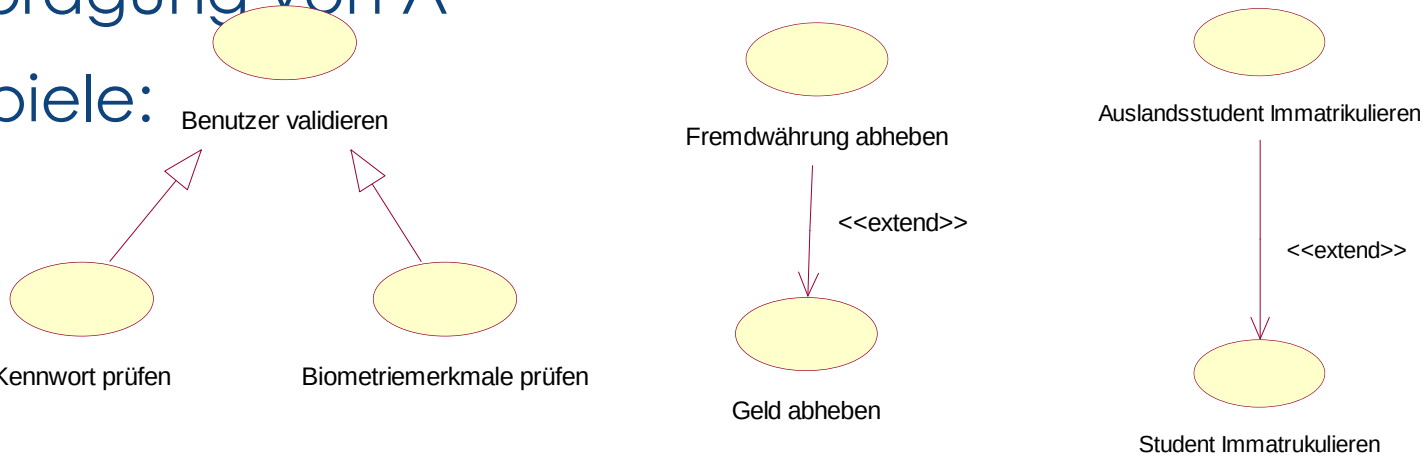
- Use Case A fügt Use Case B an einer gekennzeichneten Stelle ein
- Use Case B kennt Use Case A nicht
- Mehrfach vorkommende Use-Case-Abschnitte werden nur einmal beschrieben
- Tipp: man sollte diese Dekomposition nicht übertreiben! Jeder Use Case soll erkennbaren Nutzen behalten.

Extend und Generalisierung

Es gibt weitere Beziehungen, die **vorsichtig** verwendet werden sollten:

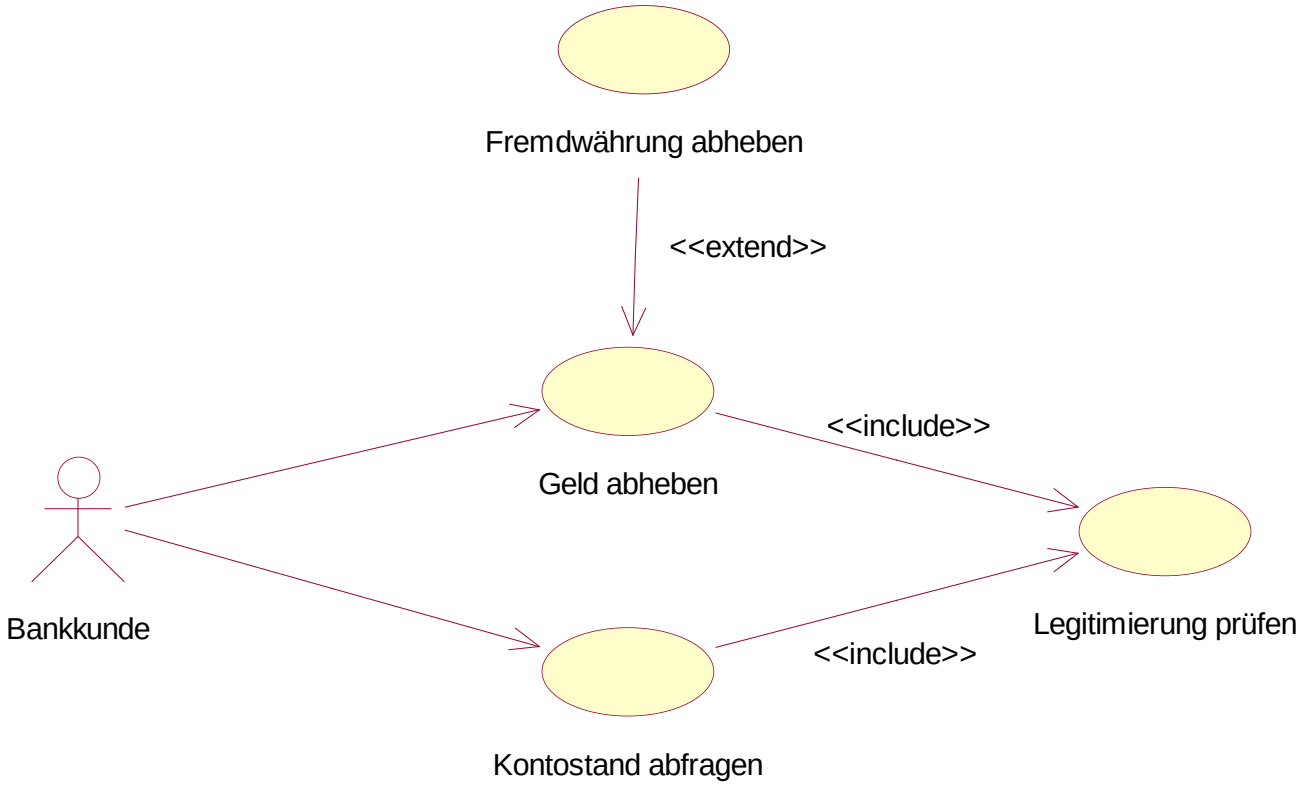


- **extend**-Beziehung: B erweitert A an einer dort festgelegten Stelle
- **Generalisierungsbeziehung**: B ist eine spezielle Ausprägung von A
- Beispiele:

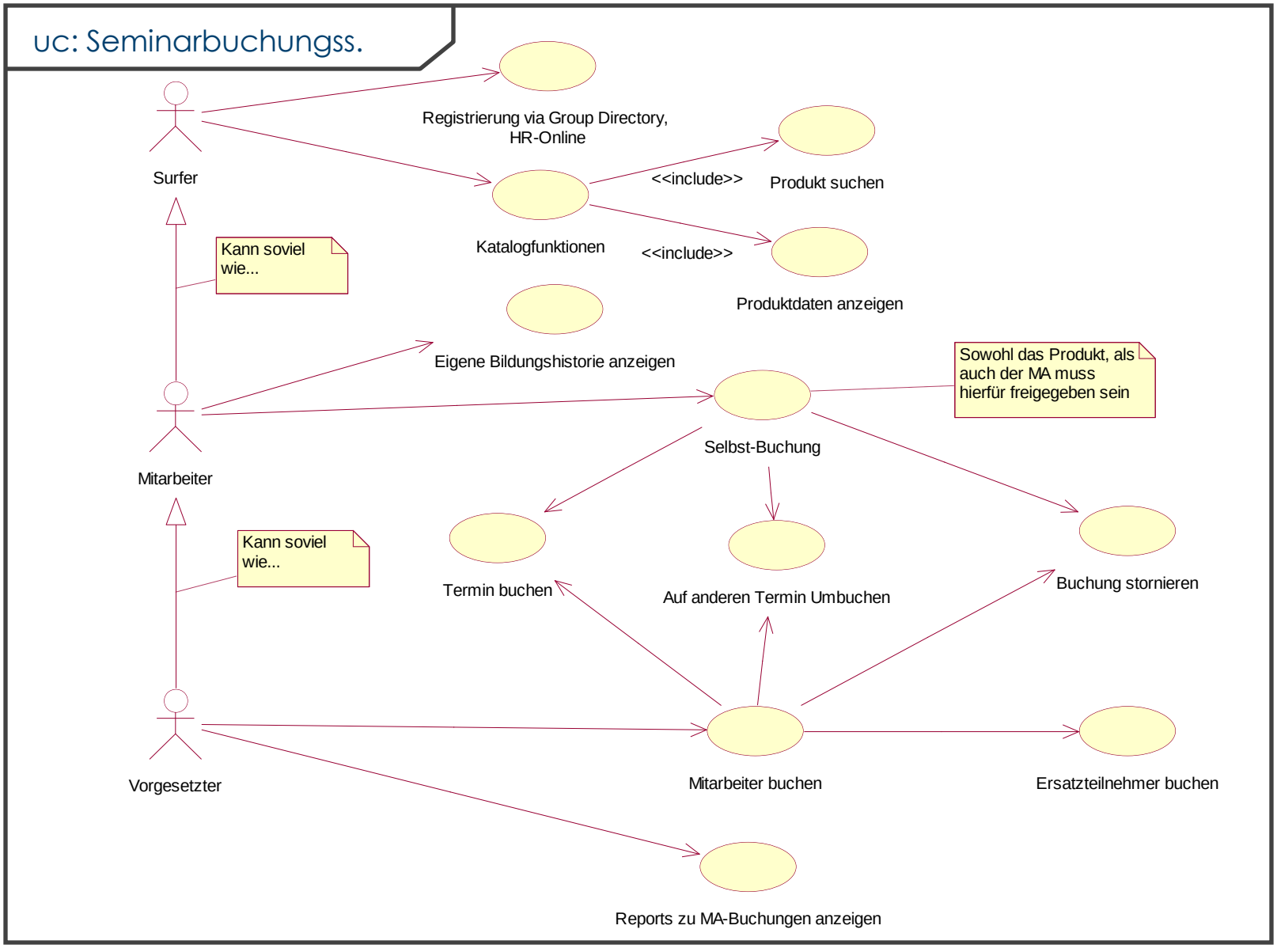


Achtung: die Semantik beider Beziehungen ist nicht scharf spezifiziert!

Beispiel: Bankomat



Beispiel: Seminarbuchungssystem



Use-Case-Beschreibung (1)

Beschreibung erfolgt

- in verständlicher Sprache
- mit konkretem Nutzen für den initiierenden Akteur
- präzise und eindeutig
- mit überprüfbaren Ergebnissen
- mit definierten Begriffen (Begriffslexikon)
- semi-formal (strukturiert)
- einheitlich

Use Case Dokumentation

Name: Seminar buchen
Akteure: Kunde

Vorbedingung:
Seminar ist freigegeben und hat noch ausreichend Plätze frei

Regulärer Ablauf:
Der Kunde selektiert ein Seminar und ...

Nachbedingung:
Der Kunde ist auf das entsprechende Seminar gebucht

Alternative Abläufe:
- das Seminar ist ausgebucht
- der Kunde hat keine Buchungsberechtigung

Use-Case-Beschreibung (2)

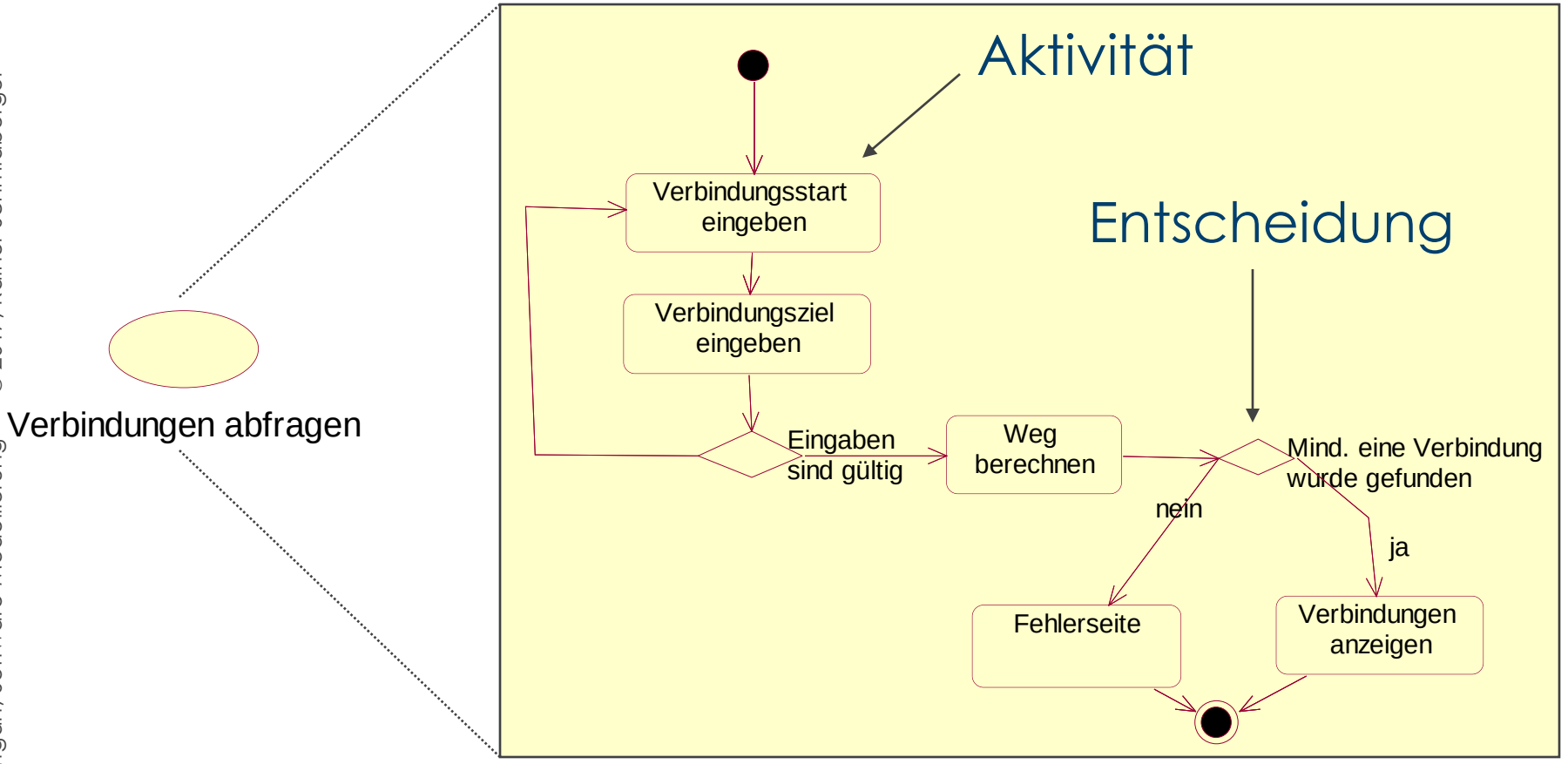
- **Name des Use Case**
- **Beteiligte Akteure**
- **Vorbedingung**
  - ⇨Voraussetzungen vor der Ausführung des Use Case
- **Regulärer Ablauf**
  - ⇨Beschreibung einzelner Aktionen innerhalb des Use Case in Form verbaler Szenarien (Einzelaktivitäten)
  - ⇨Evtl. Diagramme und Entscheidungstabellen
  - ⇨Begriffe sorgfältig verwenden (Begriffslexikon)
- **Nachbedingung**
  - ⇨Zusicherungen, die nach Use-Case-Ende eintreffen müssen
- **Alternative Abläufe**
  - ⇨Fehlerfälle
  - ⇨Sonderfälle

Use Case Beschreibung (3)

- **Name des Use Case**
  - ⇨Personalausweis beantragen
- **Vorbedingung**
  - ⇨Antragsteller hat sich legitimiert und entsprechende Aktion angstoßen. Es handelt sich nicht um einen Erstantrag
- **Regulärer Ablauf**
  - ⇨Berechtigungen des Antragstellers werden geprüft (Siehe Verordnung XY)
  - ⇨Zuständiges Amt wird über die Wohnsitzangaben ermittelt. Für Wohnsitzlose gilt Sonderregelung 4711.
  - ⇨Gebührenrechnung wird erstellt. Abwicklung abhängig vom Antragsteller über Kreditkarte oder Bankeinzug
  - ⇨Zusammenstellen des Antragpakets und weiterleiten an Stelle 0815
- **Nachbedingung**
  - ⇨Der Antragsteller hat jetzt den Status "beantragt", weitere Anträge können solange nicht entgegengenommen werden
- **Alternative Abläufe**
  - ⇨Erstanträge werden über use Case ABC behandelt

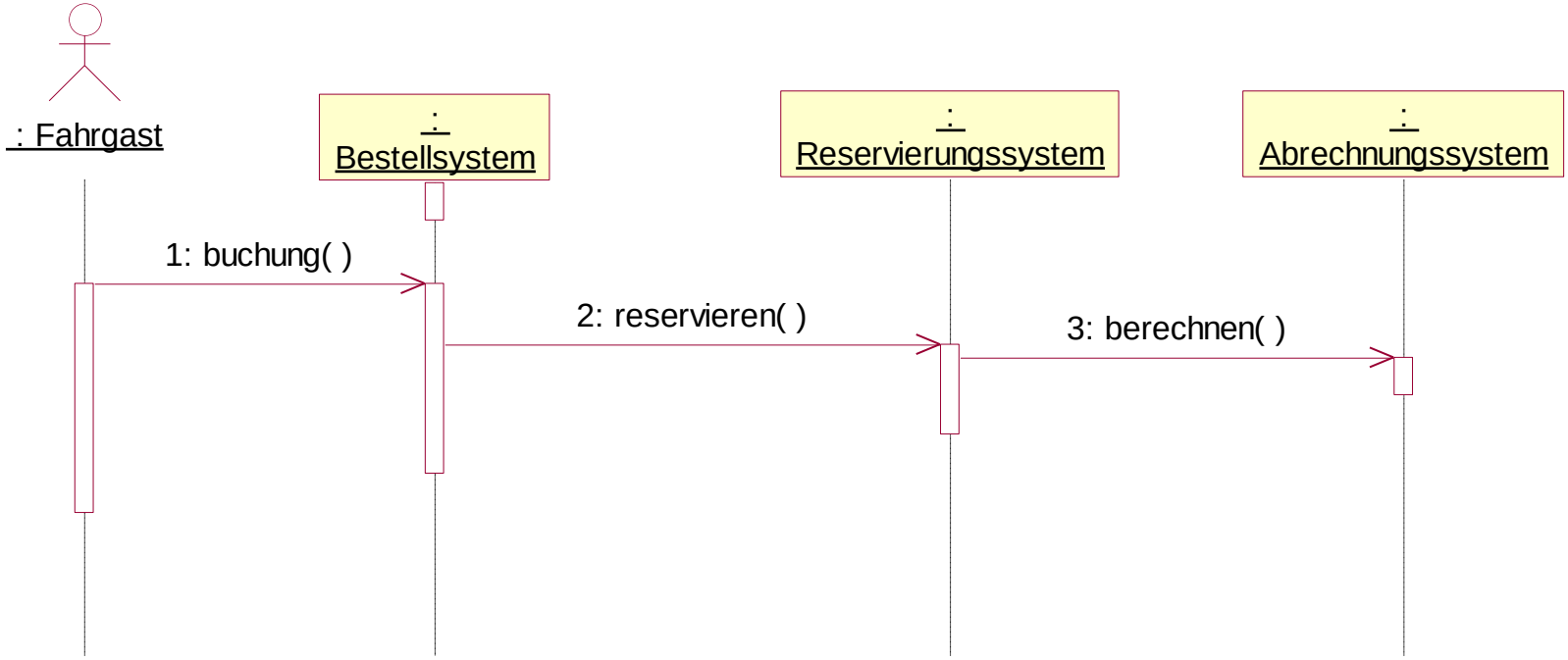
Beschreibung durch Aktivitätsdiagramm

- Beschreibt (grafisch) den Ablauf eines Use Case



Beschreibung durch Sequenzdiagramm

- Beschreibt (grafisch) die beteiligten Instanzen und den zeitlichen Ablauf eines Use Case



Vor- und Nachteile

Vorteile

- Einfache Darstellung, leicht verständlich
- Use Cases sind innerhalb der UML klar beschrieben
- Es sind (sehr gute) Werkzeuge verfügbar
- Erweiterungsmöglichkeit um Aktivitätsdiagramme und Sequenzdiagramme
- Greifen den verbreiteten Prozess-Gedanken auf

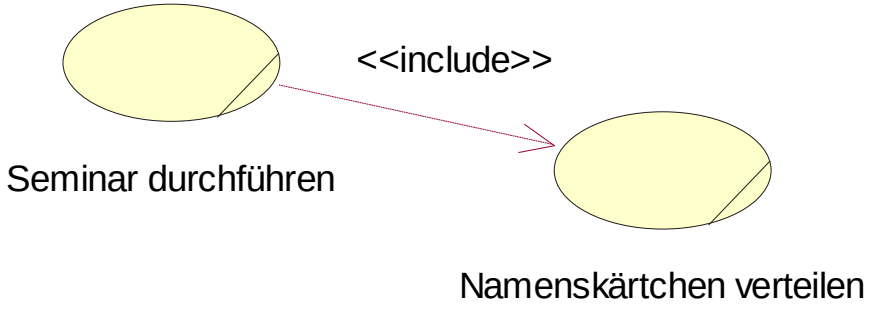
Nachteile

- Beschreibung ist nicht formalisiert
- Konsistenzen sind automatisch nicht prüfbar
- Übergang zur Modellierung der fachlichen Klassen ist brüchig

Business Use Case

Business Use Case

- Beschreibung der Fachlichkeit ohne Berücksichtigung, welche Teile des Prozesses mit IT-gestützt ablaufen und welche nicht
- Unterscheidung „manuell“ oder „IT-gestützt“ über den <<stereotyp>>



Use Case

- Eingrenzung auf die IT-gestützten Prozesse

⇨ Bei e-Commerce-Systemen entsprechen die Business Use Cases den Use Cases

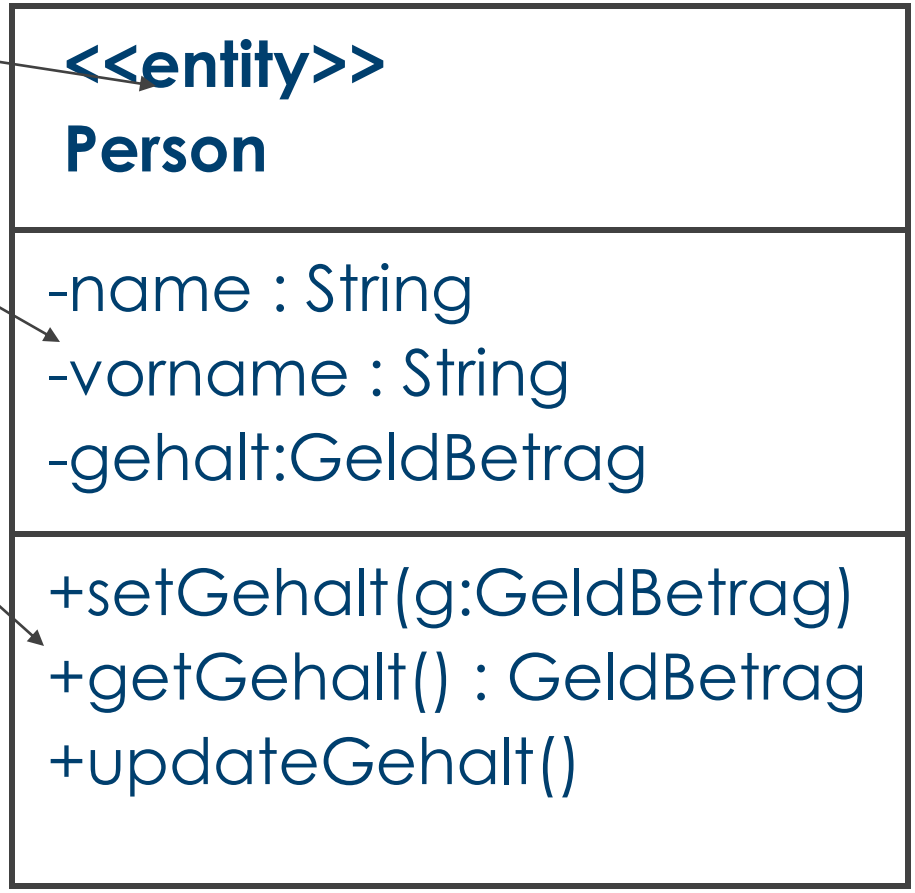
Klassen-Diagramme

UML Klassen (1)

- Die Darstellung von Klassen erfolgt als Rechteck mit den Bereichen
  - ⇨ Klassenname
  - ⇨ Attributliste
  - ⇨ Methodenliste

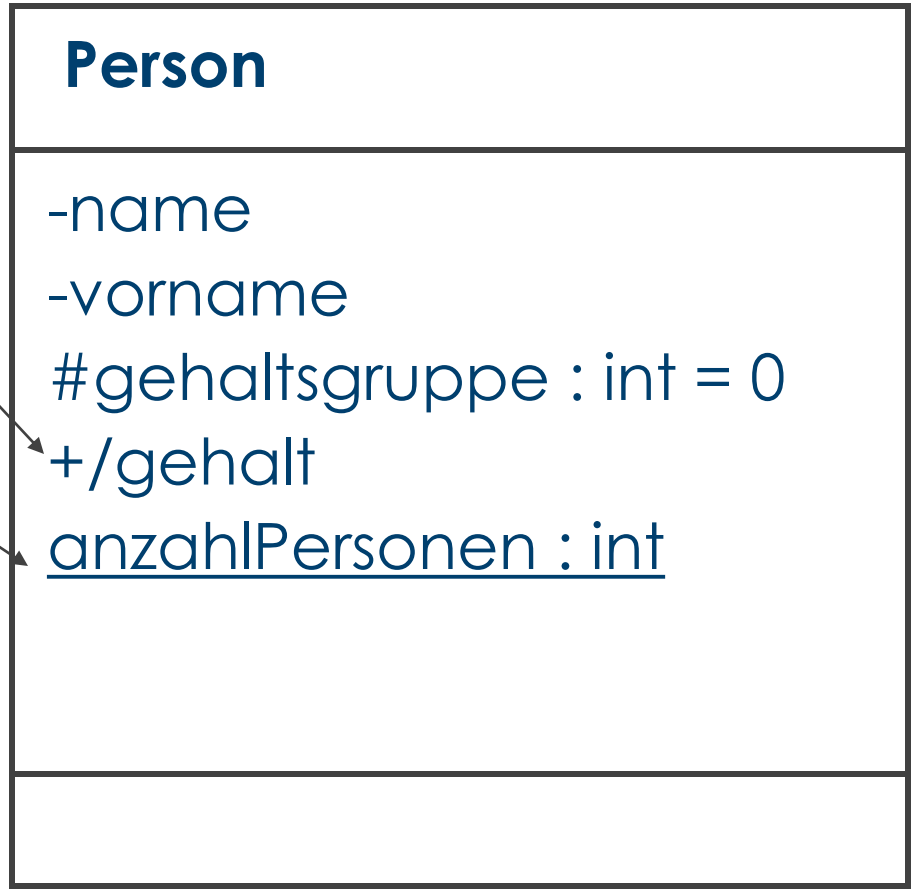
UML Klassen (2)

- Klassenname mit stereotype Angabe
- Attribute mit Datentypen und Sichtbarkeiten und Vorbelegungen
- Methoden mit Signaturen und Sichtbarkeiten
- Sichtbarkeiten:
  - ⇨ public (+)
  - ⇨ protected (#)
  - ⇨ private (-)
- Die Datentypen sind von der verwendeten Programmiersprache abhängig!



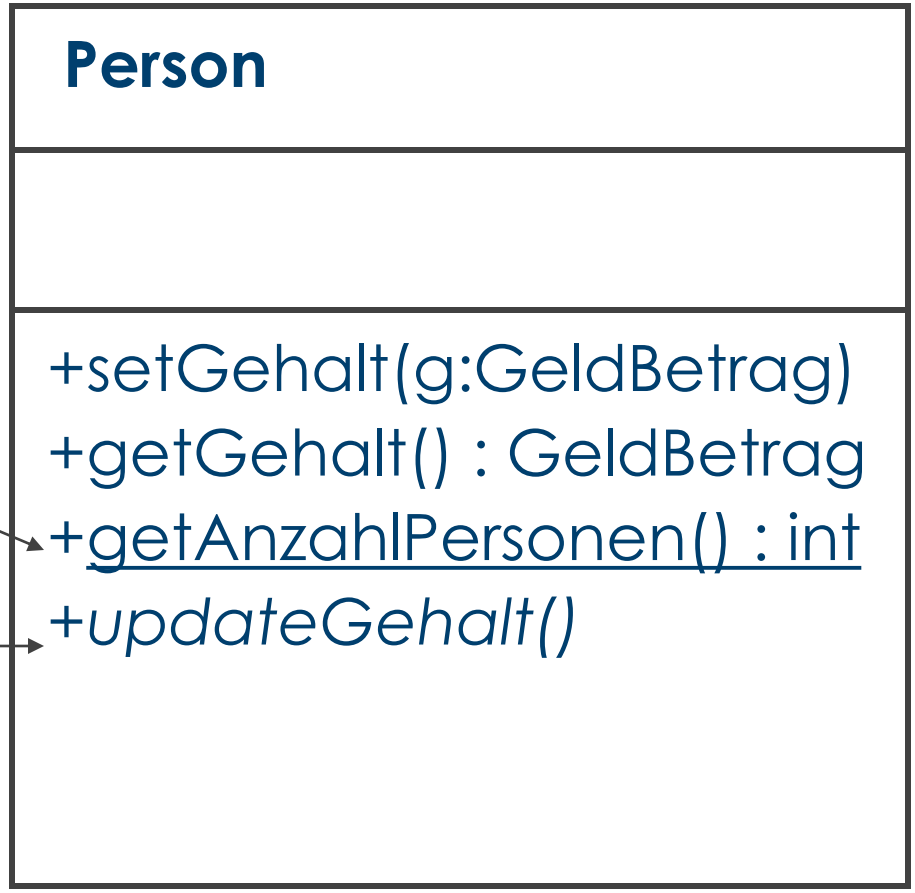
Attribute

- Berechnetes Attribut (Derived). Berechnungsformel ist nicht Teil des UML-Modells
- Klassenattribut: Attribut steht allen Objekten der Klasse Person nur einmal zur Verfügung



Methoden

- Signaturen werden oft nicht angegeben
- Private Methoden werden oft nicht angegeben
- Klassenmethode (Aufruf über Klassenbezeichner, unterstrichen)
- Abstrakte Methode (wird kursiv gedruckt), erzwingt Implementierung in den Kindklassen

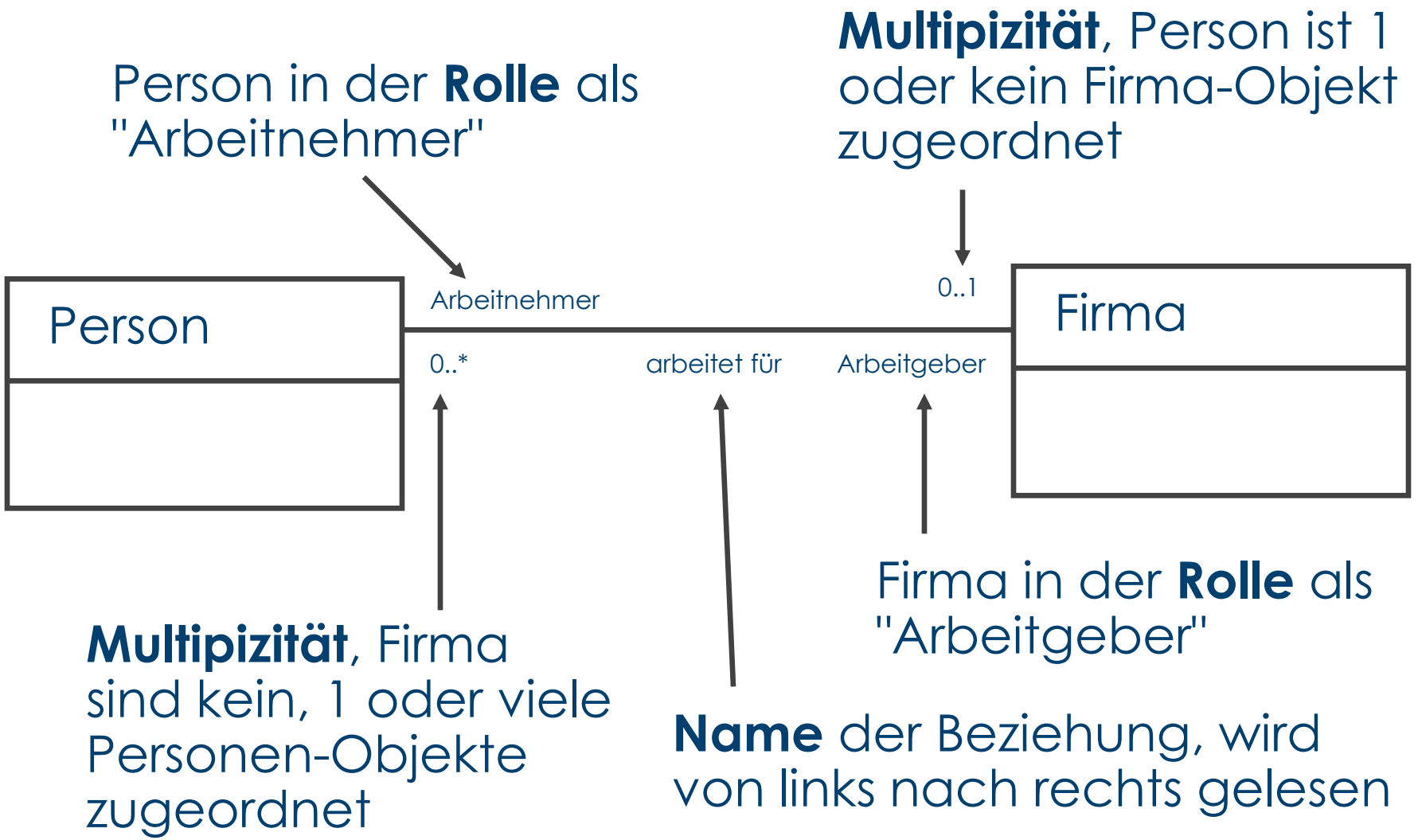


Folie 69

Assoziationen

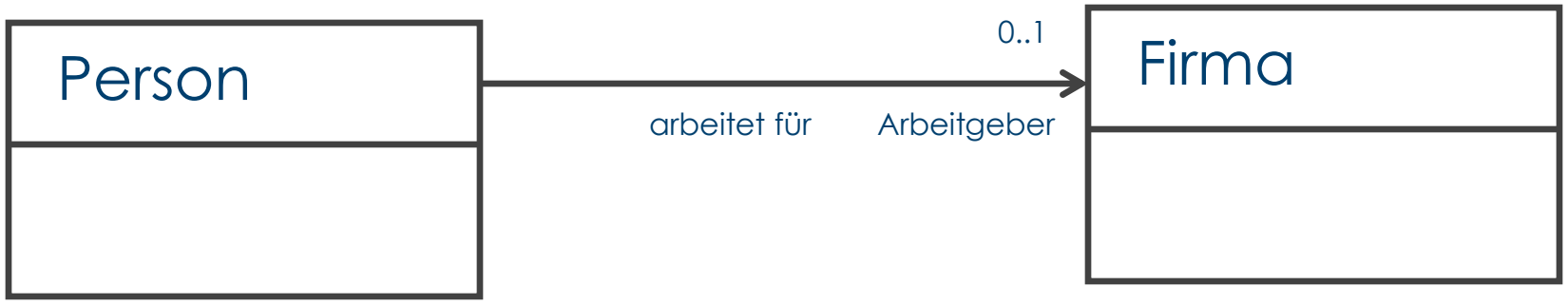
- "Beziehungen", "Verbindungen" oder "Navigationspfade" zwischen Klassen werden als Assoziation bezeichnet
- Verbindungen zwischen Objekten werden als Link bezeichnet
- In der UML werden Assoziationen sehr differenziert beschrieben
- Assoziationen sind logische Beziehungen, sie müssen nicht notwendigerweise durch ein Programmkonstrukt direkt im Code implementiert sein

Assoziationen



Navigierbarkeit

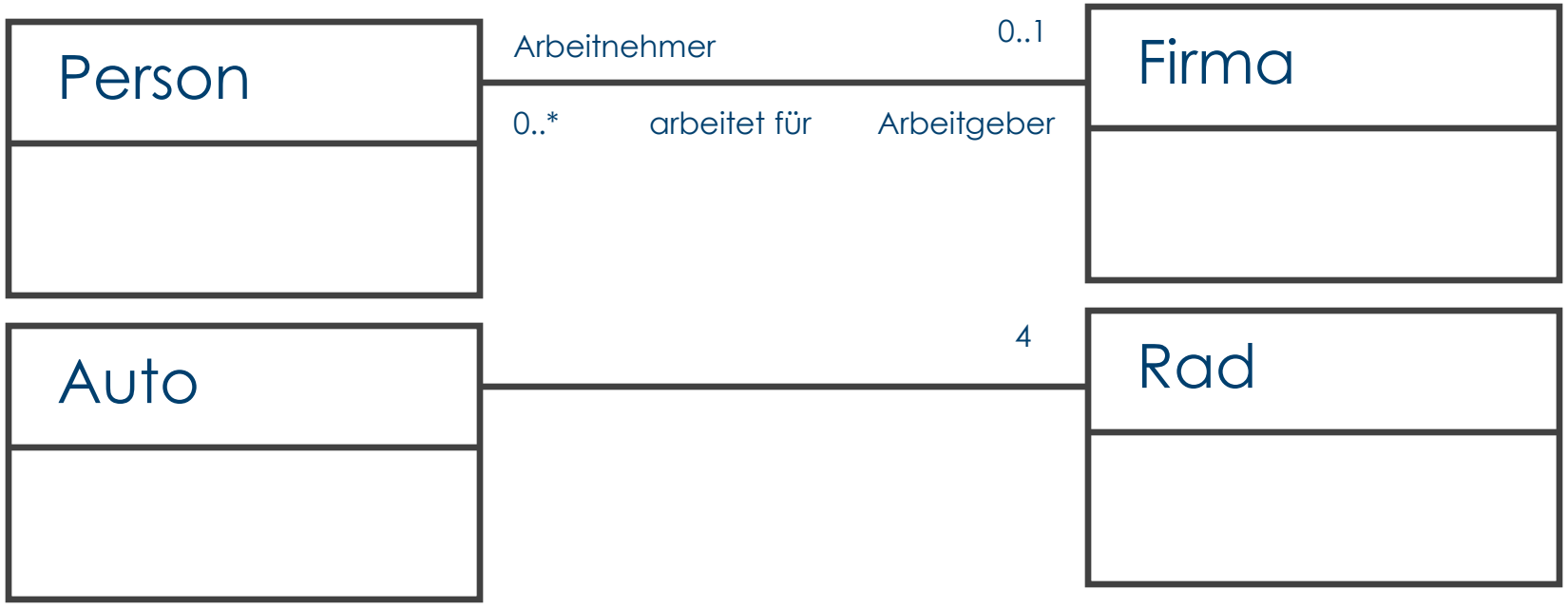
- Die Pfeilrichtung zeigt die Navigierbarkeitsrichtung an.
- Beidseitig navigierbare Assoziationen haben aber keine Pfeile.
- Nichtspezifizierte Navigierbarkeit hat ebenso keine Pfeile



⇨ Objekte der Klasse Person "kennen" ihren Arbeitgeber, Objekte der Klasse Firma "kennen" aber ihre Mitarbeiter nicht.

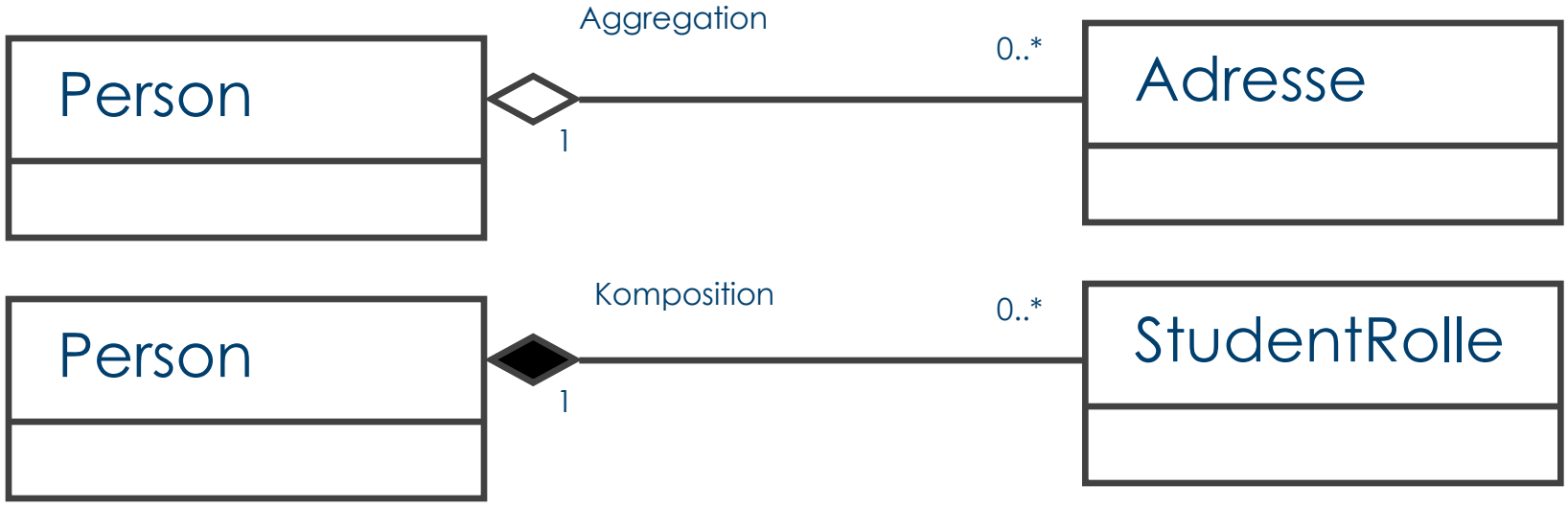
Multiplizität

- Die Multiplizität zeigt an, wie viele Objekte der jeweiligen Klasse an der Assoziation teilnehmen
  - ⇨ Typische Multiplizität: 0..1, 1, 0..*, 1..*
- Es können auch spezielle Multiplizitäten verwendet werden wie z.B. 5, 29 oder 5..290



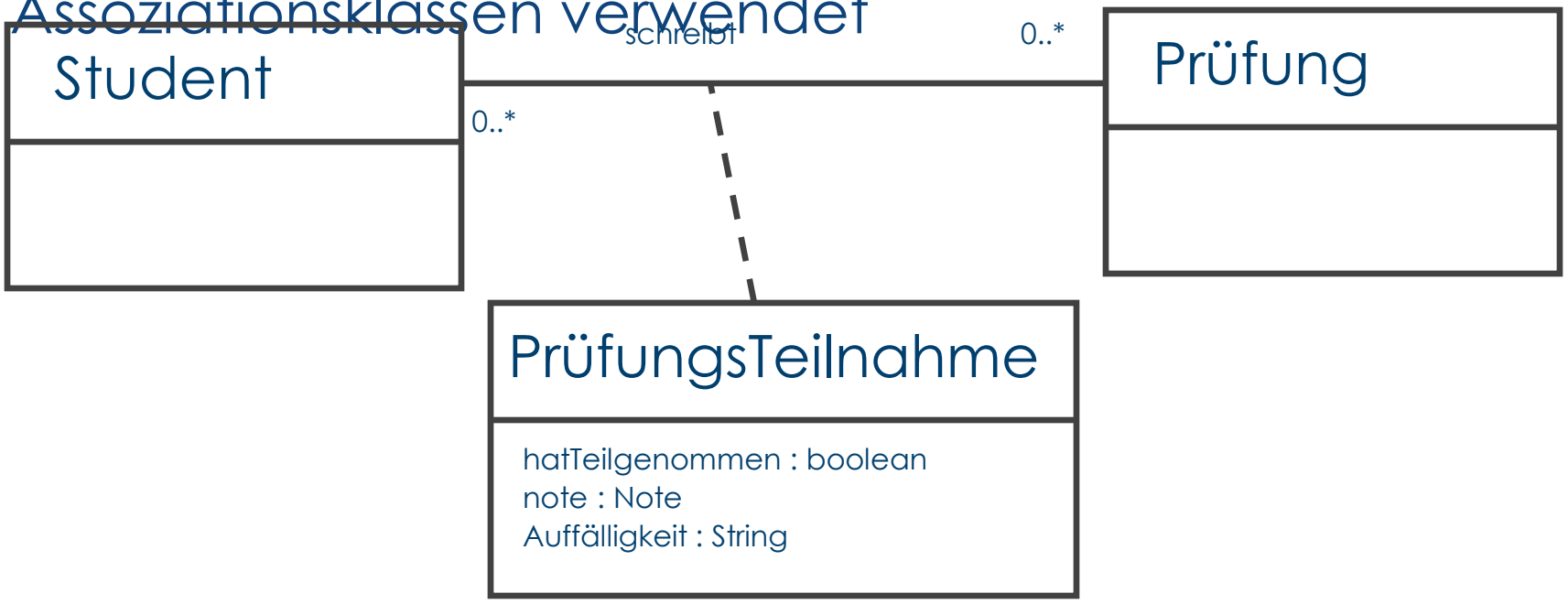
Aggregation und Komposition

- Ganzes-Teil-Assoziation
- Aggregierte Objekte werden von dem besitzenden Objekt dominiert
- I.Allg. ist kein externer Zugriff auf aggregierte Objekte möglich
- Komposition: Die Lebensdauer (Instanzierung und Freigabe) werden vom dominierenden Objekt beherrscht



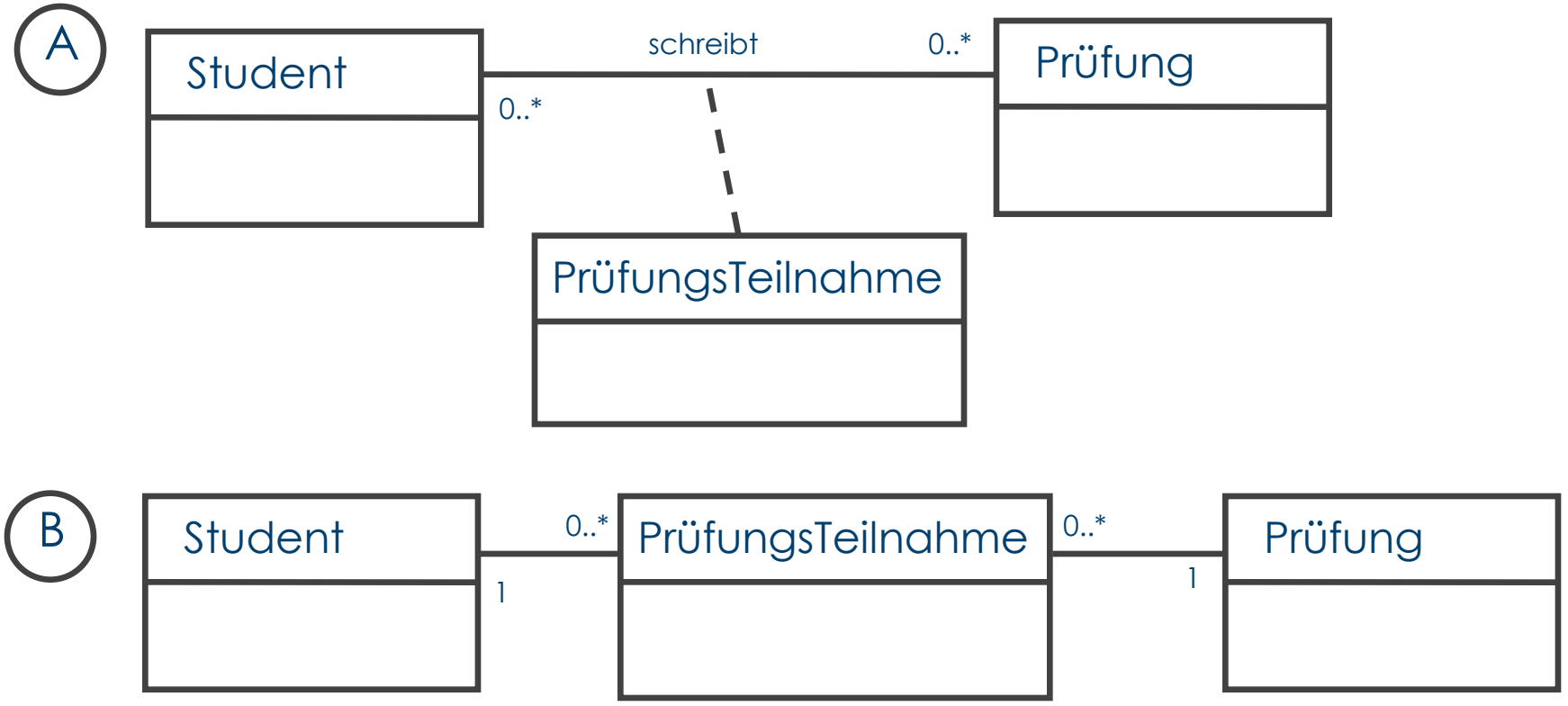
Assoziationsklassen (1)

- Assoziationen selbst können über Attribute (und auch Methoden) verfügen
- Die Assoziationsklasse repräsentiert die Assoziation
- Insbesondere bei n:m-Assoziationen werden häufig Assoziationsklassen verwendet



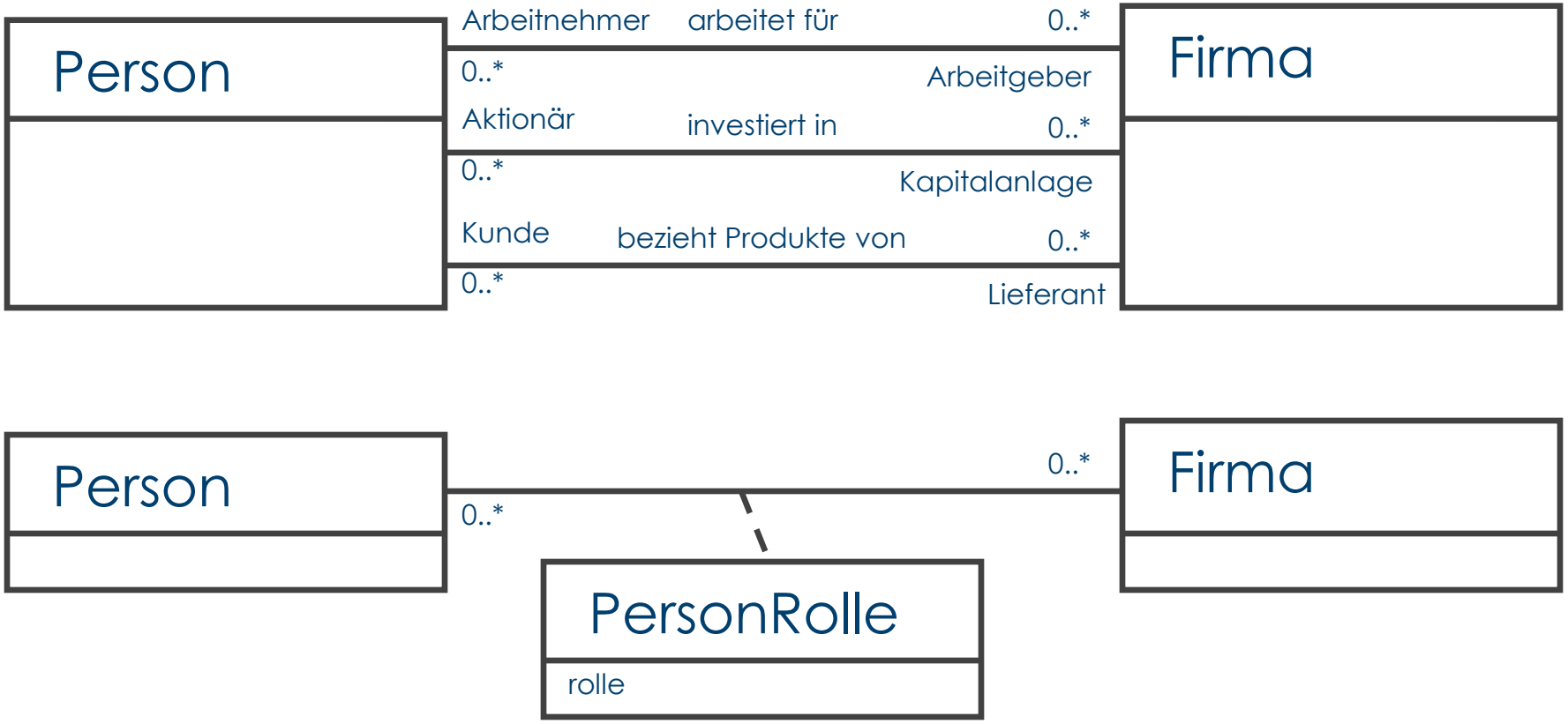
Assoziationsklassen (2)

- Eine n:m Assoziation mit Assoziationsklasse (A) ist semantisch mit einer 1:n und m:1-Assoziation (B) gleichzusetzen



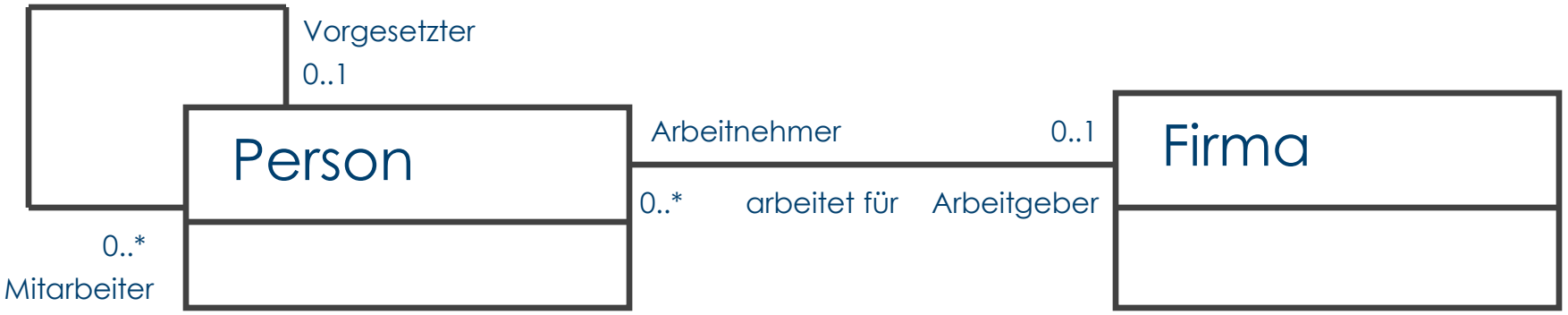
Umgang mit Assoziationen

- Bestehen mehrere Assoziationen zwischen zwei Klassen, so werden oft rollenbeschreibende Assoziationsklassen eingeführt



Rekursive Assoziationen

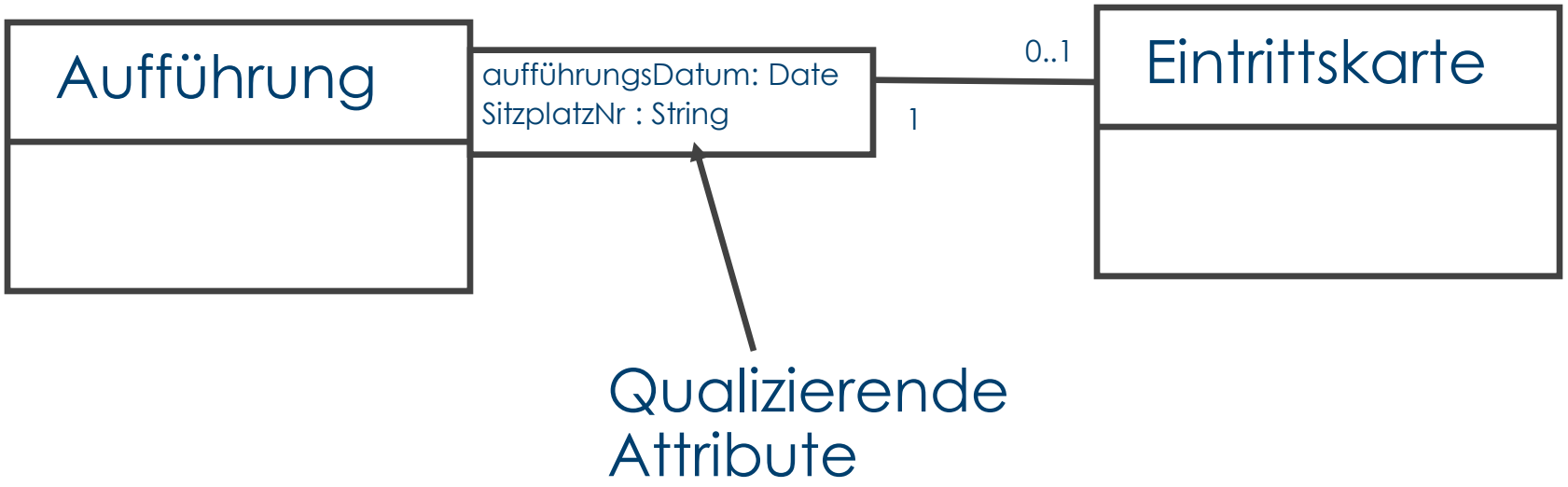
- Assoziationen können zur selben Klasse verbunden sein
- Damit ist nicht gemeint, dass ein Objekt mit sich selbst in Beziehung steht, sondern zu Objekten der selben Klasse!
- Rekursive Assoziationen können auch Assoziationsklassen haben



Folie 78

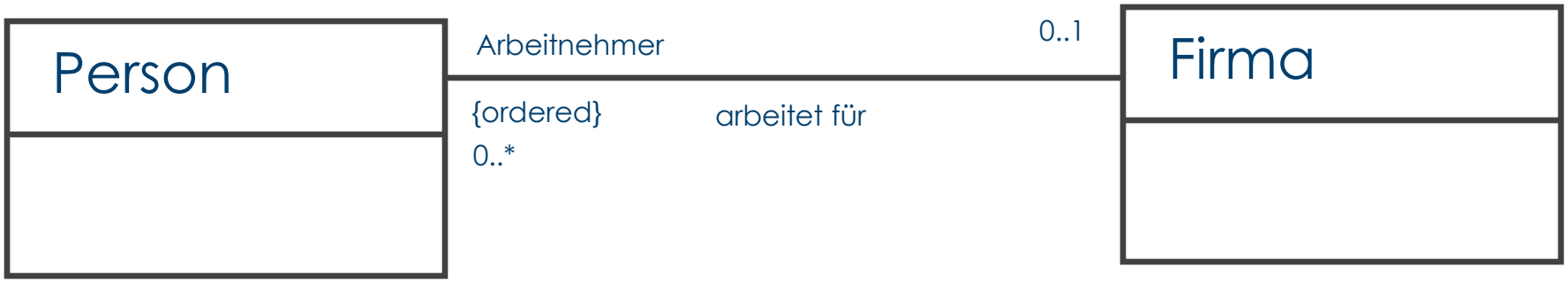
Qualifizierende Attribute

- Attribute, die wesentlich die Rolle eines Objektes in der Beziehung beschreiben oder überhaupt die Beziehung erst herstellen
- Die qualifizierenden Attribute werden der Assoziation nicht der Klasse zugeordnet



Assoziationseinschränkungen

- Spezielle Regeln einer Assoziation werden als Constraint bezeichnet
- Ab UML 2.0: Object Constraint Language OCL
- Die Darstellung erfolgt in { }
- Typische Beispiele sind Sichtbarkeit oder Änderbarkeit (z.B. ordered, addOnly)



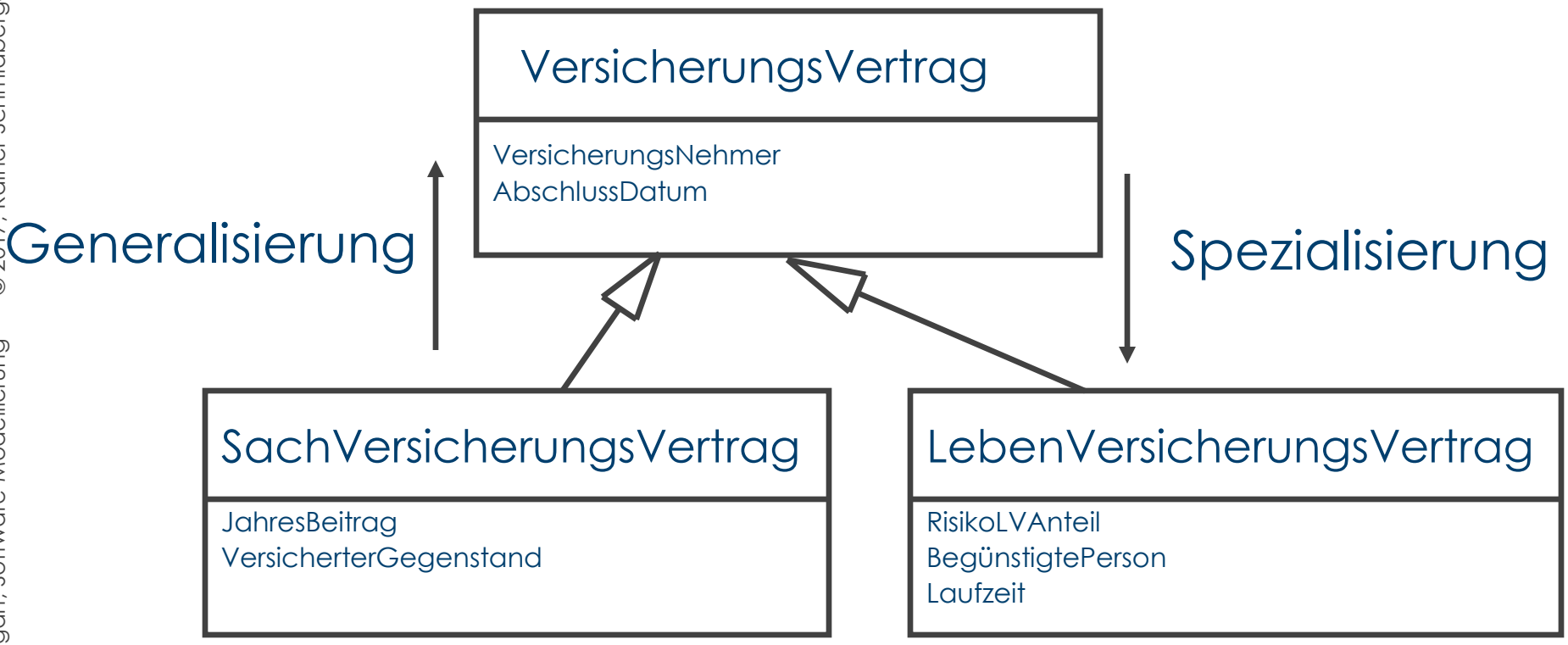
Stereotypes

- Stellen Erweiterungen des UML-Standards dar
- Vergeben einem UML-Element (wie z.B. einer Klasse) ein spezielle Ausprägung
- Darstellung des stereotyp in<< >> über dem Klassennamen
- Häufig verwendete Stereotypen:

<<boundary>>	Klassen, die mit der Außenwelt kommunizieren. Sie bilden Schnittstellen zwischen System und Umwelt
<<control>>	Klassen, die Interaktionen zwischen anderen Klassen steuern
<<entity>>	Klassen, die selbst passiv sind und keine Interaktionen auslösen (i.Allg. zur Datenhaltung)
<<utility>>	Keine Klasse, nicht instanzierbar, kapselt globale Variablen und Funktionen. Zur Abbildung Nicht-OO-Bibliotheken geeignet (z.B. Mathematik-Funktionen)

Vererbung

- Alle Eigenschaften (also Methoden, Attribute oder Beziehungen) einer Klasse (Vaterklasse) werden an die Kindklassen übertragen
- Die Kindklassen können weitere eigene Eigenschaften definieren und auch ererbte Eigenschaften überschreiben



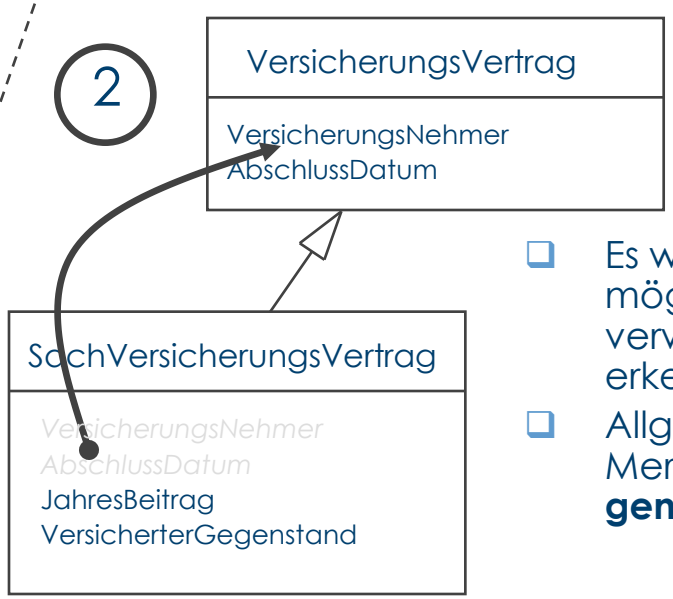
Folie 82

Entstehung von Vererbung

1

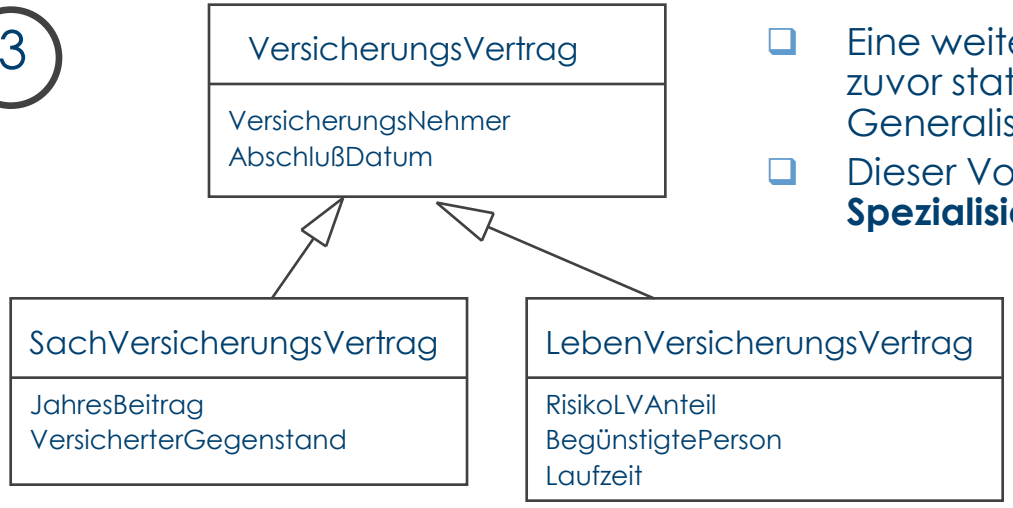
- Zunächst entsteht aus den bekannten Anforderungen eine konkrete Klasse.
- Motivation zur Vererbung besteht nicht

2



- Es wird eine mögliche Wieder-verwendbarkeit erkennbar
- Allgemeingültige Merkmale werden **generalisiert**

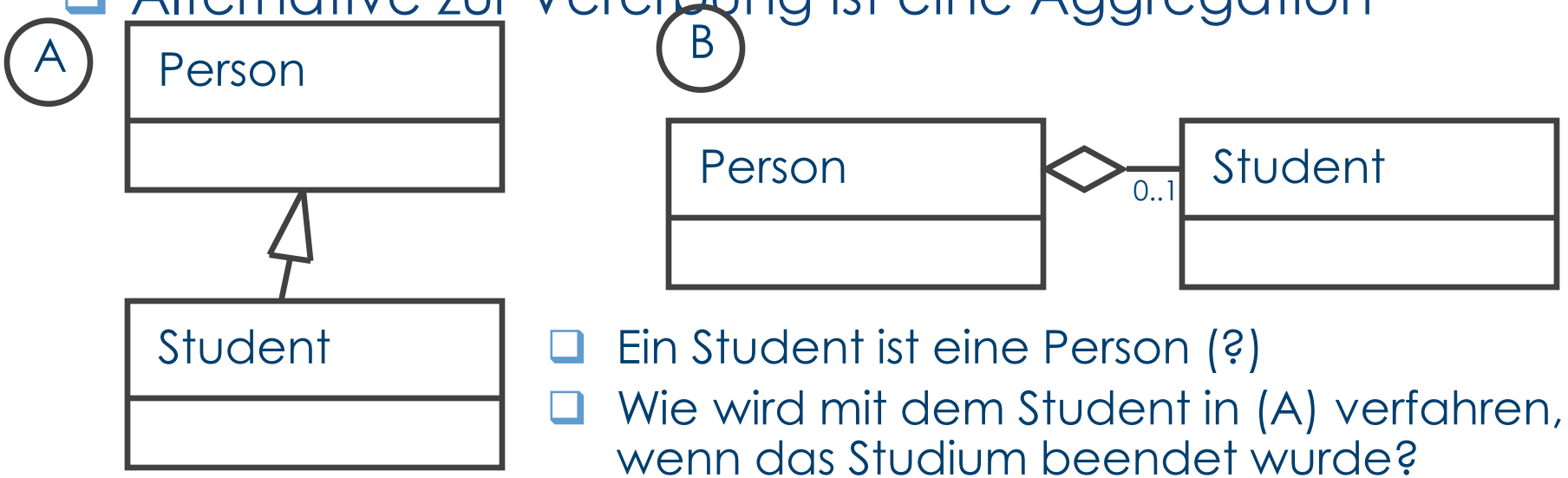
3



- Eine weitere Klasse nutzt die zuvor stattgefundene Generalisierung
- Dieser Vorgang wird nun als **Spezialisierung** bezeichnet

Vererbung vs. Aggregation

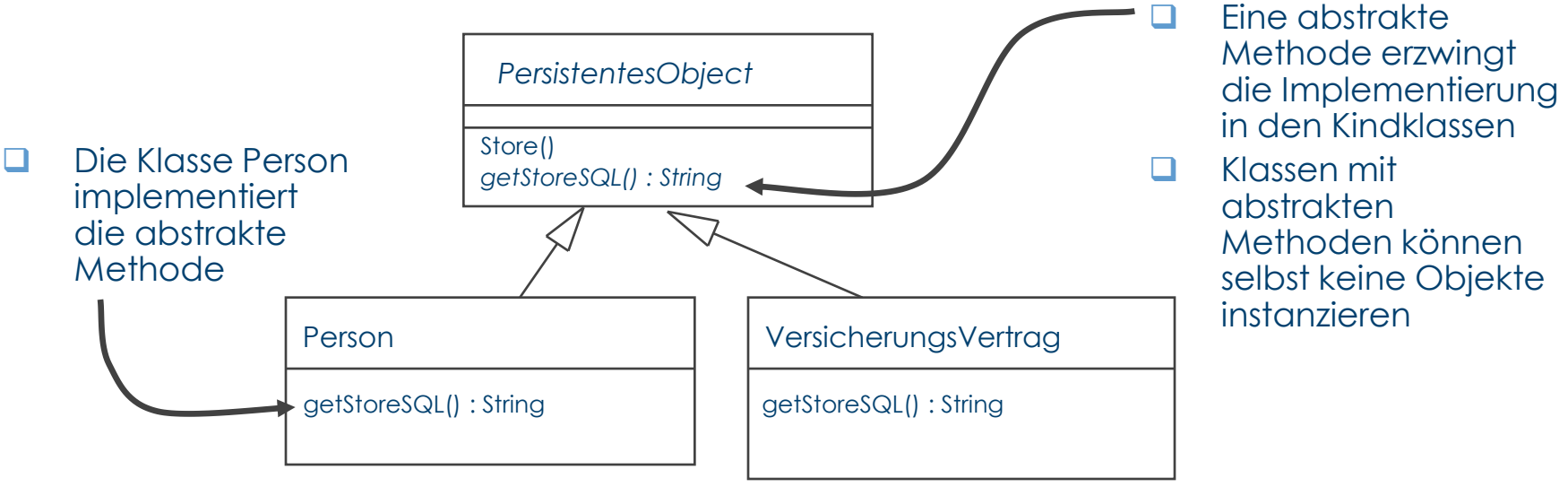
- Realisierung einer "ist-ein" Beziehung im Gegensatz zu einer "hat-ein" Beziehung der Assoziation
  - ⇨ Ein LV-Vertrag **ist** einer Versicherungsvertrag
  - ⇨ Ein Auto **hat** Räder
- Vererbung im fachlichen Kontext ist selten sinnvoll
- Alternative zur Vererbung ist eine Aggregation



- Ein Student ist eine Person (?)
- Wie wird mit dem Student in (A) verfahren, wenn das Studium beendet wurde?

Vererbung abstrakter Methoden

- Wird eine abstrakte Methode vererbt, wird deren Implementierung in den Kindklassen erzwungen
- Vorteil: lässt sich auch keine generische Implementierung festlegen, so ist doch ein gemeinsames Interface erzwingbar

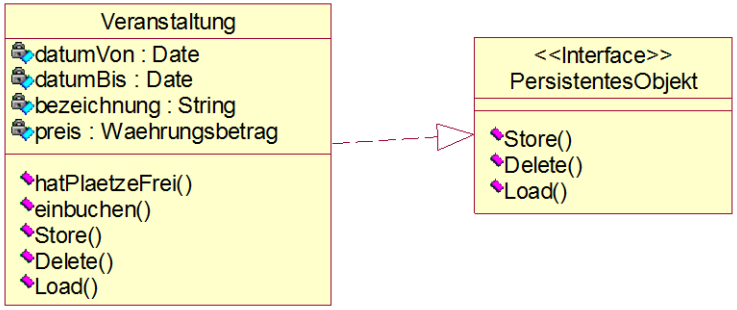


- Die Klasse Person implementiert die abstrakte Methode
- Eine abstrakte Methode erzwingt die Implementierung in den Kindklassen
- Klassen mit abstrakten Methoden können selbst keine Objekte instanzieren

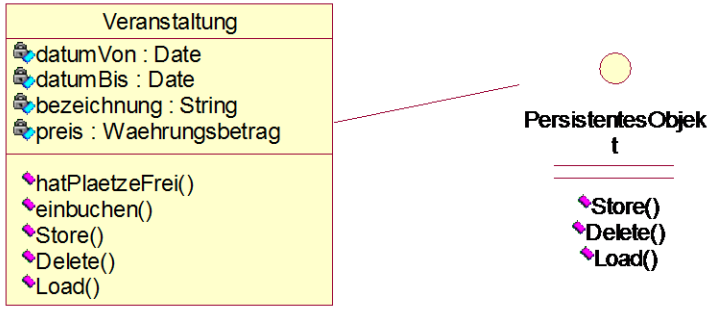
Interface

- Ein Interface ist einer Klasse mit ausschließlich abstrakten Methoden gleichzusetzen
- Interfaces werden "realisiert"
- Eine Klasse kann viele Interfaces realisieren
- Interfaces definieren Methoden, die von den Realisierungen implementiert werden müssen

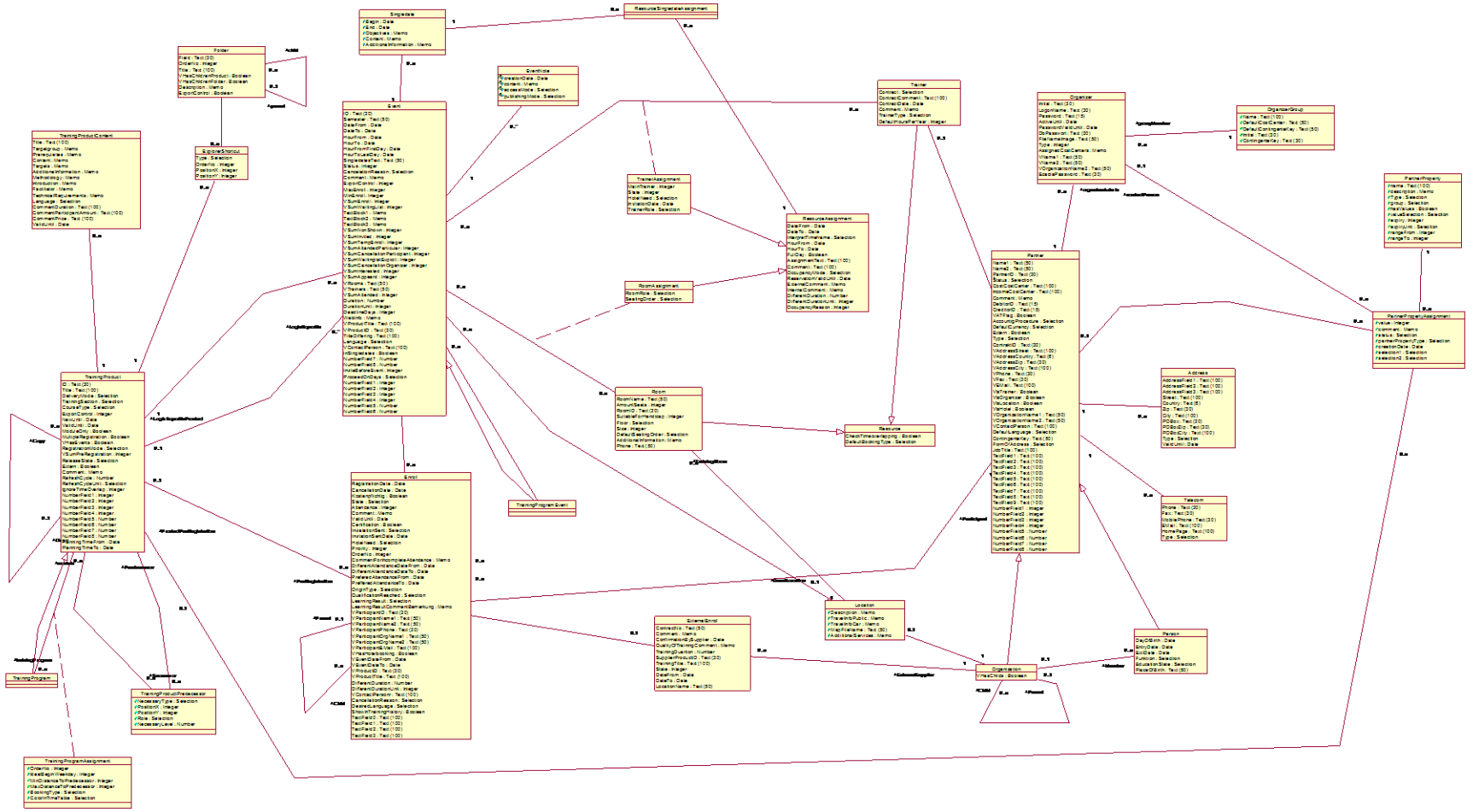
Label-Darstellung:



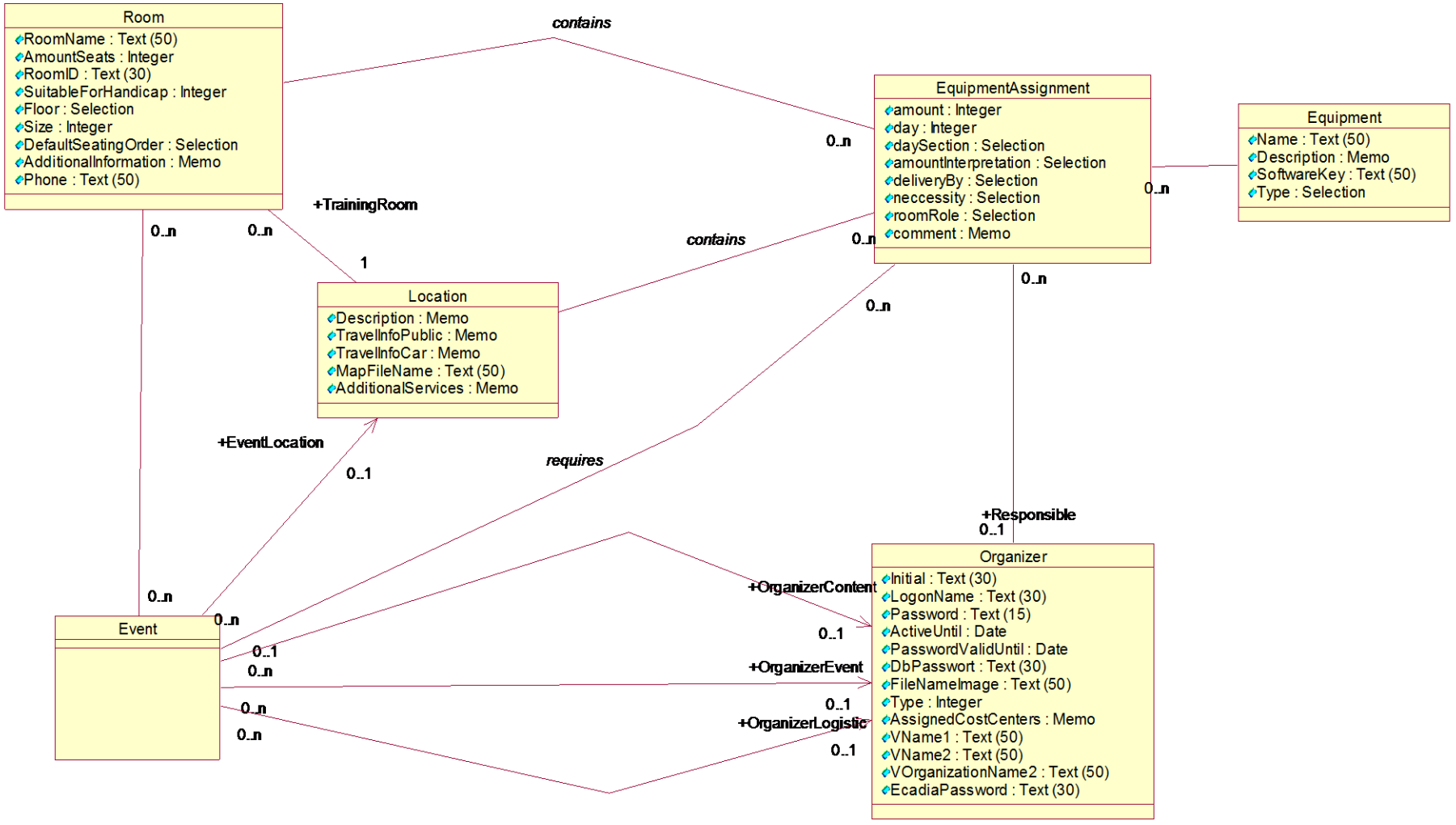
Icon-Darstellung:



Beispiel eines Klassendiagramms (1)



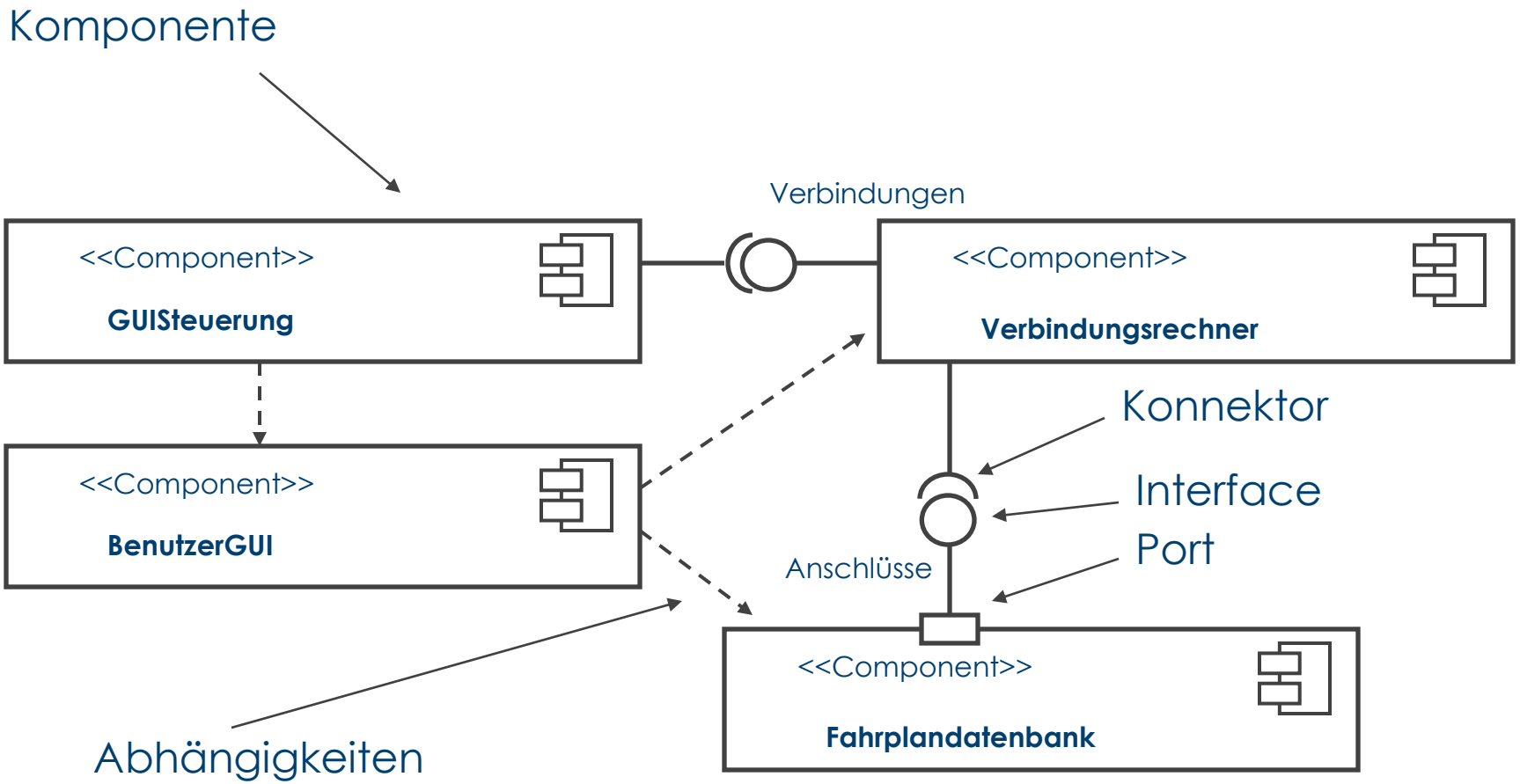
Beispiel eines Klassendiagramms (2)



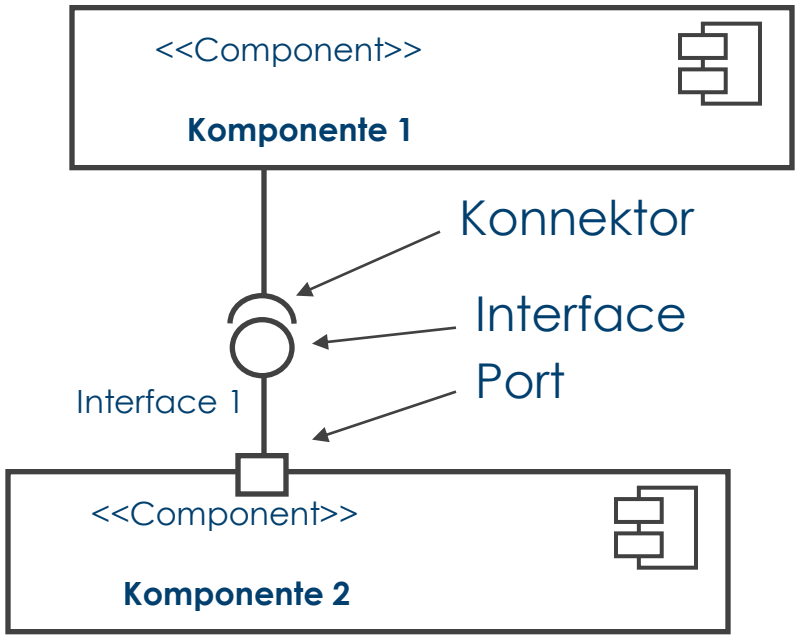
Folie 88

Komponenten Diagramm

- Modelliert die Software-Architektur mit Bibliotheken, Komponenten, Datenbanken, usw.



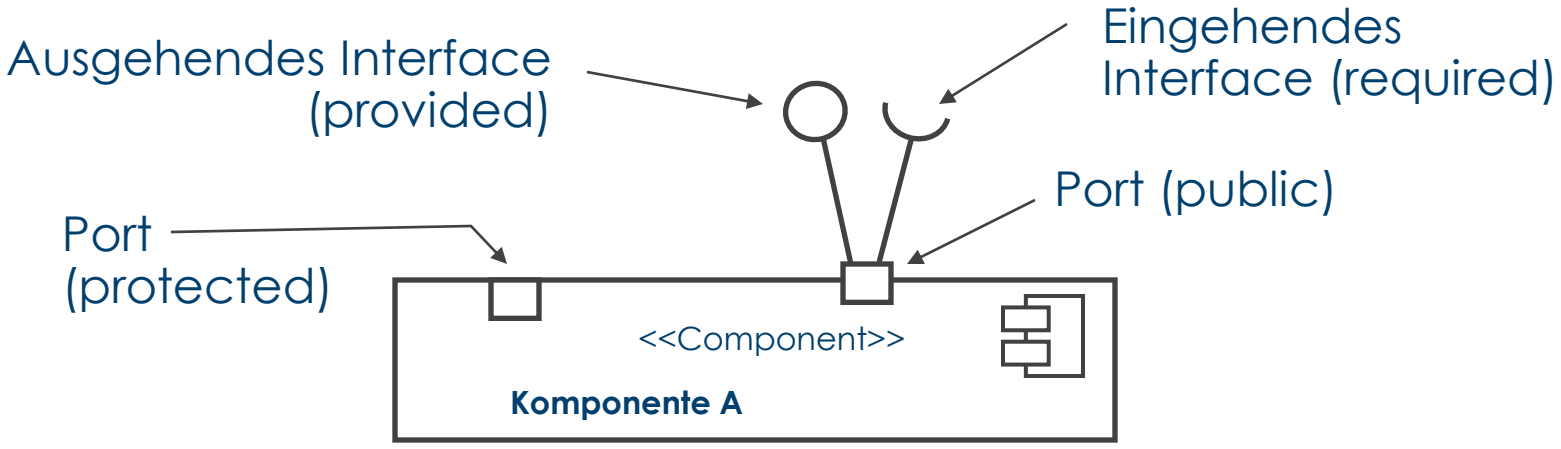
Komponenten Diagramm (2)



- Komponente: Abgegrenzter Teil des Gesamtsystems
- Konnektor (=required Interface): eine von der Komponente benötigte Schnittstelle
- Interface (=provided Interface): eine von der Komponente bereitgestellte Schnitttstelle
- Port: Interaktionspunkt, kann mehrere (provided) Interfaces enthalten

Port

- Interaktionspunkt zwischen einem Classifier (Klasse, Paket, Komponente, ...) und seiner Umgebung
  - ⇨ Gruppierung einer beliebigen Anzahl von Interfaces zu einem Dienst, der einen bestimmten Dienst/ Zweck verfolgt
  - ⇨ Kapselung des Classifiers gegenüber seiner Umgebung
  - ⇨ Für ein- und ausgehende Signale
- Sichtbarkeit
  - ⇨ public für eingehende Nachrichten
  - ⇨ protected, private für interne Nachrichtenverteilung



UML Spezifikation zu Komponente

- Siehe UML Superstructure, Kapitel 8

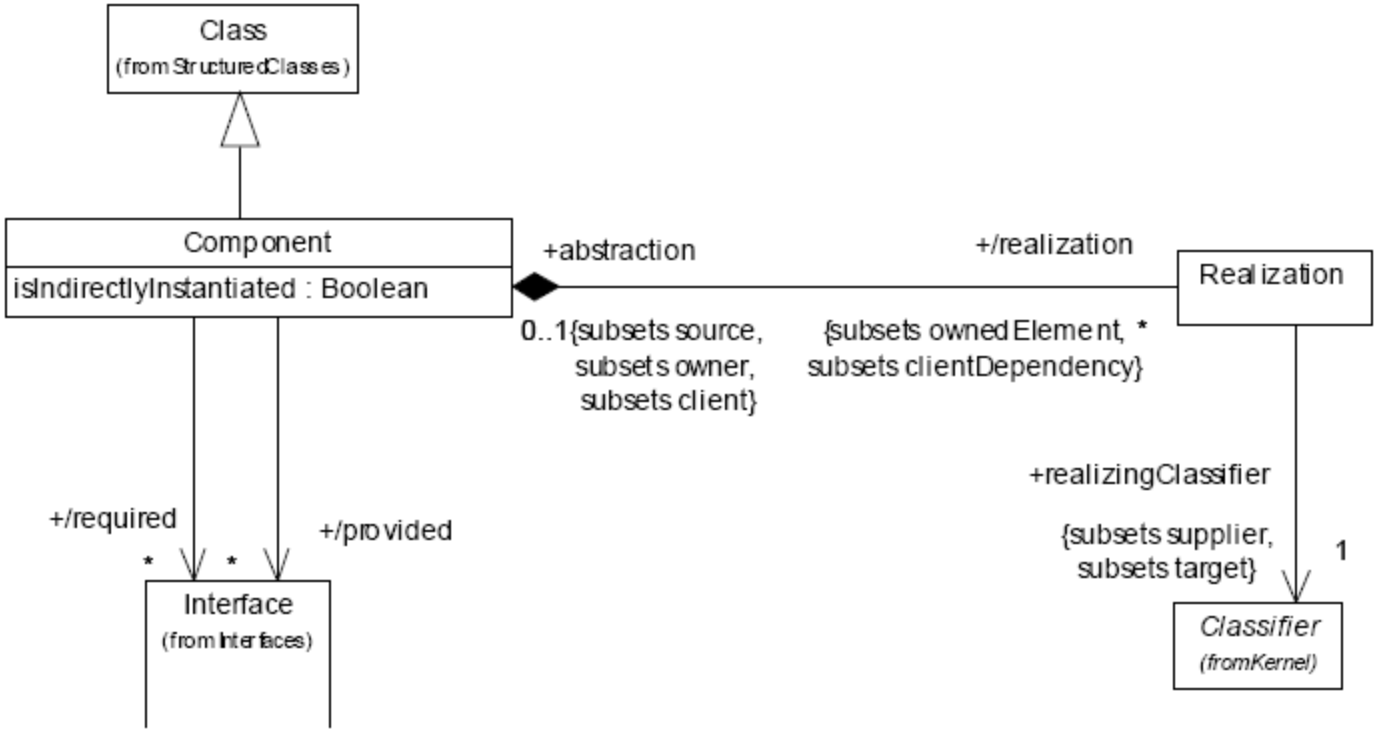
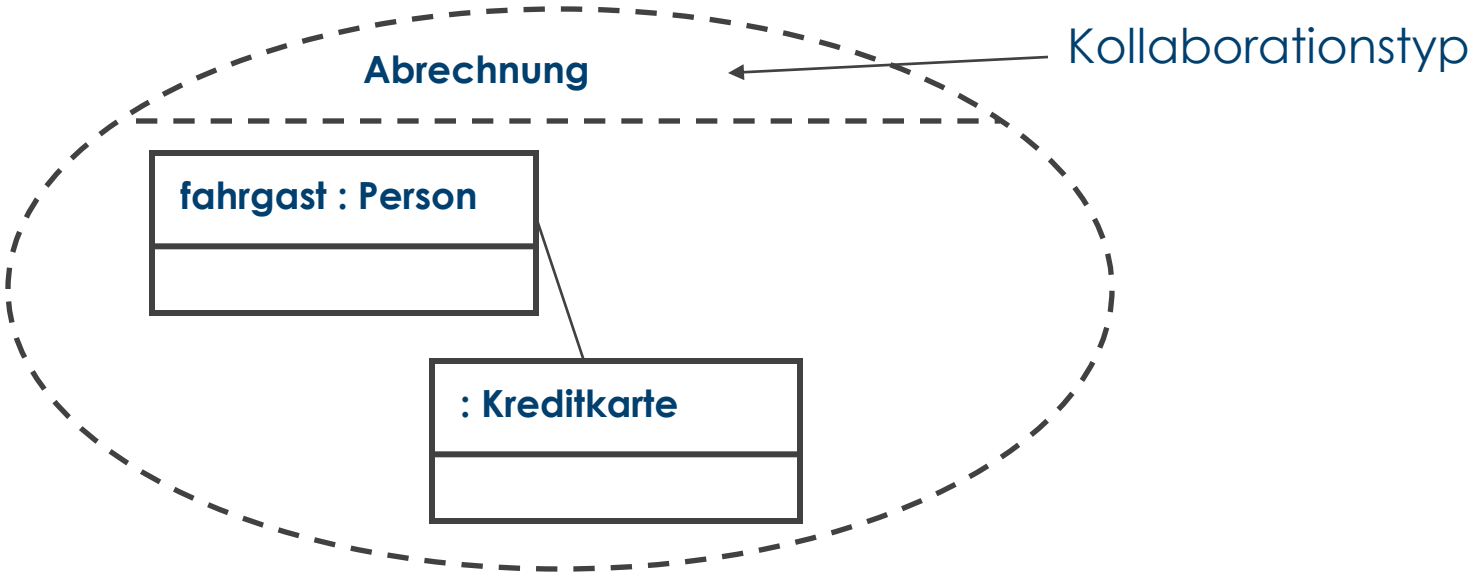


Figure 8.2 - The metaclasses that define the basic Component construct

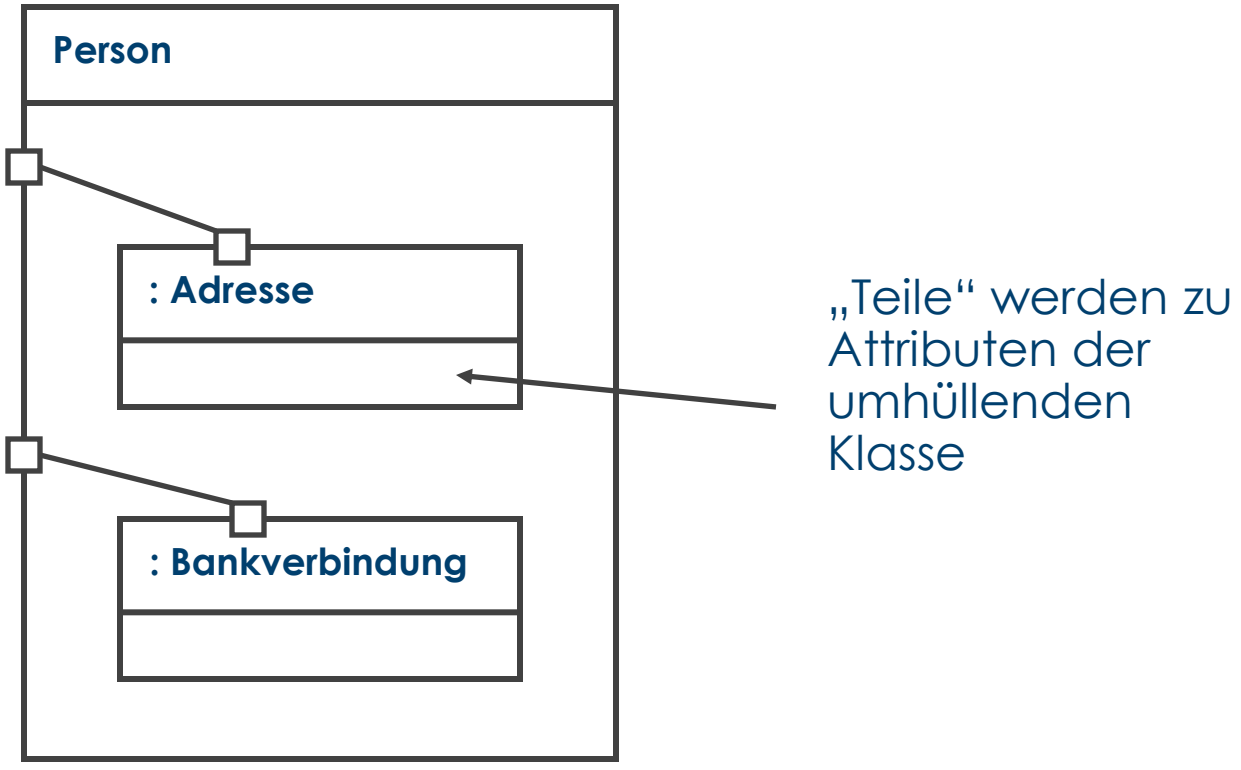
Kompositions-Struktur-Diagramm (1)

- Ab UML 2.0, Composite Structure
- „Kontext“-bezogenes Klassen-, Objekt oder Komponentendiagramm
- Zeigt eine „Ausprägung“ oder eine bestimmte Konfiguration



Kompositions-Struktur-Diagramm (2)

- Modelliert eine „Teil von“-Beziehung, ähnlich wie bei der Aggregation
- Die „Teile“ können über definierte Schnittstellen (Ports) mit der Umgebung kommunizieren



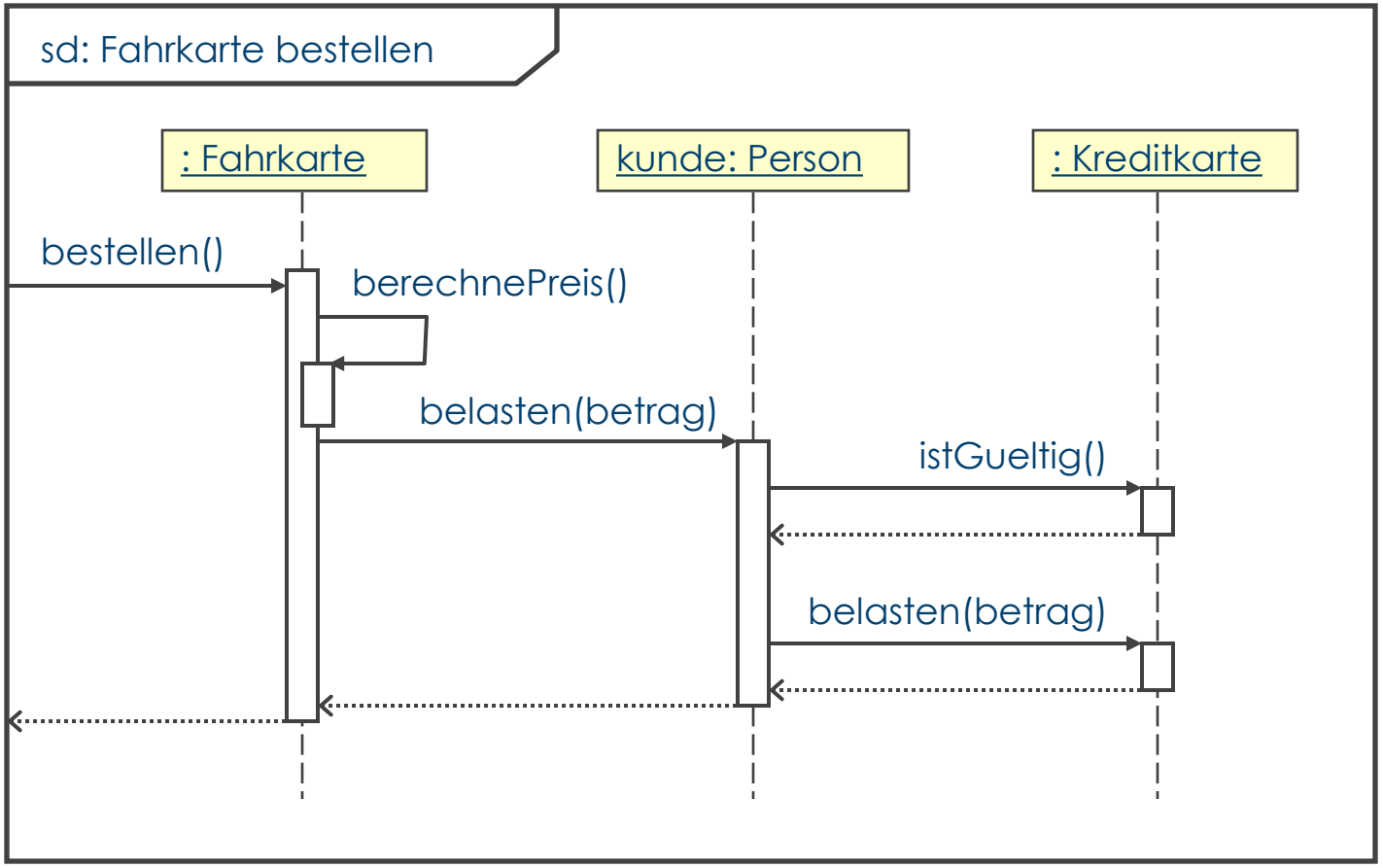
Verhaltensdiagramme

Sequenzdiagramm

- Beschreibt einzelne, genau definierte Abläufe
- Dient der Beschreibung von Use Cases und zur Beschreibung des dynamischen Verhaltens von Objekten (Achtung: der Objekte, nicht der Klassen!)
- Zeigt über die Abfolge (von oben nach unten) und auch über die Nummerierung die zeitliche Reihenfolge der Botschaften an
- Die Summe aller Sequenzdiagramme beschreibt das dynamische Verhalten des Systems
- Botschaften werden bei der Klasse des Empfänger-objekts als Methode implementiert

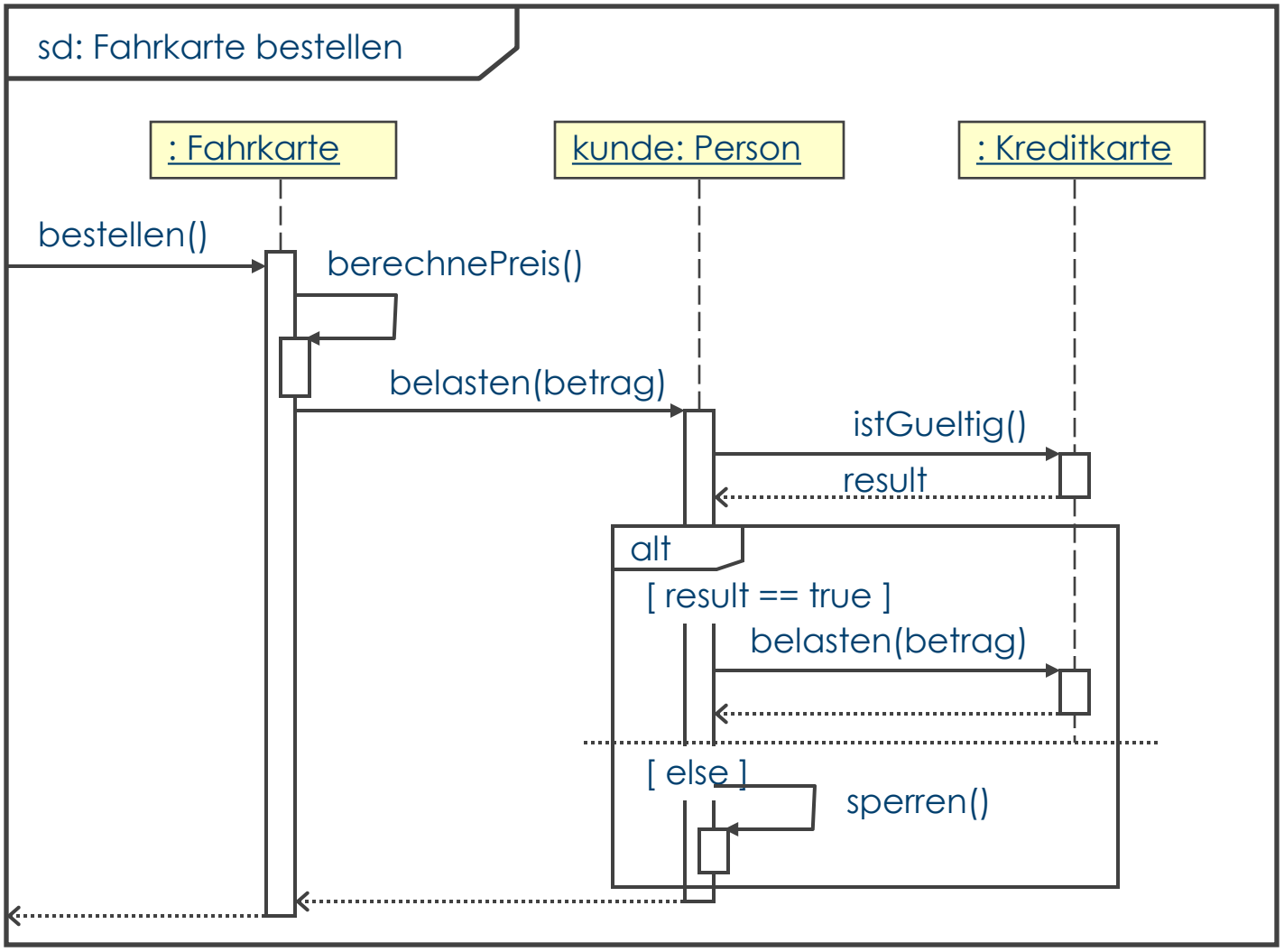
Sequenzdiagramm (1)

- Beispiel: „Fahrkarte bestellen“



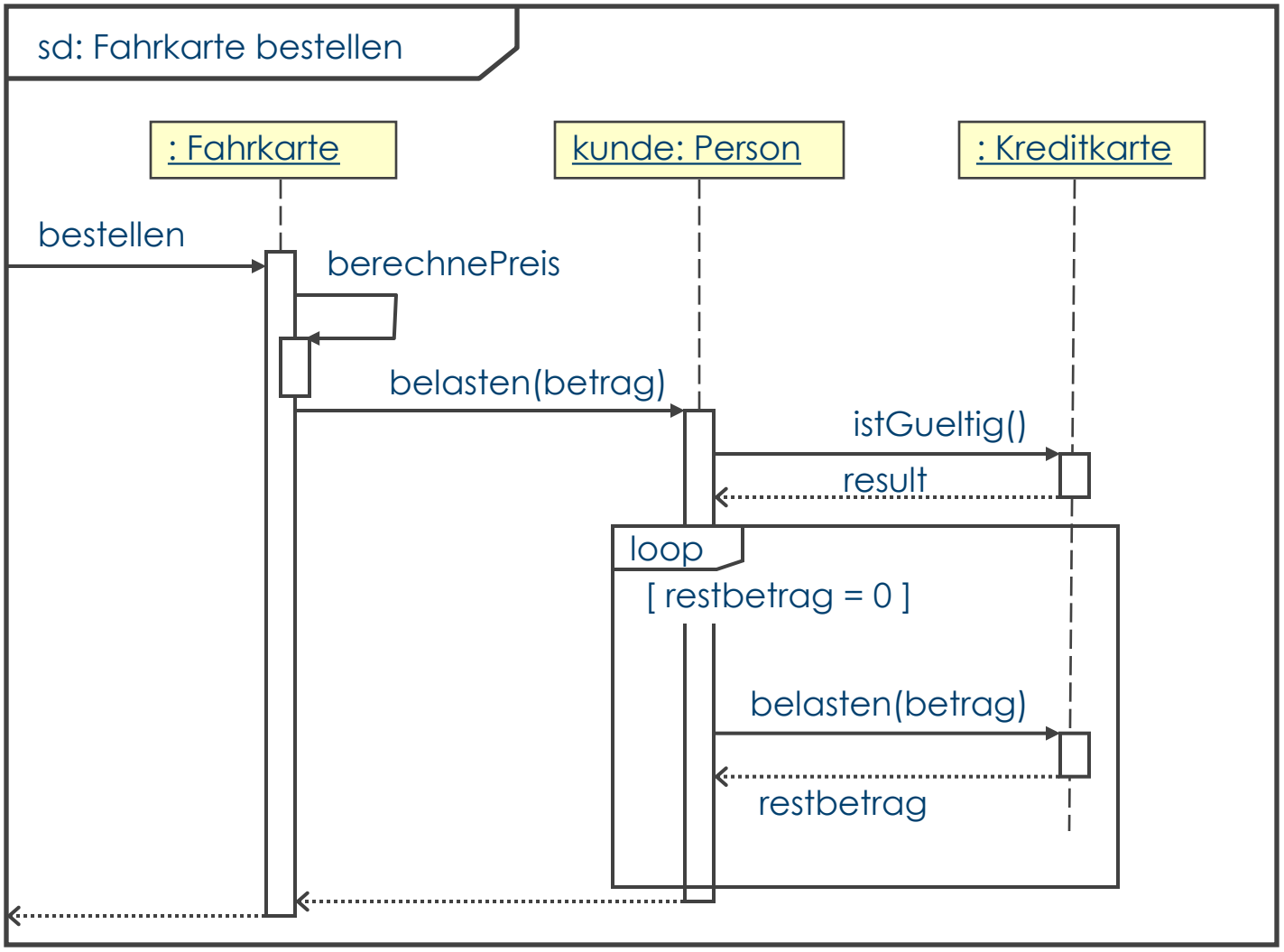
Sequenzdiagramm (2)

- Alternativen



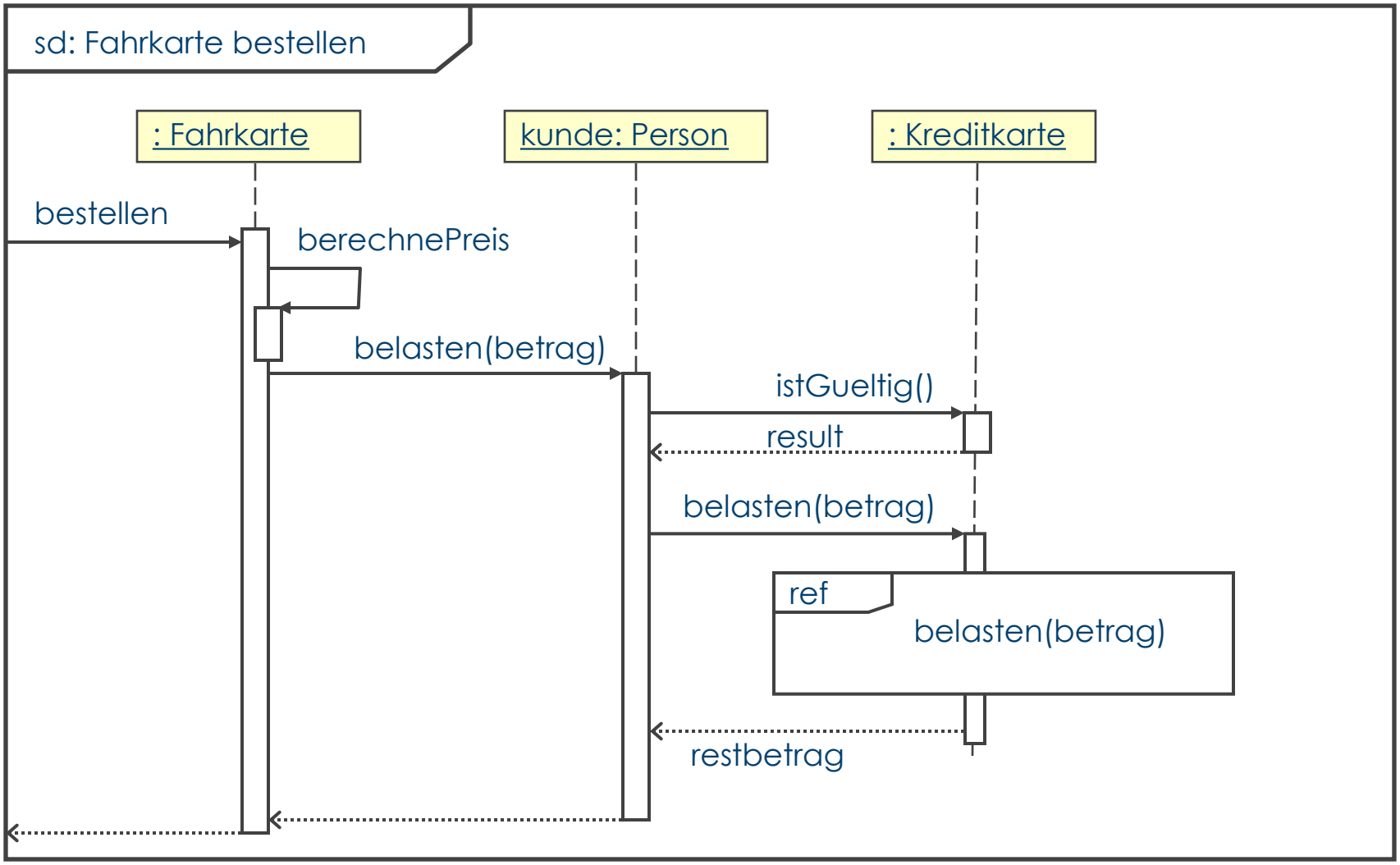
Sequenzdiagramm (3)

- Schleifen



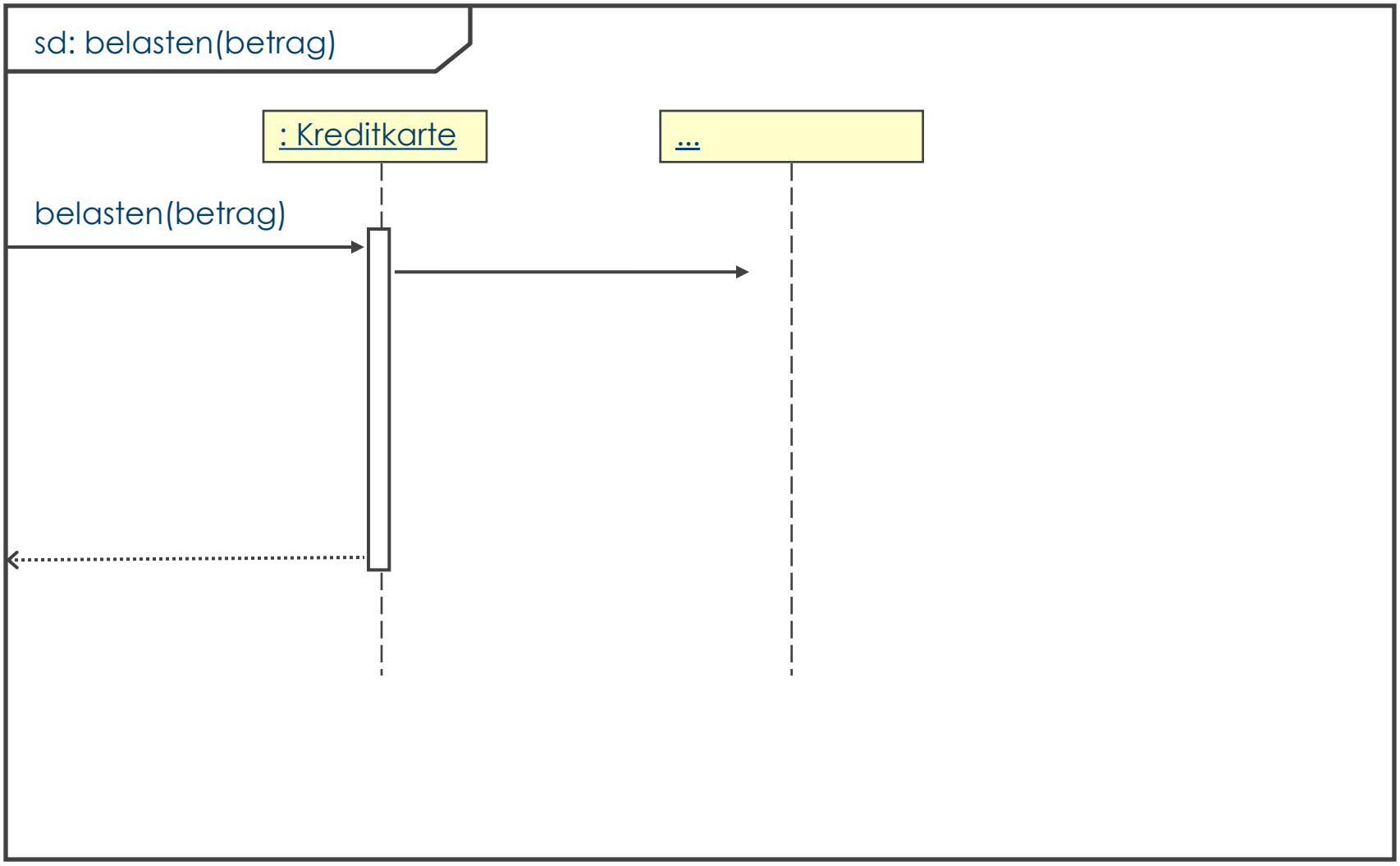
Sequenzdiagramm (4)

- Referenzen



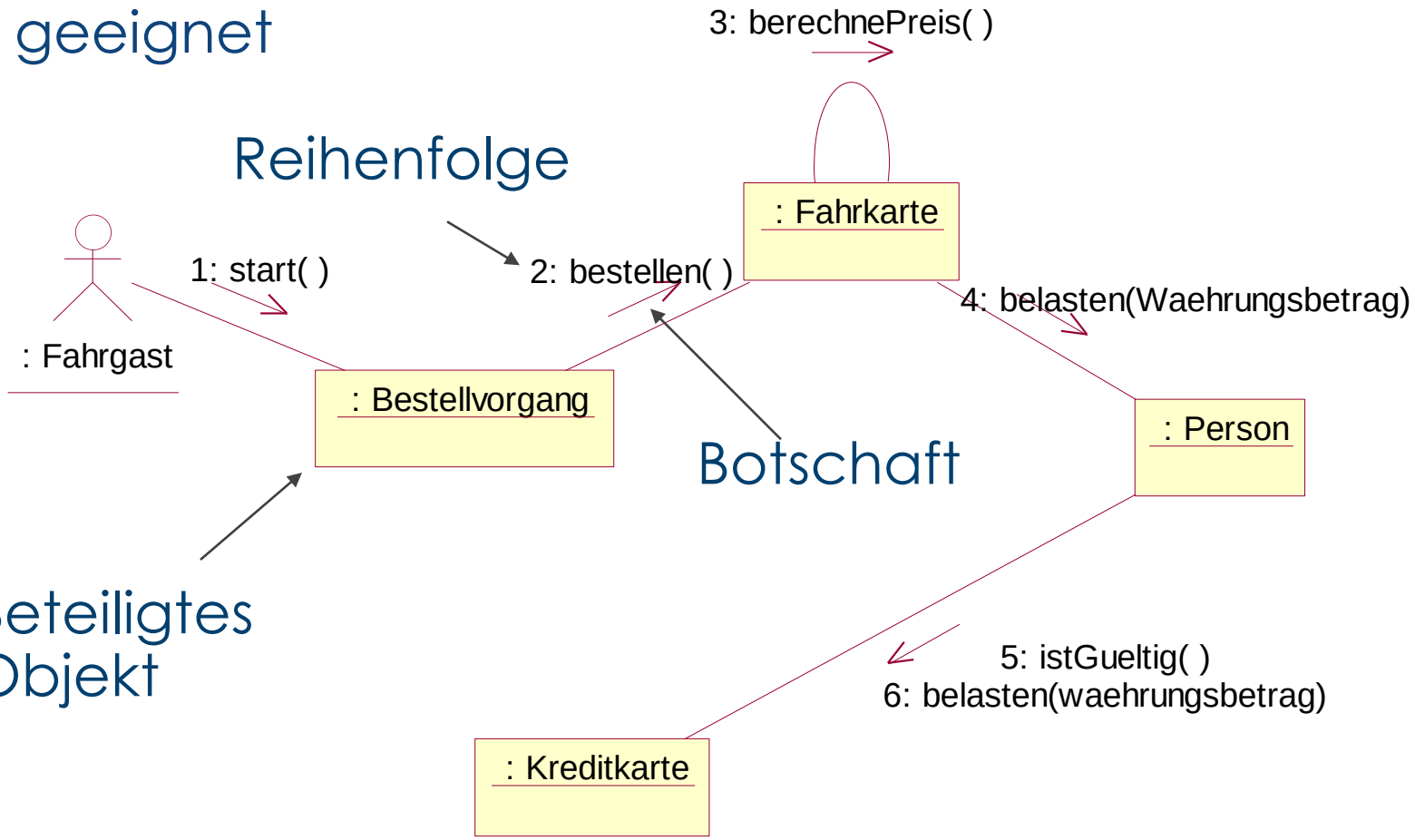
Sequenzdiagramm (5)

- Referenz



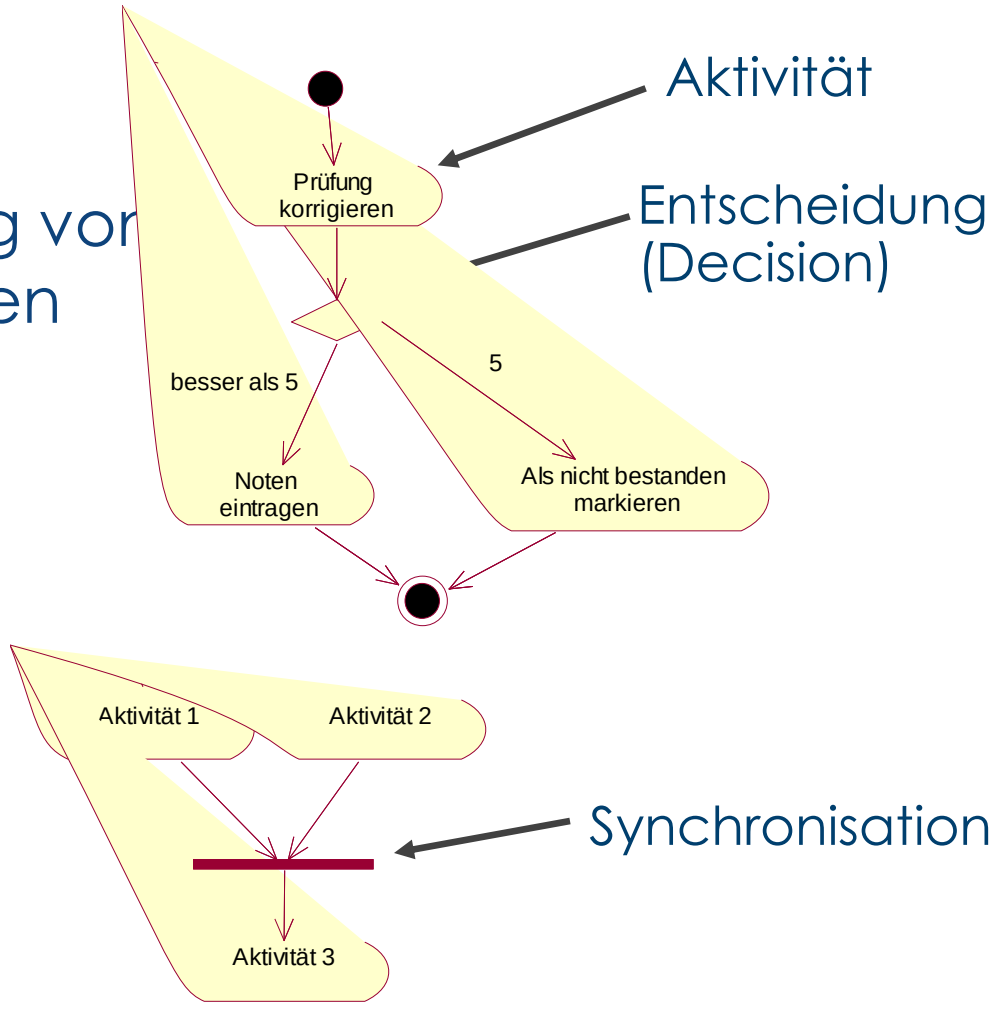
Kommunikationsdiagramm

- Ähnlicher Inhalt wie beim Sequenzdiagramm
- Bei vielen Objekten mit wenig Botschaftsfluss geeignet



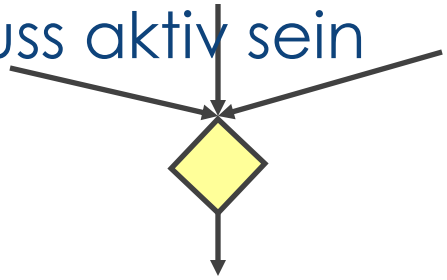
Aktivitätendiagramm

- Aktivitätenfolge mit Kontrollfluss
- Wird zur Beschreibung von Use Cases oder Klassen eingesetzt
- Wie bei Zustandsübergängen kann ein Ereignis je Transition hinterlegt werden
- Verzweigungen und Zusammenführungen

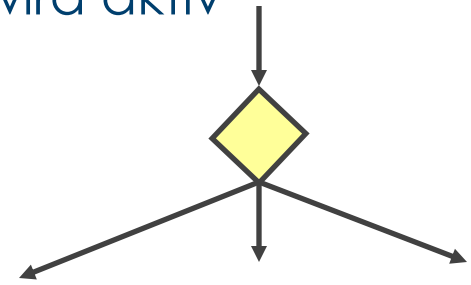


Verzweigung / Vereinigung

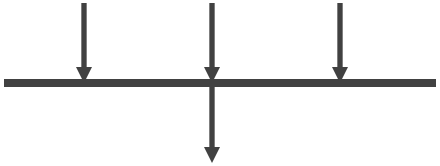
- Vereinigung (Merge Node)
  - ⇨ Einer der Eingänge muss aktiv sein



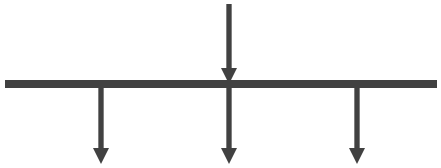
- Verzweigung (Decision Node)
  - ⇨ Einer der Ausgänge wird aktiv



- Zusammenführung (Join Node)
  - ⇨ Alle Eingänge müssen aktiv sein



- Gabelung (Fork Node)
  - ⇨ Alle Ausgänge werden aktiv



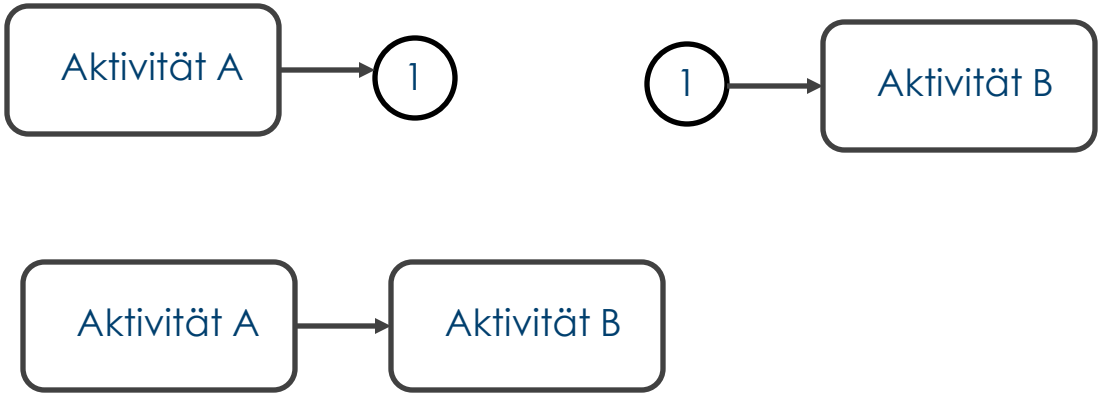
Knoten

Start- und Endknoten

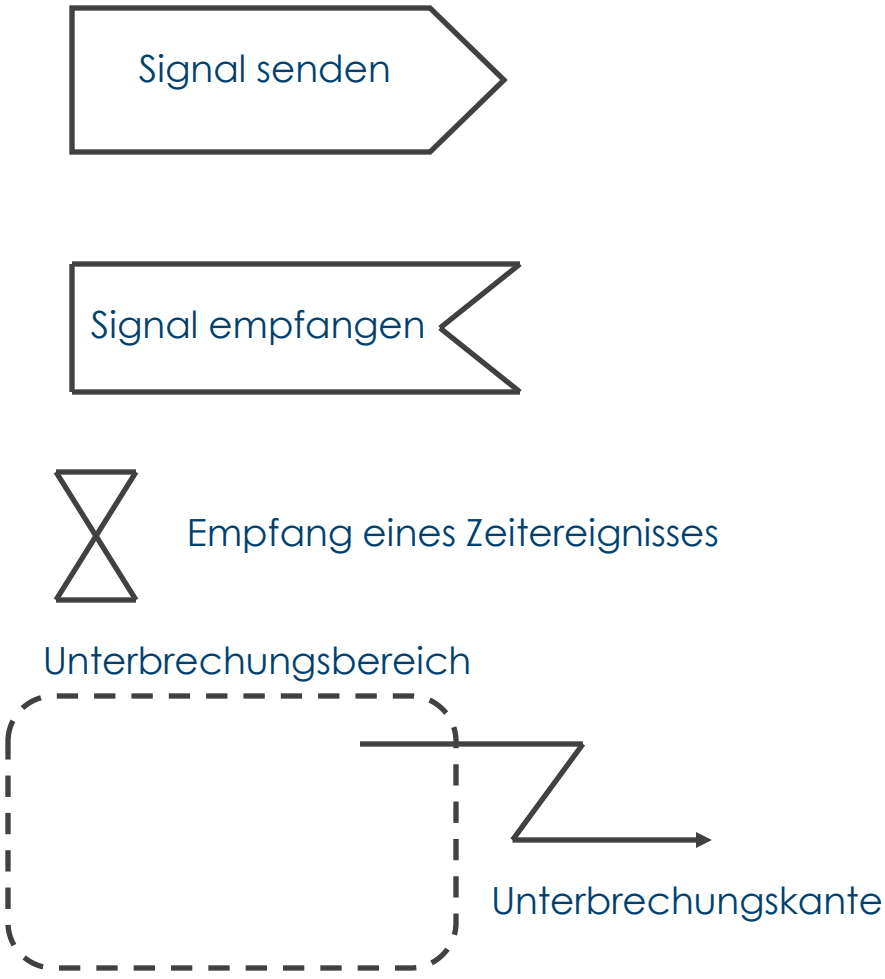
(beendet die gesamte Aktivität)

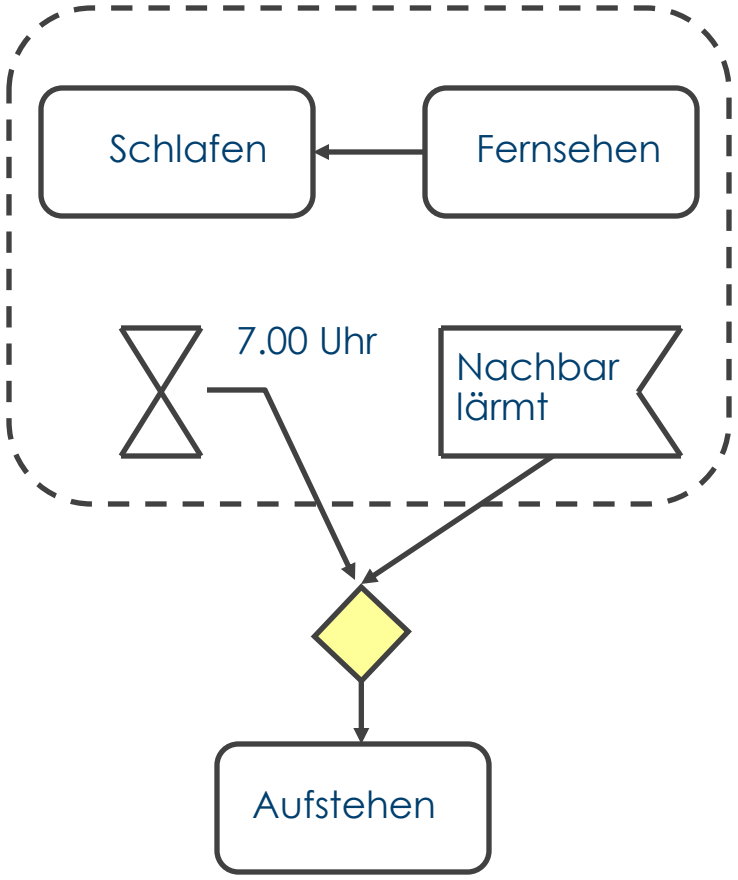
(beendet den Fluss an der entsprechenden)

Sprungmarken

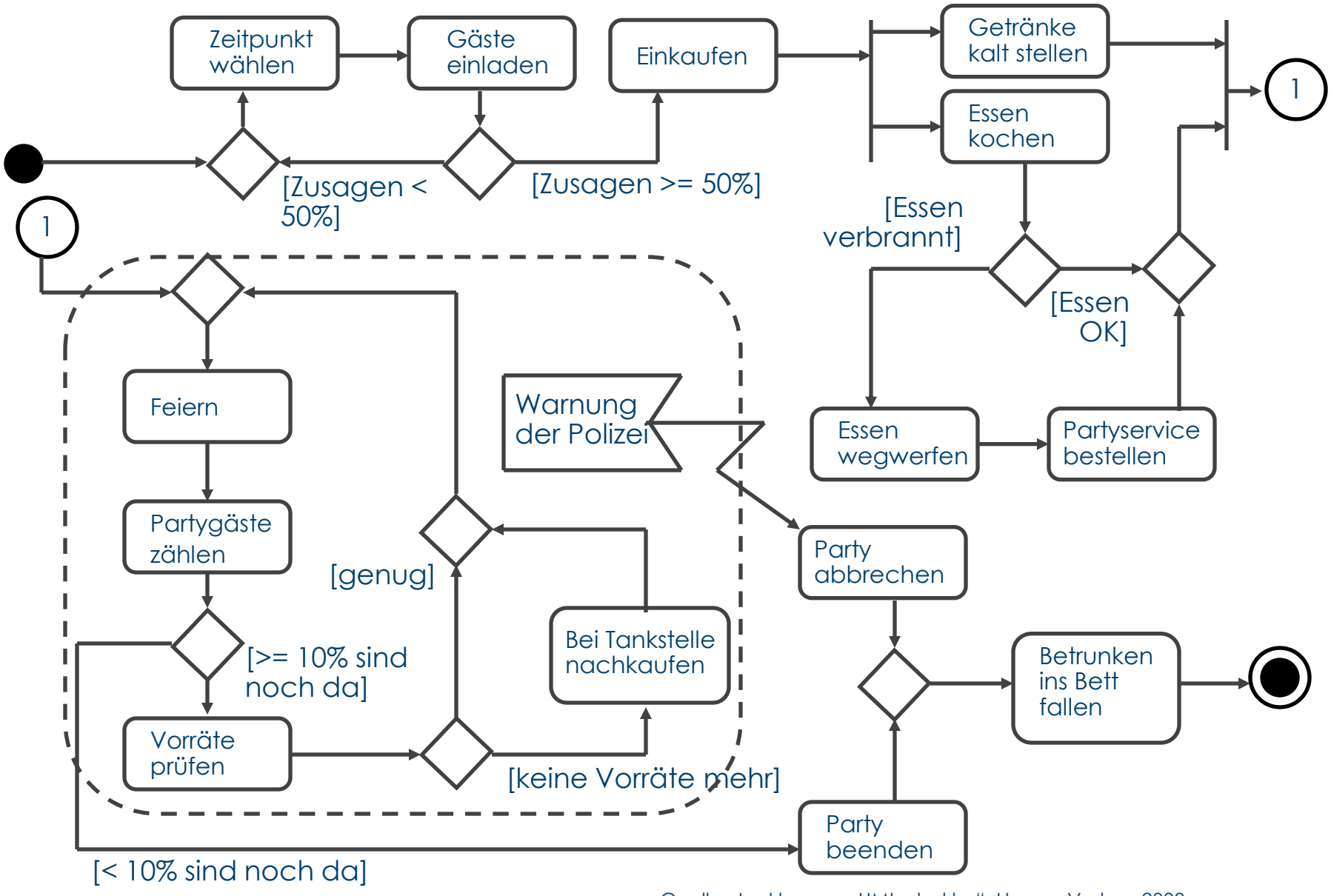


Signale im Aktivitätsdiagramm





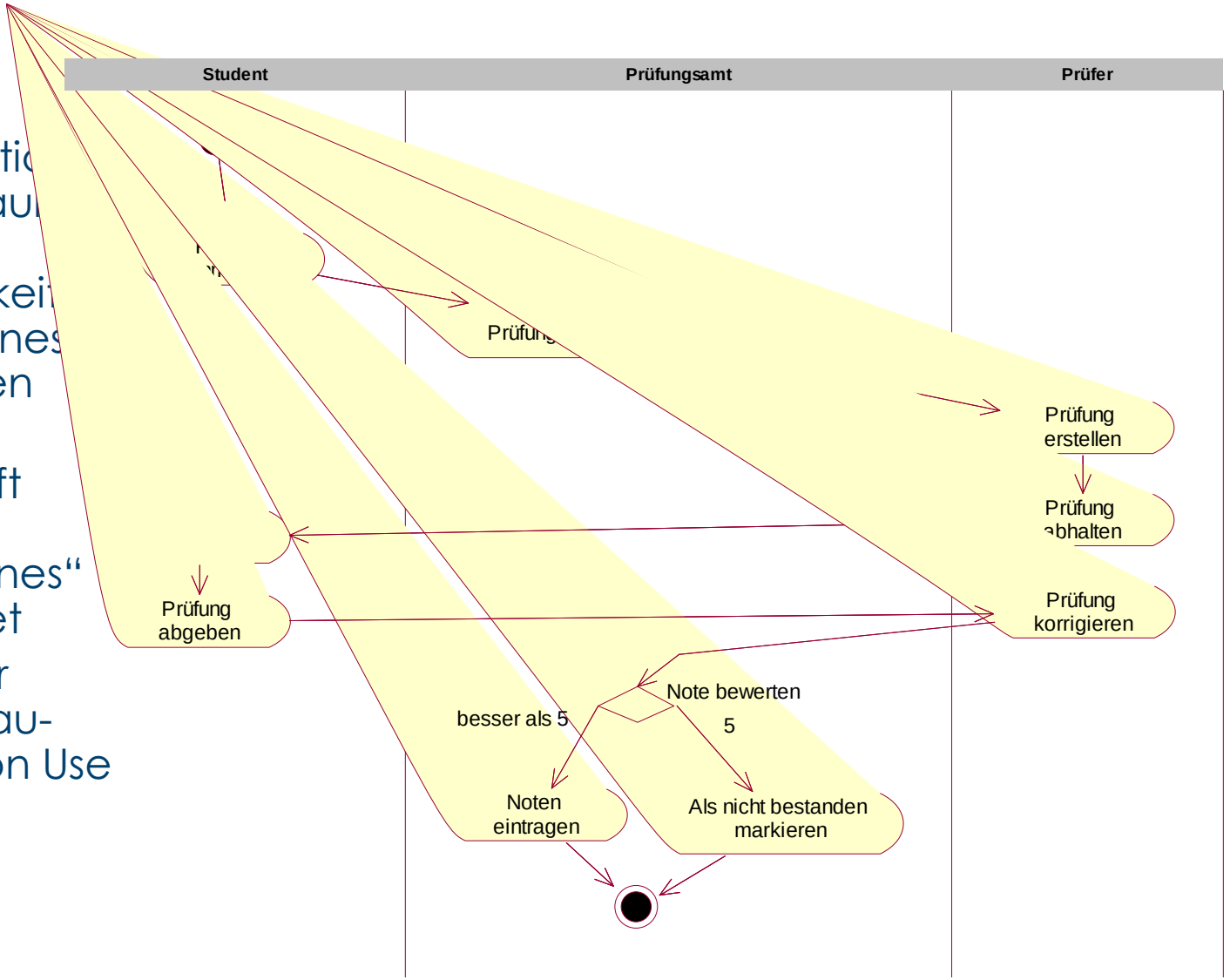
Verlauf einer typischen Party



Quelle: Jeckle u.a., „UML glasklar“, Hanser Verlag, 2003

Aktivitätsdiagramm: Partitions

- Eine „Partitio
  veranschau
  den
  Zuständigkeit
  bereich eines
  bestimmten
  Objekts
- Werden oft
  auch als
  „Swimmlanes“
  bezeichnet
- Speziell zur
  Veranschau-
  lichung von Use
  Cases

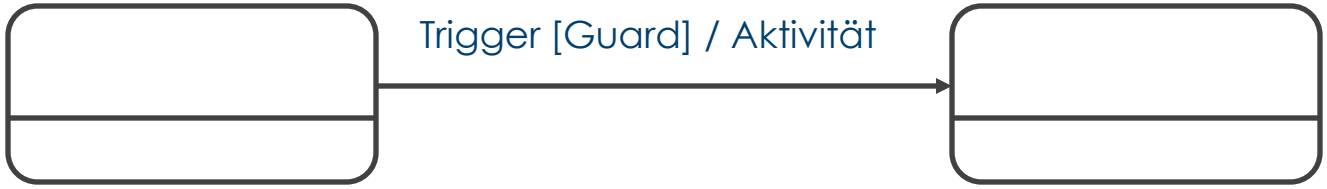


Folie 108

Zustandsautomat

- Zeigt die Zustände und Zustandsübergänge innerhalb eines Objektes an
- Elemente:

- Zudem können noch Trigger mit Bedingungen festgelegt werden, die Aktivitäten auslösen

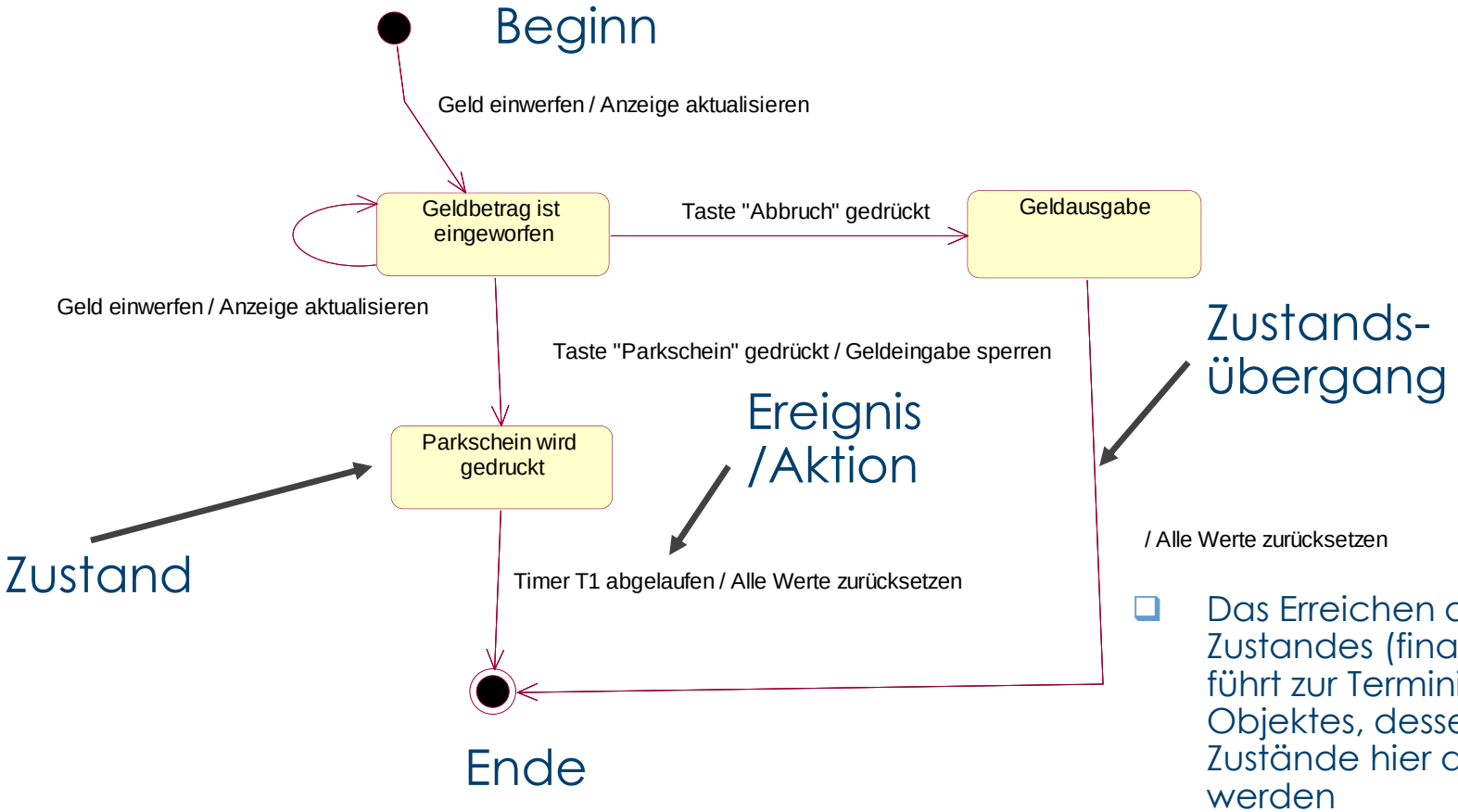


- Trigger: Ereignisse, die den Zustandswechsel herbeiführen
- Guard: Bedingung, die zum Zustandswechsel erfüllt sein muss
- Aktivität: Aktivität, die beim Zustandswechsel durchgeführt wird

Folie 109

Zustandsautomat

- Zeigt die Zustände und Zustandsübergänge innerhalb eines Objektes an

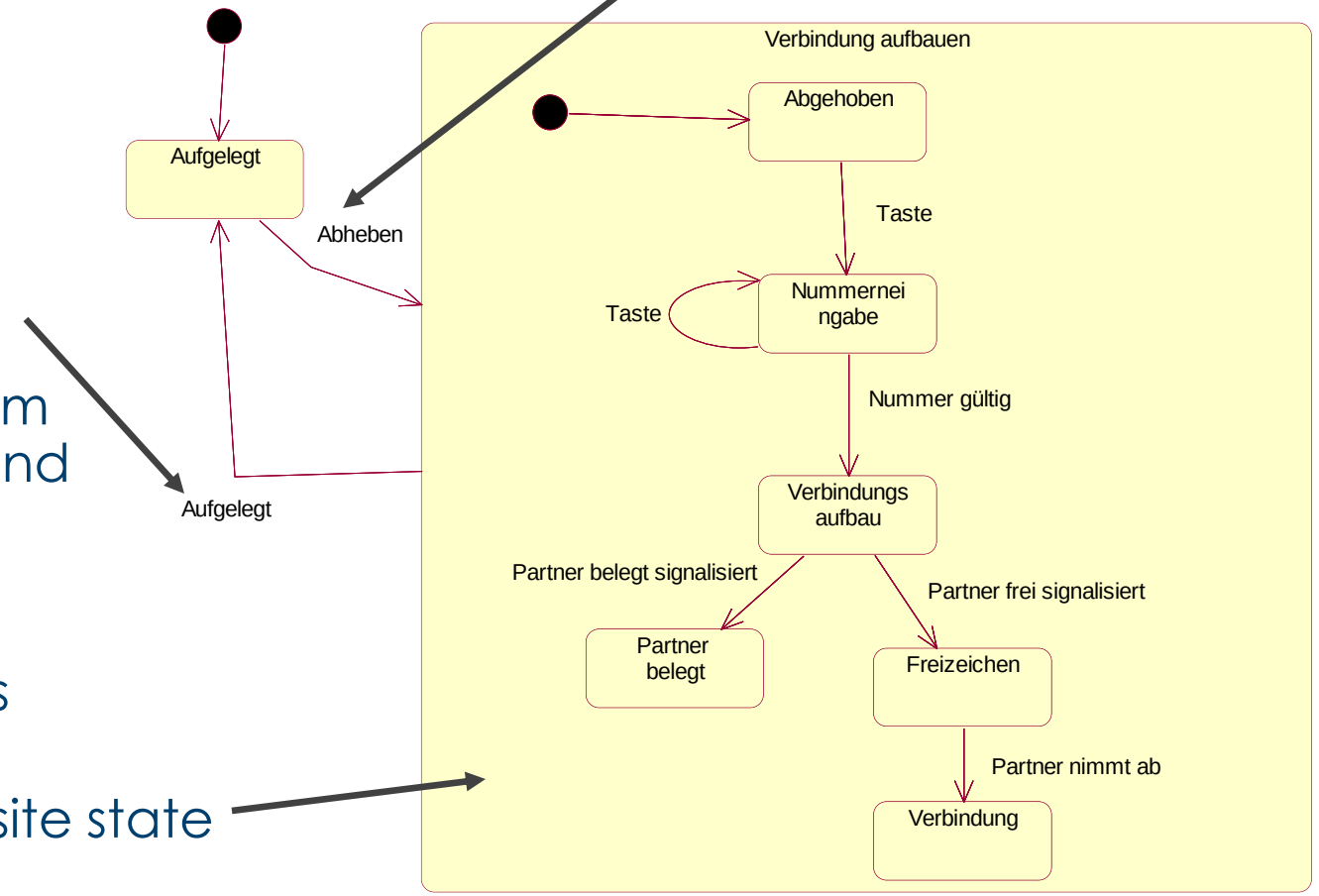


- Das Erreichen des Ende-Zustandes (final state) führt zur Terminierung des Objektes, dessen Zustände hier dargestellt werden

Folie 110

Verbundzustand

- Composite state
- Spezialisierung von Zuständen



Folie 111

Object Constraint Language (OCL)

OCL

- Die Object Constraint Language ist Teil von UML 2.0
- OCL ist eine formale, prädikatenlogikbasierte Sprache zur Ergänzung von UML-Modellen
- OCL Ausdrücke sind leicht zu lesen (leichter als beispielsweise die von Z oder die Prädikatenlogik)
- OCL Ausdrücke sind deklarativ, d.h. sie ändern den Systemzustand nicht
- Dient zur Formulierung von Zusicherungen
  - ⇨ Invarianten von Klassen
  - ⇨ Vor- und Nachbedingungen von Methoden
- Nutzen von OCL
  - ⇨ Die UML-Modelle können präziser beschrieben werden
  - ⇨ Programmcode kann generiert werden

OCL Constraints (1)

- OCL constraints werden relativ zu einem Systemzustand *S* ausgewertet
- OCL constraints sind vom Typ Boolean; sie sind wahr oder falsch bezüglich *S*
- OCL constraints beschränken die erlaubten Systemzustände eines UML Klassendiagramms

OCL Constraints (2)

- Invarianten:
  - ⇨ Eine Invariante *I* für eine Klasse *C* ist ein Boolscher OCL-Ausdruck.
  - ⇨ Semantik: *I* gilt in jedem feststellbaren Zustand (snapshot) jeder Instanz von *C*.
- Vor- und Nachbedingungen:
  - ⇨ Vor- und Nachbedingungen *(v, n)* für eine Operation *p* ist ein Paar von Boolschen OCL-Ausdrücken.
  - ⇨ Semantik: *v* muss zum Zeitpunkt, wenn *p* aufgerufen wird, *true* sein. *n* muss zum Zeitpunkt, wenn *p* terminiert, *true* sein

Der Kontext

- Der Kontext legt fest, auf welches Element sich ein OCL Constraint bezieht.

```
context  [ kontextInstanz : ] typeName
inv: OclAusdruck1
...
inv: OclAusdruckn
```

- Beispiel:

```
context Person
inv: self.alter > 0
```

`self` (die referenziert per default auf die Kontext-Instanz)

Soll bedeuten: zu keinem Zeitpunkt darf es ein Objekt der Klasse Person geben, das einen negativen Wert im Attribut `alter` hat

Boolean Operationen

Operation	Notation	Ergebnistyp
Oder | Und | Exklusiv Oder	a **or** | **and** | **xor** b	Boolean
Negation	**not** a	Boolean
Vergleich	a **=** b	Boolean
Ungleichheit	a **<>** b	Boolean
Impliziert (wenn a true ist, muss auch b true sein	a **implies** b	Boolean

Integer/Real Operationen

Operation	Notation	Ergebnistyp
Vergleich	a **=** b, a<>b, a<b, a>b, a<=b, a>=b,	Boolean
Rechnen	a+b, a-b, a*b, a/b	Integer/Real
Modulo	a.**mod**(b)	Integer/Real
Min, Max	a.max(b), a.min(b)	Integer/Real
...		

Spezielle OCL-Operatoren

- oclIsTypeOf (t : OclType) : Boolean
- oclIsKindOf (t : OclType) : Boolean
- oclInState (s : OclState) : Boolean
  - ⇨ Prüfung von Zuständen bei Zustandsautomaten (State D.)
- oclIsNew () : Boolean
  - ⇨ für Nachbedingungen, prüft, ob Objekte innerhalb einer Methode neu erzeugt wurden
- oclAsType (t : OclType) : instance of OclType
- Beispiel:

```
context Person
inv: self.oclIsTypeOf( Person ) -- ist true
inv: self.oclIsTypeOf( Firma) -- ist false
```

OCL Kollektionen

- Set(Menge), OrderdSet
- Bag (Liste)
- Sequence (geordnete Liste)

Instanzierung:

- als Literal
  - ⇨ Set { 1, 2, 4, 42}
  - ⇨ Sequence {1..10} entspricht Sequence {1,2,3,4,5,6,7,8,9,10}
- durch Navigation
- durch Aufruf von Kollektionsmethoden
- durch Klasse.allInstances()

OCL Kollektionen (Auszug)

Operator	Ergebnistyp	Beschreibung
count(objekt)	integer	Anzahl, wie oft das Objekt in der Kollektion enthalten ist
size	integer	Anzahl d. Elemente
isEmpty	boolean	wahr, wenn Menge leer ist
notEmpty	boolean	falsch, wenn Menge leer ist
forAll(<ausdruck>)	boolean	wahr, wenn <*ausdruck*> für jedes Element gilt
exists(<ausdruck>)	boolean	wahr, wenn <*ausdruck*> für mind. ein Element gilt
select(<ausdruck>)	Menge	Teilmenge, für die <*ausdruck*> gilt
includesAll (Menge2)	boolean	wahr, wenn alle Elemente der Menge *Menge2* enthalten sind

OCL Kollektionen (2)

c->forAll(objekt1, ... Objektn | <*ausdruck* >)	liefert true falls <ausdruck> für alle Elemente true ist.

```
context Firma
inv:
self.Mitarbeiter->forAll(p1, p2 |
      p1 <> p2 implies p1.Name <> p2.Name)
```

c->exists(objekt1, ... Objektn | <*ausdruck*>)	liefert true falls <ausdruck> für mindestens eines der Elemente true ist.

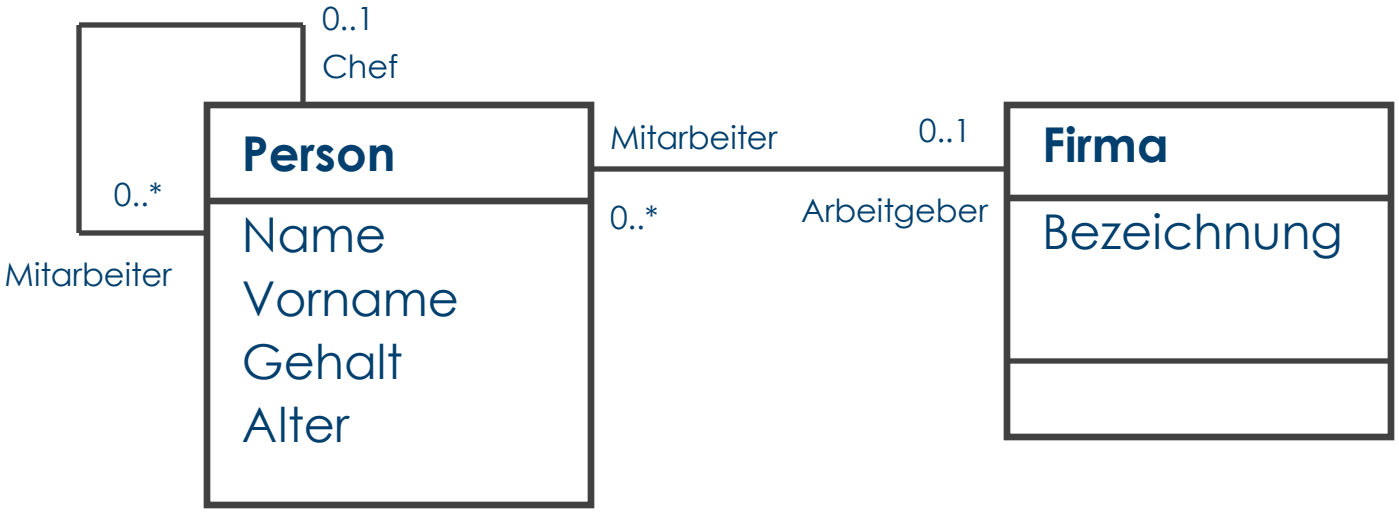
```
context Firma
inv:
self.Mitarbeiter->exists(p | p.Gehalt > 100000)
```

OCL - Beispiel

- Die Mitarbeiter sind mind. 16 und max. 65 Jahre alt
- Jeder hat ein Gehalt > 0

```
context Firma
inv:
self.Mitarbeiter->forAll(m | m.Alter>=16 and m.Alter<=65)
self.Mitarbeiter->forAll(m | m.Gehalt >= 0)
```

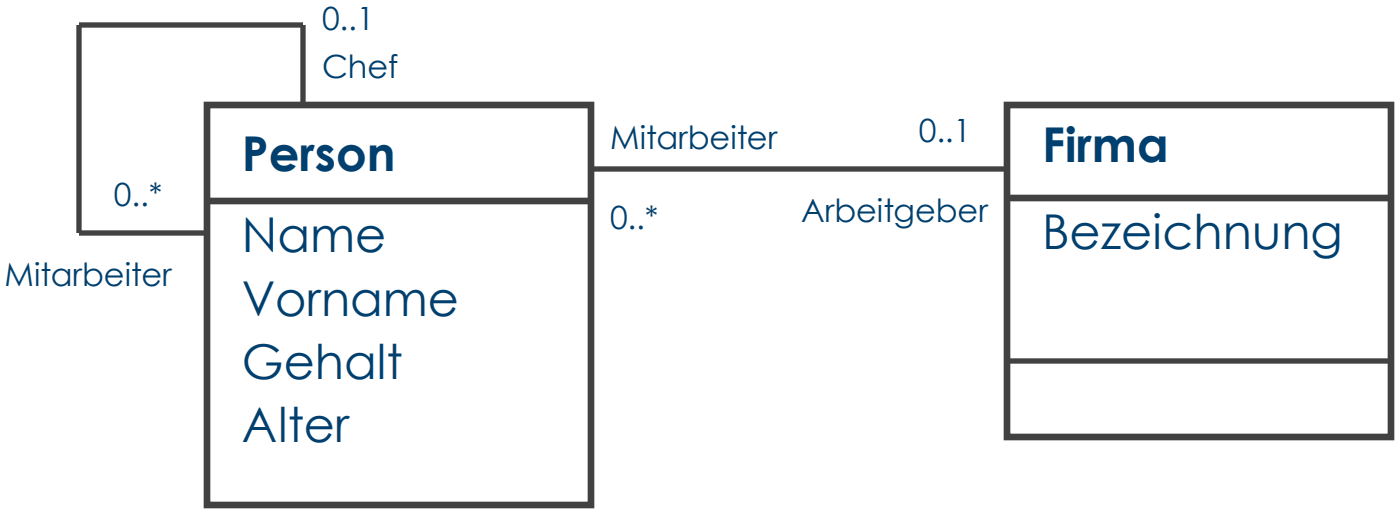
OCL - Beispiel



- Der Chef verdient mehr als sein Mitarbeiter

```
context Person
inv:
self.Chef->notEmpty() implies
self.Chef.Gehalt > self.Gehalt
```

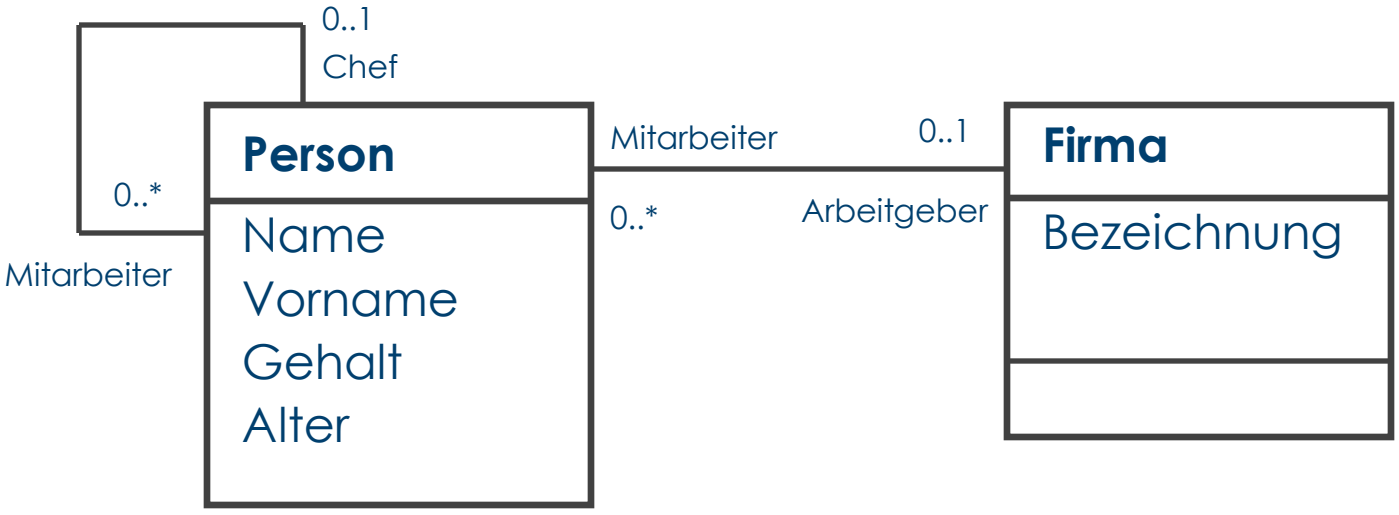
OCL - Beispiel



- Der Chef arbeitet bei der selben Firma wie sein Mitarbeiter

```
context Person
inv:
self.Chef->notEmpty() implies
self.Chef.Arbeitgeber = self.Arbeitgeber
```

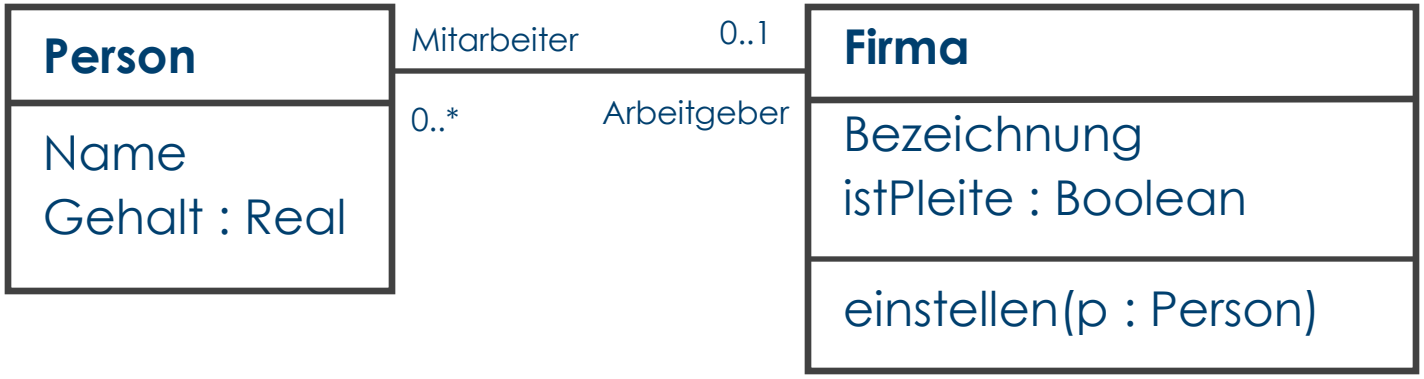
OCL - Beispiel



- Es gibt mindestens einen „Oberchef“

```
context Firma
inv:
self.Mitarbeiter->select(Chef->isEmpty())->size >= 1
```

Vor- und Nachbedingungen




```
context Firma::einstellen(p : Person)
pre:
not self.istPleite and      -- Firma ist nicht pleite
p.Arbeitgeber->isEmpty()    -- Person ist nicht bereits
                            -- anderswo beschäftigt
post:
p.Arbeitgeber = self and    -- jetzt eingestellt
p.Gehalt > p.Gehalt@pre     -- Stellenwechsel mit
                            -- Gehaltserhöhung
```

OCL Messages

- Um auszudrücken, dass eine „Botschaft“ versendet wurde, dient der hasSent (‘^’) Operator:

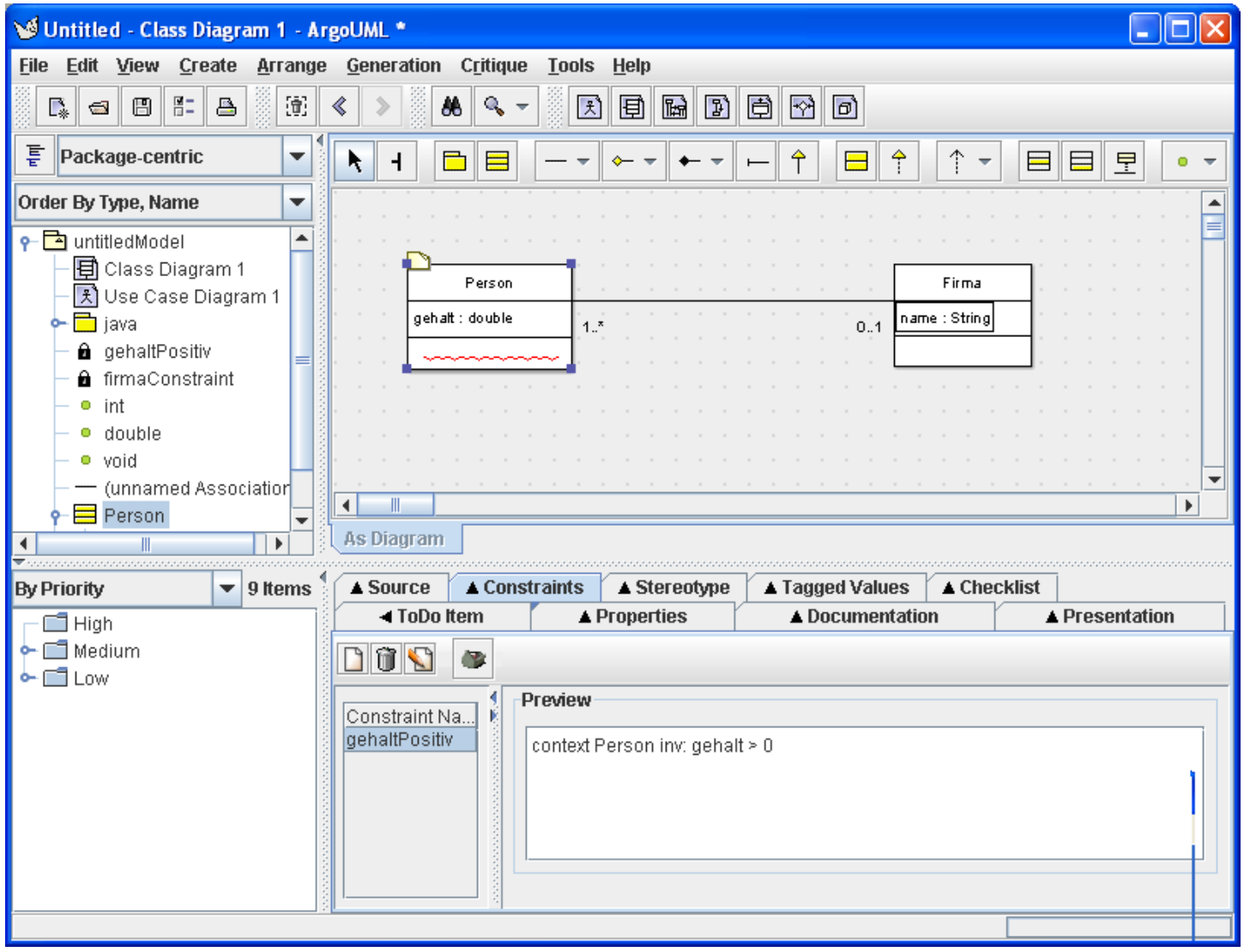
```
context Firma::einstellen(p : Person)
post: p^setArbeitgeber(self)
```

- Bei unspezifizierten Parametern:

```
context Subject::hasChanged()
post: observer^update(? : Integer, ? : Integer))
```

- Die Menge der Aufrufe einer Methode an einen Empfänger wird über ^^ ausgedrückt

OCL Werkzeuge

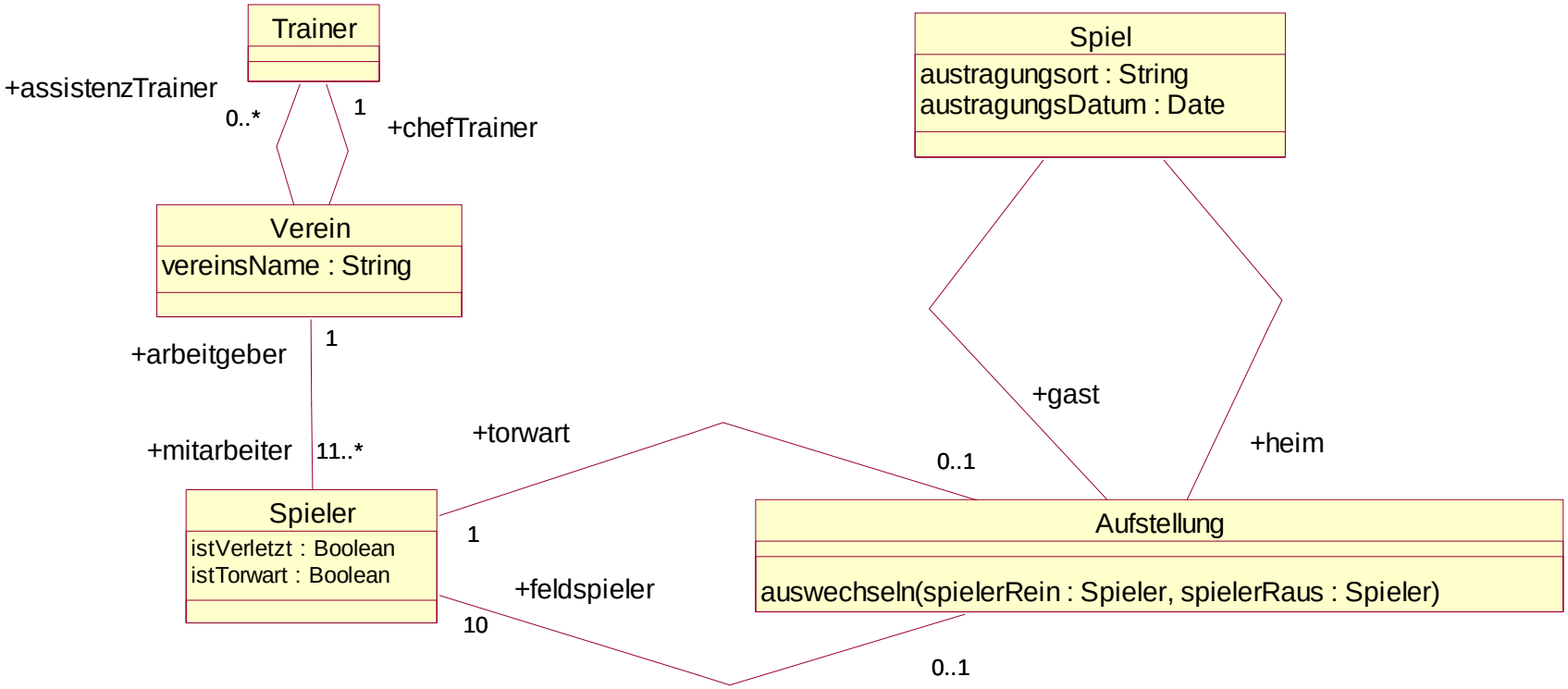


OCL Literatur

- http://www.omg.org/technology/documents/modeling_spec_catalog.htm
- Jos Warmer, Anneke Kleppe, „*The Object Constraint Language*“, second edition, Addison-Wesley, 2003.

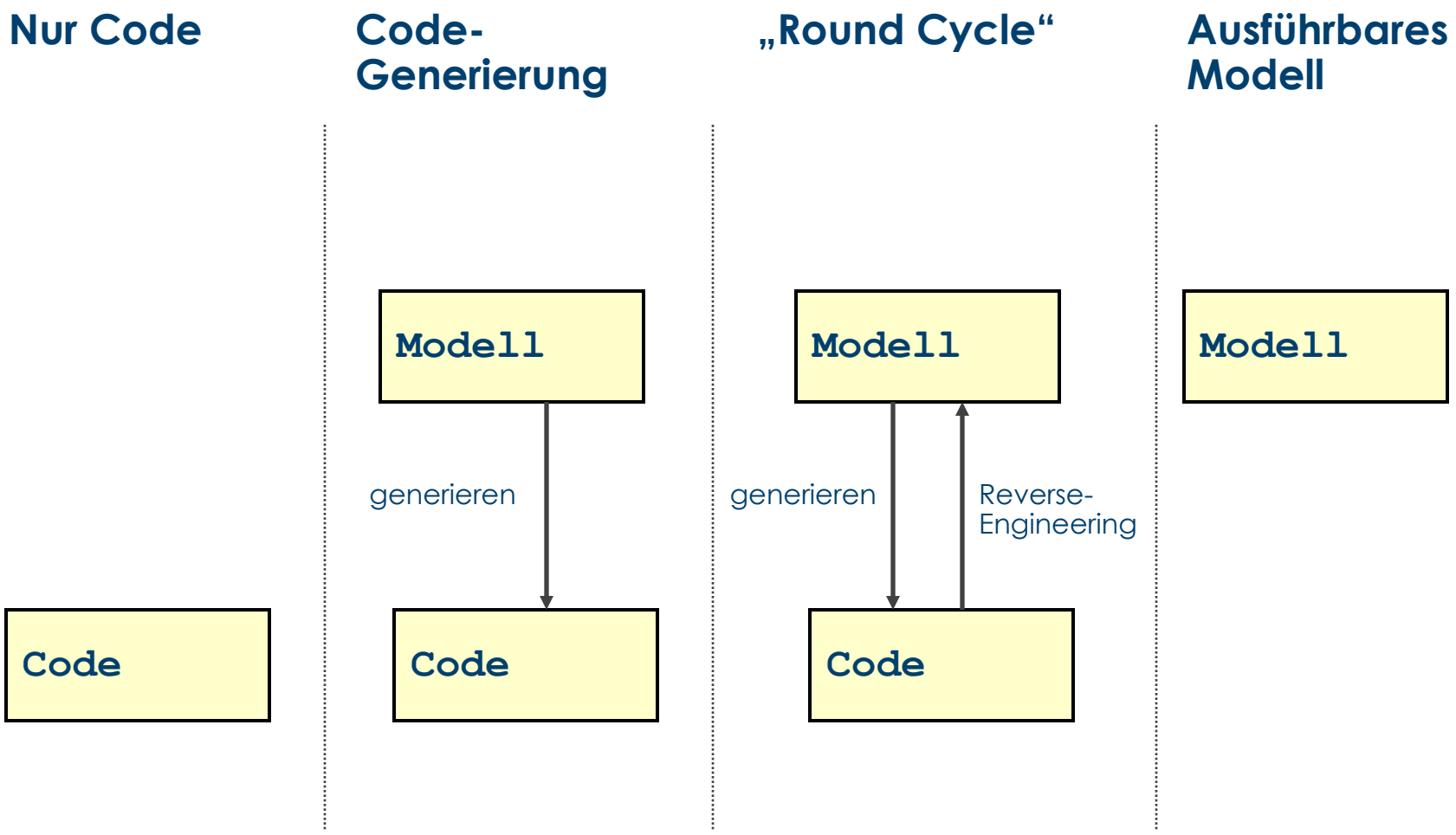
Folie 130

Übung

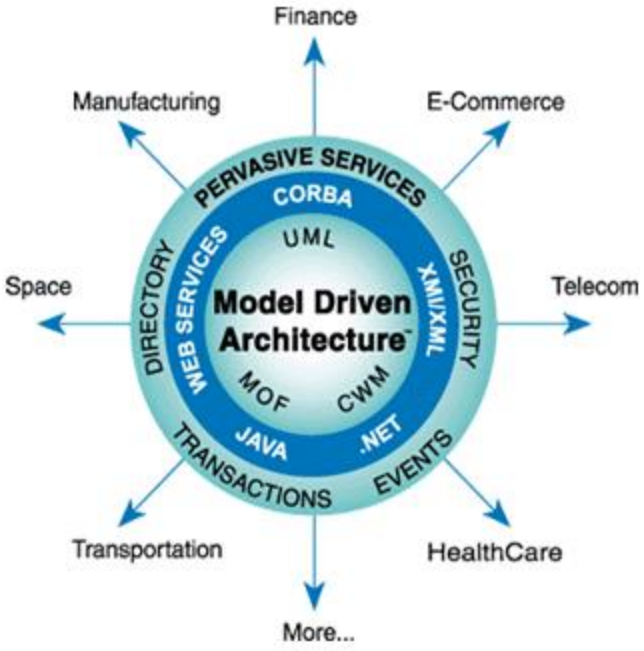


Model Driven Architecture MDA

Modelle im Entwicklungsprozess



MDA



- www.omg.org/mda
- Modelle sind zentrales Element des Entwicklungsprozesses
- Grundgedanke ist eine Modelltrennung
  - ⇨ Platform independent model (PIM): Technologieneutrale Bestandteile wie z.B. Geschäftsprozesse, Fachlogik, Domänenmodell etc.
  - ⇨ Platform specific model (PSM): Modell für die speziell zum Einsatz kommende Systemlandschaft wie z.B. J2EE
- Die Vision besteht in einer Ausführbarkeit dieser Modelle
- Viele Werkzeuge werden angeboten

UML-Profile

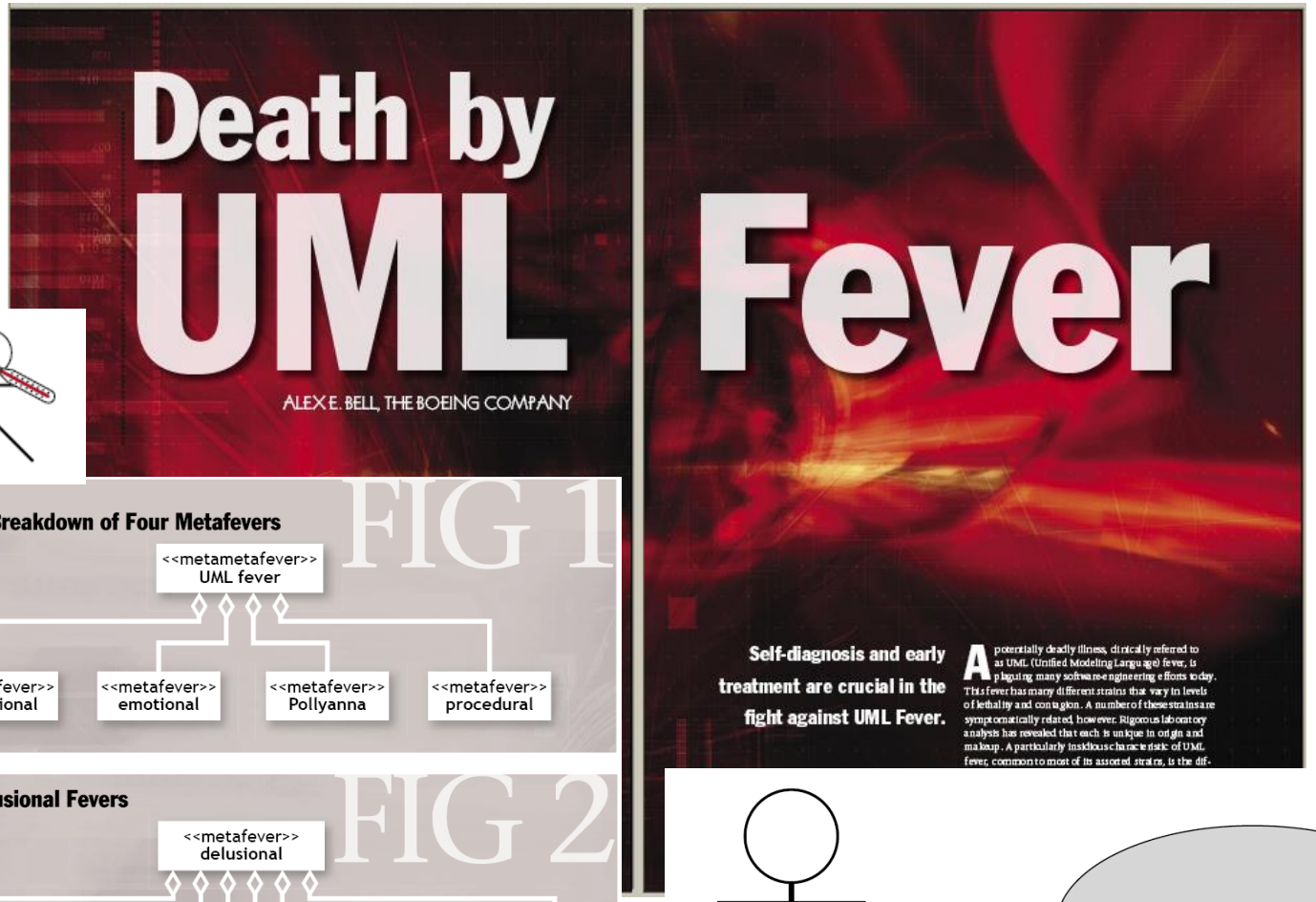
- UML-Profile spezifizieren domänenspezifische Typen, Stereotypes usw.
- UML Testing Profile
  - ⇨ The UML Testing Profile defines a language for designing, visualizing, specifying, analyzing, constructing and documenting the artifacts of test systems. [...] The UML Testing Profile can be used stand alone for the handling of test artifacts or in an integrated manner [...]
- UML Profile for Systems Engineering (SysML)
  - ⇨ [...] supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities. (www.sysml.org)
- Weitere
  - ⇨ UML Profile for Schedulability, Performance and Time
  - ⇨ UML Profile for System on a Chip (SoC)
  - ⇨ UML Profile for Enterprise Application Integration (EAI)

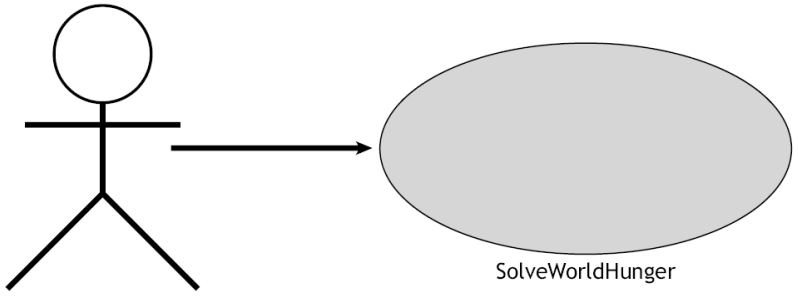
Bewertung von UML

- umfassend
- Industriestandard
- Großes Angebot an Büchern, Unterlagen und Schulung
- durch alle großen Werkzeuganbieter unterstützt
- offener, erweiterbarer Standard

- erhebliche konzeptionelle Schwächen:
  - ⇨ fehlende Systemdekomposition
  - ⇨ (zu)viele Konstrukte ohne klare Bedeutung
- Anything goes; öffnet der Beliebigkeit Tür und Tor
- (Zu) großer Sprachumfang
- UML-Modelle sind Sammlungen von Einzelmodellen: Konsistenzprobleme, Problem des Zusammensuchens relevanter Information
- Gefahr der erneuten Sprachverwirrung durch unkontrollierte und undisziplinierte Erweiterungen

Nach Martin Glinz, Universität Zürich





ACM, Alex E. Bell, Death by UML Fever, Queue, v.2 n.1, March 2004

Was UML nicht modellieren kann ...

- Projektplan, Arbeitspakete, Rollen, Aktivitäten
- Personalplanung
- Budget
- Nichtfunktionale Anforderungen (z.B. Stabilität, Wartbarkeit, Performance, Qualität, ...)
- Relationales Datenmodell (ER-Diagramm)
- Testplan, Testfälle, Testbericht
- Risiken, Qualitätssicherung
- Begriffslexikon (Glossar)
- GUI-Entwürfe
- Configuration Management
- ...