Kapitel 3

Die Unified Modeling Language (UML)

- Stand: 4.11.2015 -

3.1 Modellierung und UML

3.2 Beispiel:
→ Modellierung einer Uhr

3.3 Strukturdiagramme im Kleinen:
→ Objekt-, Klassen-, Kompositionsstruktur-, Paket-Diagramme

3.4 Verhaltensmodellierung:
→ Sequenz-, Kommunikations-, Aktivitäts- und Interaktionsübersichts- und Zustands-Diagramme

3.5 Zusammenfassung und Ausblick

3.1 Modellierung und Unified Modelling Language

Ausgangspunkt

Warum Modellierung?

Warum objektorientierte Modellierung?

Warum mit der UML?

UML-Überblick

Problemstellung

- Ausgangssituation: Sie …
  - kennen 2-3 Programmiersprachen
  - haben bisher allein oder in sehr kleinen Gruppen (<5) gearbeitet

- Neue Herausforderungen: Sie sollen …
  - mit Kollegen zusammenarbeiten, die andere Programmiersprachen nutzen
  - mit Kunden kommunizieren, die nichts von Informatik verstehen
  - komplexe, evt. verteilte Systeme realisieren, mit einer Vielzahl von Klassen, Subsystemen, verschiedensten Technologien, …
  - Entwürfe auf einer höheren Abstraktionsebene als Quellcode kommunizieren und dokumentieren
  - auch die Software-Verteilung, -Einführung und den Betrieb planen

- Lösungsweg: Abstraktion des zu realisierenden Systems
  - Erst modellieren, dann programmieren!

Warum Software modellieren?

Software ist schon eine Abstraktion, also warum noch modellieren?

- Software wird umfangreicher, nicht kleiner
  - Windows NT 5.0 ~ 40 Millionen Zeilen Code
  - Kein Programmierer kann diese Menge Code alleine bewältigen.
- Code ist oft von Entwicklern, die an der Entwicklung nicht mitgewirkt haben, nicht direkt zu verstehen
- Wir brauchen einfachere Repräsentationen für komplexe Systeme
  - Modellierung ist ein Mittel um mit Komplexität umzugehen

Systeme, Modelle, und Sichten

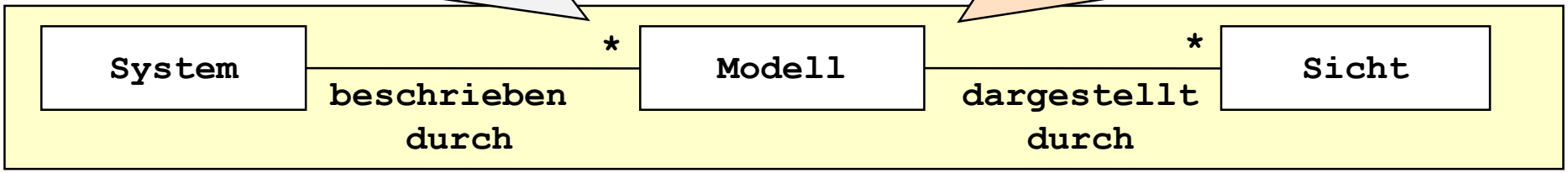
- Ein *Modell* ist eine Abstraktion, welche ein *System* oder einen Teil davon beschreibt
- Eine ***Sicht*** stellt ausgewählte Aspekte eines Modells dar.
- Modelle eines Systems können sich überschneiden. Sichten auch.

- Eine *Notation* ist ein Satz von Regeln (grafisch oder textuell) zur Darstellung von Sichten

Systeme, Modelle, und Sichten (UML)

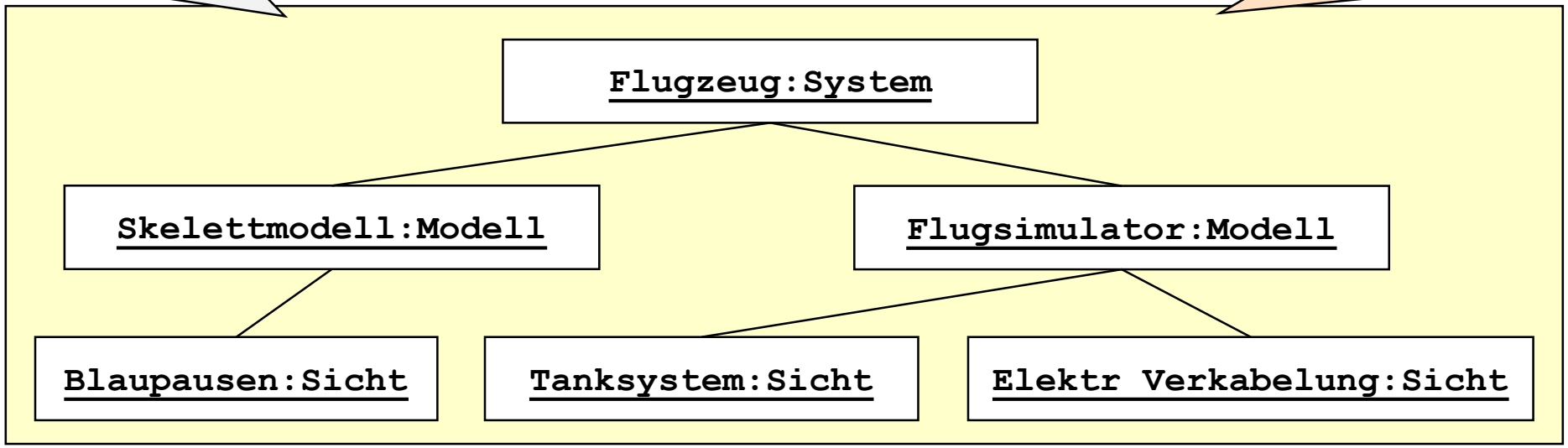
Dies ist eine statische Sicht eines Modells von Systemen, Modellen und Sichten

Sie ist in der „UML“-Notation beschrieben



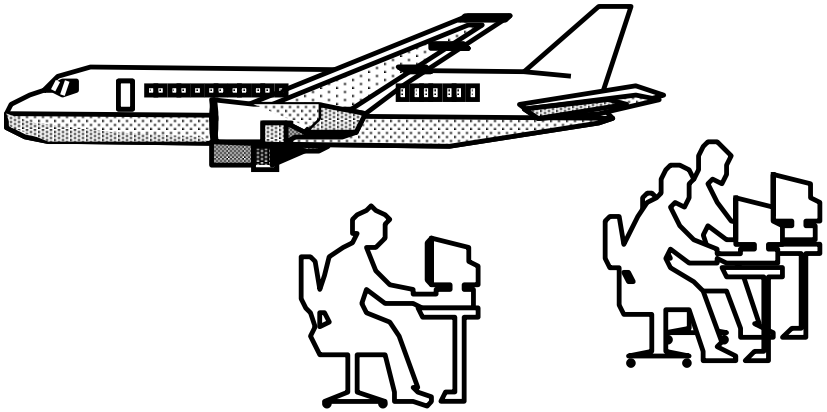
... und dies ist ihre Instanz für das vorige Beispiel.

Sie ist ebenfalls in der „UML“-Notation beschrieben

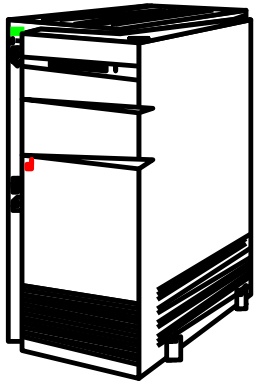


Anwendungsdomäne → Lösungsdomäne

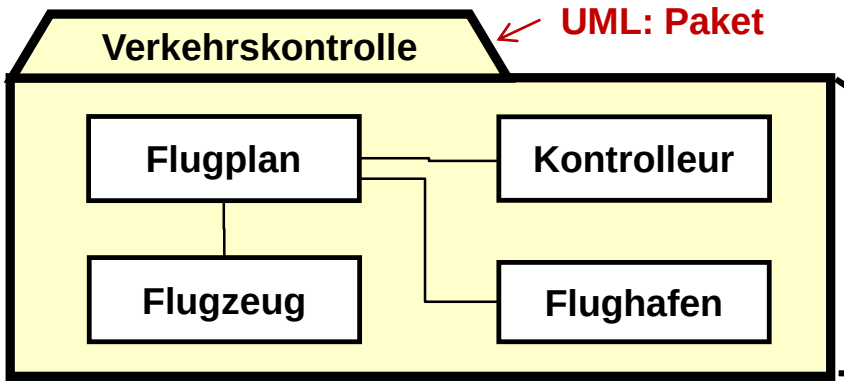
- Anwendungsdomäne (Anforderungsanalyse):
  - Die Arbeitsumgebung des Systems



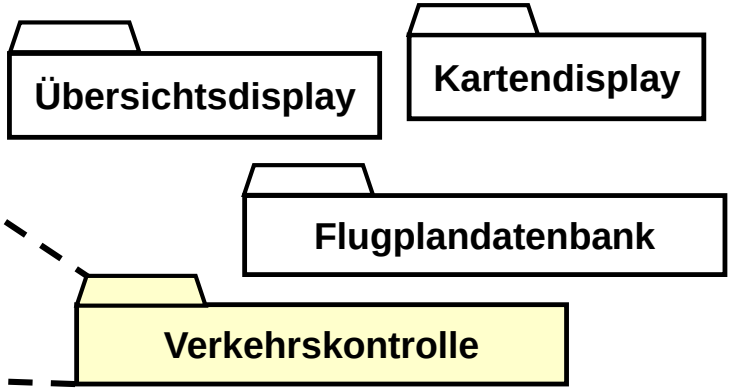
- Lösungsdomäne (Design):
  - Die zum Bau verfügbaren Technologien



- Modell der Anwendungsdomäne (Ausschnitt)



- Systemmodell (Ausschnitt)



Probleme im traditionellen SW-Prozess

Workflow	Dokumente
Analyse	Pflichtenheft (Text)
Entwurf	Flussdiagramme, ...
Implementierung	Programmcode
Test	Programmcode

Unified Modeling Language (UML)

- Standardisierte graphische Notation für alle Aktivitäten der Softwareentwicklung
  - Nutzen für Anforderungserhebung, Entwurf, Implementierung, Einsatz („deployment“), …
  - Besondere Unterstützung für objektorientierte Modellierung
  - Standardisiert durch die OMG (Object Management Group) → www.omg.org

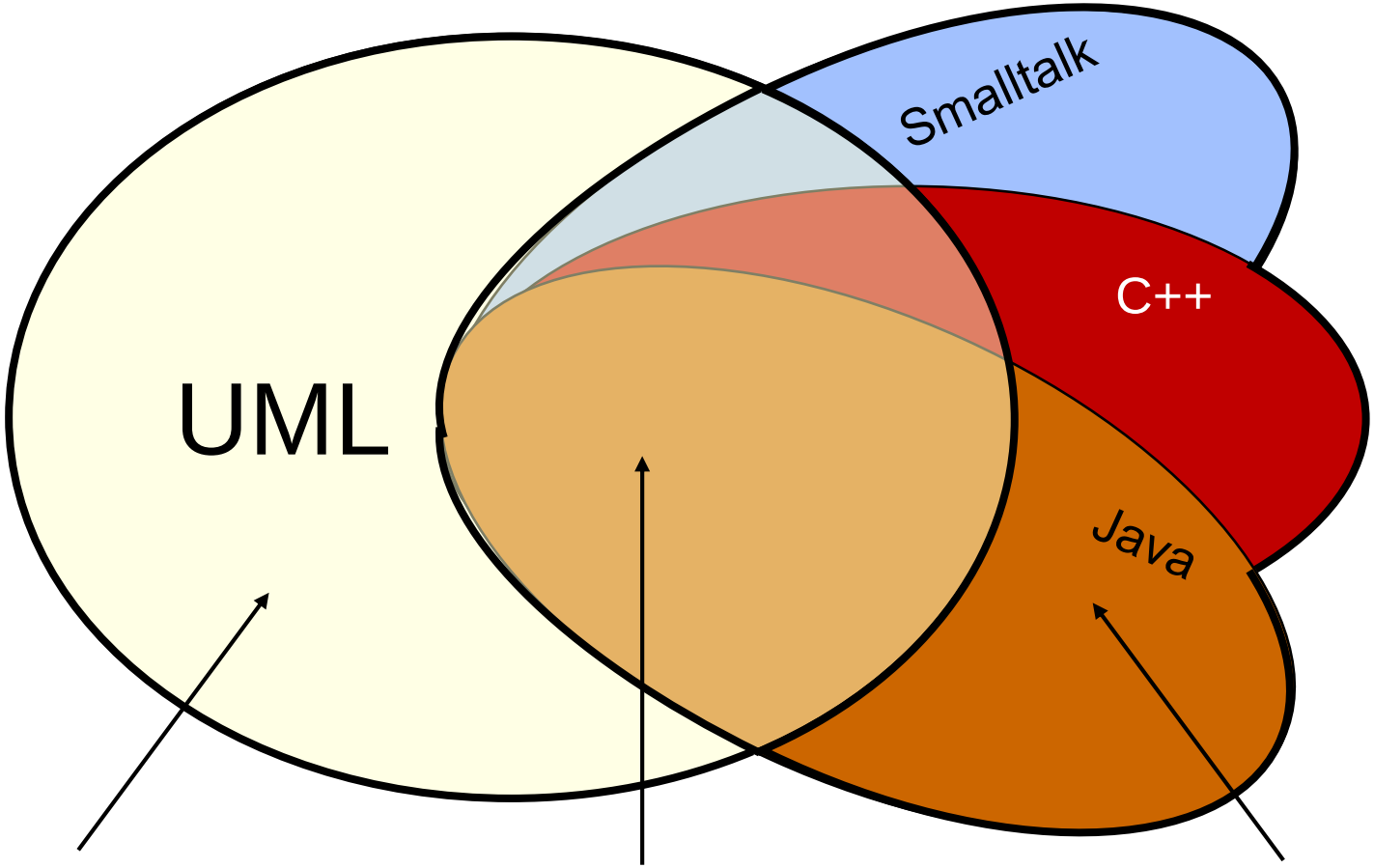
- Bietet zahlreiche Diagrammtypen für verschiedene Sichten von Software
  - statische Sichten auf die Struktur
  - dynamische Sichten auf das Verhalten

- Mittlerweile breite Werkzeugunterstützung für…
  - … die Erstellung von UML Diagrammen
  - … Code-Generierung aus Diagrammen (Forward Engineering)
  - … Diagramm-Generierung aus Code (Reverse Engineering)
  - … nahtlose Synchronisation von Diagrammen und Code (Round-Trip Engineering)

Nutzen der UML

- Kommunikation mit Anwendern
  - Wenig formal
  - Einfache graphische Notation
  - Konzentration auf das Wesentliche
- Kommunikation mit Kollegen
  - Austausch von Designs, ...
  - Abstraktion von Sprachdetails
  - Schutz vor übereilter Implementationssicht
- Bandbreite
  - unterstützt alltägliche und exotische Konzepte
  - manuell, rechner-unterstützt und kombiniert einsetzbar

Bezug von UML zu gängigen OO Sprachen



Konzepte ohne direkter Entsprechung in gängigen Sprachen

direkt ineinander abbildbarer gemeinsamer Kern

sprachspezifische Details

UML: Geschichte und Literatur

- Entstanden aus der Zusammenführung dreier führender objekt-orientierten Methodologien
  - OMT (James Rumbaugh) – Klassendiagramme, Sequenzdiagramme, …
  - OOSE (Ivar Jacobson) – Anwendungsfalldiagramme, …
  - Booch (Grady Booch) – Software-Prozess → „Unified Process“

- Literatur: Standardwerke
  - “The Unified Modeling Language User Guide” (Booch & al 1999)
  - “The Unified Modeling Language Reference Manual” (Rumbaugh & al 1999)
  - alle im Addison Wesley / Pearsons Verlag)
- Empfehlung
  - “UML Distilled” (Fowler & al. 2000, Addison Wesley) – kurz und gut!
  - „UML@Work“ (Hitz & Kappel 2005, dpunkt) – UML 2.0, deutsch, ausführlich!

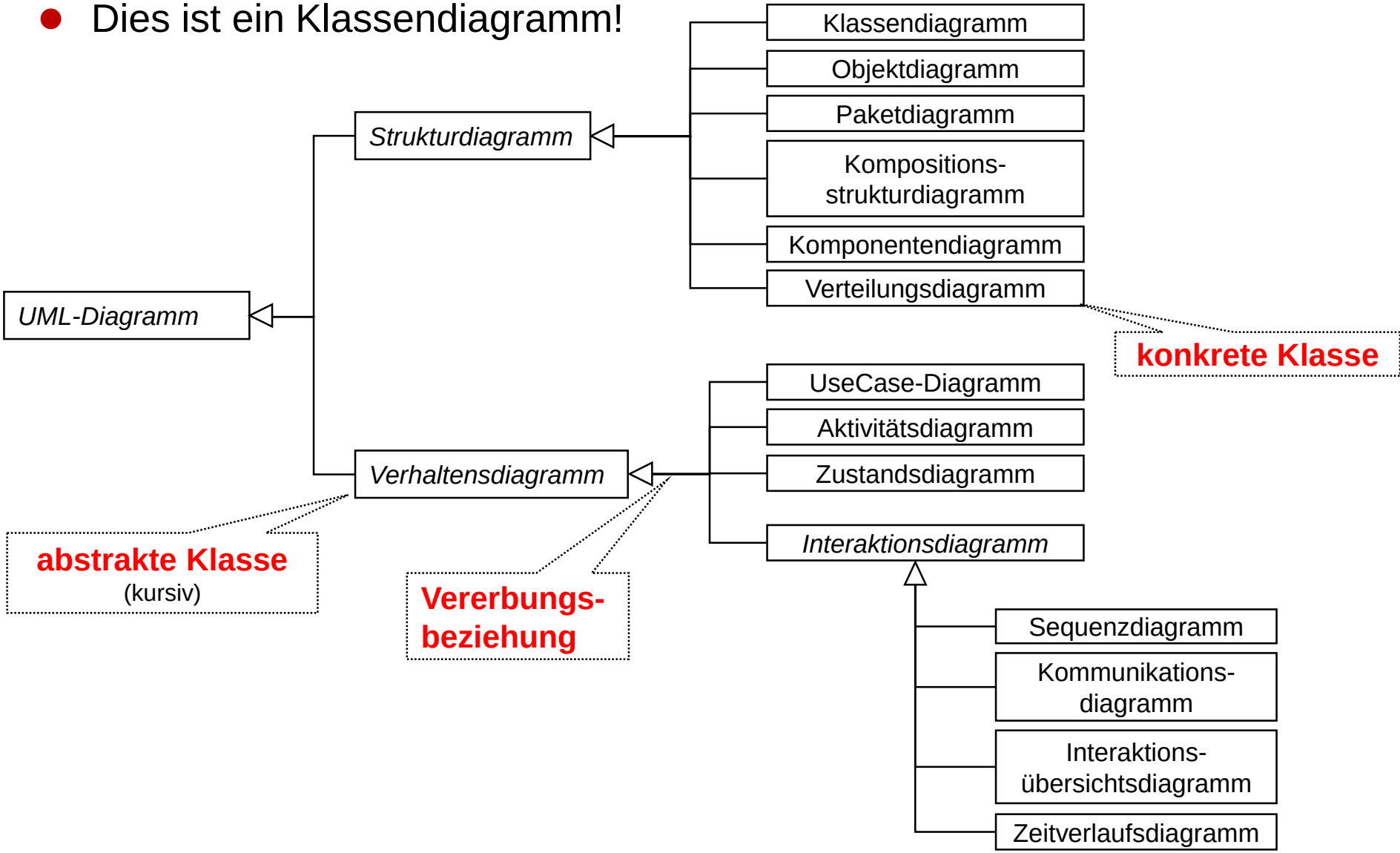
UML Versionen

- UML 1 (1997) / UML 1.1 (1998)
  - OMG-Spezifikation eines Modellierungsstandards
    - ⇨ OMG (Object Management Group): Verband führender Softwareunternehmen und Standardisierungsgremium für objektorientierte Systementwicklung

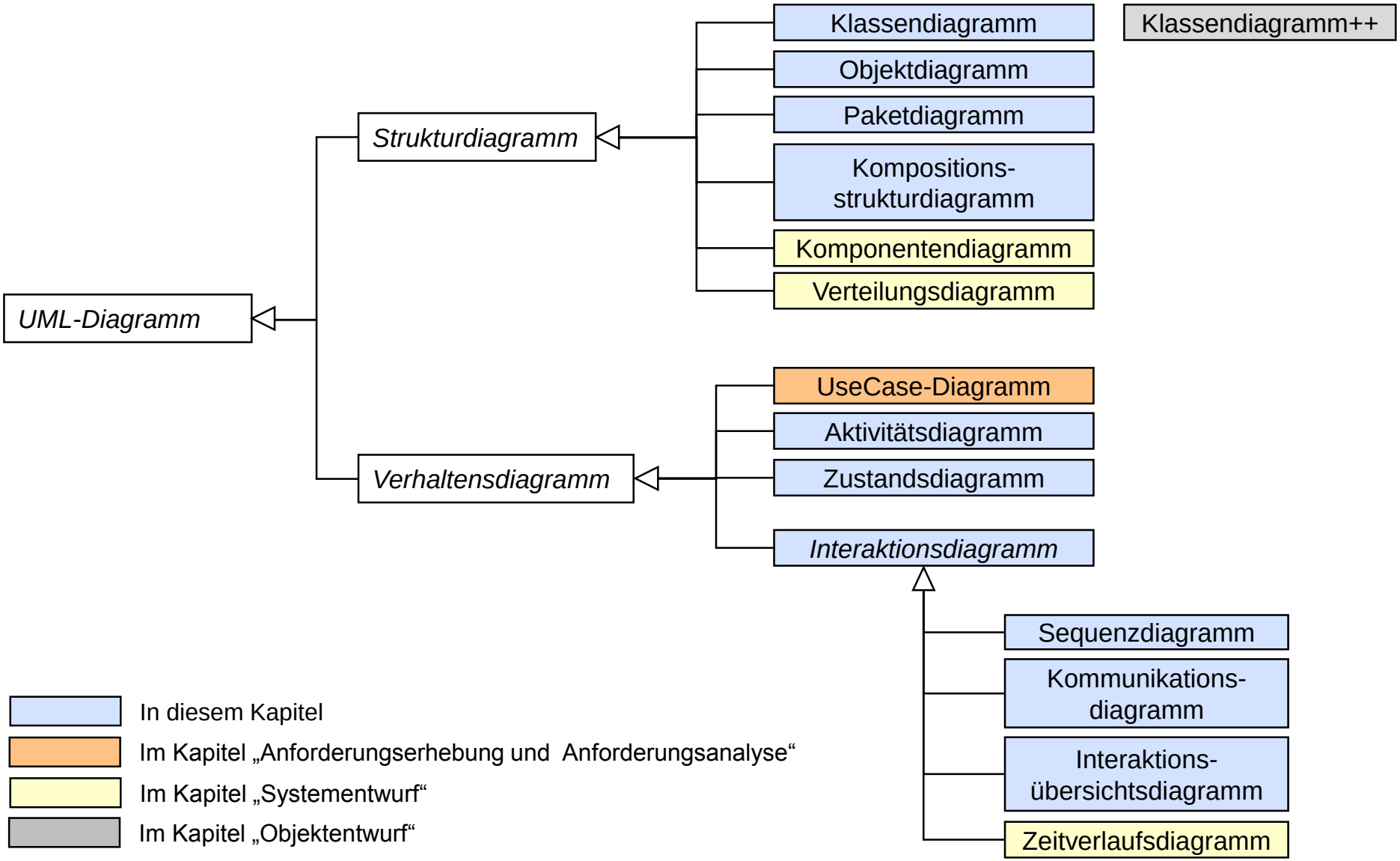
- UML 2 (2005)
  - Neue Diagrammtypen
    - ⇨ Sollen den Anforderungen heutiger Softwarearchitekturen gerecht werden
      - – Komponenten- und Serviceorientierte (SOA) Architekturen, Geschäftsprozessmanagement, Echtzeitsysteme, …
  - Modifikationen bestehender Diagrammtypen
    - ⇨ Konvergenz der Diagrammtypen, vor allem der dynamischen Diagramme
  - Grundidee: Modell-Driven Software Engineering (MDSE)
    - ⇨ Quellcode als Hauptartefakt ablösen und …
    - ⇨ …durch Modelle ersetzen, aus denen die Programme generiert werden
    - ⇨ Modelltransformationen, Generative Programmierung

UML Diagrammtypen

- Dies ist ein Klassendiagramm!



UML Diagrammtypen: Was wird wo erläutert?



3.2 Beispiel: Modellierung einer Uhr

Kurzüberblick elementarer UML-Notationen

Use-Case Diagramm

Klassendiagramm

Sequenzdiagramm

Zustandsdiagramm

Beispiel: Eine einfache LCD-Uhr

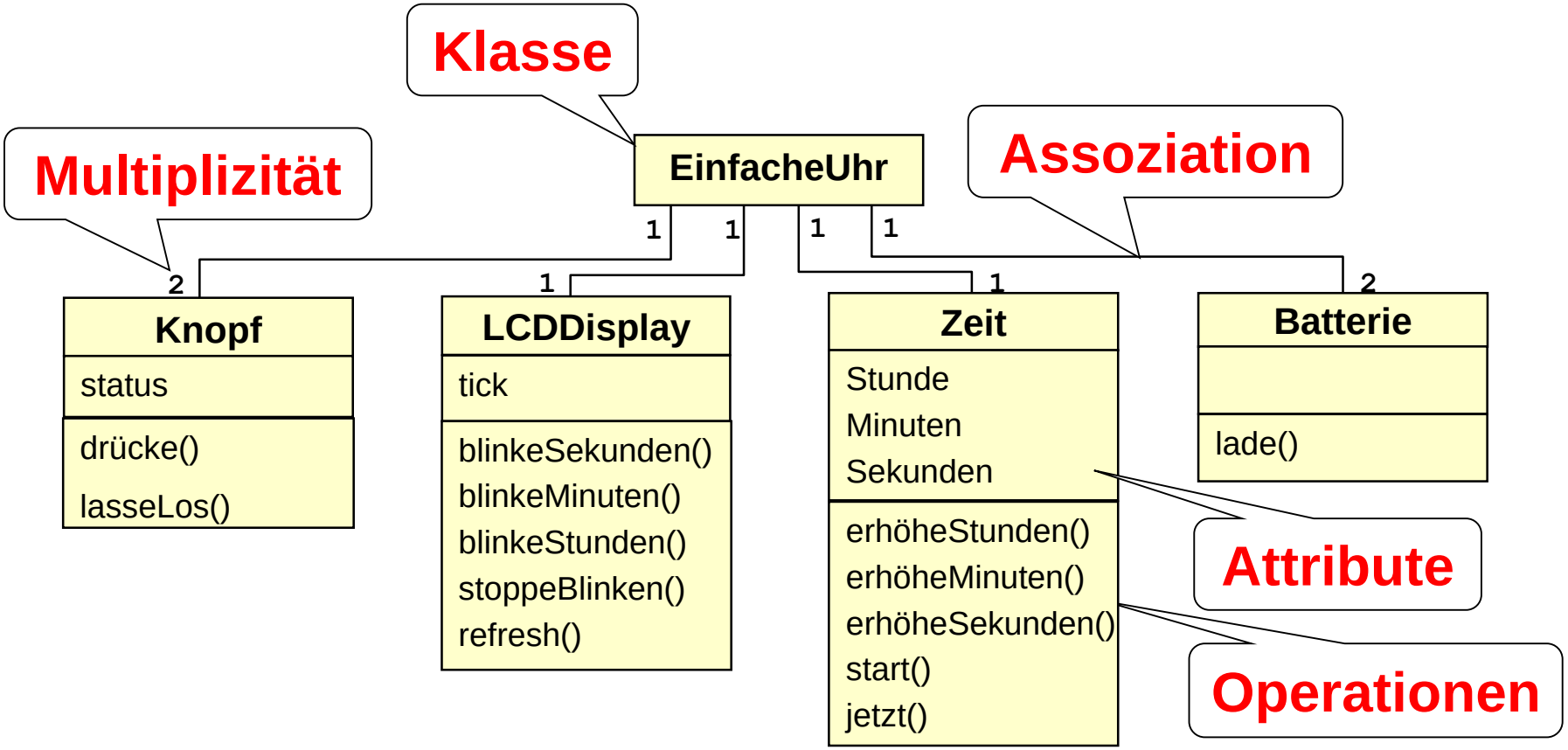
Struktur

- 1 Display
- 2 Knöpfe
- 1 Batterie
- …

Verhalten

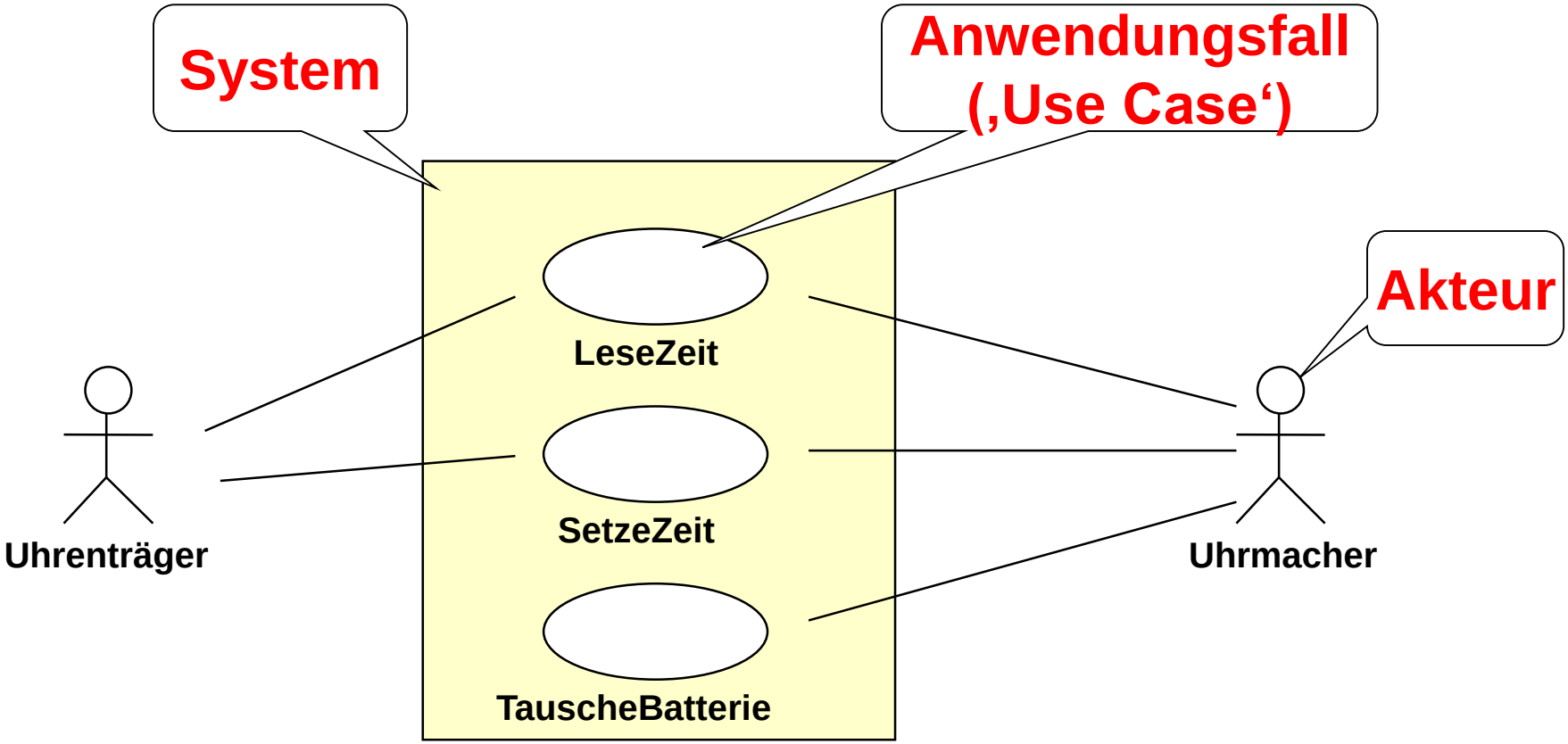
- Knopf 1 drücken
  - startet Stunden-Einstellung
  - Jeder weitere Druck schaltet weiter zu Minuten, Sekunden, Stunden, ….
- Knopf 2 drücken
  - erhöht den Wert des aktuell einzustellenden Elementes.
- Beide Knöpfe drücken
  - beendet die Einstellung

UML-Kurzübersicht: Klassen-Diagramme



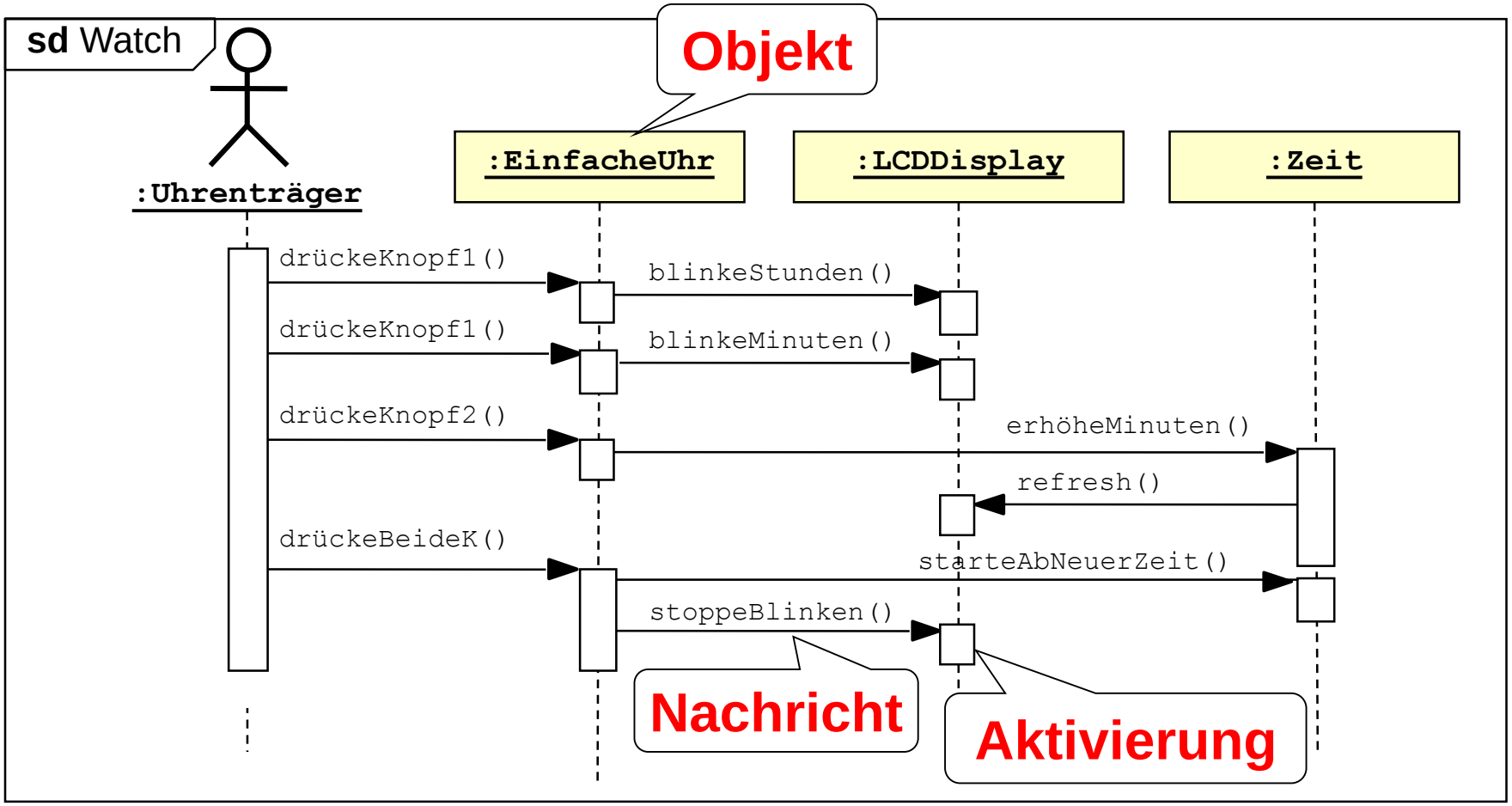
Klassendiagramme repräsentieren die Struktur eines Systems

UML-Kurzübersicht: Use-Case Diagramme



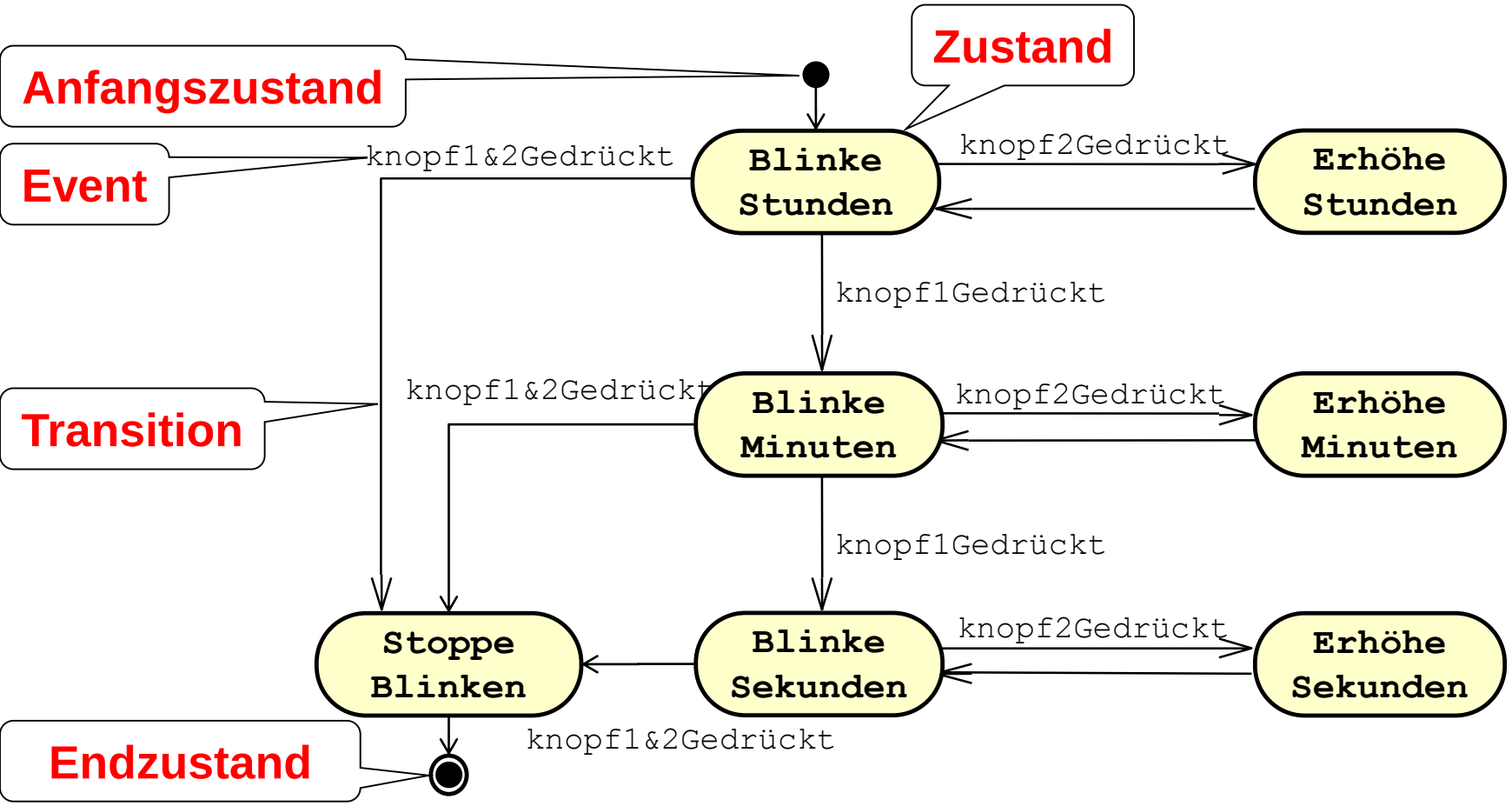
Use-Case Diagramme stellen die Funktionalität eines Systems aus Sicht der Benutzer dar.

UML-Kurzübersicht: Sequenz-Diagramme



Sequenzdiagramme stellen die Interaktion von Objekten dar.
Hier: Einstellen der Zeit an einer LCD-Uhr.

UML-Kurzübersicht: Zustands-Diagramme



Zustandsdiagramme stellen die interne Sicht von Objekten dar:
Zustände und Zustandsübergänge ➔ endlicher Automat.

UML Notationskonventionen

- Diagramme sind Graphen
  - Knoten sind Konzepte
  - Kanten sind Beziehungen zwischen Konzepten
- Rechtecke sind Typen oder Instanzen
- Instanzen werden durch „Rolle : Typ“ und Unterstreichung gekennzeichnet
  - `meineUhr:EinfacheUhr`
  - `Joe:Feuerwehrmann`
- Bei Typen werden die Namen nicht unterstrichen
  - `EinfacheUhr`
  - `Feuerwehrmann`

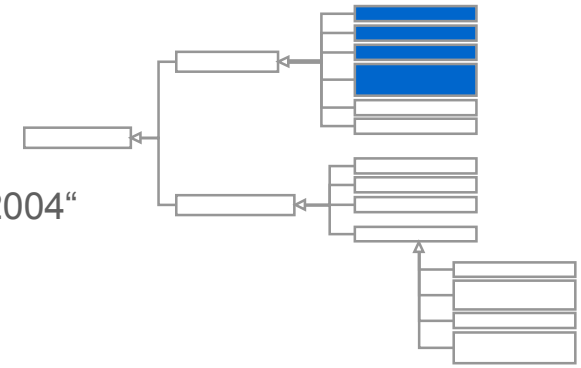
3.3 Strukturdiagramme „im Kleinen“

3.3.1 Objektdiagramme – „UML@Work“, Seite 118 - 120

3.3.2 Klassendiagramme – „UML@Work“, Seite 52 - 101

3.3.3 Kompositionsstrukturdiagramme – „JOT, Vol. 3, Nr 10, 2004“

3.3.4 Paketdiagramme – „UML@Work“, Seite 120 – 134



JOT = Journal of Object Technology

Strukturdiagramme „im Kleinen“

Beschreiben die kleinsten in der objektorientierten Modellierung erfassten Teile eines Entwurfes

- Klassendiagramme
  - Klassen: definieren Struktur und Verhalten von Instanzen
  - Assoziationen: definieren Beziehungen zwischen Instanzen der Klassen
- Objektdiagramme
  - zur Verdeutlichung von Objektbeziehungen, die sich auf Klassenebene (durch Assoziationen) nicht ausreichend beschreiben lassen
- Paketdiagramme
  - Gruppen von Klassen, die thematisch zusammengehören
  - … allerdings nicht unbedingt zur Laufzeit gemeinsam auftreten

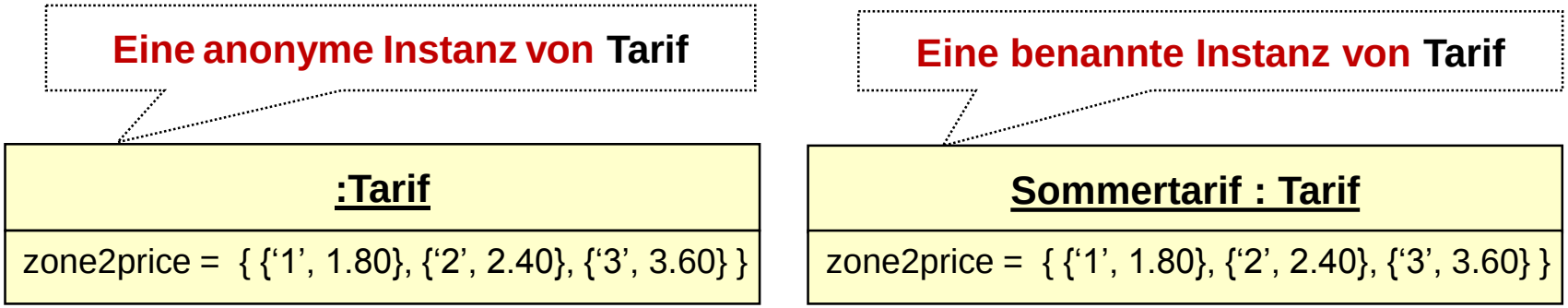
Diagrammtypen, die die Struktur „im Großen“ beschreiben, werden im Kapitel „Systementwurf“ behandelt. Sie beschreiben Einheiten, die für die modulare Komposition, die Verteilung und die Laufzeit von Software Bedeutung haben

3.3.1 Objektdiagramme

Objektdiagramm

Objekte werden auch durch Rechtecke dargestellt, aber:

- Das Namensfeld einer Instanz ist <u>unterstrichen</u> und besteht aus:
  - Einem **Namen für die Rolle** dieser Instanz (optional)
  - Einem **Doppelpunkt** als Trenner
  - Dem **Typ** der Instanz
- Die Attribute werden zusammen mit ihren jeweiligen Werten angegeben
- Methoden werden nicht angegeben, da sie nicht objektspezifisch anders sein können, als die Klasse es vorgibt.



3.3.2 Klassendiagramme

3.3.2.1 Klassen

3.3.2.2 Stereotypen

3.3.2.3 Assoziationen

3.3.2.4 Abbildung Klassendiagramm → Code

3.3.2.5 Assoziationsklassen

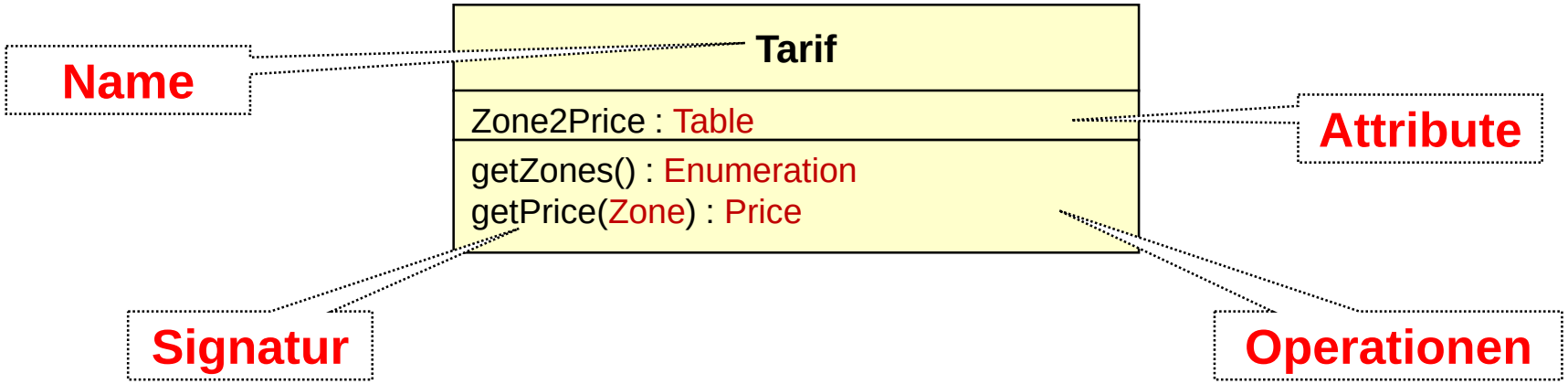
3.3.2.6 Qualifizierte Assoziationen

3.3.2.7 Aggregation und Komposition

3.3.2.8 Vererbung und Interfaces

3.3.2.1 Klassen

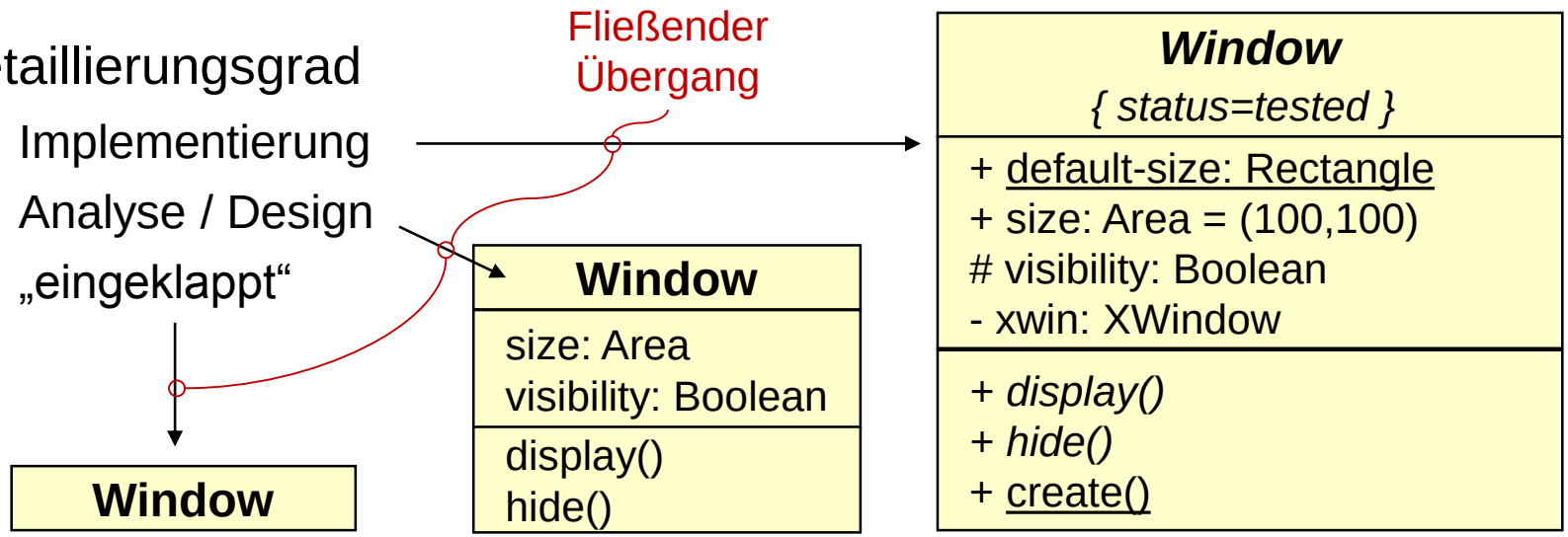
Klassendiagramm ▶ Klassen



- Eine ***Klasse*** repräsentiert ein Konzept.
- Eine Klasse kapselt Zustand ***(Attribute)*** und Verhalten ***(Operationen).***
- Jedes Attribut hat einen ***Typ.***
- Jede Operation hat eine ***Signatur.***
- Der Klassenname ist die einzige unverzichtbare Information
  - Hinzufügen weiterer Information beim Fortschreiten des Modellierungsprozesses
  - Es ist möglich, unwichtige Details in einer teilweisen Sicht des Modells zu verstecken

Klassendiagramm ▶ Verschiedene Sichten

- Detaillierungsgrad
  - Implementierung
  - Analyse / Design
  - „eingeklappt“



- Sichtbarkeit (Implementierungs-Ansicht)
  - public: **+**
  - protected: **#**
  - private: **–**

- Weitere Notationen
  - Klassenvariable / -methode: unterstrichen
  - abstrakte Klasse / Methode: *kursiv*

Klassendiagramm ▶ Abstrakte Klassen und Interfaces

Abstrakte Klassen

- Definiton
  - Implementierung unvollständig
  - Enthalten abstrakte Methoden
- Notation
  - Kursivschrift für Namen abstrakter Typen und Methoden
  - Alternativ: Constraint {abstract} in geschweiften Klammern
- Äquivalente Beispiele

Flugzeug
tank
starten() *beladen()* *tanken()*

Flugzeug { abstract }
tank
starten() *beladen()* *tanken()*

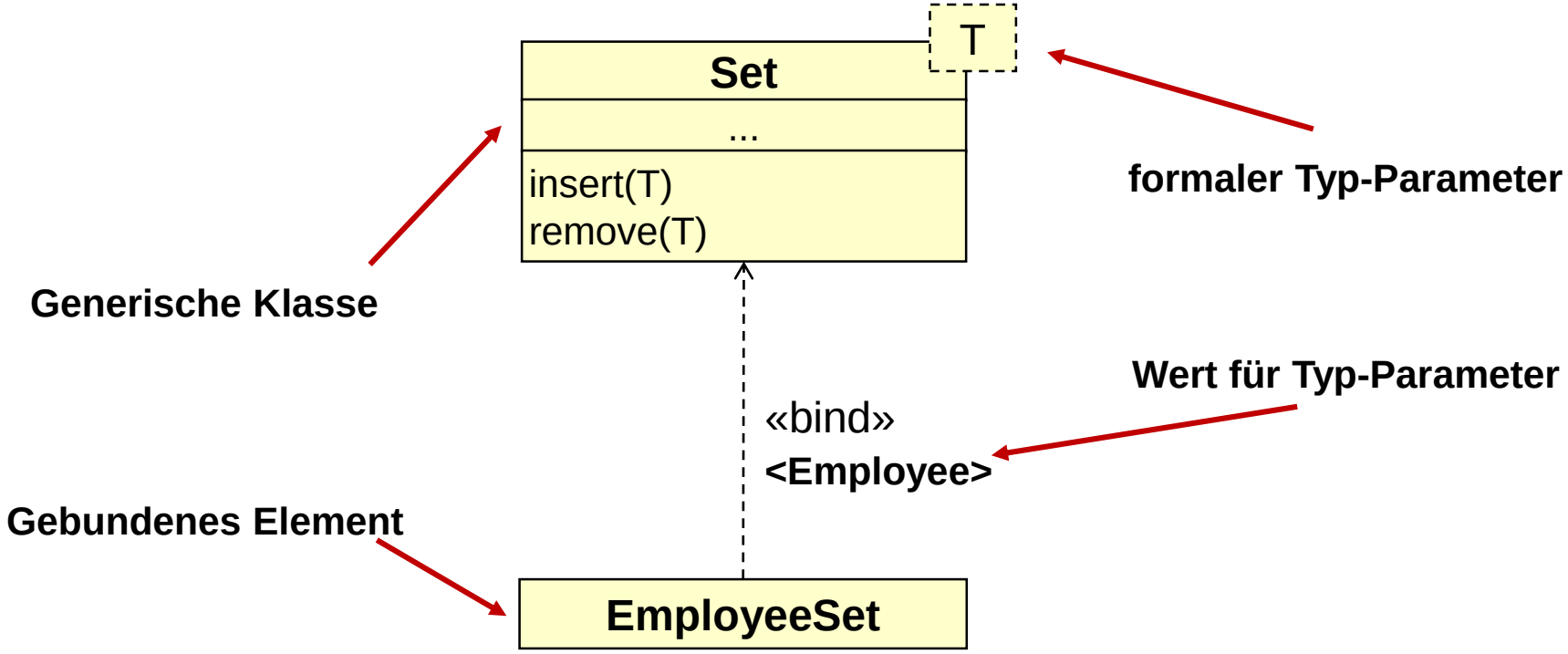
Interfaces

- Definition
  - Ganz abstrakter Typ
  - Keine Attribute und keine Methodenimplementierungen
- Notation
  - wie Abstrakte Klasse
  - evtl. zusätzlich Angabe des Stereotyps <<interface>>
- Beispiel

Flugobjekt <<interface>>
starten() *fliegen(Ziel)* *landen()*

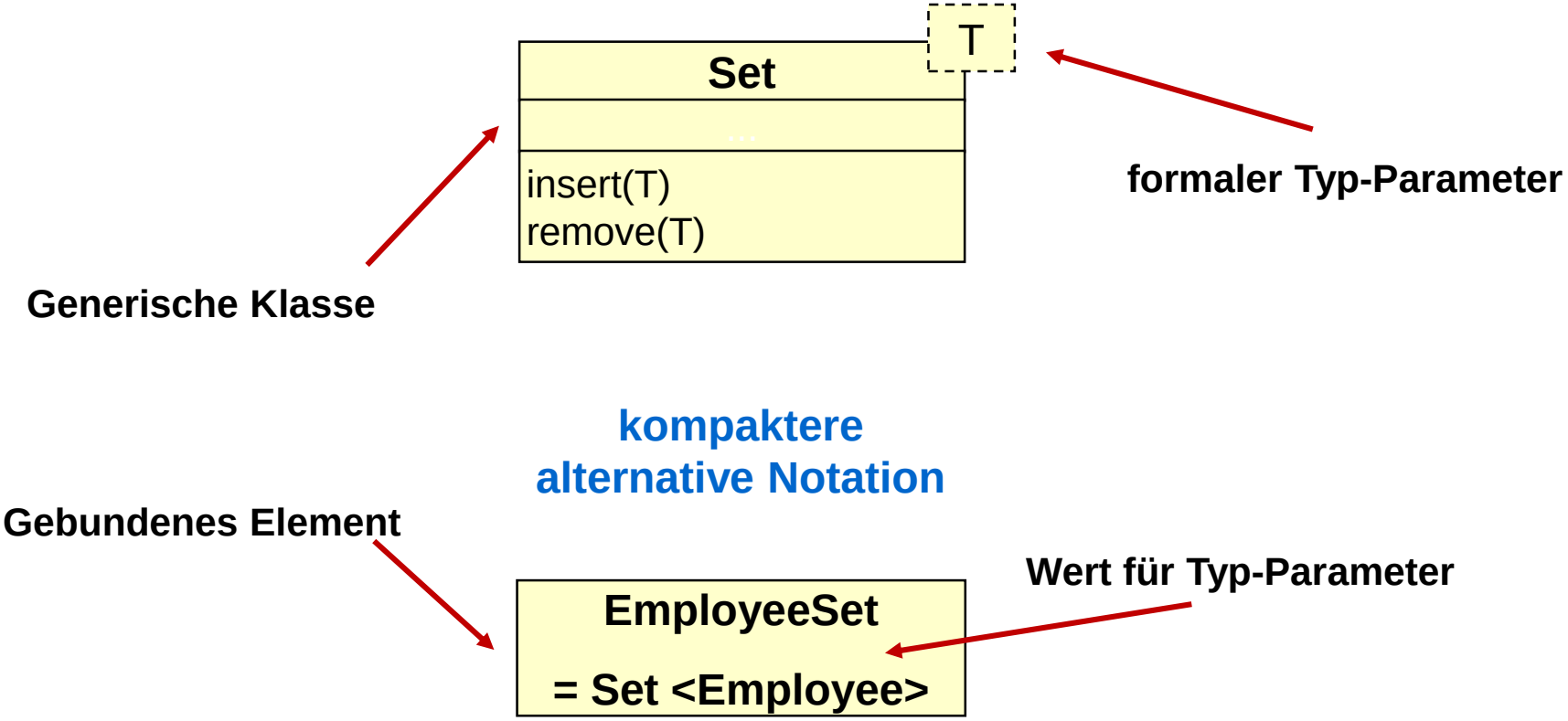
UML, statisches Modell: Generische Klassen

- Klassen mit Typ-Parameter
  - C++: "Templates"
  - Java: ab JDK 1.5



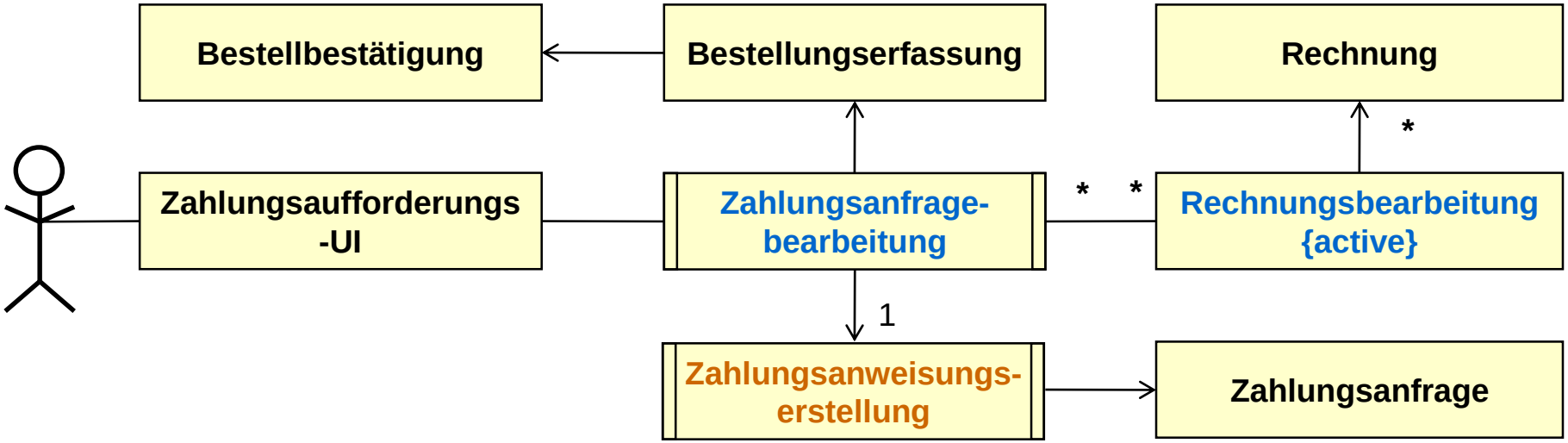
UML, statisches Modell: Generische Klassen

- Klassen mit Typ-Parameter
  - C++: "Templates"
  - Java: ab JDK 1.5



Aktive Klassen / Objekte

- Aktive Klasse: Jede ihrer Instanzen ist ein eigener Thread.
- Notation
  - UML 2.0: Klasse/Objekt mit doppeltem vertikalem Rand oder {active}
- Beispiel



  - Zahlungsanfragebearbeitung, Rechnungsbearbeitung und Zahlungsanweisungserstellung sind aktive Klassen
  - Hingegen sind z.B. Bestellerfassungen nur auf Anfrage aktiv

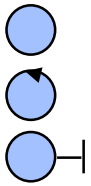
3.3.2.2 Stereotypen

Stereotypen gibt es nicht nur in Klassendiagrammen

Sie werden hier aber erstmalig eingeführt

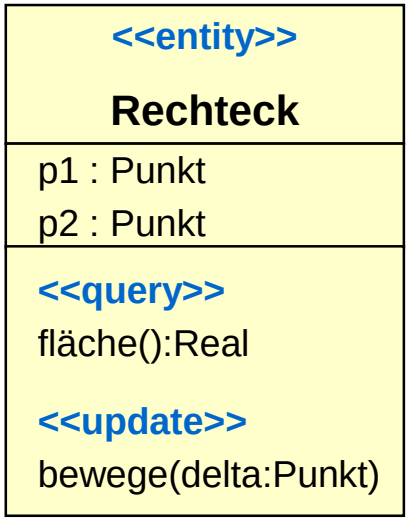
Stereotypen ▶ Spezielle semantische Kategorien

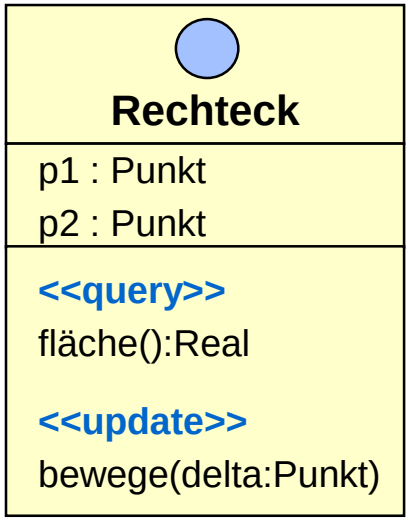
- Bisher nur Notationen für eine feste Menge von Konzepten
  - Die Welt ist aber unendlich…
- Stereotyp
  - Hinweis auf semantische Kategorie für die es keine spezifische Notation gibt
  - Ansatzpunkt für anwendungsspezifische Erweiterungen
- Anwendbarkeit
  - Allgemein (Klassen, Variablen, Operationen, Beziehungen, ...)
- Notation
  - “<<stereotyp>>“oder graphisches Symbol, z.B.
    - ⇨ <<entity>>
    - ⇨ <<controller>>
    - ⇨ <<boundary>>
    - ⇨ <<interface>>
    - ⇨ <<selbst definierte Kategorie>>



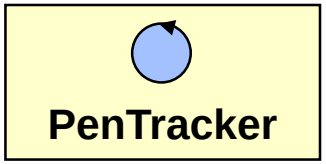
Stereotypen ▶ Beispiele

- Klasse mit Stereotypen





- Drei Äquivalente Darstellungen der Klasse PenTracker



3.3.2.3 Assoziationen

Assoziationen

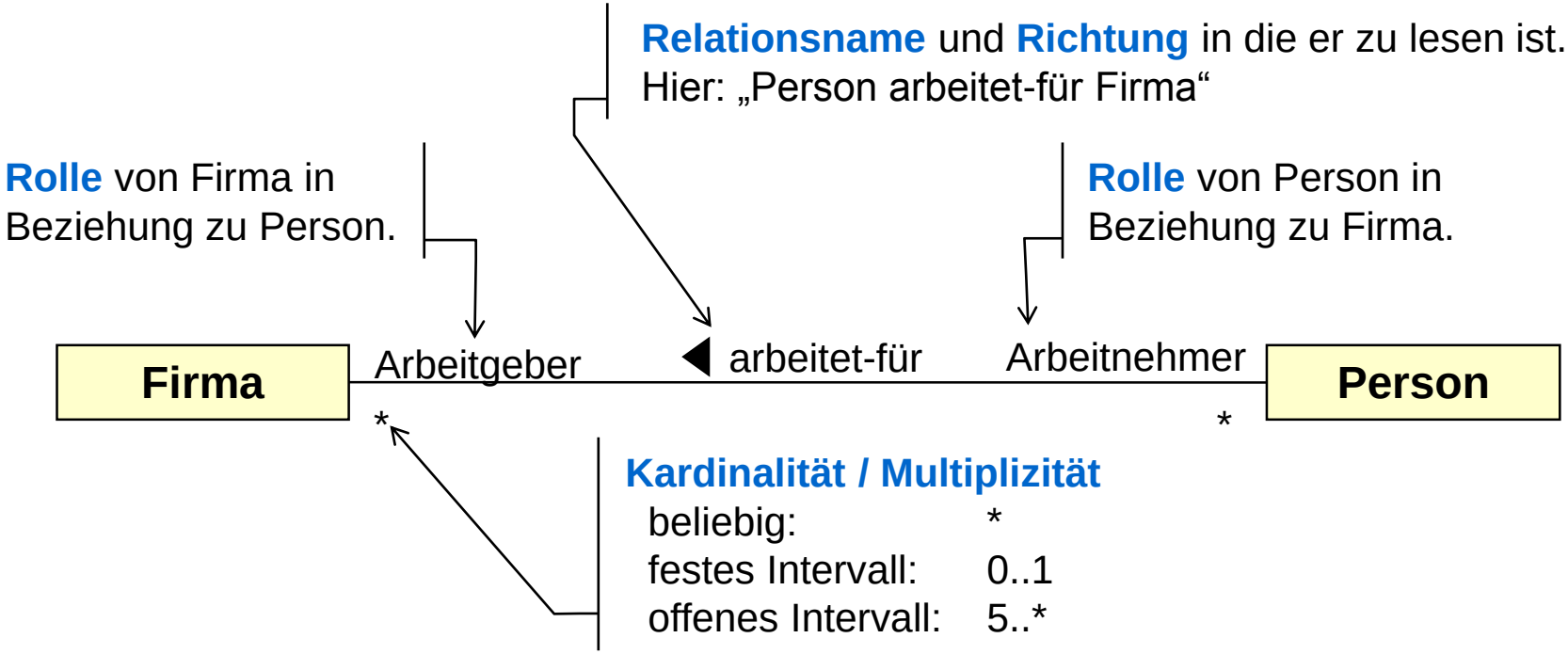
Rollen

Multiplizitäten

Aussagen über legale / illegale Objektbeziehungen

Klassendiagramm ▶ Assoziationen

- Assoziationen definieren Beziehungen zwischen Instanzen von Klassen.

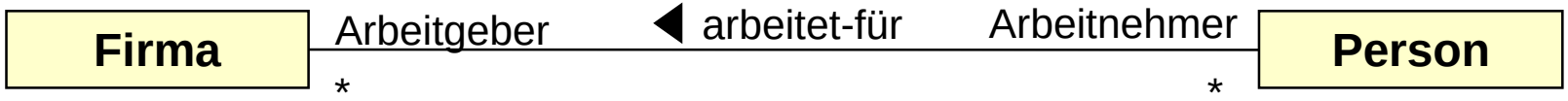


- Die Multiplizität am jeweiligen Ende einer Assoziation gibt an, mit wie vielen Zielobjekten ein Quellobjekt in Beziehung stehen kann.

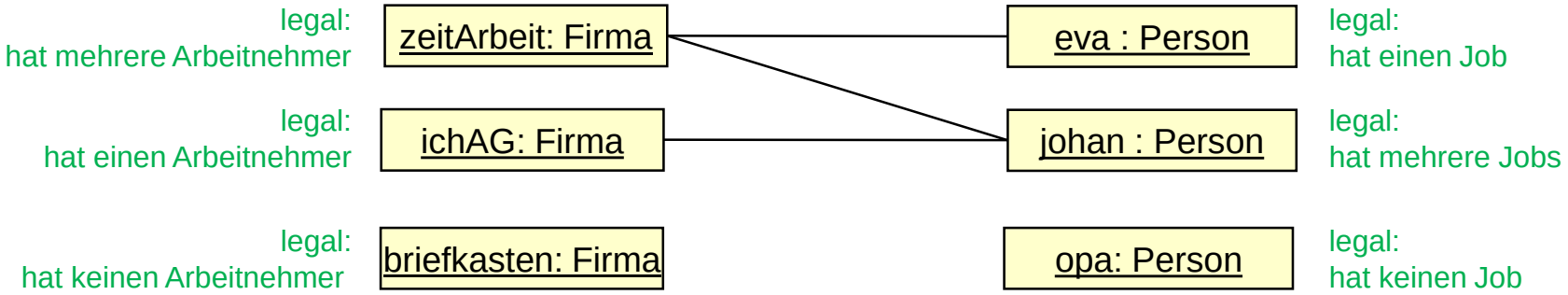
Klassendiagramm ▶ Assoziationen ▶ Multiplizität

- Assoziationen definieren Beziehungen zwischen Instanzen von Klassen. Die Multiplizität am jeweiligen Ende einer Assoziation gibt an, mit wie vielen Zielobjekten ein Quellobjekt in Beziehung stehen kann.

- Beispiel: N zu M Assoziation



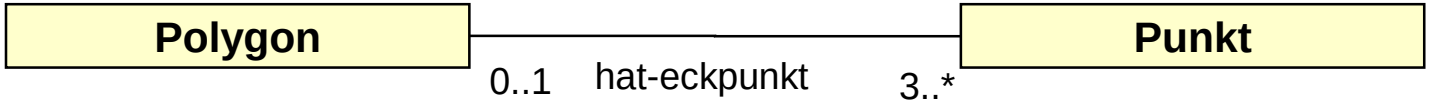
- Dazu passendes Objektdiagramm



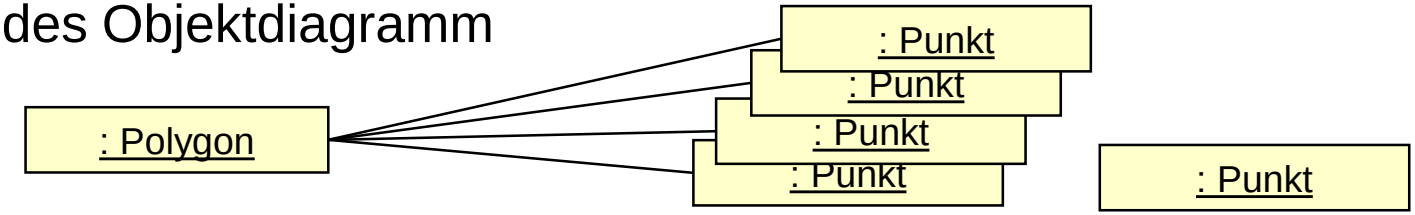
Klassendiagramm ▶ Assoziationen ▶ Multiplizität

- Assoziationen definieren Beziehungen zwischen Instanzen von Klassen. Die Multiplizität am jeweiligen Ende einer Assoziation gibt an, mit wie vielen Zielobjekten ein Quellobjekt in Beziehung stehen kann.

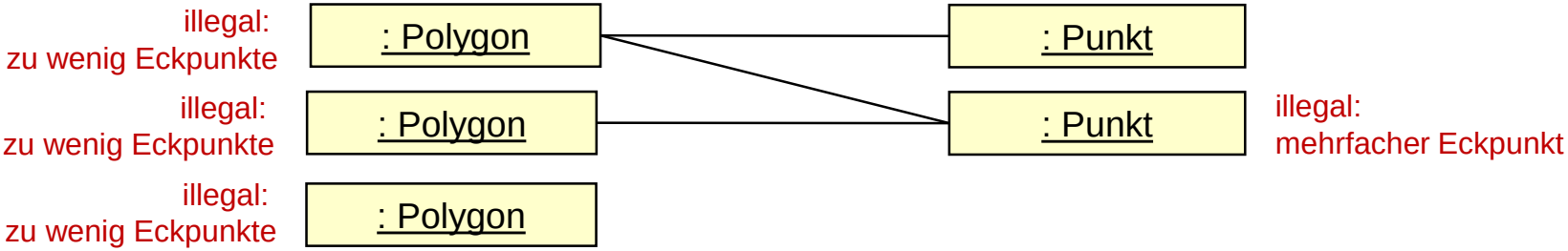
- Beispiel: 1 zu N Assoziation



- Dazu passendes Objektdiagramm



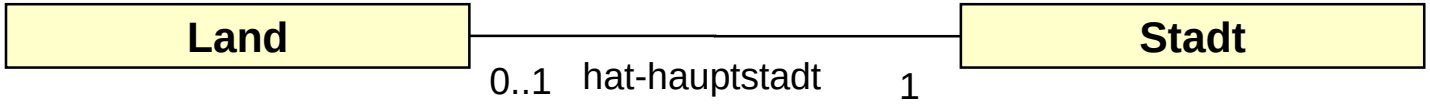
- Illegales Objektdiagramm



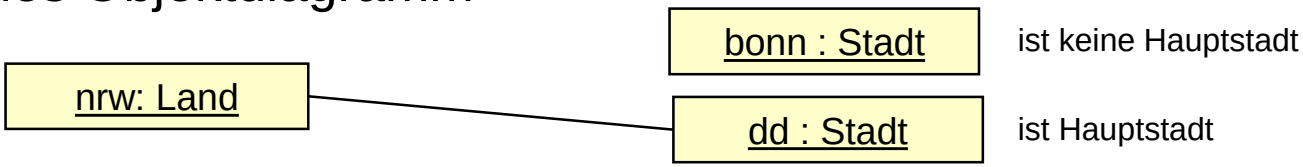
Klassendiagramm ▶ Assoziationen ▶ Multiplizität

- Assoziationen definieren Beziehungen zwischen Instanzen von Klassen. Die Multiplizität am jeweiligen Ende einer Assoziation gibt an, mit wie vielen Zielobjekten ein Quellobjekt in Beziehung stehen kann.

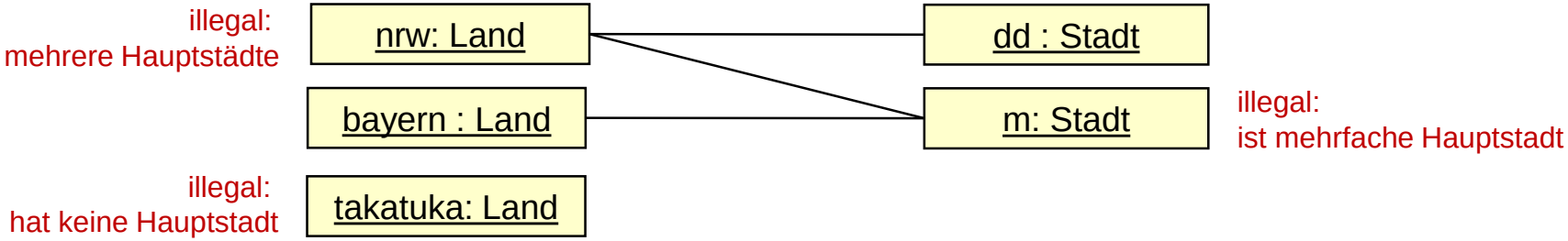
- Beispiel: 1 zu 1 Assoziation



- Dazu passendes Objektdiagramm



- Illegales Objektdiagramm

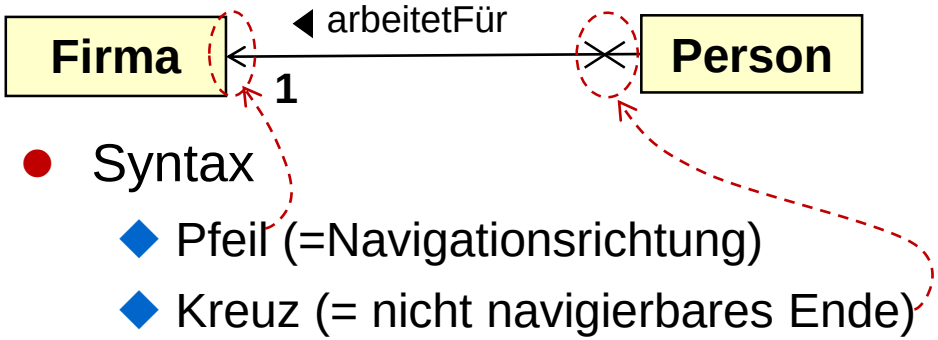


Klassendiagramm ▶ Assoziationen ▶ Navigation

Bidirektionale Assoziationen

- Syntax
  - Einfache Linie
  - … ohne Pfeilen und Kreuzen
- Semantik
  - Navigation in beide Richtungen möglich
  - Im Beispiel
    - ⇨ Person kennt ihren Arbeitgeber
    - ⇨ Firma kennt ihre Angestellten
- Nutzung
  - Konzeptuelle Sicht

Unidirektionale Assoziationen



- Syntax
  - Pfeil (=Navigationsrichtung)
  - Kreuz (= nicht navigierbares Ende)
- Semantik
  - Navigation nur in Pfeilrichtung möglich
  - Im Beispiel
    - ⇨ Person kennt ihren Arbeitgeber
    - ⇨ Firma kennt ihre Angestellten <u>nicht</u>
- Nutzung
  - Implementierungssicht

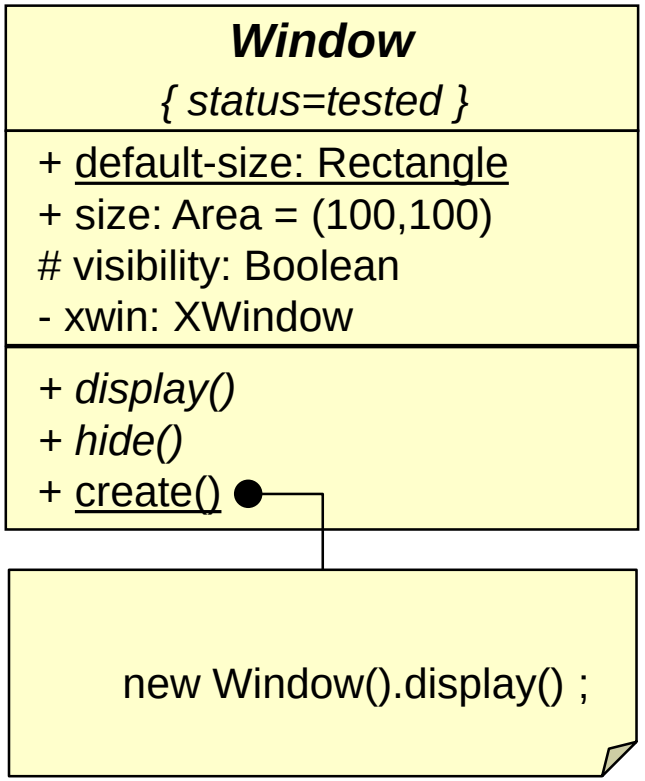
3.3.2.4 Klassendiagramm → Code

Klassendiagramm mit Assoziationen → Klassendiagramm ohne Assoziationen → Code

Assoziationen sind (implizite) Klassen

UML → Code

UML ohne Assoziationen



Java-Quellcode

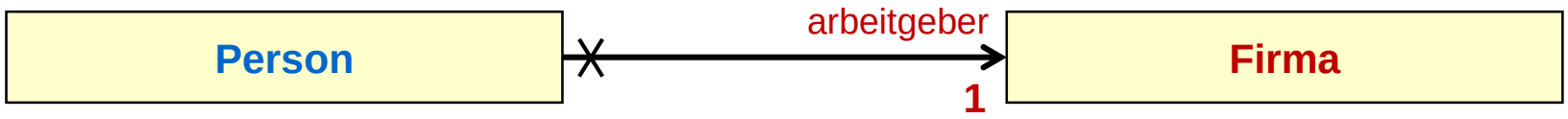
```
@status{ tested = true }
abstract class Window {

    public static Rectangle default-size ;
    public Area size = new Area(100,100) ;
    protected Boolean visibility ;
    private Xwindow xwin ;

    public abstract display() {}
    public abstract hide() {}
    public static create() {
        Window w = new Window() ;
        w.display() ;
        return w;
    }
}
```

▶ Unidirektionale Assoziation (1:1)

Objektentwurfsmodell mit Assoziation

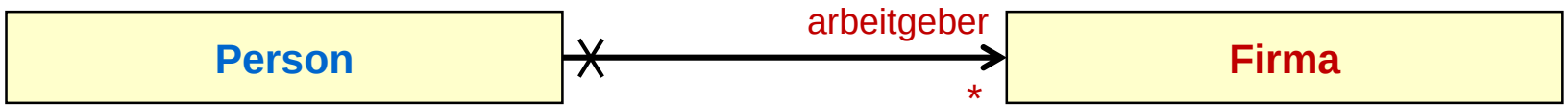


Objektentwurfsmodell ohne Assoziation

- Unidirektionale 1:1 Assoziationen sind einfach:
  - Durch Instanzvariable in der „referenzierenden“ Klasse implementieren
    - ⇨ Der Name der Instanzvariablen ist der Rollenname am Pfeilende
    - ⇨ Der Typ der Instanzvariablen ist die referenzierte Klasse.
  - Die „referenzierte“ Klasse hat **keine** solche Instanzvariable

▶ Unidirektionale Assoziation (1:N)

Objektentwurfsmodell mit Assoziation



Objektentwurfsmodell ohne Assoziation

- Unidirektionale 1:N Assoziationen sind genauso einfach:
  - Durch Instanzvariable in der „referenzierenden“ Klasse implementieren
    - ⇨ Der Name der Instanzvariablen ist der Rollenname am Pfeilende
    - ⇨ Der Typ der Instanzvariablen ist eine **Collection** von Elementen der referenzierte Klasse.
  - Anmerkung: In Java ist *Collection* das Interface zu Klassen wie *ArrayList*, *Set*, etc.

▶ Bidirektionale Assoziation (1:1)

Objektentwurfsmodell mit Assoziation

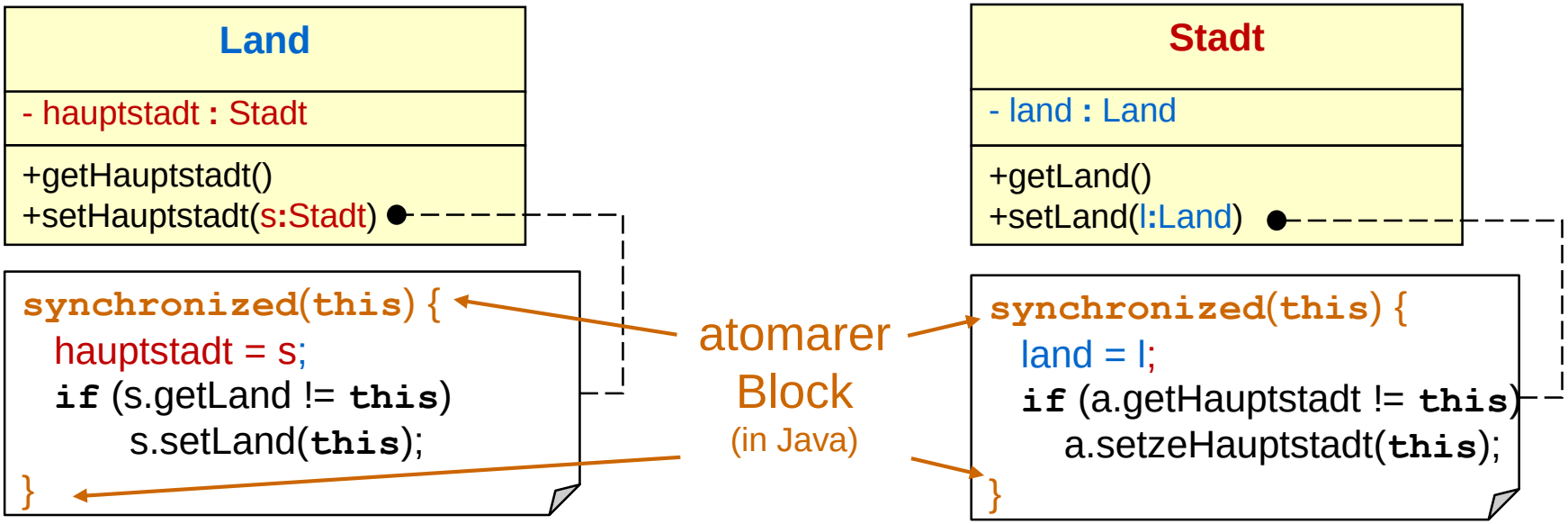
Objektentwurfsmodell ohne Assoziation

- Im Prinzip ist eine bidirektionale Assoziation das Gleiche wie zwei gegengleich zeigende Unidirektionale
- Problem: Bei Setzen einer Referenz muss der Rückverweis mit gesetzt werden
  - Man muss sicherstellen, dass beide Zuweisungen atomar geschehen
  - In Java: „synchronized“ Block oder „synchronized“ Methode (→ nächste Folie)

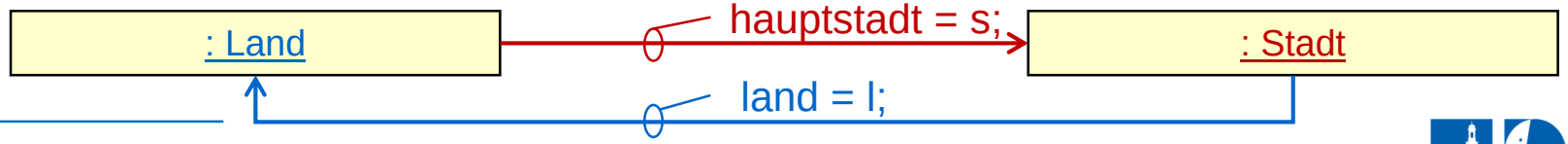
▶ Bidirektionale Assoziation (1:1)

Objektentwurfsmodell mit Assoziation

Objektentwurfsmodell ohne Assoziation



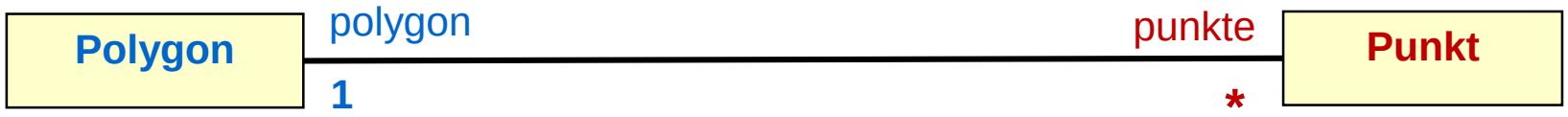
Beispiel-Instanziierung



© 2000-2015 Dr. G. Kniesel

▶ Bidirektionale Assoziation (1:N)

Objektentwurfsmodell mit Assoziation



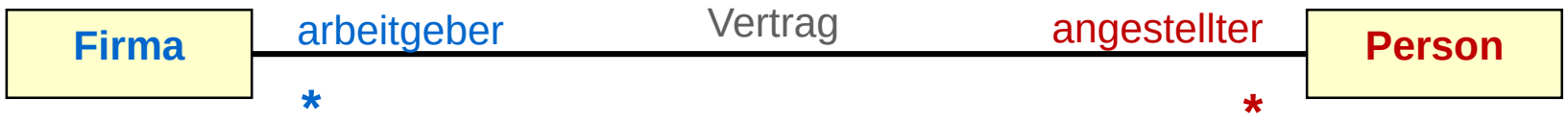
Objektentwurfsmodell ohne Assoziation

Beispiel-Instanziierung

Synchronisation der Zuweisung von Werten für ‚punkte‘ und ‚polygon‘ wie bei bidirektionaler 1:1 Assoziation → analog zu 1:1-Fall

▶ Bidirektionale Assoziation (N:M Naiv)

Objektentwurfsmodell vor der Transformation



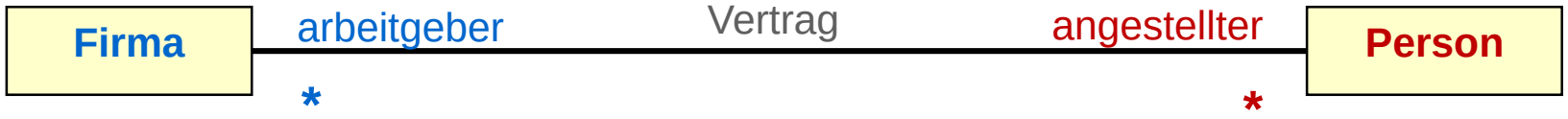
Objektentwurfsmodell ohne Assoziation

Beispiel-Instanziierung nach der Transformation

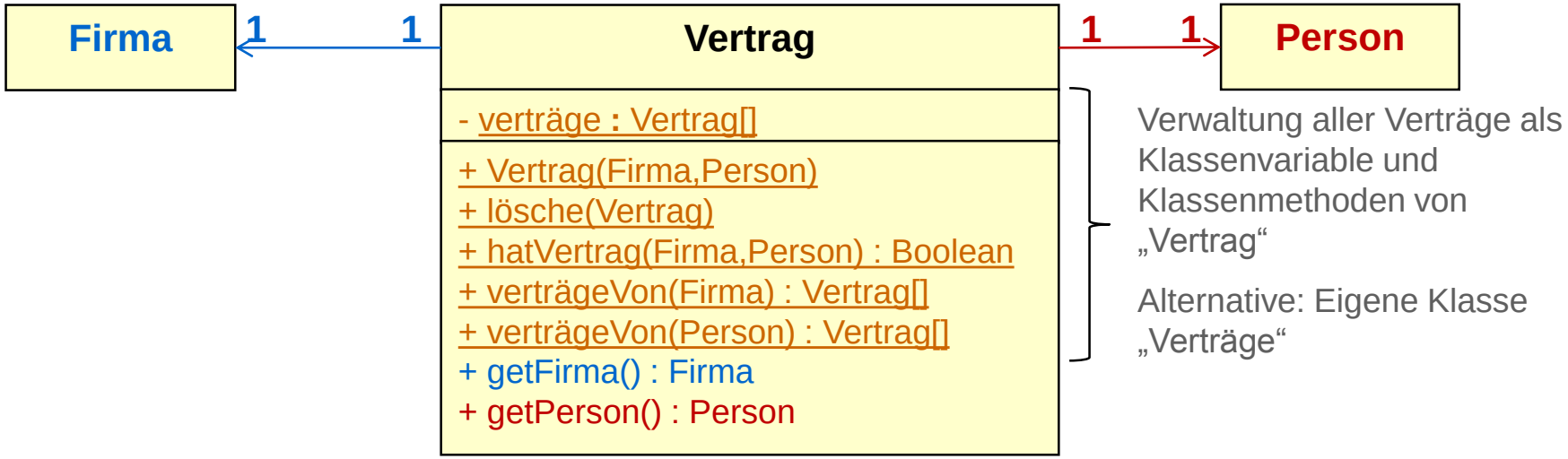
- Problem 1: Deadlockfreies setzen von Rückreferenzen nicht mehr trivial.
- Problem 2 (aller bisherigen Varianten): Die Beziehung wird fest in den Klassen verdrahtet. Das schafft zusätzliche Abhängigkeiten.

▶ Bidirektionale Assoziation (N:M als eigene Klasse)

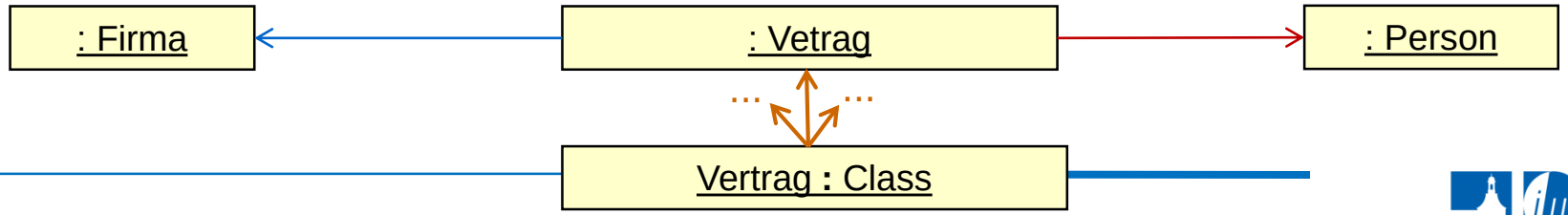
Objektentwurfsmodell mit Assoziation



Objektentwurfsmodell ohne Assoziation

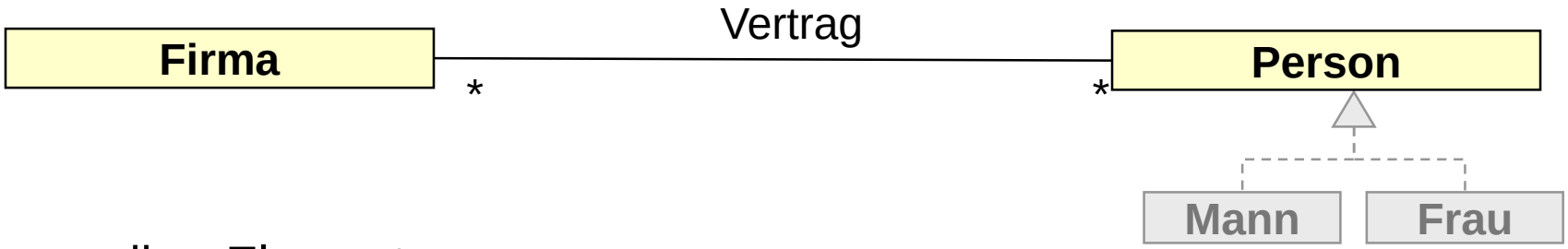


Beispiel-Instanziierung

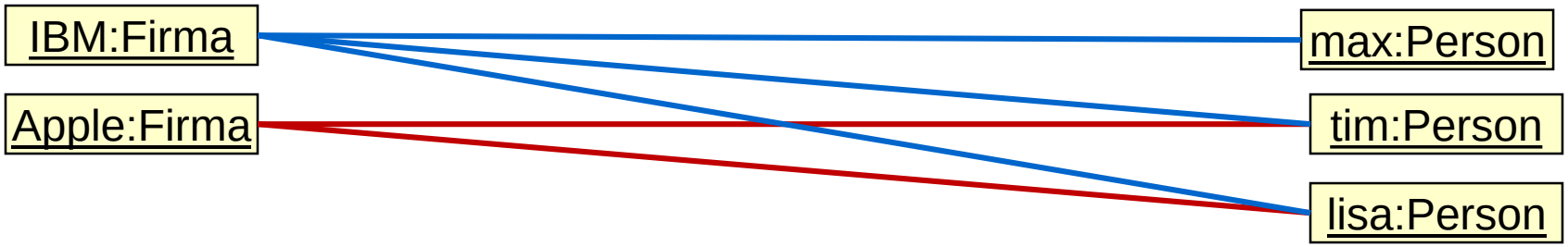


Assoziationen sind implizite Klassen!

- Eine Assoziation



- ... ihre Elemente



- ... können aufgefasst werden als Instanzen einer Klasse „Vertrag"

a11:Vertrag	a12:Vertrag	a13:Vertrag	a22:Vertrag	a23:Vertrag
Firma = IBM	Firma = IBM	Firma = IBM	Firma = Apple	Firma = Apple
Person = max	Person = tim	Person = lisa	Person = tim	Person = lisa

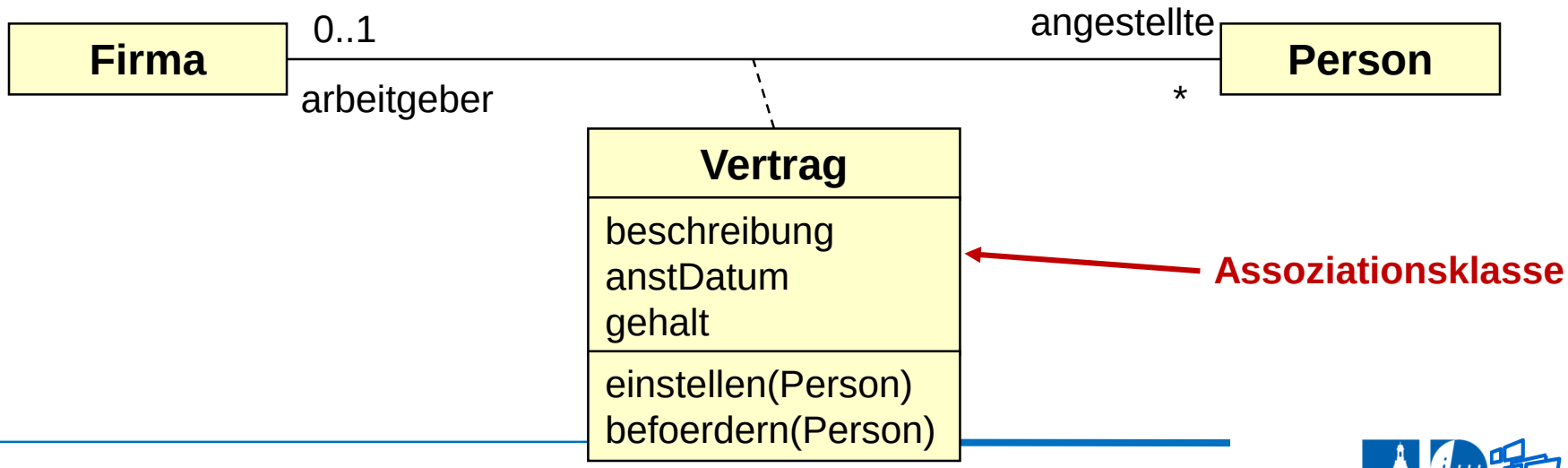
3.3.2.5 Assoziationsklassen

▶ Assoziationsklassen (1)

- Modellieren Assoziationen mit eigener Struktur und eigenem Verhalten
  - Machen die implizite Klasse explizit wenn zusätzliche Attribute und Operationen der Beziehung modelliert werden müssen.

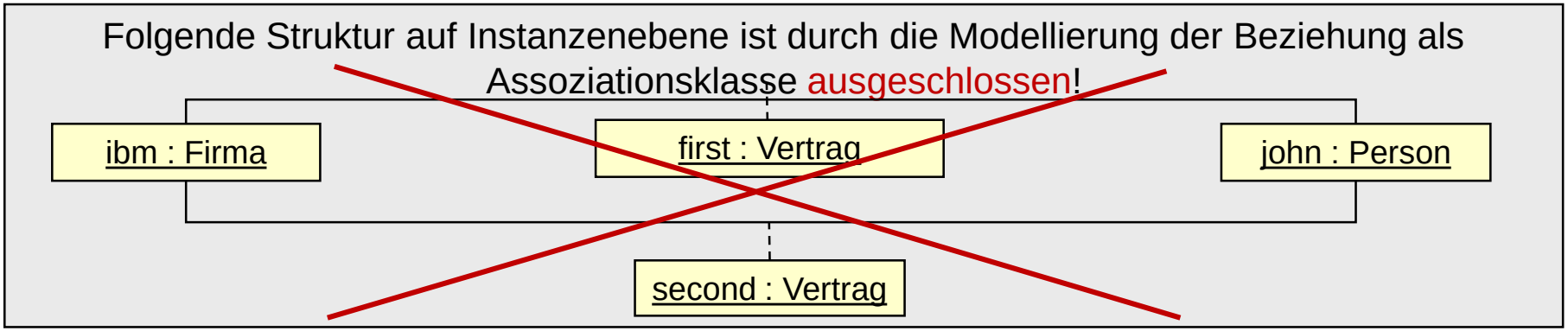
- Beispiel
  - Das Datum der Einstellung (Attribut „anstDatum")
  - Der Vorgang der Einstellung (Operation „einstellen()")

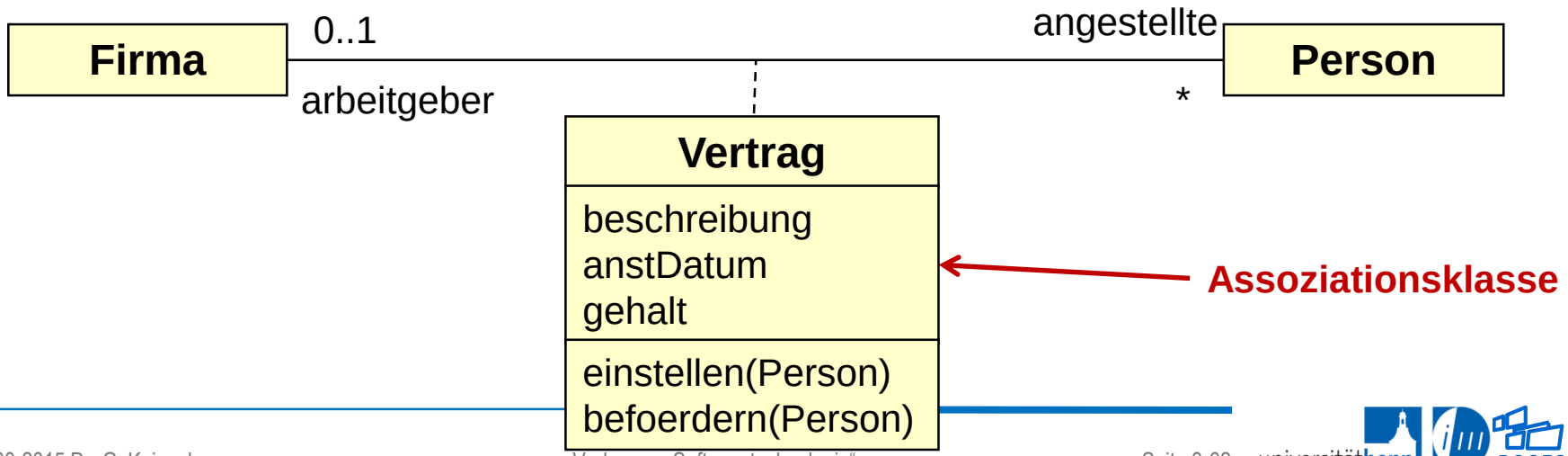
  ... sind Eigenschaften der Employment-Beziehung



▶ Assoziationsklassen-Constraint

- Es darf nur eine einzige Ausprägung der Beziehung zwischen zwei Objekten geben!
  - Z.B.: Eine Person dürfte nicht mehrere Anstellungen bei der gleichen Firma haben

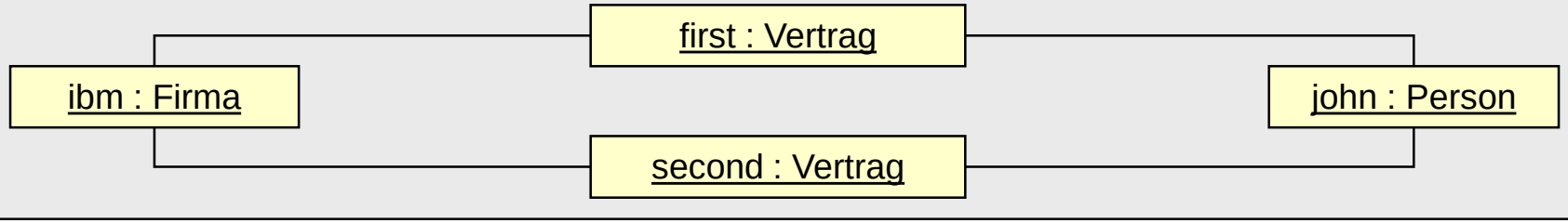


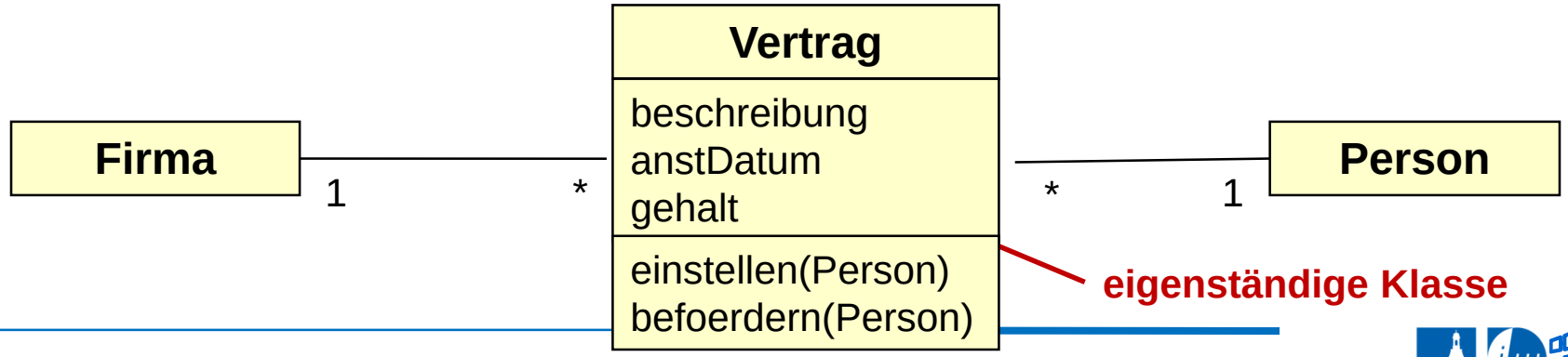


Komplexe Beziehung als eigenständige Klasse

- Wenn die gleiche Beziehung zwischen den selben Objekten mehrfach auftreten kann → eigenständige Klasse statt Assoziationsklasse
  - Beispiel: Jede Person kann beliebig oft in der gleichen Firma arbeiten
  - Es kann daher beliebig viele Verträge zwischen einer Person und einer Firma geben

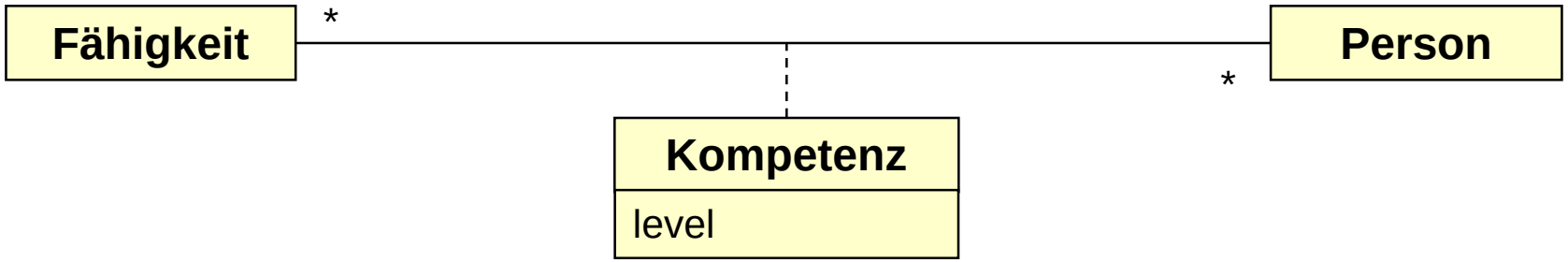
Folgende Struktur auf Instanzenebene ist durch die Modellierung der Beziehung als eigenständige Klasse möglich!



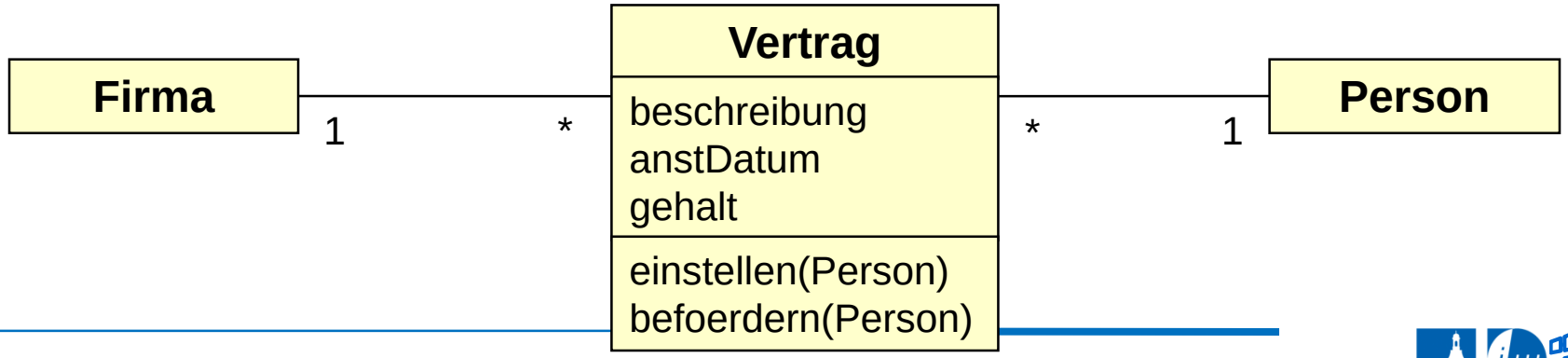


▶ Assoziationsklassen versus Klassen

- Fallbeispiel 1:
  - Jede Person hat pro Fähigkeit nur eine Kompetenz
  - ➔ “Competency” als Assoziationsklasse



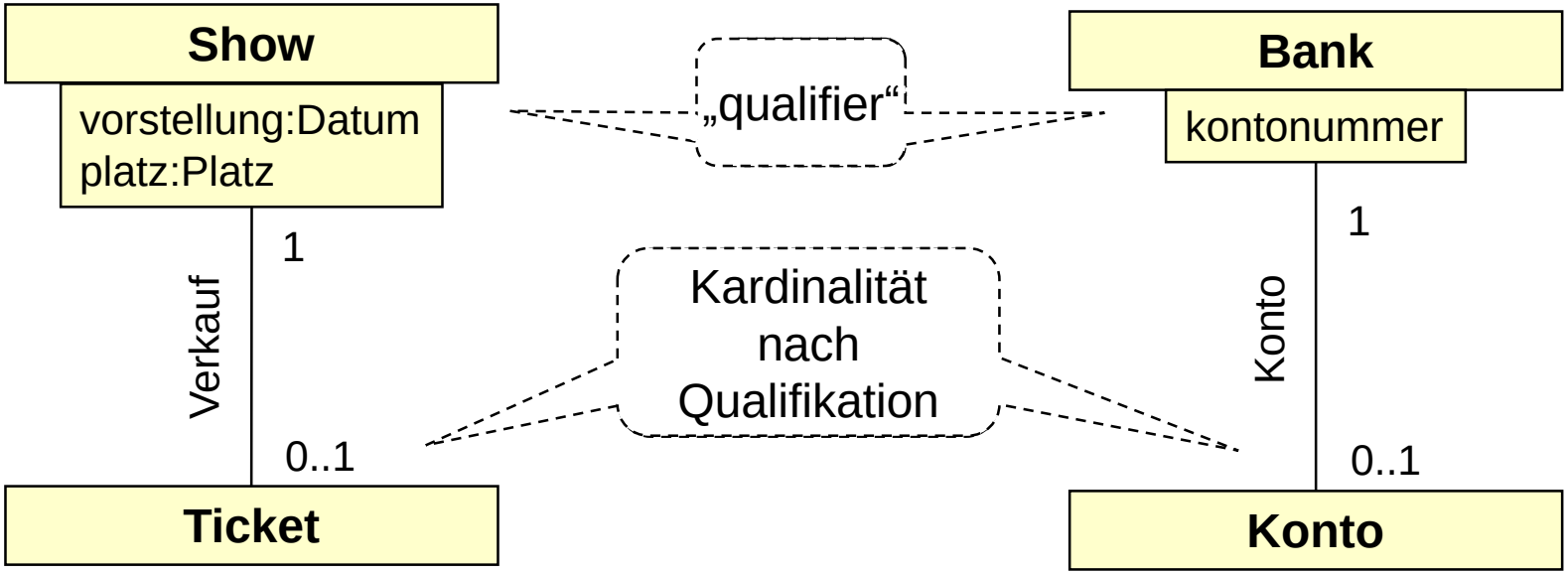
- Fallbeispiel 2:
  - Eine Person hat in verschiedenen Arbeitsperioden unterschiedliche Jobs.
  - ➔ “Employment” als eigenständige Klasse



3.3.2.6 Qualifizierte Assoziationen

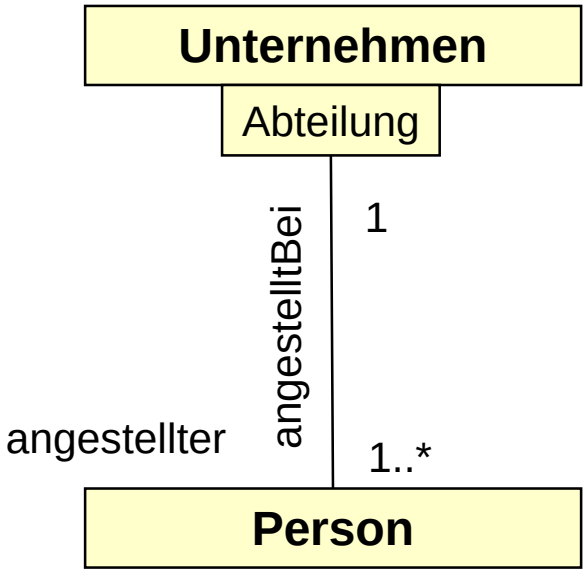
▶ Qualifizierte Assoziation

- Qualifier
  - modelliert Indizierung aus Sicht der einen Klasse
  - gehört zur Assoziationen, nicht zu den Partner-Klassen
  - Effekt: Kardinalität „am anderen Ende“ der Beziehung wird reduziert
    - ⇨ Oft nur noch 0 oder 1: Entweder gibt es kein passendes Objekt oder es ist durch den Qualifier eindeutig bestimmt



▶ Qualifizierte Assoziation

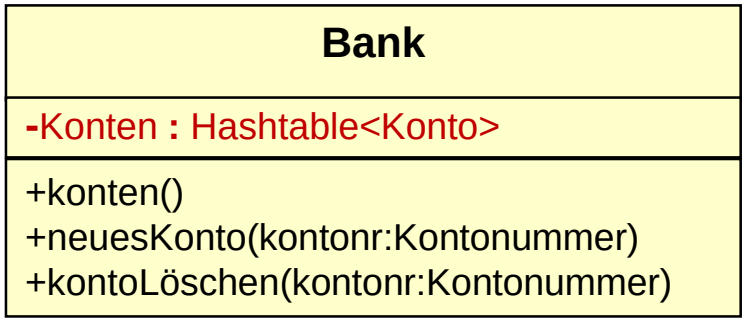
- Qualifier können auch partielle Indizes sein
  - Sie bestimmen evtl. nicht ein einziges Element, sondern eine (Teil)Menge
  - Beispiel: In einer Abteilung können mehrere Angestellte beschäftigt sein

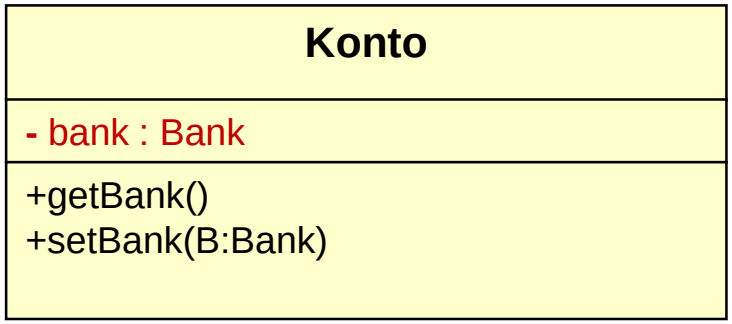


▶ Implementierung

Objektentwurfsmodell vor der Transformation

Objektentwurfsmodell nach der Transformation





- Hastabelle implementiert Indizierung
- Der Qualifier „kontonr“ wird zum Schlüssel (“key”) in der Hashtabelle.

3.3.2.7 Aggregation und Komposition

„Ist Teil von“-Beziehung

Kaskadierende Operationen

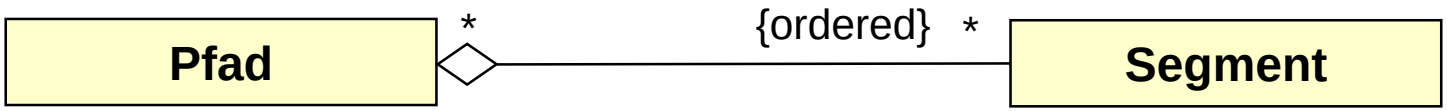
Exklusivität und Existenzabhängigkeit

Die vier Kategorien

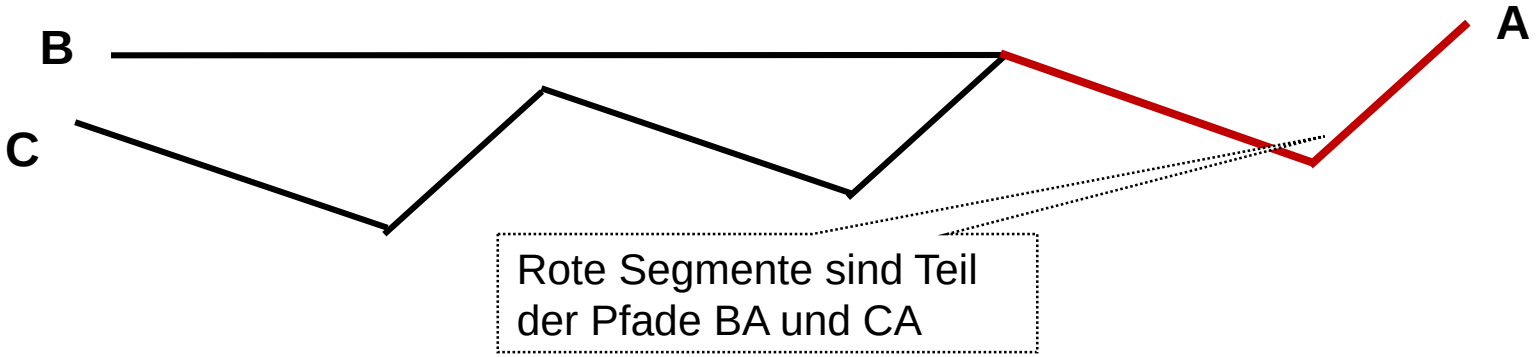
Constraints und Abhängigkeiten

Klassendiagramm ▶ Aggregation

- Aggregation = „ist Teil von“-Beziehung
  - Teil darf in <u>mehreren</u> „Ganzen“ enthalten sein
  - Seine Lebensdauer ist nicht von der des Ganzen abhängig

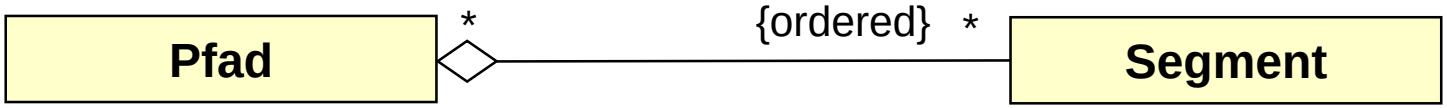


- Dazu passende Veranschaulichung der realen Welt

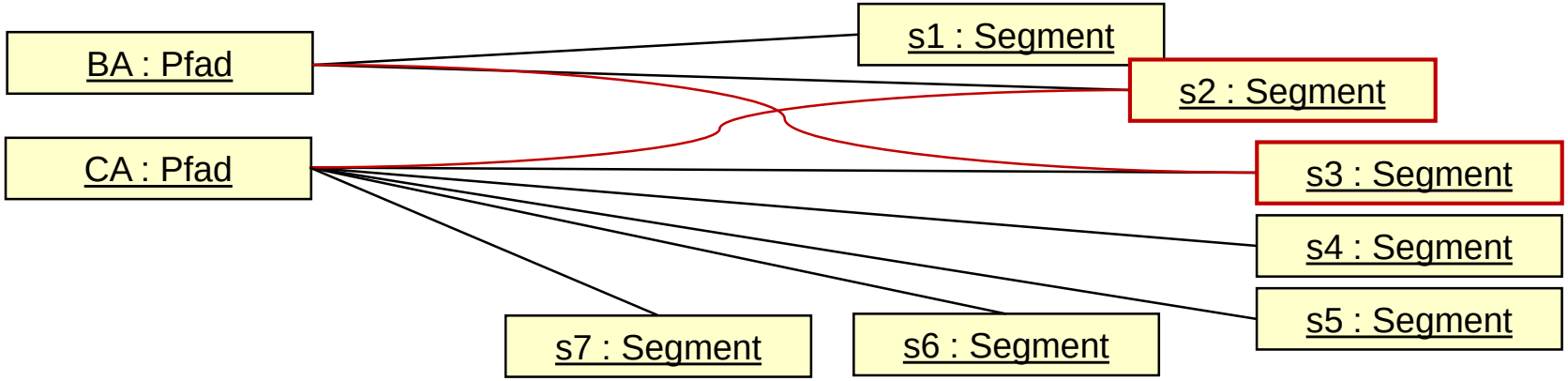


Klassendiagramm ▶ Aggregation

- Aggregation = „ist Teil von“-Beziehung
  - Teil darf in <u>mehreren</u> „Ganzen“ enthalten sein
  - Seine Lebensdauer ist nicht von der des Ganzen abhängig

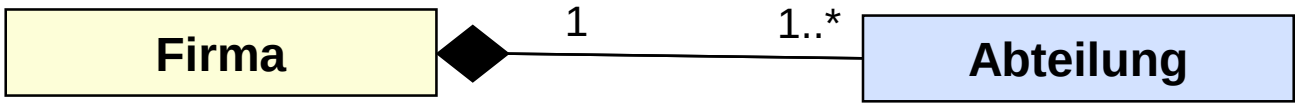


- Dazu passendes Objektdiagramm

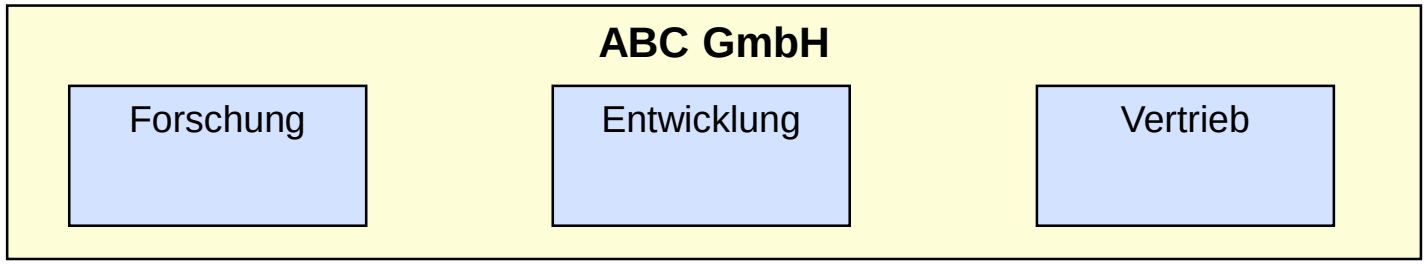


Klassendiagramm ▶ Komposition → Firma-Abteilung

- Komposition = „ist ausschließliches Teil von“-Beziehung
  - Teil kann nur in <u>einem</u> Ganzen enthalten sein
  - Lebensdauer des Teils endet mit Lebensdauer des Ganzen
  - Beispiel: Abteilung gehört zu genau einer Firma und kann ohne Firma nicht existieren

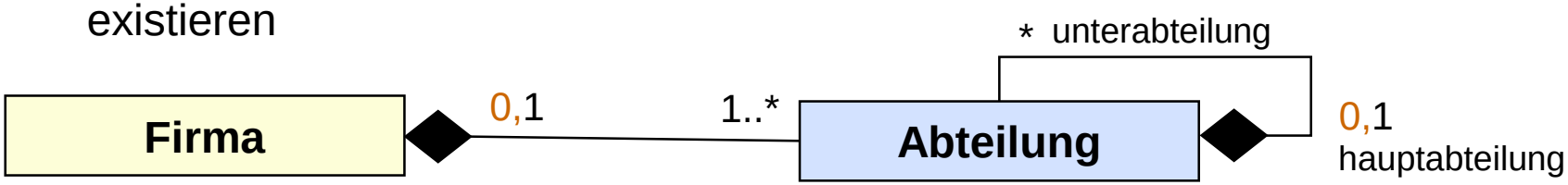


- Dazu passende Veranschaulichung der realen Welt



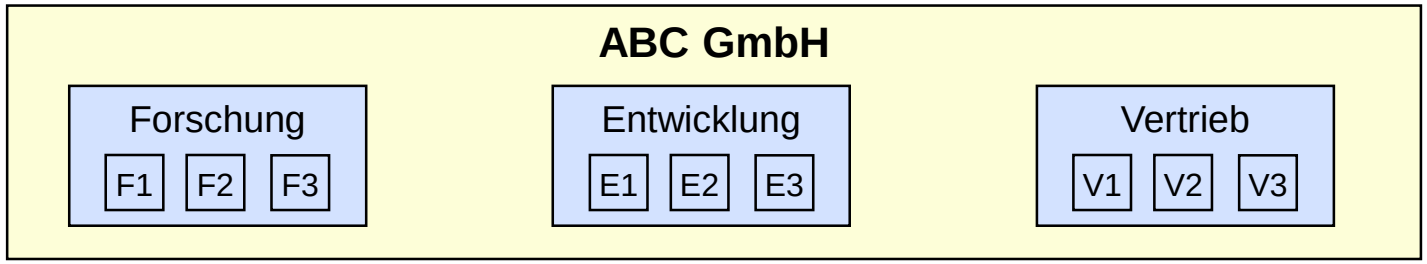
Klassendiagramm ▶ Komposition → Firma-Abteilung-Unterabteilung

- Komposition = „ist ausschließliches Teil von“-Beziehung
  - Teil kann nur in <u>einem</u> Ganzen enthalten sein
  - Lebensdauer des Teils endet mit Lebensdauer des Ganzen
  - Beispiel: Abteilung gehört zu genau einer Firma und kann ohne Firma nicht existieren



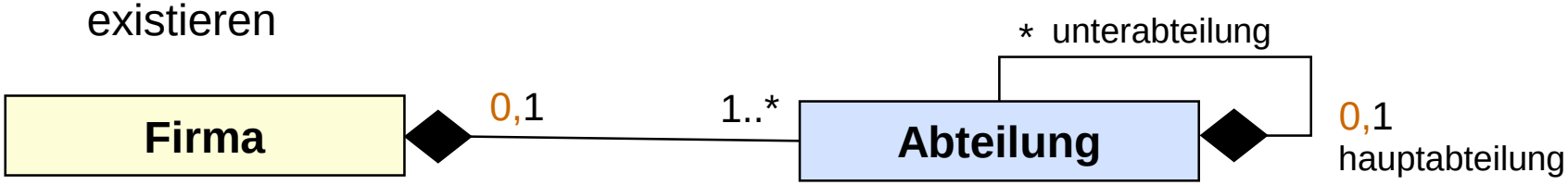
  - Unterabteilung ist nicht gleichzeitig direkt in Firma enthalten
    - ⇨ Kardinalitäten
    - ⇨ explizite Constraints { **context** Abteilung : (Firma <> null) implies (hauptabteilung = null) }
      { **context** Abteilung : (hauptabteilung <> null) implies (Firma = null) }

Komplette OCL-Syntax: http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FOverview.html



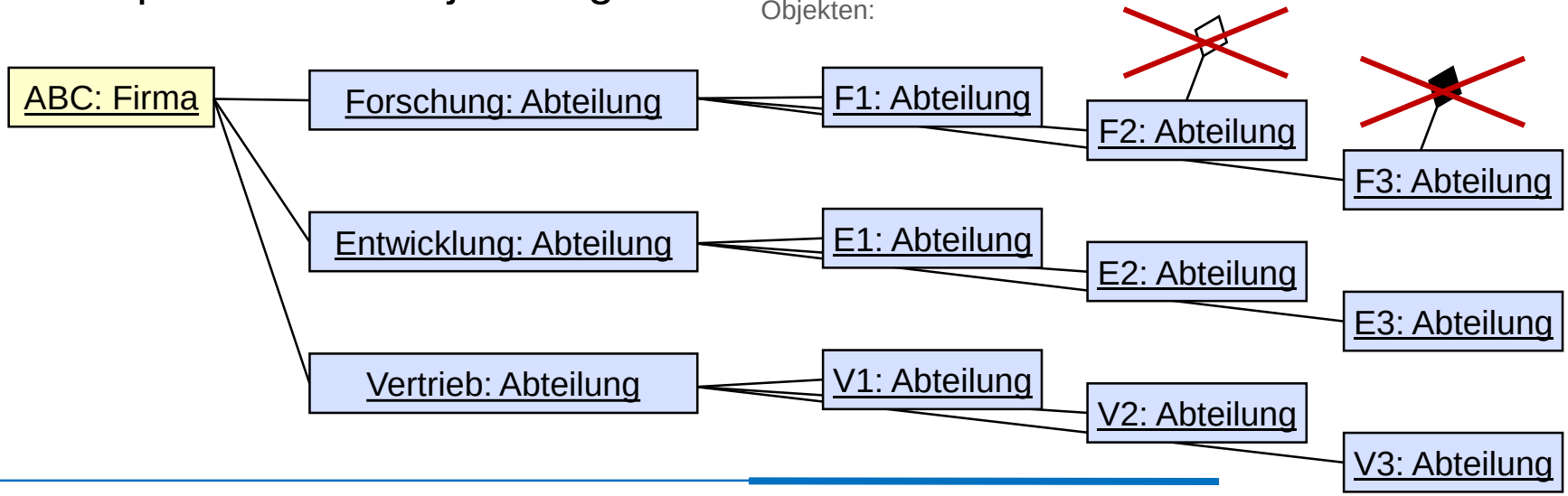
Klassendiagramm ▶ Komposition

- Komposition = „ist ausschließliches Teil von“-Beziehung
  - Teil kann nur in <u>einem</u> Ganzen enthalten sein
  - Lebensdauer des Teils endet mit Lebensdauer des Ganzen
  - Beispiel: Abteilung gehört zu genau einer Firma und kann ohne Firma nicht existieren



- Dazu passendes Objektdiagramm

Keine Aggregations- oder Kompositionsbeziehungen zu anderen Objekten:



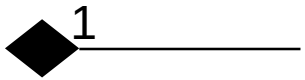
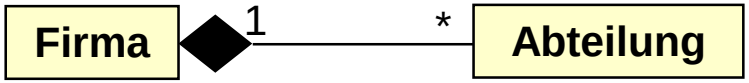
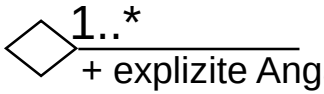
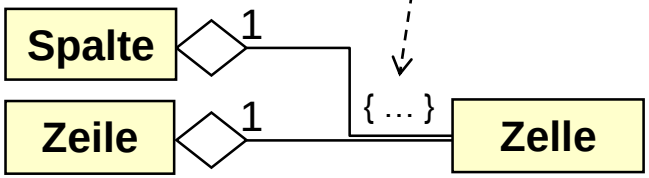
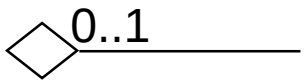
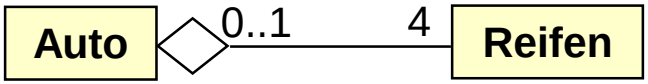
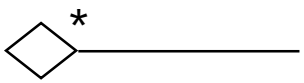
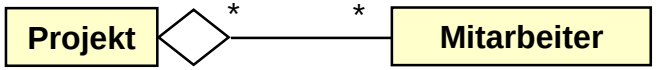
Klassendiagramm
▶ Assoziation, Aggregation, Komposition

- Spezialisierungen
  - Aggregation ist spezielle Assoziation
  - Komposition ist spezielle Aggregation
- Modellierung
  - Alles was für Assoziationen gilt, gilt für die beiden anderen auch
  - Im Zweifelsfall die allgemeinere Notation wählen und Intention durch zusätzliche Bedingungen explizit machen (Kardinalitäten, Constraints, ...)
- Aggregation versus Assoziation
  - Assoziation mit Kardinalitätsangaben und Constraints
- Komposition versus Aggregation
  - Aggregation mit Kardinalitätsangaben und Constraints

Aggregation = „Ist Teil von“-Beziehung

- Modelliert Einheit eines "Ganzen" mit seinen "Teilen"
- Impliziert oft kaskadierende Operationen
  - Operation auf Ganzem wird auch auf allen Teilen durchgeführt
  - Beispiel 1: Verschiebung eines Rechtecks verschiebt alle seine Kanten
  - Beispiel 2: Laden des Ganzen von Platte lädt alle seine Teile
- Impliziert oft Existenz-Abhängigkeit
  - Teil existiert nicht ohne Ganzem
  - Entspricht dem Kaskadierungsverhalten beim Löschen: Teile werden mit gelöscht oder an anderes Ganzes weitergegeben
- Impliziert oft Exklusivität
  - Teil ist in genau einem Ganzen enthalten
- Mögliche semantische Kategorien
  - (existenz-)abhängig, exklusiv
  - (existenz-)abhängig, nicht exklusiv
  - (existenz-)unabhängig, exklusiv
  - (existenz-)unabhängig, nicht exklusiv

▶ Die vier Kategorien

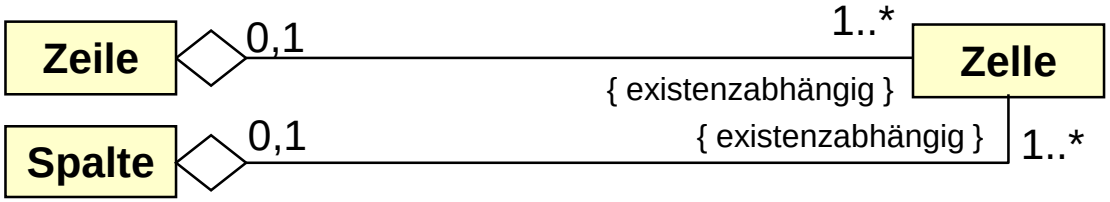
	exklusiv (nur in einem Ganzen)	nicht exklusiv (in mehreren Ganzen)
existenz-abhängig (Propagierung der Löschoperation)	 	 + explizite Angabe der Propagierungs-Semantik als Constraints (Seite 47) 
existenz-unabhängig (keine Propagierung der Löschoperation)*	 	 

* Sonderfall siehe Seite 48

▶ Kardinalität 0..1

- Kardinalität 0 bedeutet hier:
  - Teile können temporär unabhängig existieren
  - Teile können insbesondere unabhängig von Ganzen erzeugt werden
    - ⇨ Sehr praxisrelevanter Fall: Vieles wir „bottom-up“ produziert
- Kardinalität 1 bedeutet hier:
  - Sobald die Teile einem Ganzen zugeordnet werden, sind sie exklusiv darin enthalten

Bedingungen (‚Constraints')

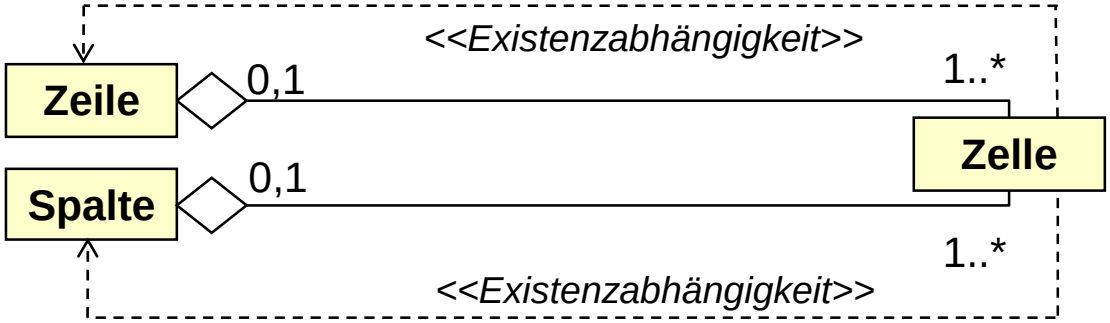
- Bisher wurden nur besonders häufige Bedingungen modelliert, über
  - Spezialnotationen (z.B. „Komposition") oder
  - Kardinalitäten (z.B. 1..4)
- Das reicht oft nicht aus! Es gibt daher auch eine Notation für beliebige Bedingungen
  - Angabe von Bedingung in geschweiften Klammern: { Bedingung }
- Bedingungssprache
  - vordefinierte Eigenschaften:
    - ⇨ abstract, readOnly, query, leaf, ordered, unique, set, bag, …
  - beliebige umgangssprachliche Formulierung
    - ⇨ 

  - Object Constraint Language
    - ⇨ Kapitel „Objektmodellierung"

Bedingungen (‚Constraints'): Angabe als Abhängigkeiten

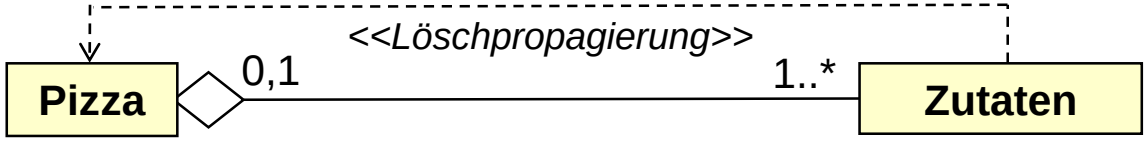
- Manche Constraints geben nur eine Abhängigkeit an
  - Beipiel: {existenzabhängig} -- „Nomen est omen"!
  - Man kann sie daher auch als Abhängigkeitspfeile mit passendem Stereotyp modellieren

- Beispiel 1: „Zellen sind existenzabhängig von Zeilen und Spalten"



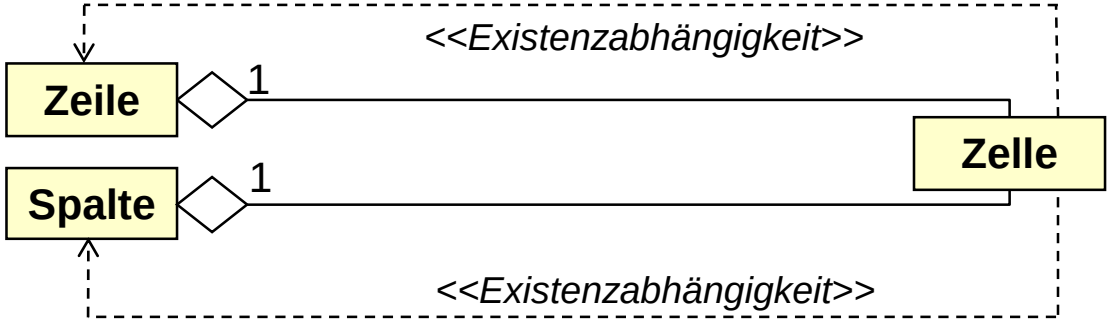
Der Pfeil gibt immer die Richtung der Abhängigkeit an, nicht die Richtung de Löschpropagierung!

- „Beispiel 2": Zutaten sind löschpropagierungsabhängig von Pizzen"

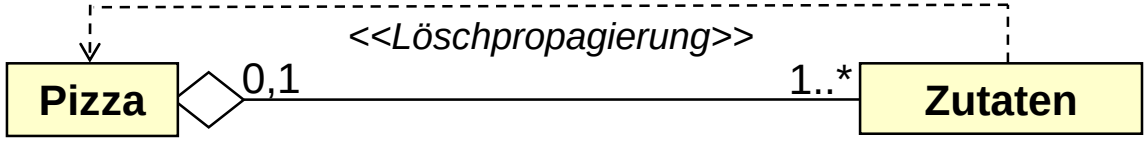


Bedingungen (‚Constraints'): Existenz- und Löschabhängigkeit

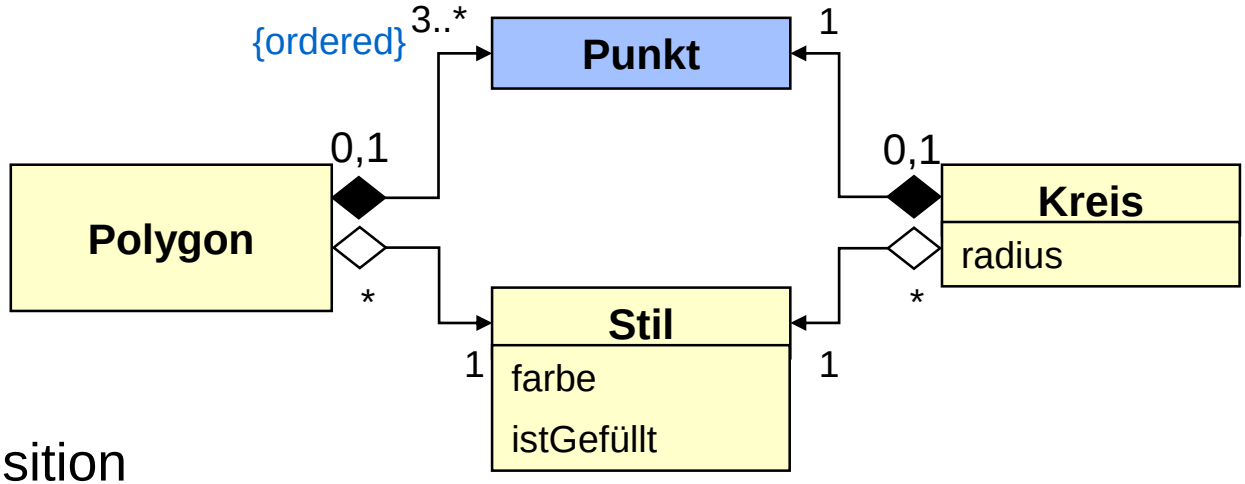
- Existenzabhängigkeit impliziert Löschpropagierung
  - → Zellen können nicht außerhalb von Zeilen und Spalten existieren
  - → Wenn eine Zeile gelöscht wird, werden alle Zellen dieser Zeile gelöscht!



- Löschpropagierung impliziert aber keine Existenzabhängigkeit!
  - → Wenn die Pizza gegessen (also „gelöscht") wird, werden auch die Zutaten gegessen („gelöscht")!
  - → Jedoch können die die Zutaten auch ohne der Pizza existieren!



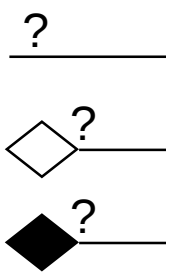
Statisches Modell: Aggregation und Komposition – Gemeinsames Beispiel



- Komposition
  - Für Beziehung Polygon → Punkt und Kreis → Punkt …
  - … weil man nicht möchte, dass der Mittelpunkt des Kreises auch Eckpunkt eines Polygons sein kann.
  - Dann würde nämlich ein Verschieben des Kreises das Polygon verformen!
- Aggregation
  - Für Beziehung Polygon → Stil und Kreis → Stil …
  - … weil man gern möchte, dass anpassen eines gemeinsamen Stils alle Formen ändert die ihn benutzen

Denksportaufgabe

- Wie würden Sie die Beziehung einer Datei zu einem Ordner modellieren?
  - Assoziation, Aggregation oder Komposition?



- Bedenken Sie:
  - Die Datei ist zu jedem Zeitpunkt exklusiver Teil eines Ordners,
  - … kann aber in einen anderen Ordner verschoben werden
  - Wenn der Ordner gelöscht wird, wird die Datei mit gelöscht.

3.3.2.8 Vererbung und Interfaces

Vererbung

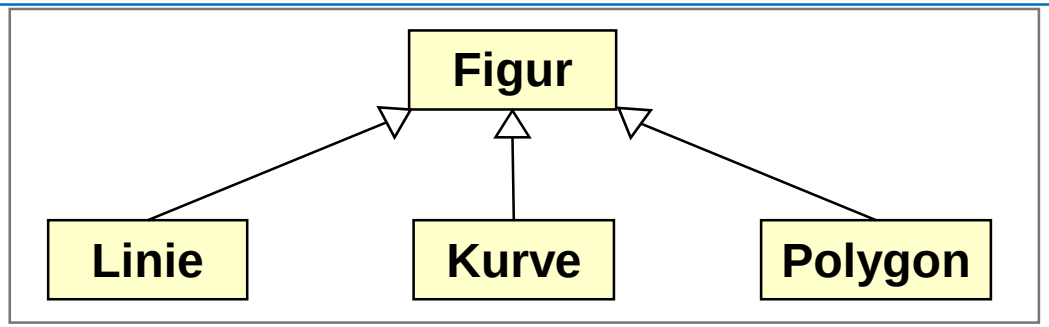
Interfaces

„Bietet“-Beziehung

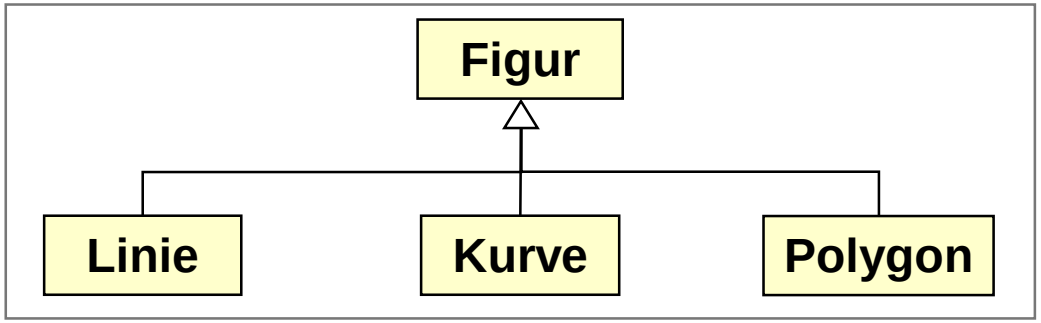
„Braucht“-Beziehung

Klassendiagramm: Vererbung

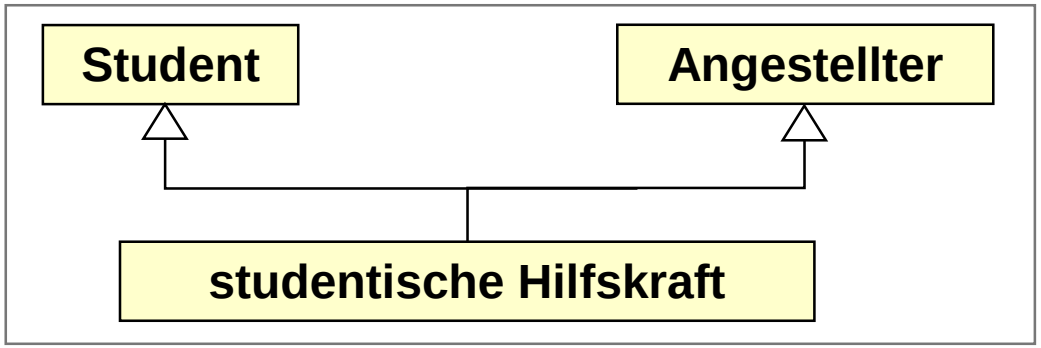
- "direkte" Darstellung



- "zusammengefasste" Darstellung

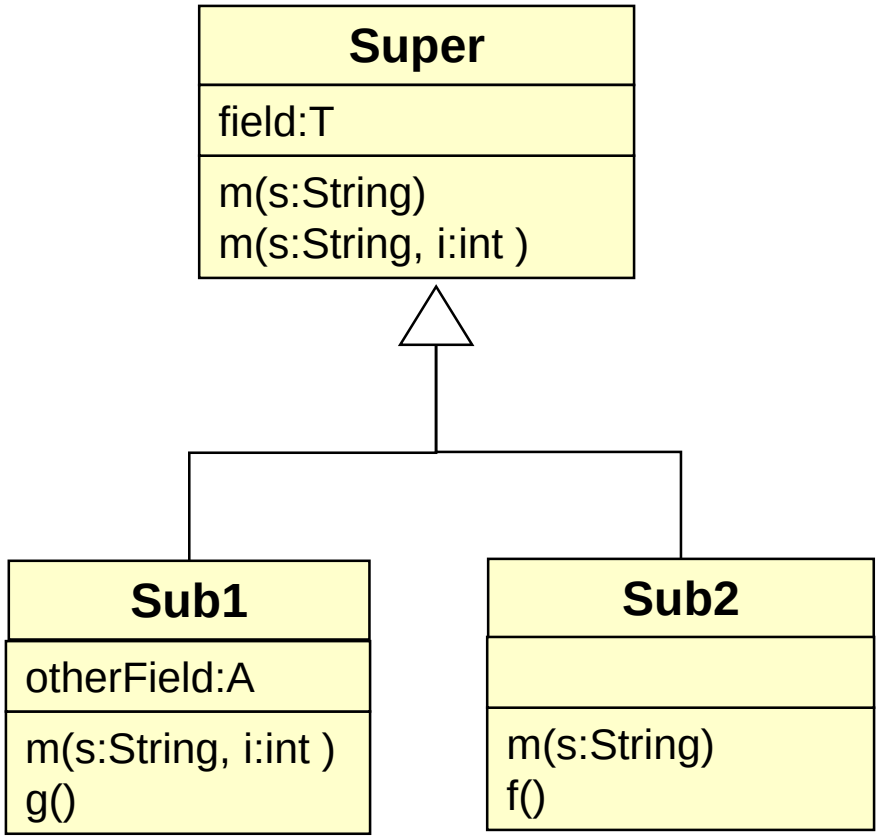


- Multiple Vererbung
  - Eine Klasse mit mehreren Oberklassen
  - Bsp: C++



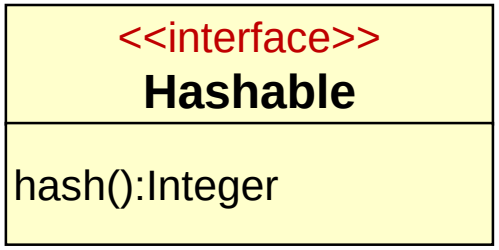
Klassendiagramm: Vererbung

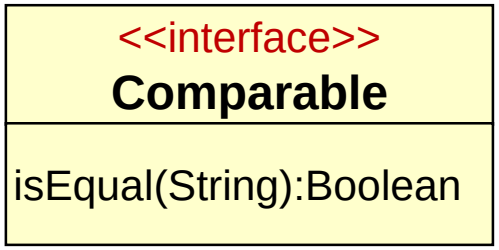
- Vererbung
  - geerbte Methoden / Attribute im Diagramm der Unterklasse nicht wiederholen
- Overriding
  - überschriebene Methoden im Diagramm der Unterklasse wiederholen
- Overloading
  - alle verschiedenen Signaturen angeben
  - auch überladene Methoden können in Untertypen überschreiben werden



Statisches Modell: Interfaces

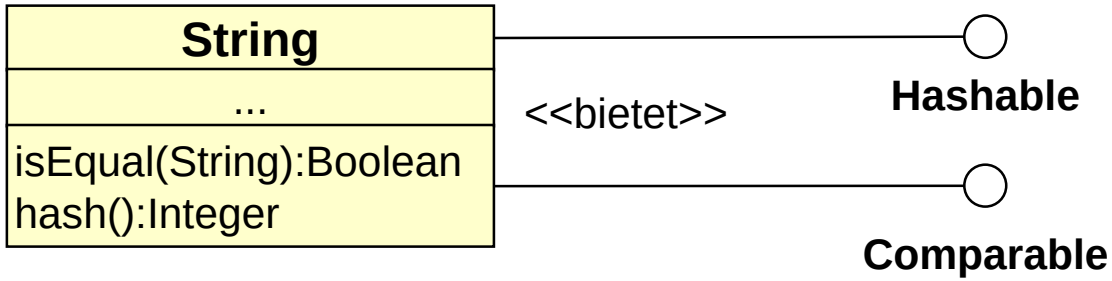
- Semantik
  - Interfaces enthalten nur Methodensignaturen
- Notation
  - Klassensymbol mit stereotyp <<interface>> im Namensfeld
  - leeres Attributfeld kann weggelassen werden

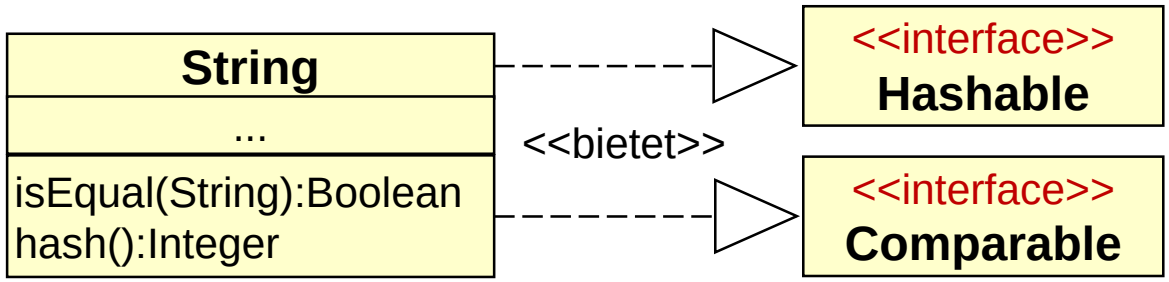




Statisches Modell: Implementierungs-Beziehung

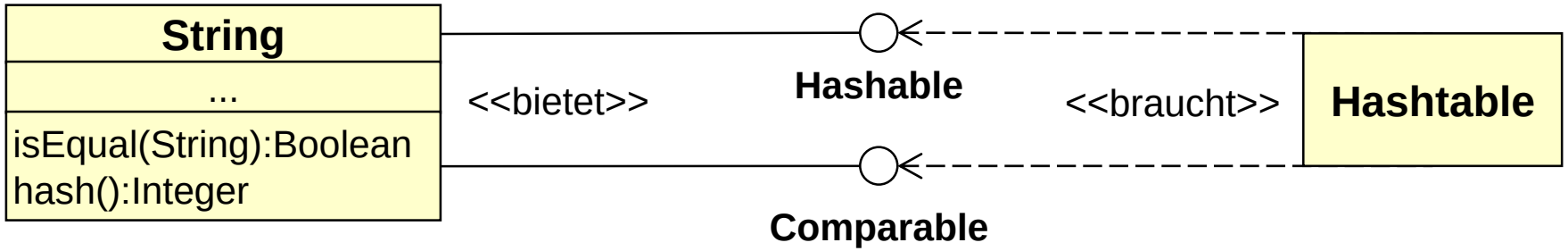
- Semantik
  - „Komponente / Klasse implementiert Interface“
- Notation
  - gestrichelter Vererbungs-Pfeil oder „Lollipop“

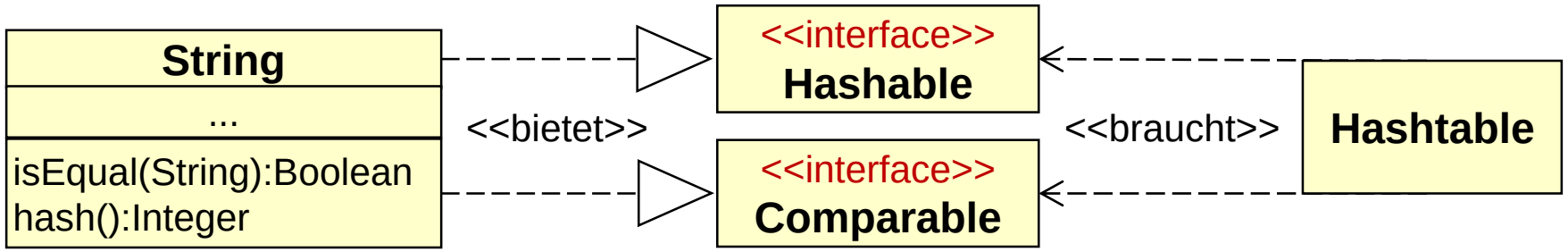




Statisches Modell: „Braucht"-Beziehung

- Semantik
  - „Klasse benötigt Interface"
- Notation
  - gestrichelter spitzer Pfeil („Abhängigkeitspfeil") + Stereotyp





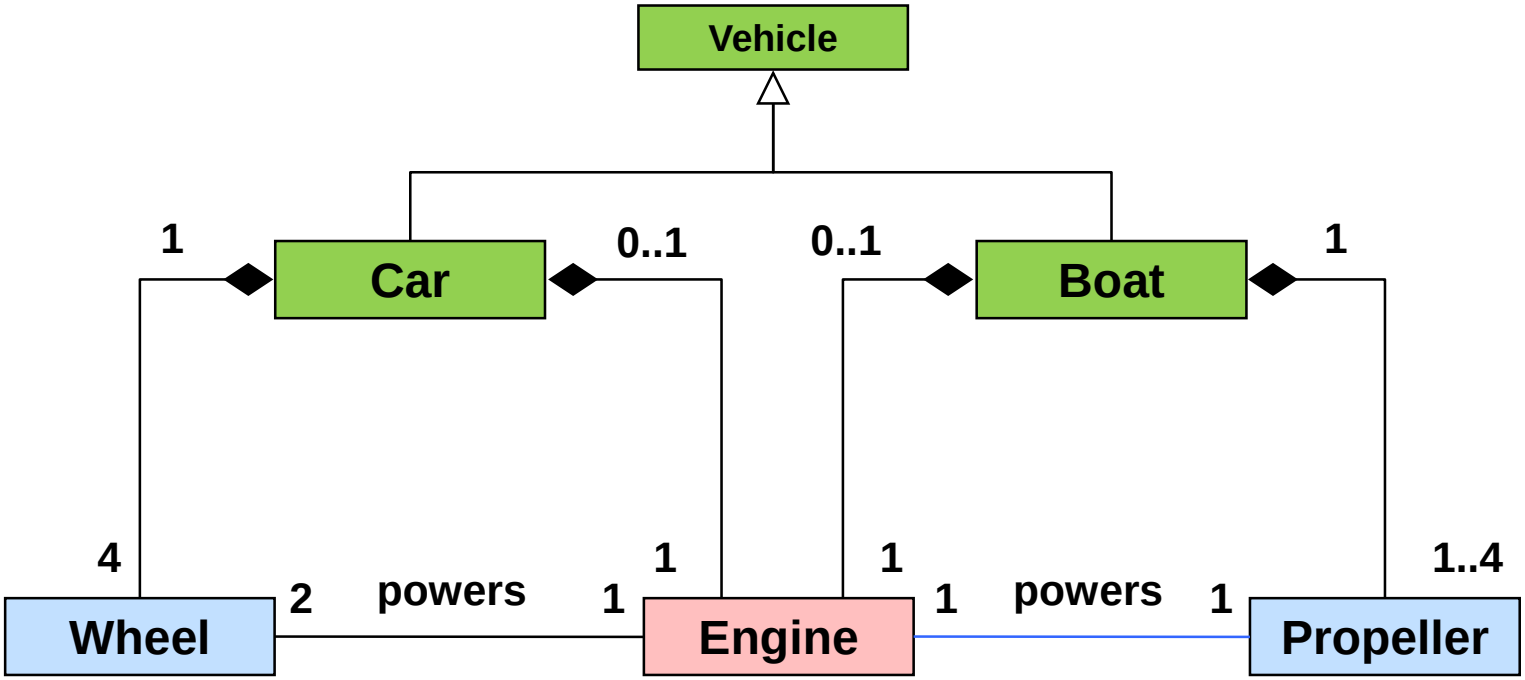
3.3.3 Kompositionsstrukturdiagramme

Motivation

Notation

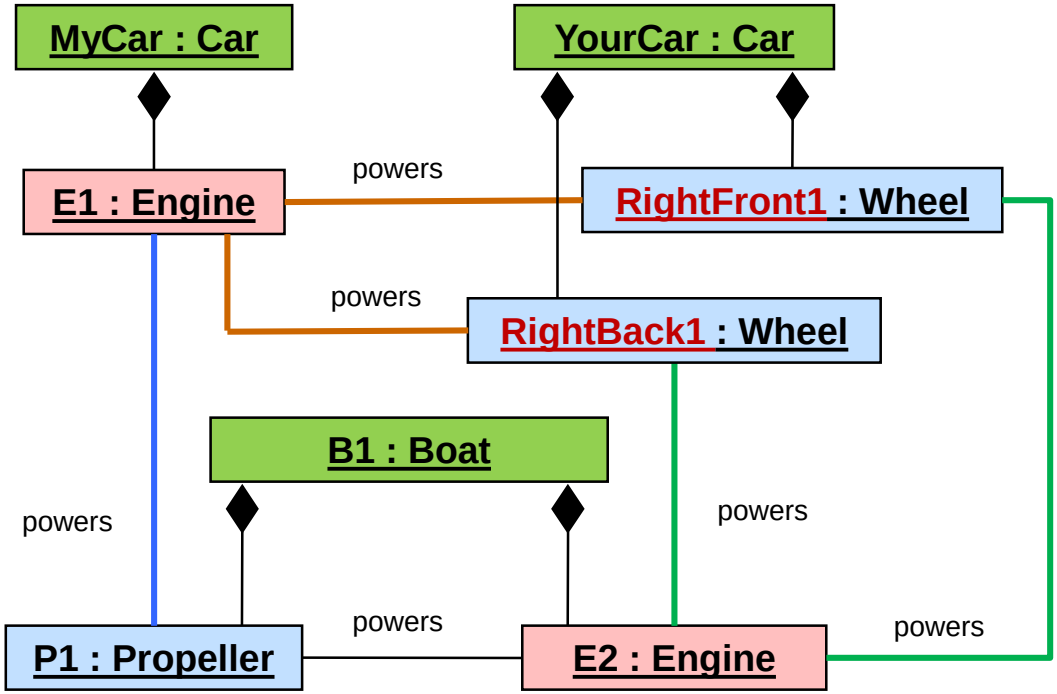
Bedeutung

Rückblick: UML Komposition



Was ist seltsam an diesem Beispiel?

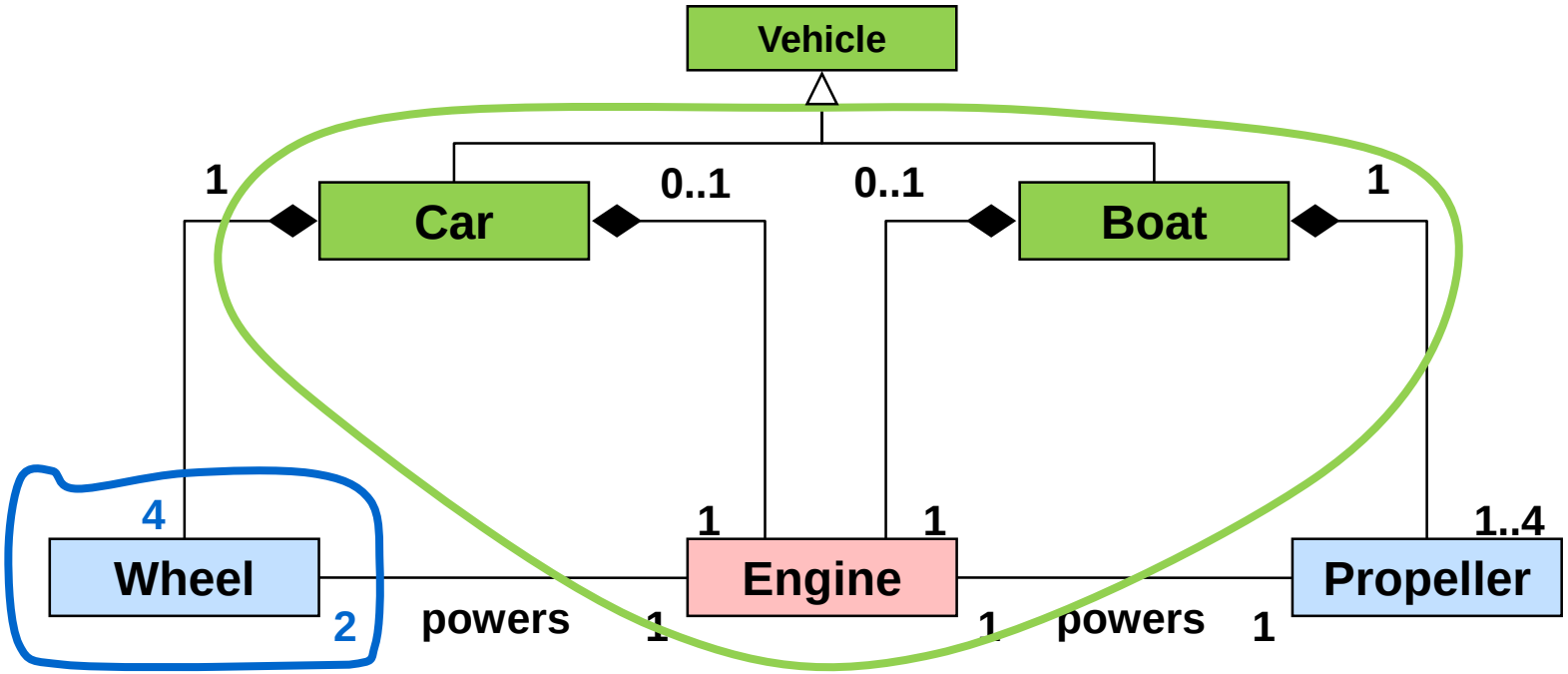
Laut vorherigem Klassendiagramm legales Objektdiagramm



- Sinnlose Beziehungsinstanzen möglich
  - Der Motor meines Autos treibt die Räder eines anderen Autos an
  - Statt zweier Vorderräder werden zwei rechte Räder angetrieben
  - Der Motor meines Autos treibt den Propeller des Bootes an
  - Der Motor des Bootes treibt die Räder eines Autos an

Fazit: Bisherige Notation ist zu allgemein

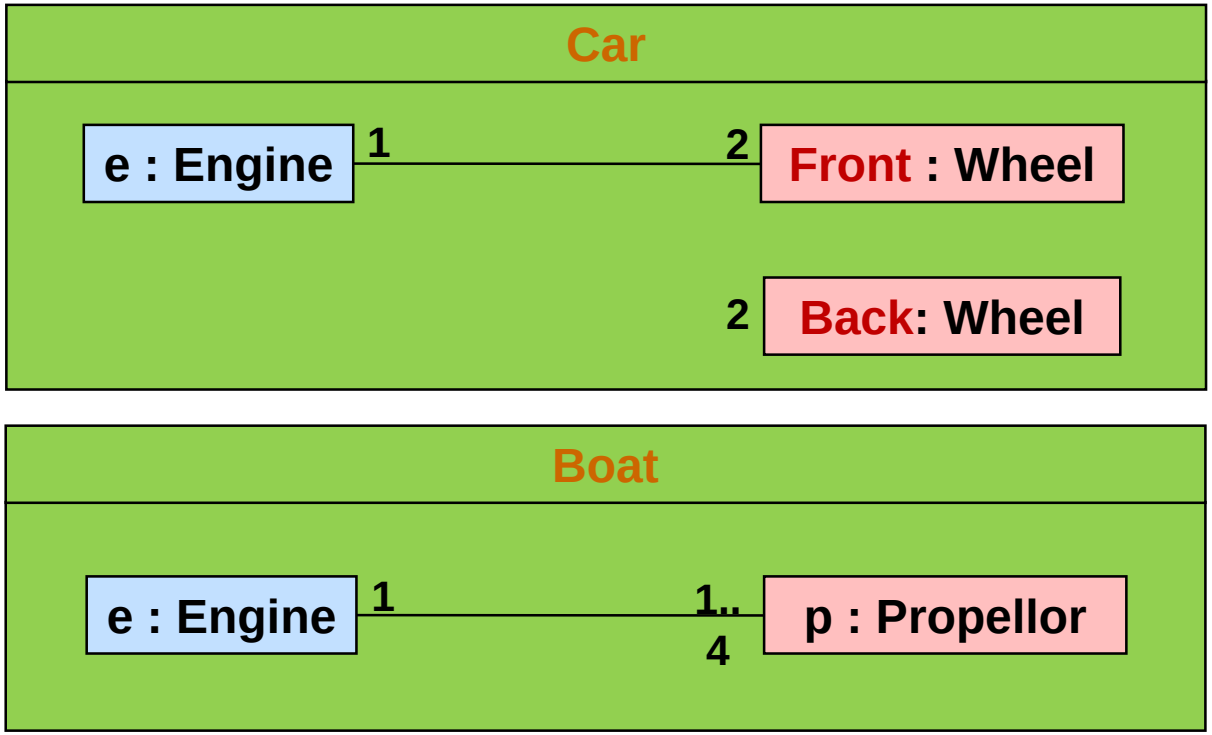
- Problem 1: Wie stellt man kontextabhängige Semantik dar?
  - Motor im Auto versus Motor im Boot



- Problem 2: Wie stellt man rollenabhängige Semantik dar?
  - Antrieb der Vorderräder versus Antrieb der Hinterräder (Rollen von ‚Wheel‘)

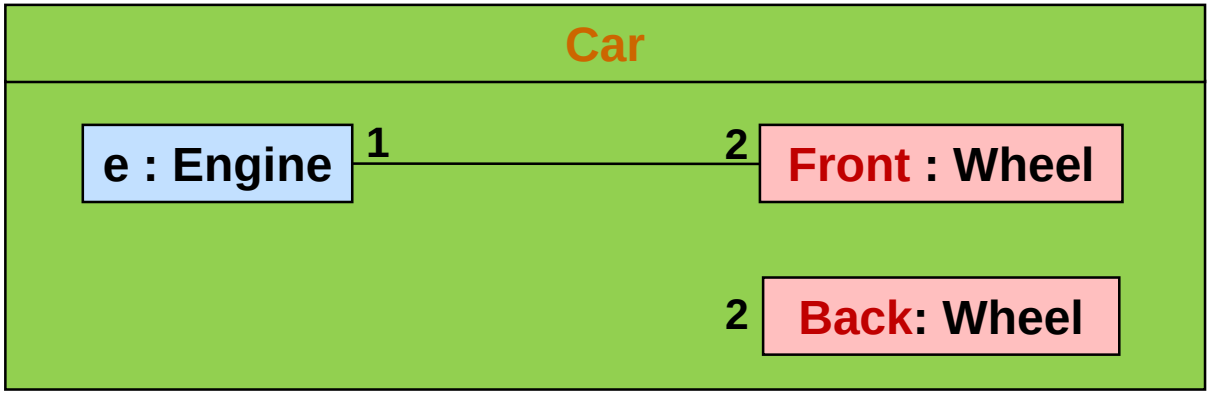
Kompositionsstruktur-Diagramm

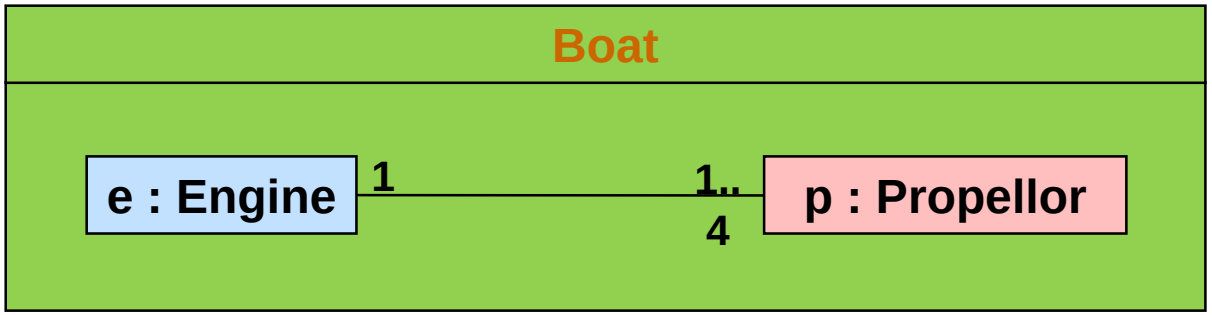
- Die interne Kompositionsstruktur der im vorherigen Diagramm grün hervorgehobenen Klassen wird hier explizit gemacht
  - Kontextabhägigkeiten werden explizit dargestellt
  - Rollenabhänigkeiten werden explizit dargestellt



Kompositionsstruktur-Diagramm: Bedeutung

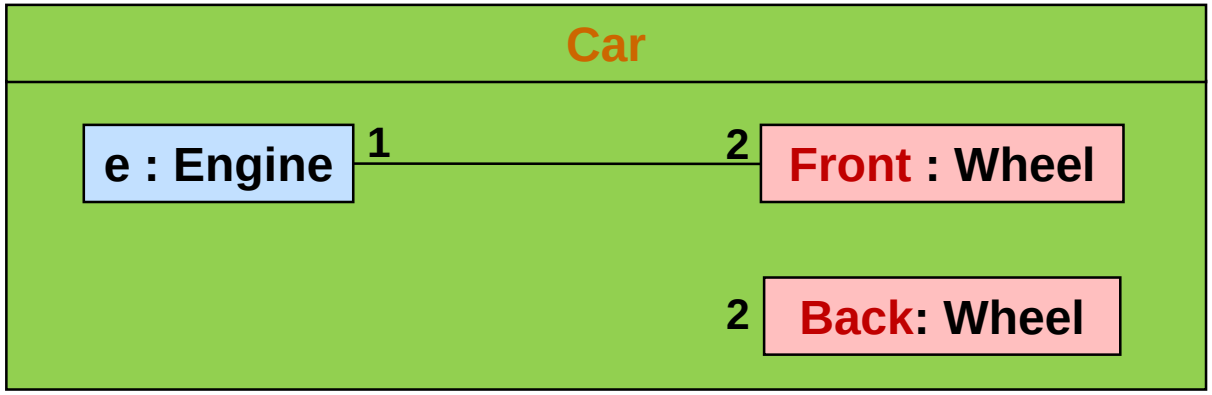
- In jedem Auto gilt:
  - 1. Es existiert ein Motor, zwei Vorder- und zwei Hinterräder.
  - 2. Der Motor treibt die Vorderräder im selben Auto an in dem er existiert.
  - 3. Der Motor treibt nichts anderes an im Auto, in anderen Autos oder Booten.

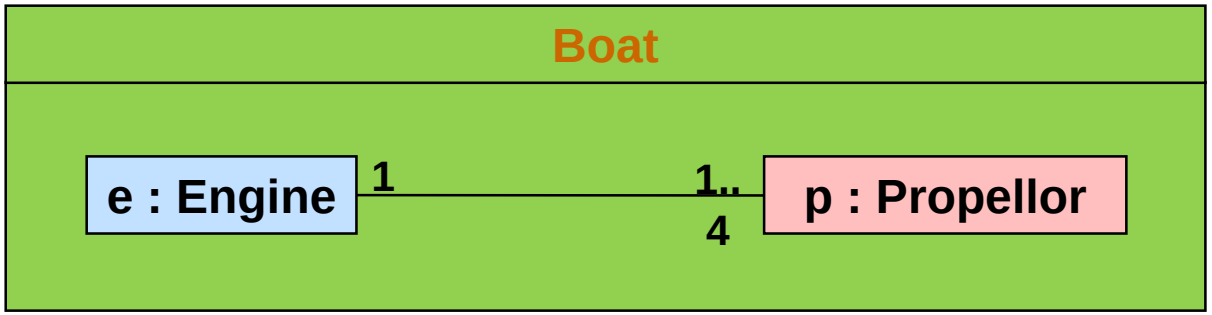




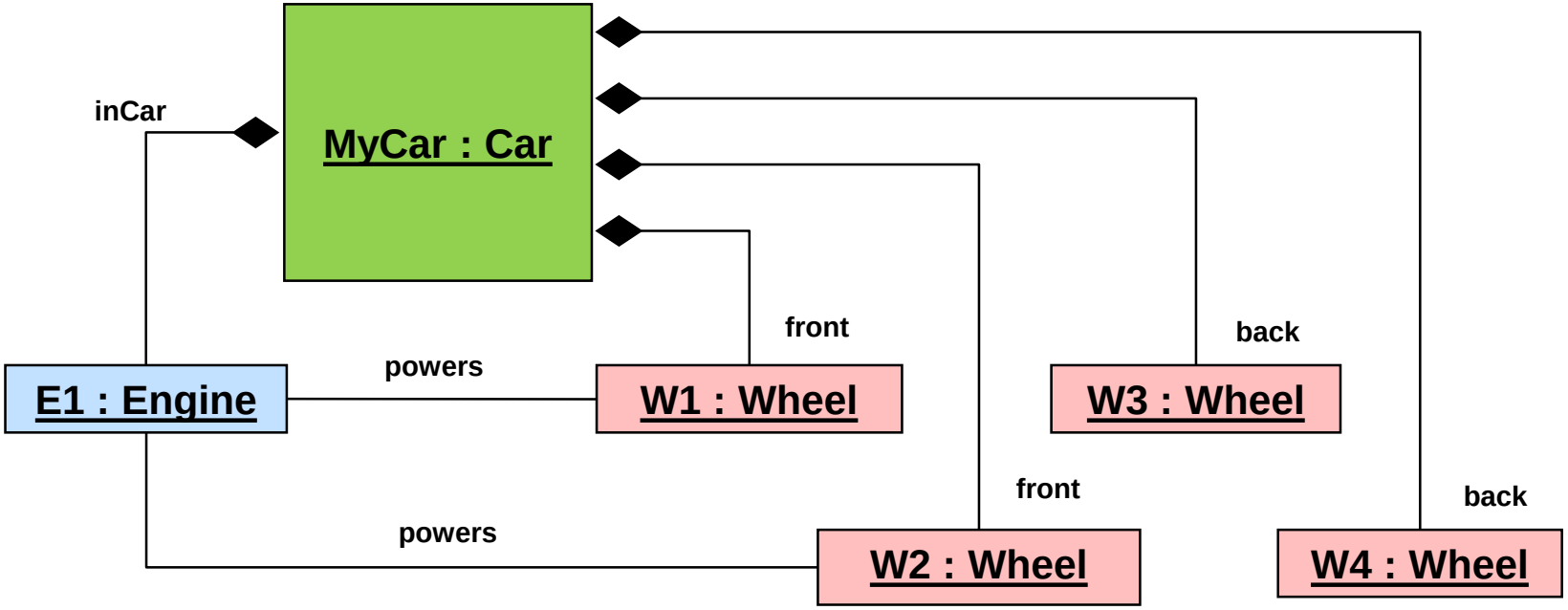
Kompositionsstruktur-Diagramm: Bedeutung

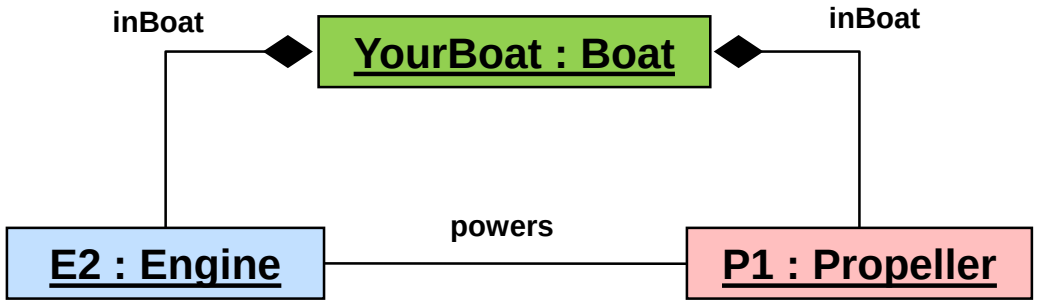
- In jedem Boot gilt:
  - 1. Es existiert ein Motor und ein bis vier Propeller.
  - 2. Der Motor treibt die Propeller im selben Boot an indem er existiert.
  - 3. Der Motor treibt nichts anderes an im Boot, in anderen Booten oder in Autos.



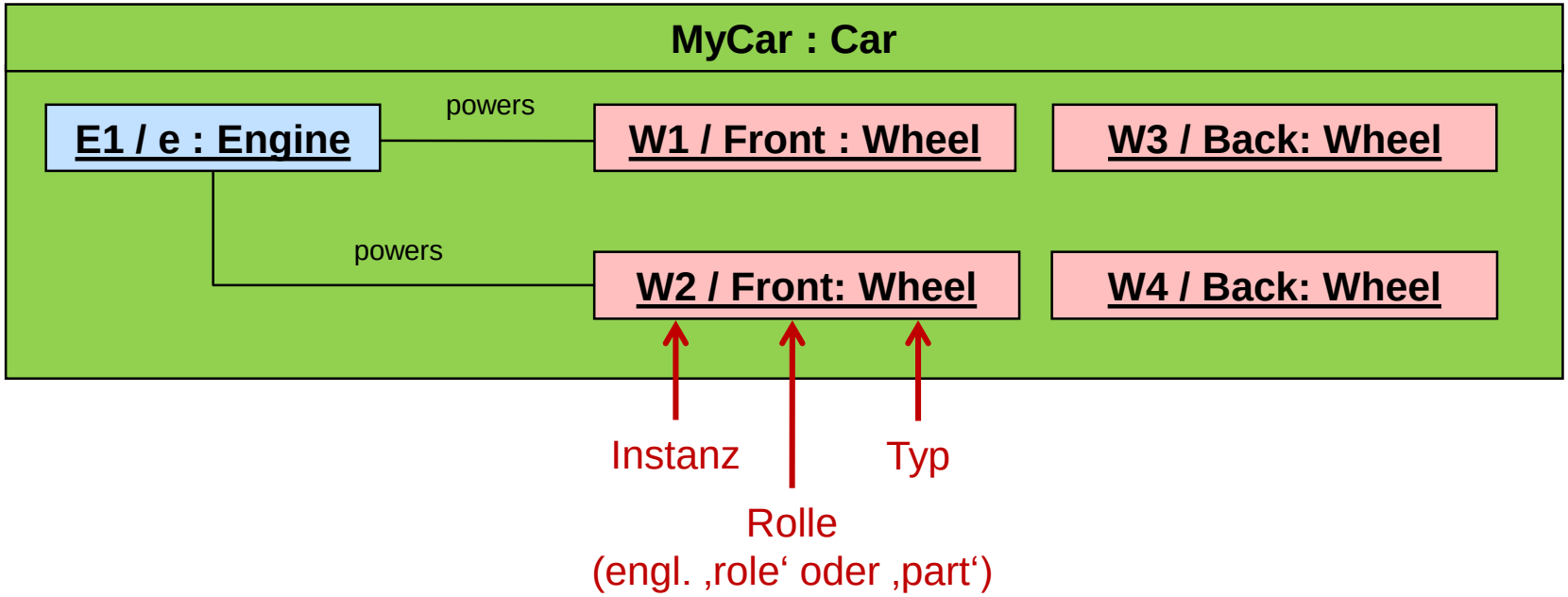


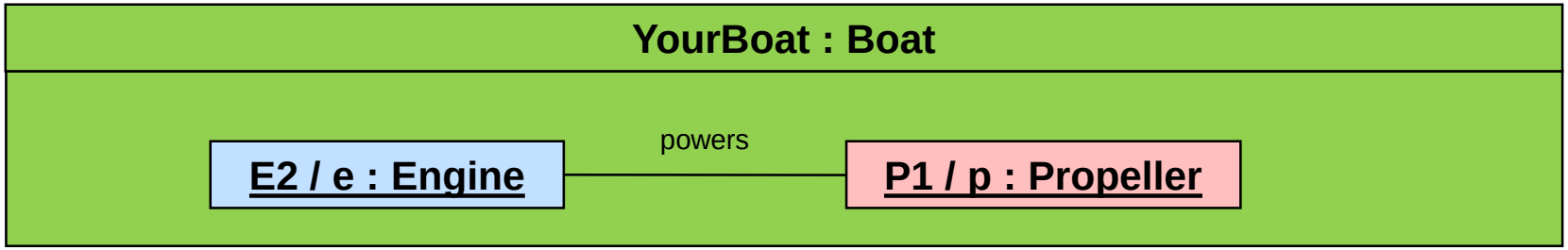
Legale Instanzierungen des Kompositionsstrukturdiagramms





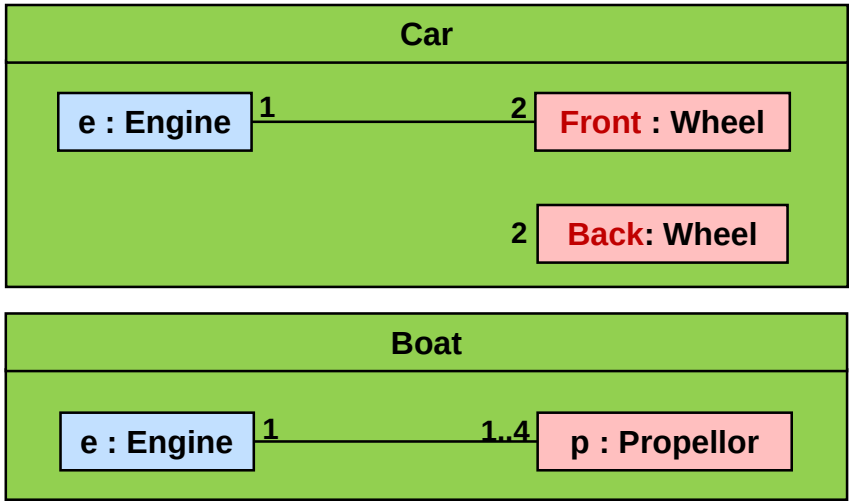
Kompositionsdiagrammnotation auf Objektebene (äquivalent zur vorherigen)



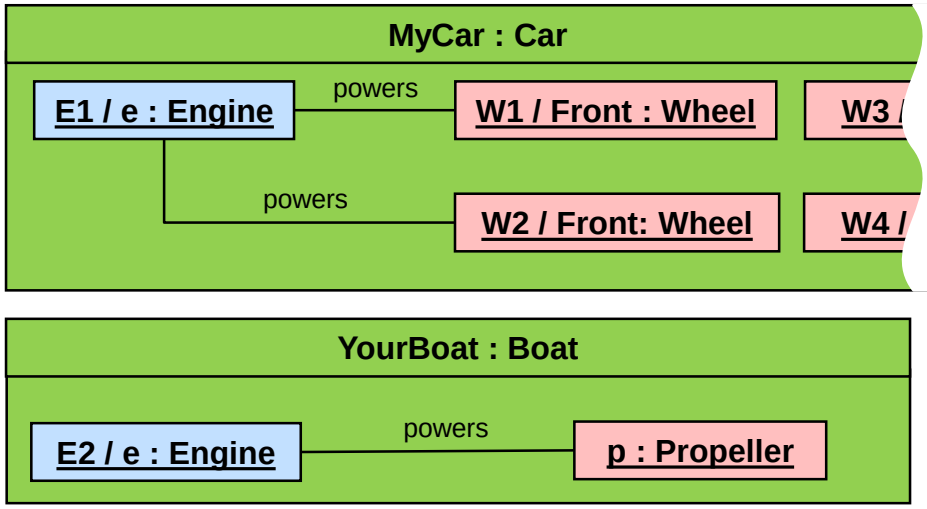


Zusammenfassung

Kompositionsstrukturdiagramm



Kompositionsobjektdiagramm



- Kontextabhängige Komposition
  - Beispiel: Motor im Auto oder Boot
  - Notation: Kontext = Schachtelung = Kompositionsbeziehung
- Rollenabhängige Komposition
  - Beispiel: Antrieb nur für Vorderräder
  - Notation: Klasse mehrfach im Klassendiagramm in verschiedenen Rollen
  - Notation: Unterscheidung Name / Rolle : Typ im Objektdiagramm

Literatur zu Kompositionsstrukturdiagramm

JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©JOT, 2004

Vol. 3, No. 10, November-December 2004

→ http://www.jot.fm/issues/issue_2004_11/column5/

→ http://www.jot.fm/issues/issue_2004_11/column5/column5.pdf

UML 2 Composition Model

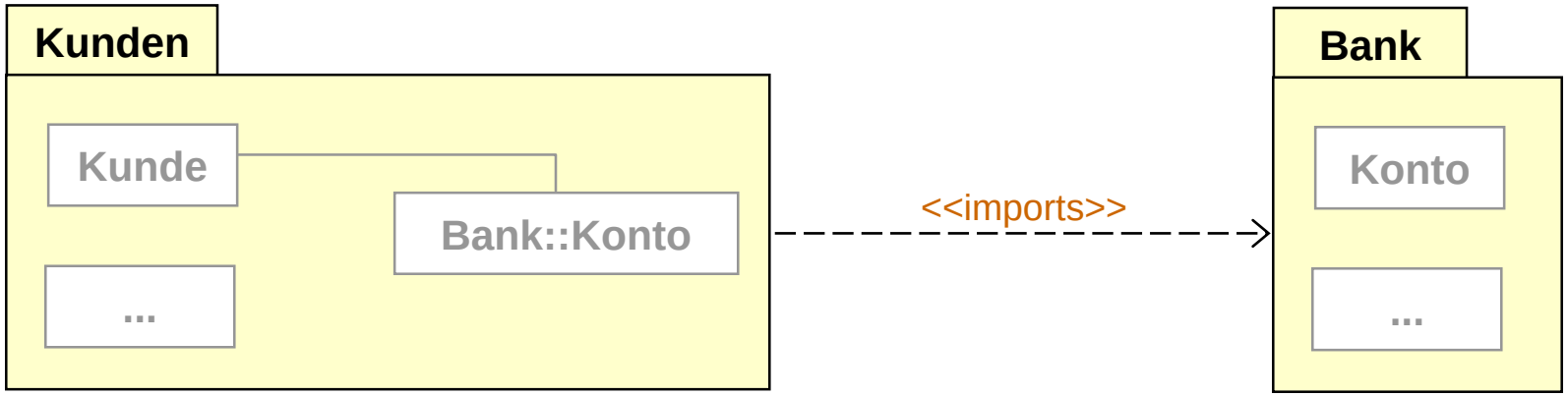
Conrad Bock, U.S. National Institute of Standards and Technology

The composition model in the Unified Modeling Language, version 2 (UML 2), is a major upgrade to the familiar "black diamond" composition of earlier versions. It supports connections between parts at the same level of decomposition, in addition to the usual part-whole associations. Complex networks of entities can be represented within a single class, inherited to subclasses, with links maintained between objects playing parts at runtime. The model also supports connections between parts of parts (ports), enabling more detailed structural modeling and message forwarding, which increases independence of reused applications and provides better plug-compatibility for components.

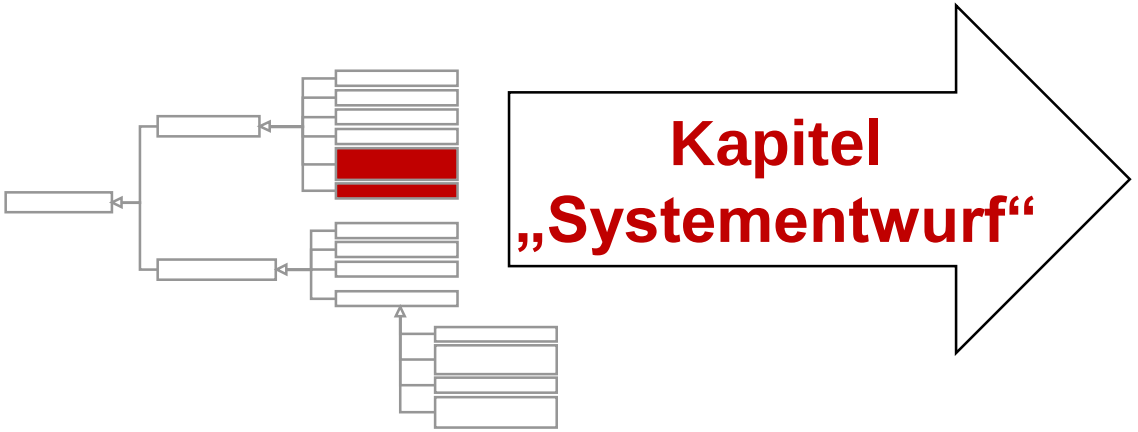
3.3.4 Paketdiagramme

Paketdiagramme

- Bedeutung
  - Gruppierung thematisch zusammengehöriger Klassen
- Darstellung
  - “Karteikarten”, auf denen die dazugehörigen Klassen aufgemalt sind
- Referenz auf eine Klasse in einem anderen Package
  - package::class
- Import aus anderem Package
  - Gestrichelter Abhängigkeits-Pfeil mit stereotyp <<imports>>



Strukturdiagramme „im Großen“

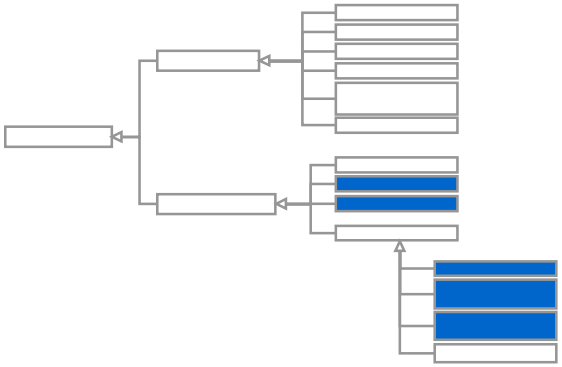


Komponentendiagramme

Verteilungsdiagramme

3.4 Verhaltensmodellierung

3.4.1 Sequenzdiagramme
3.4.2 Kommunikationsdiagramme
3.4.3 Aktivitätsdiagramme
3.4.4 Interaktionsübersichtsdiagramme
3.4.5 Zustandsdiagramme



3.4.1 Sequenzdiagramme

Einführung

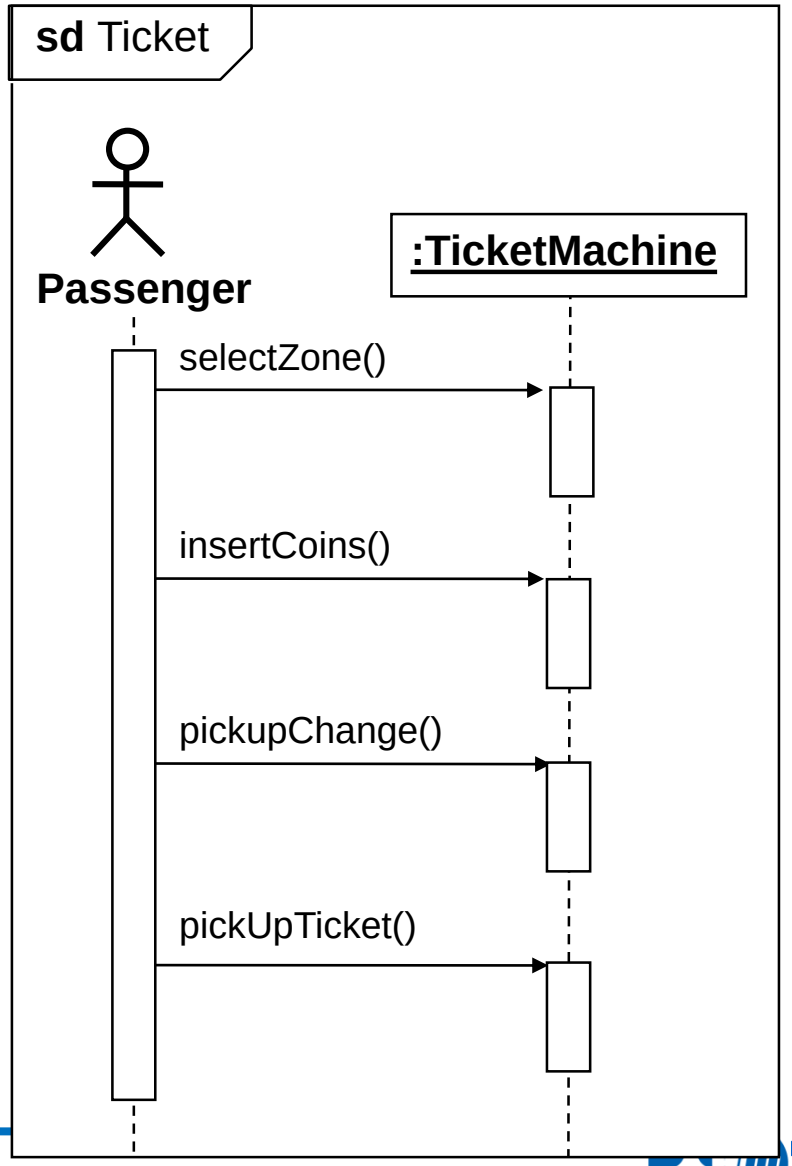
Kontroll- und Datenfluss

Interaktionsfragmente

→ „UML@Work“, Seite 251 - 287

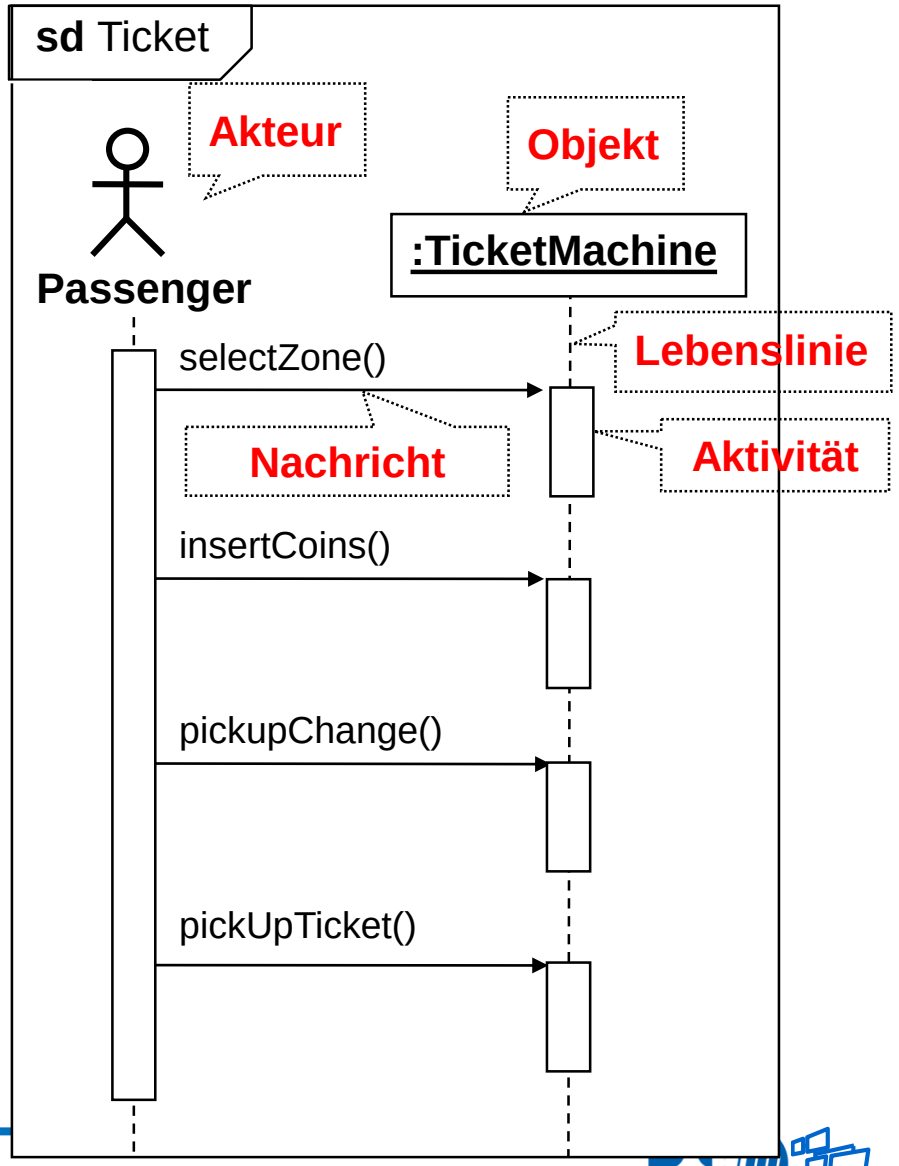
Sequenzdiagramm (‚sequence diagram')

- Visualisiert Nachrichten entlang einer vertikalen Zeitachse
- Modelliert Interaktion zwischen
  - Akteuren und Objekten
  - verschiedenen Objekten
  - einem Objekt mit sich selbst
- Bietet Darstellung von
  - Datenfluss
  - Zeitpunkte und –dauer
  - Nebenläufigkeit
  - Verzweigungen / Schleifen
  - Filterungen / Zusicherungen



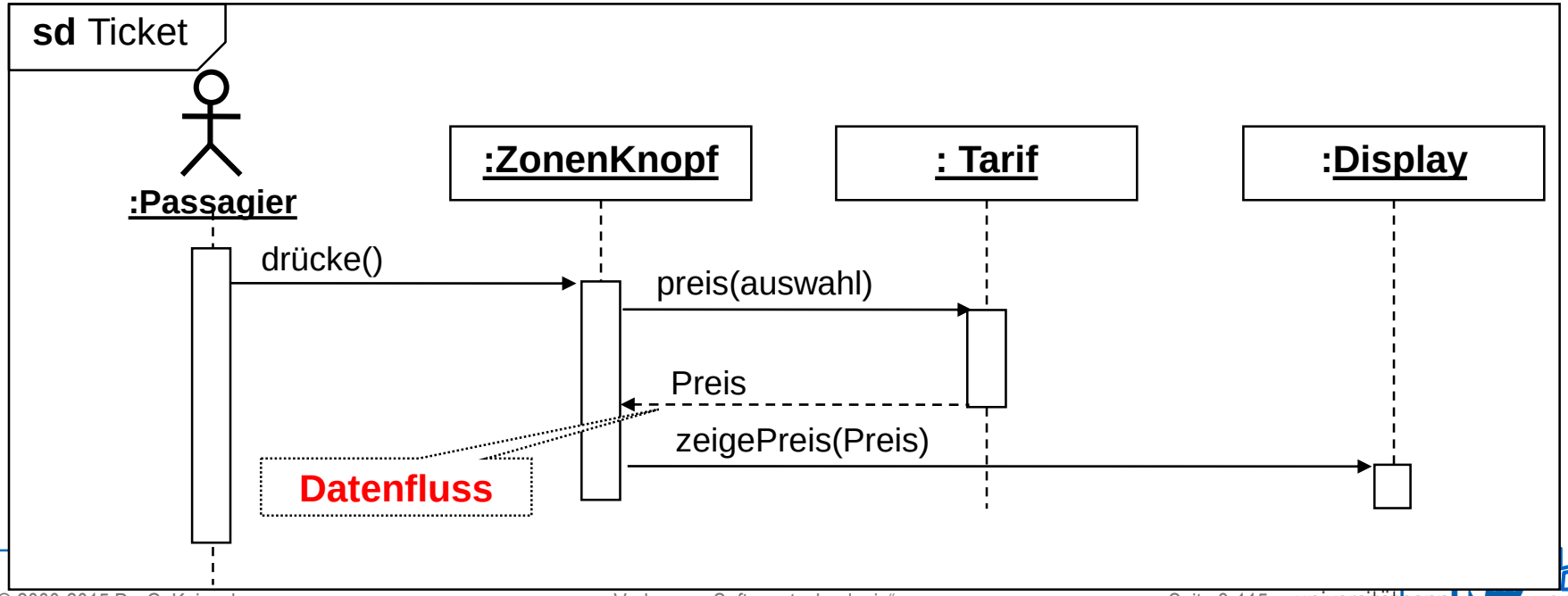
Sequenzdiagramm (‚sequence diagram')

- Akteure
  - Wie im Use-Case-Diagramm
- Objekte
  - Objektdiagramm-Notation für passive oder aktive Objekte
- Lebenslinie
  - Zeitraum in dem das Objekt existiert
  - Später: Explizite Notation für Objekt-"Geburt" und -"Tod"
- Aktivität / Aktivierung
  - Zeitraum in dem das Objekt etwas tut
  - Ausgelöst durch Nachricht bei passiven Objekten
  - Thread bei aktiven Objekten



Sequenzdiagramme: Kontroll- und Datenfluss

- Jeder Pfeil beginnt an der Aktivierung, welche die Nachricht gesendet hat
- Die Aktivierung an der der Pfeil endet beginnt direkt am Pfeilkopf
- Eine Aktivierung dauert (mindestens) so lange wie alle geschachtelten Aktivierungen
- Für synchrone Nachrichten erfolgen Rückgaben implizit am Ende einer Aktivierung
  - Pfeile für Rückgaben werden nur benutzt, um den Datenfluss explizit zu machen



Sequenzdiagramme (wichtige Elemente)

UML 2.0

- Interaktionspartner
  - passives Objekt (aktiv nur als Reaktion auf Nachrichten)
  - aktives Objekt (Prozess/Thread)

rolle:Typ

rolle:Typ

- Nachricht
  - synchron (Aufrufender wartet auf Ende der Aktivierung)
  - asynchron (Aufrufender sendet Nachricht und macht sofort weiter)

m1

m2

- Antwortnachricht
  - ohne Ergebnis (bei synchronen Nachrichten meist weggelassen)
  - mit Rückgabewert

wert

- Zeiteinschränkung
  - Absoluter Zeitpunkt
  - Relativer Zeitpunkt
  - Intervall

at(12.00)

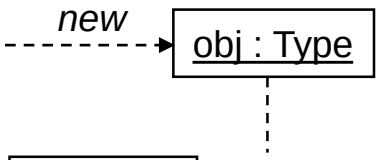
after(12.00)

{12.00…13.00}

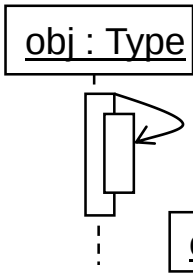
Sequenzdiagramme (wichtige Elemente)

UML 2.0

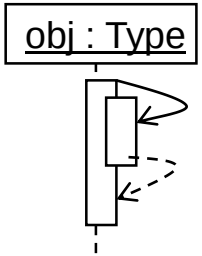
- Objekt wird durch Nachricht erzeugt



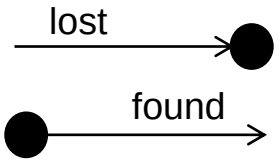
- Nachricht an sich selbst



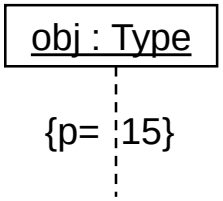
- Rekursiver Aufruf



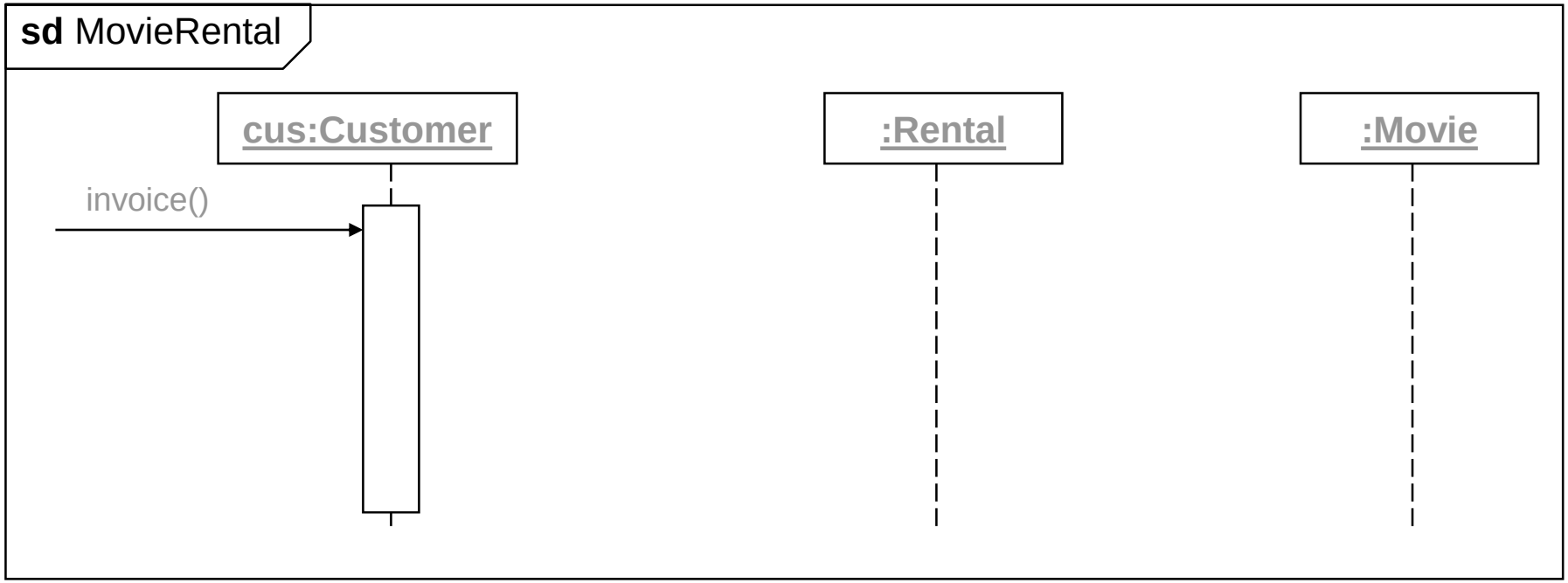
- Nachrichten ohne explizit modellierte Interaktionspartner



- Zustandsinvariante
  - Bedingung, die zu einem gewissen Zeitpunkt erfüllt sein muss

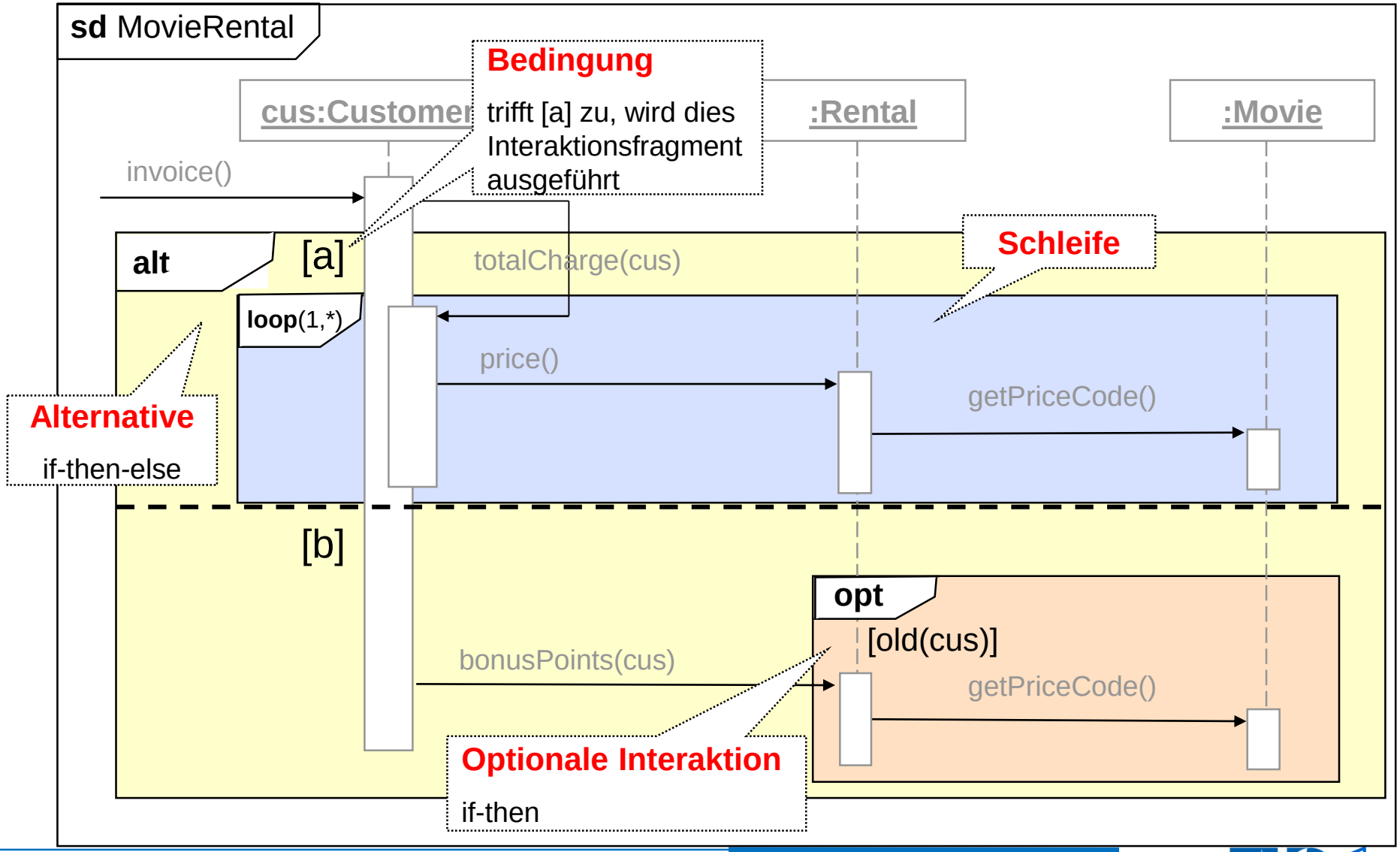


Sequenzdiagramme: Kontrollstrukturen in UML 2 → „Kombinierte Fragmente“



- Szenario: Ausdrucken einer Rechnung für Kunden eines Videoverleihs
  - Alternative: Falls *a* Gesamtkosten drucken, falls *b* Bonuspunkte drucken
  - Schleife: Gesamtkosten sind kosten aller Ausleihvorgänge (:Rental)
  - Optionale Aktionen: Bonuspunkte für Altkunden anders berechnen

Sequenzdiagramme: Kontrollstrukturen in UML 2 → „Kombinierte Fragmente“



Sequenzdiagramme: Interaktionsfragmente

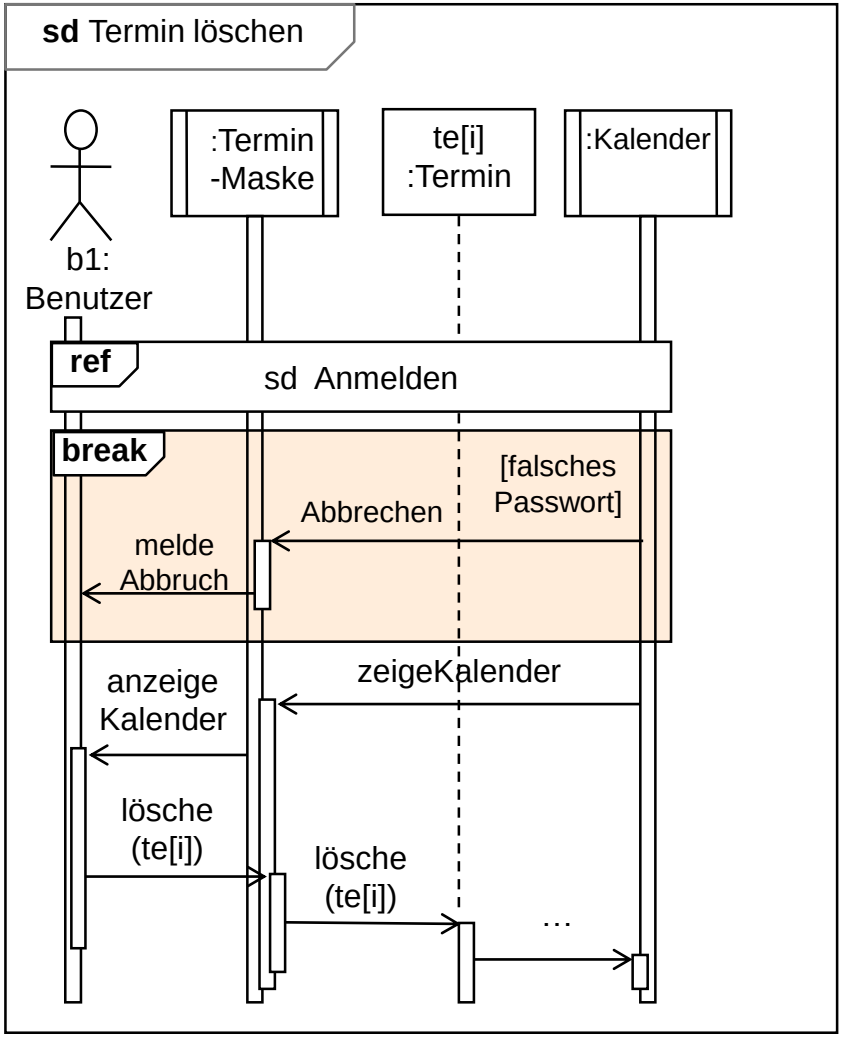
- Komplexere Kontrollstrukturen werden durch Operatoren auf Interaktionsfragmenten (Ausschnitte des Sequenzdiagramms) realisiert

	Operator	Zweck
Verzweigungen und Schleifen	alt	Alternative Interaktionen – if-then-else
	opt	Optionale Interaktionen – if-then
	break	Ausnahme-Interaktion – umgebendes Interaktionsfragment verlassen
	loop	Iterative Interaktionen
Nebenläufigkeit und Ordnung	seq	Sequentielle Interaktionen mit schwacher Ordnung (Default)
	strict	Sequentielle Interaktionen mit strenger Ordnung
	par	Nebenläufige Interaktionen
	critical	Atomare Interaktionen
Filterung und Zusicherung	ignore	Irrelevante Interaktionen
	consider	Relevante Interaktionen
	assert	Zugesicherte Interaktionen
	neg	Ungültige Interaktionen

- Anwendung siehe Beispiel auf der nächsten Folie

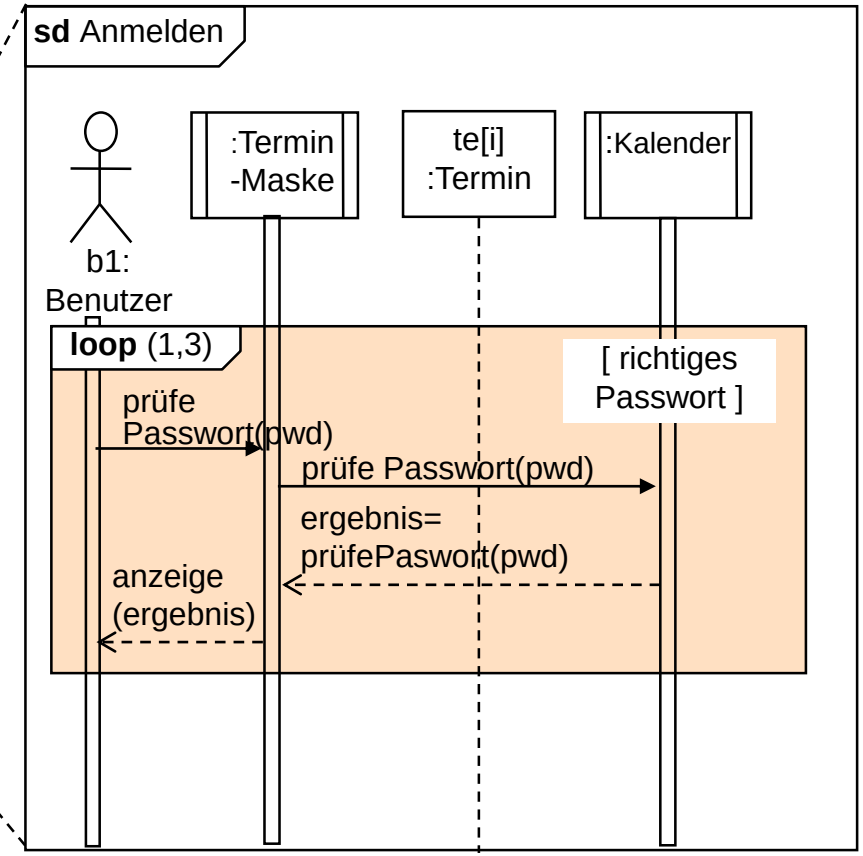
Sequenzdiagramm: Kalender-Beispiel

- Gemeinsamer Kalender
  - À la Google-Kalender
  - Man muss sich anmelden
- Hier: „Löschen eines Termins“
  - Erst anmelden …
  - dann Terminanzeige …
  - dann Selektion des Termins
  - … und Löschen
- Quelle
  - „UML@Work“, Abbildung 4-46



Sequenzdiagramm: Kalender-Beispiel

- Was versteckt sich hinter der Referenz?
  - Ein beliebiges dynamisches Diagramm
  - Hier: Der Anmeldevorgang



- loop(1,3) [cond]
  - Kombination von for- und do-until-Schleife
  - Wiederhole Fragment maximal 3 mal aber hör auf sobald „cond“ erfüllt ist,
    - ⇨ d.h. for (i=1..3) {...; if cond break;}

Sequenzdiagramme: Fazit

- Sequenzdiagramme…
  - …repräsentieren Verhalten bezüglich Interaktionen.
  - …ergänzen die Klassendiagramme, welche die Struktur repräsentieren.
  - …sind nützlich, um
    - ⇨ das Verhalten eines Anwendungsfalls auf die beteiligten Objekte abzubilden
    - ⇨ beteiligte Objekte zu finden
    - ⇨ Verhalten präzise zu beschreiben

- Der Zusatzaufwand lohnt sich auf jeden Fall bei komplexen und unklaren Abläufen
  - Aber: keine Zeit vergeuden, um Trivialitäten und Offensichtliches mit Sequenzdiagrammen zu modellieren

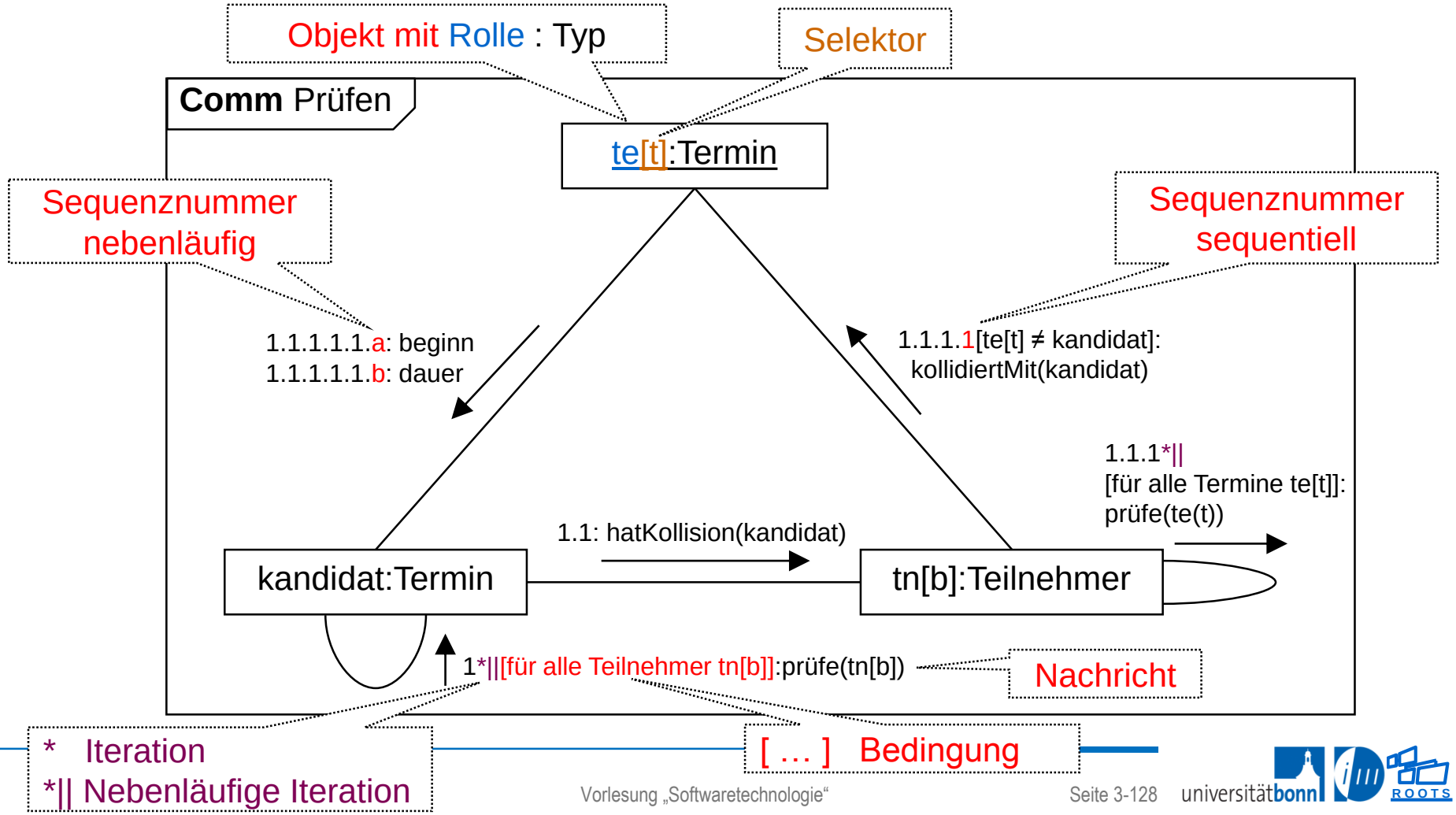
3.4.2 Kommunikationsdiagramme

→ „UML@Work“, Seite … - …

Kommunikationsdiagramm (‚Communication diagram')

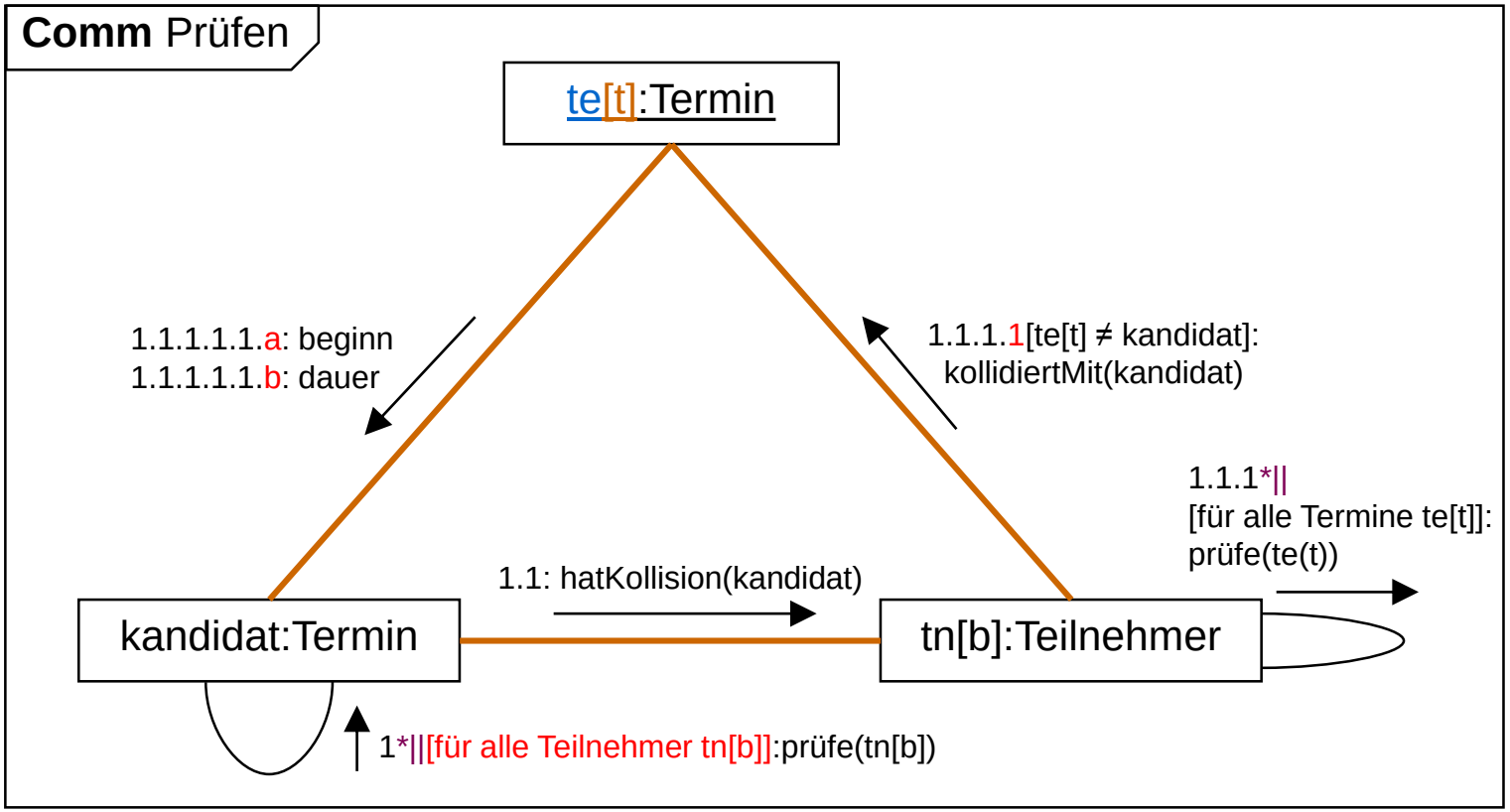
In UML1: Kollaborationsdiagramm

Beschreibt Interaktion zwischen Objekten einschließlich struktureller Informationen (Beziehungen der Objekte)



Kommunikationsdiagramm (‚Communication diagram')

- Beziehungen im Kommunikationsdiagramm
  - permanente Verbindungen: Assoziationen oder globale Variablen
  - temporäre Verbindungen: Parameter oder lokale Variablen



Kommunikationsdiagramm

- Beschreibt Interaktion zwischen Objekten
  - Zeigt auch strukturelle Informationen (Beziehungen der Objekte)
- Zeitlicher Verlauf wird durch Nummern angegeben
  - Syntax: „Nummer : Nachricht“
  - Beispiel: „1 : foo, 2 : bar“ → zuerst wird „foo“ gesendet, dann „bar“
- Nummer zeigt sequenzielle oder parallele Verarbeitung
  - Sequentielle Bearbeitung → Zahlen
    - ⇨ 1.1, 1.2 sind sequentielle Teilabläufe von 1
  - Nebenläufige Bearbeitung → Namen
    - ⇨ 1.a, 1.b sind nebenläufige Teilabläufe von 1
- Iteration wird nach der Sequenznummer angegeben:
  - Sequentiell: 2.3 * [i=1..n]: nachricht(i)
  - Nebenläufig: 2.3 *|| [i=1..n]: nachricht(i)

Vergleich zu Sequenzdiagramm

- Kommunikationsdiagramme sind schwieriger
  - Zeitliche Abfolge muss aus Nummerierung rekonstruiert werden
  - Bei Änderungen muss viel neu Nummeriert werden
    - ⇨ Gute Werkzeugunterstützung ist daher essentiell

- Kommunikationsdiagramme sind einfacher
  - Strukturelle Informationen zusätzlich darstellbar
  - Objekte können überall platziert werden
    - ⇨ Weniger Überkreuzungen → übersichtlicher
  - Leichter kollaborativ am Whiteboard zu erstellen

3.4.3 Aktivitätsdiagramme

Einführung

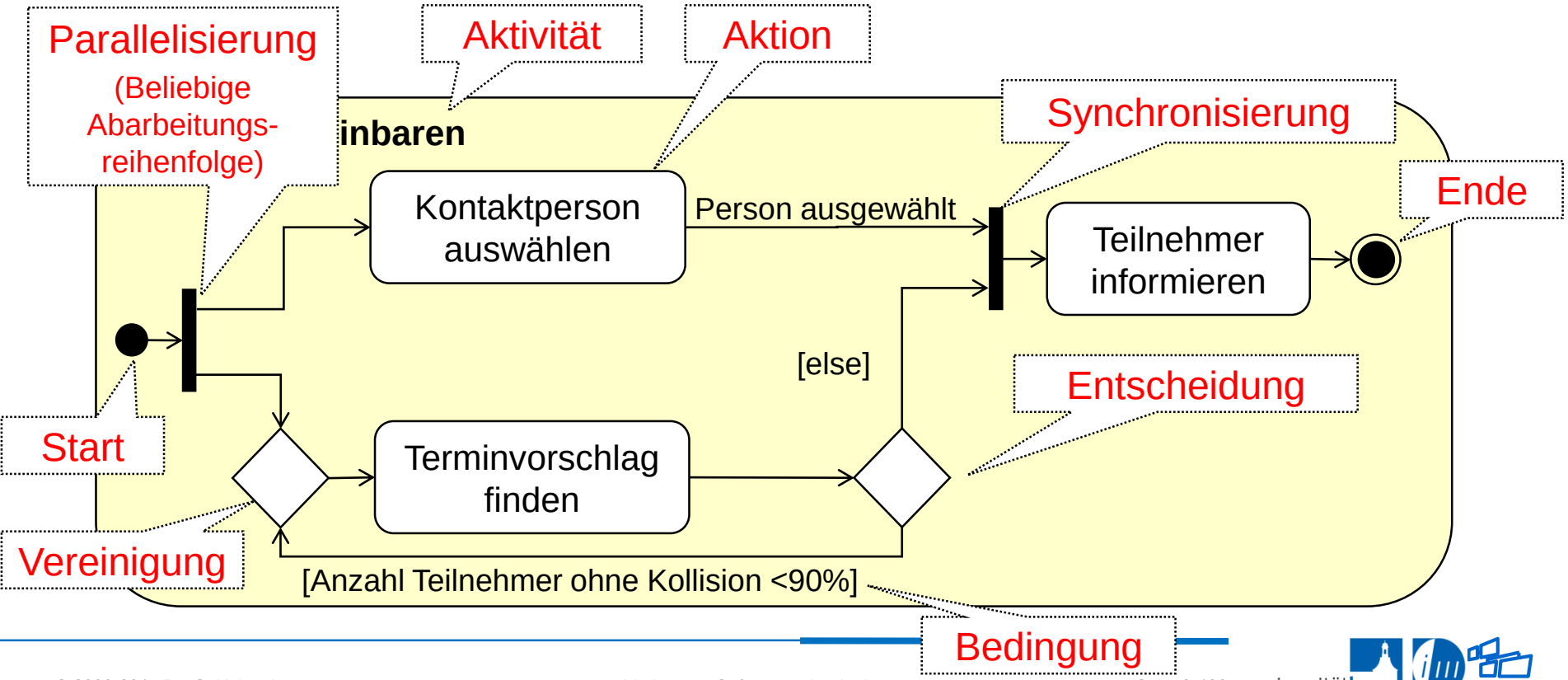
Tokenbasierte Semantik (Petri-Netze)

Ablaufende versus Endzustand

Notation im Detail

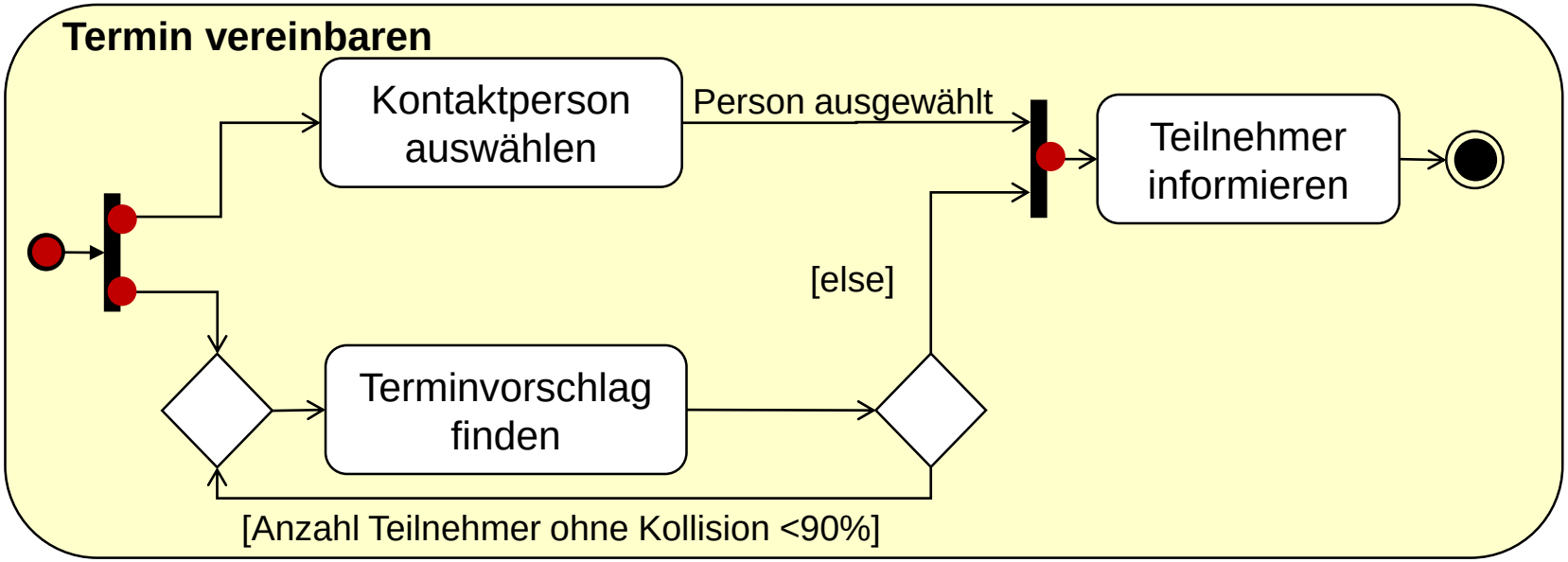
Aktivitätsdiagramme

- Eine Aktion ist ein elementarer Arbeitsschritt
  - atomar (d.h. Effekte werden bei Unterbrechung rückgängig gemacht)
- Eine Aktivität spezifiziert den Kontroll- und Datenfluss zwischen Aktionen



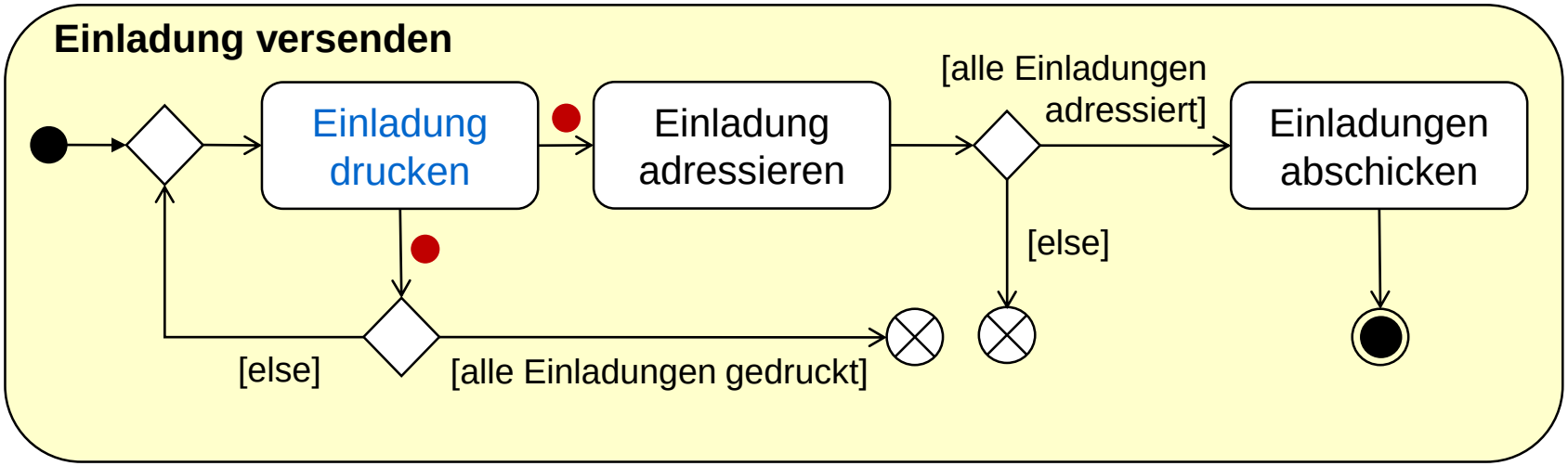
Aktivitätsdiagramme: Tokenbasierte Semantik

- Operationales Modell: Token, die Abläufen entsprechen, fließen durch das Diagramm ab dem Startknoten.
  - Parallelisierung: Auf jeder Ausgangskante fließt je ein Token weiter.
  - Synchronisation: Auf jeder Eingangskante muss ein Token ankommen.
  - Vereinigung: Jedes ankommende Token wird weitergeleitet.
  - Entscheidung: Token fließt nur auf der Ausgangskante weiter, deren Bedingung wahr ist.



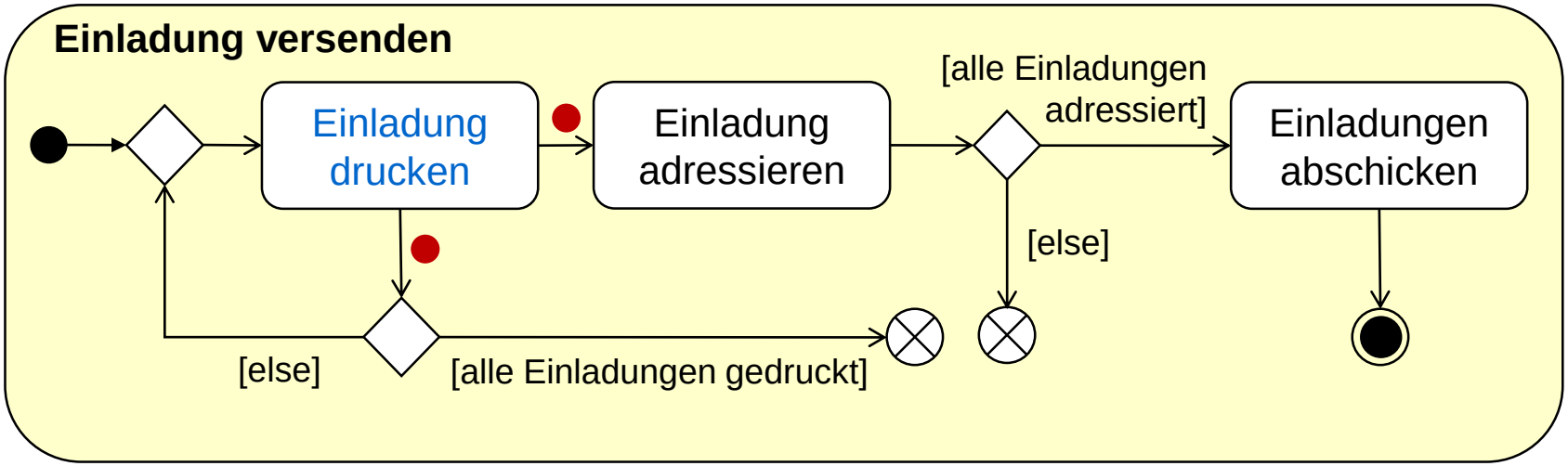
Ablaufende versus Endzustand

- Ablaufende ⊗
  - Ende eines Ablaufs (andere können weiterlaufen)
    - ⇨ Nur ein einzigen Token wird beendet
- Endzustand ◉
  - Ende aller Abläufe, die gerade das Diagramm durchlaufen
    - ⇨ Bildlich: Wenn ein Token den Endzustand erreicht, werden alle anderen Token mit beendet



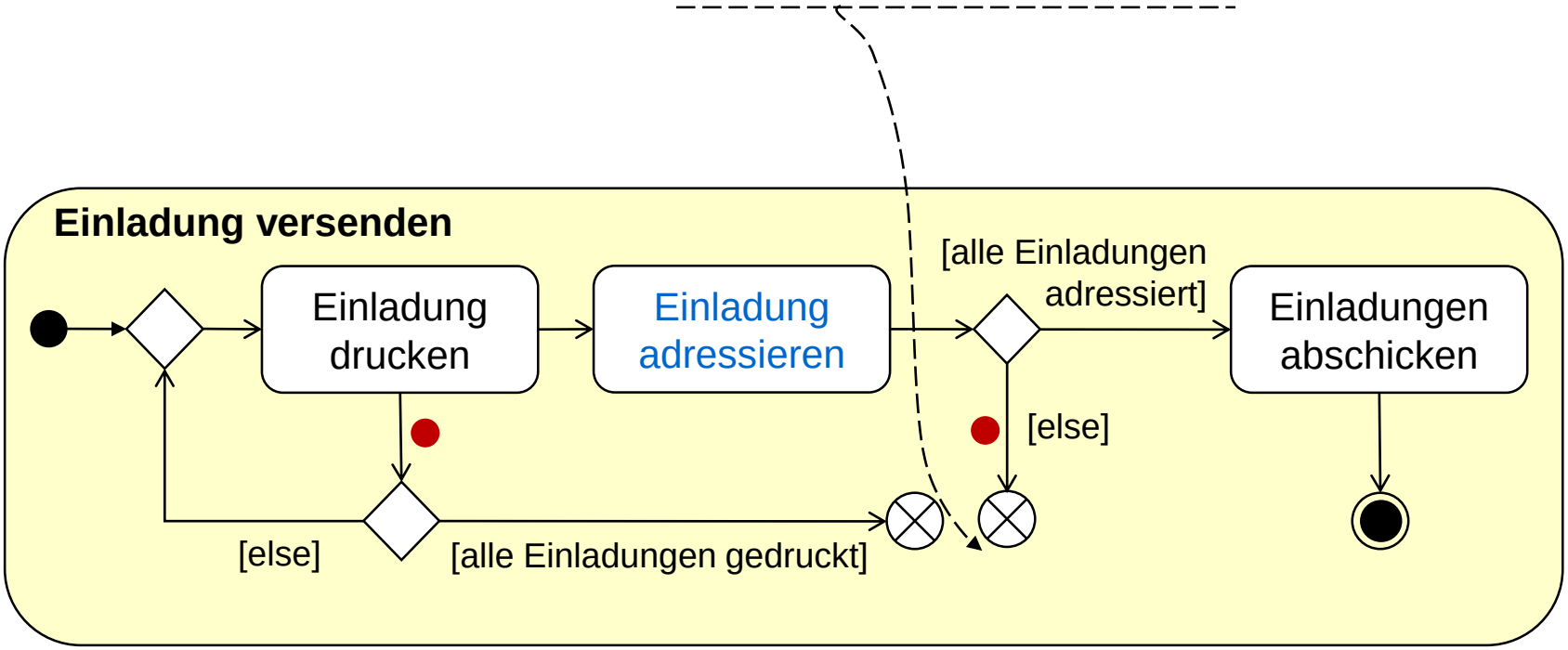
Endzustand versus Ablaufende: Beispiel

- Drucken, Adressieren und Versenden von Einladungen
  - Wenn eine Einladung gedruckt ist, fließen 2 Token weiter, eines auf jeder Ausgangskante (implizite Parallelisierung)
  - Die Einladung wird daher als Nächstes adressiert und es werden gleichzeitig weitere Einladungen gedruckt



Endzustand versus Ablaufende: Beispiel

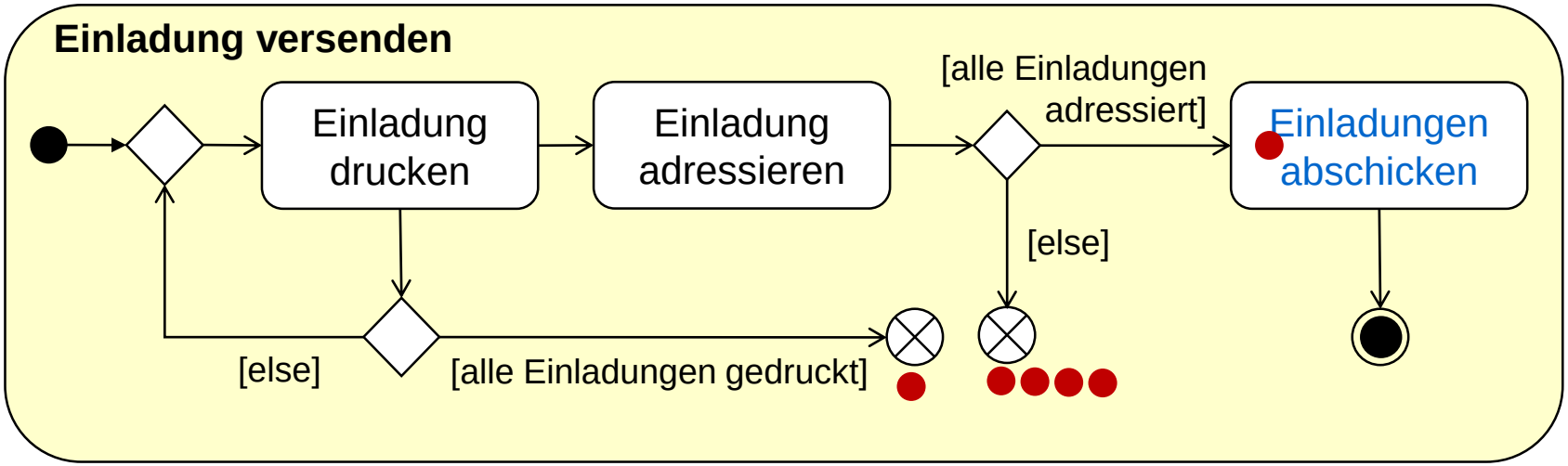
- Drucken, Adressieren und Versenden von Einladungen
  - Wenn eine Einladung adressiert ist und es nicht die letzte war, ist der entsprechende Ablauf beendet.
  - Das Token verschwindet im entsprechenden Ablaufende



Endzustand versus Ablaufende: Beispiel

- Wenn die letzte Einladung gedruckt ist verschwindet das „Drucken“-Token im entsprechenden Ablaufende
- Wenn die letzte Einladung addressiert ist, werden alle (gedruckten und adressierten) Einladungen gemeinsam verschickt

Hier die Momentaufnahme beim Verschicken von 5 Einladungen:



- Die gesamte Aktivität ist danach beendet (Token erreicht Endzustand)

Aktivitätsdiagramme

- Ablauforientierte Notation
  - basierend auf Petri-Netzen und BPEL (Business Process Engineering Language)
- Auch für nicht objekt-orientierte Systeme
  - Aktivitäten sind unabhängig von Objekten
  - Anwendbar auch auf Geschäftsprozesse, Funktionsbibliotheken, …
- Elementare Konzepte
  - Aktivität = benanntes Verhalten
  - Aktion = atomare Aktivität (einzelner Arbeitsschritt)

Aktivitätsdiagramme: Aktivitäten

- Aktivitätsdiagramm
  - Graph bestehend aus Aktivitätsknoten und Aktivitätskanten

- Aktivitätsknoten
  - Kontrollknoten: Alternativen, Parallelisierung, Synchronisation
  - Datenknoten: Parameter, Puffer, Datenbanken
  - Aktionsknoten: Elementare, atomare, meist vordefinierte Aktionen

- Aktivitätskanten
  - Kontrollflusskanten: Verbindungen von Aktions- und Kontrollknoten
  - Datenflusskanten: Verbindungen von Datenknoten

Aktivitätsdiagramme: die wichtigsten Elemente

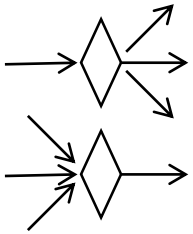
- Aktivität

Name der Aktivität

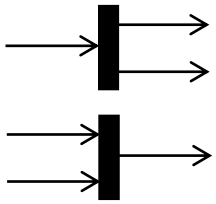
- Start- und Endknoten
  - Start
  - Aktivitätsende (Ende aller Abläufe)
  - Ablaufende (Ende eines bestimmten Ablaufs)

3 von 44 vordefinierten Aktionsknoten

- Alternative Abläufe
  - Entscheidungsknoten
  - Vereinigungsknoten



- Nebenläufige Abläufe
  - Parallelisierungsknoten
  - Synchonisierungsknoten



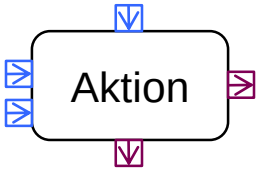
Kontrollknoten

Aktivitätsdiagramme: die wichtigsten Elemente

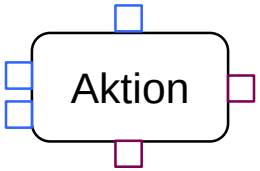
- Datenknoten
  - Parameter von Aktivitäten
  - Parameter von Aktionen (Pins)
  - Temporäre Puffer
  - Persistente Puffer
- Eingabeparameter
  - Explizit: Pfeil im Datenknoten hin zur Aktivität / Aktion
  - Implizit: Parameter links oder oben
- Ausgabeparameter
  - Explizit: … weg von der …
  - Implizit: … rechts oder unten …

Aktivitätsparameter überlappen den Rand!

Pins liegen außen!



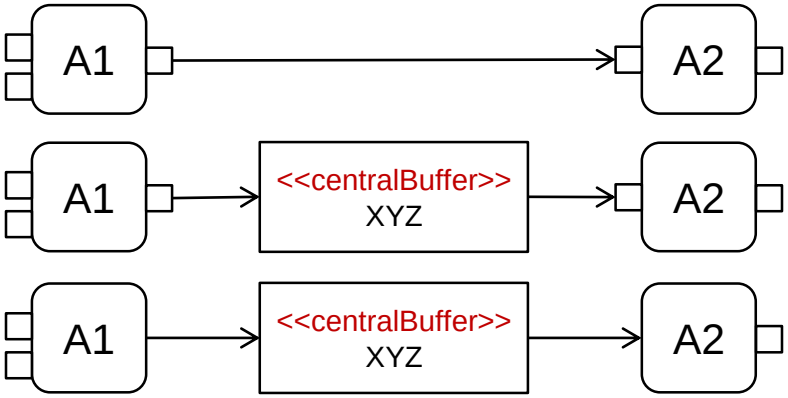
Explizit



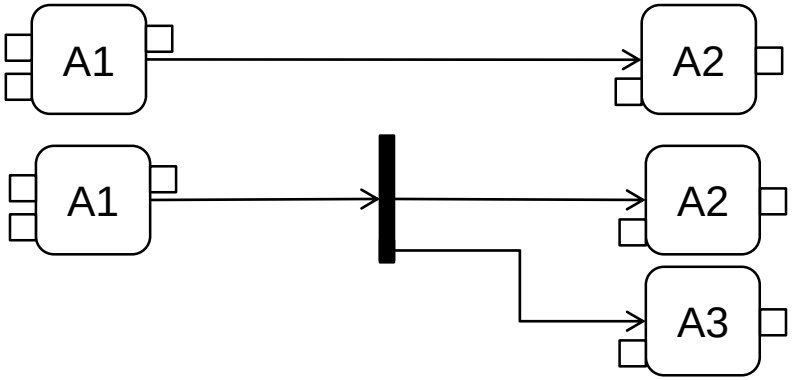
Implizit

Aktivitätsdiagramme: die wichtigsten Elemente

- Datenfluss
  - Durchgezogene spitze Pfeile zwischen Datenknoten
  - … oder direkt zwischen Aktionen, Aktivitäten und Kontrollknoten



- Kontrollfluss
  - Durchgezogene spitze Pfeile zwischen Aktionen, Aktivitäten und Kontrollknoten



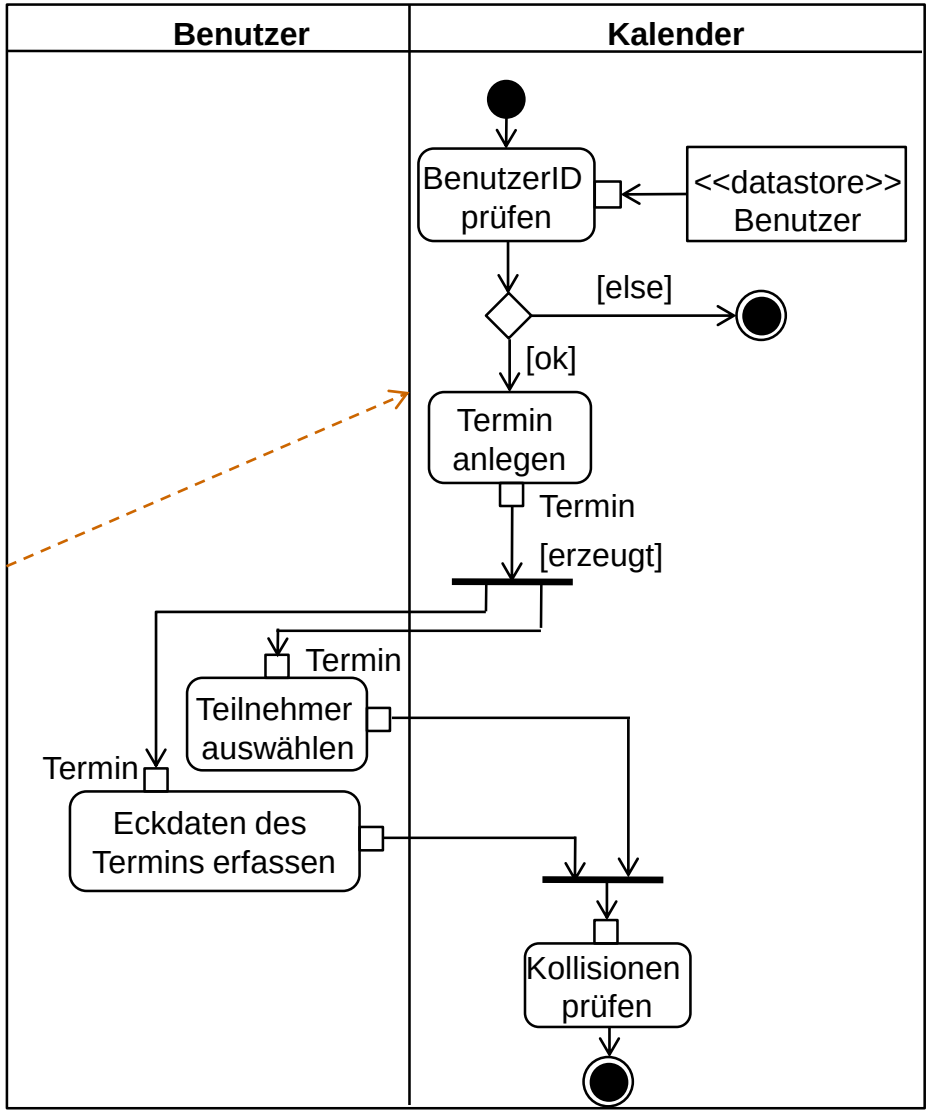
- Mischformen
  - Datenfluss hat auch impliziter Kontrollfunktion

Aktivitätsdiagramme: Aktionen

- Aktion = atomare Aktivität
  - Atomar = kann unterbrochen werden, macht dann aber jegliche Effekte rückgängig
- 44 vordefinierte Aktionen in 4 Kategorien
  - Alle sprachunabhängig definiert. Sonderfall: *OpaqueAction* als Hinweis auf Aktion die in einer bestimmten Implementierungssprache definiert ist
- Kommunikationsbezogenen Aktionen
  - *sendObjectAction, sendSignalAction*
    - ⇨ Objekt oder Signal an Empfänger senden
    - ⇨ asynchron, Ergebnis wird ignoriert
    - ⇨ Notation: Blockpfeile **Informiere Teilnehmer**
- Für die meisten Aktionen gibt es lediglich textuelle Stereotypen (keine eigene graphische Notation)

Aktivitätsdiagramm: Partitionen

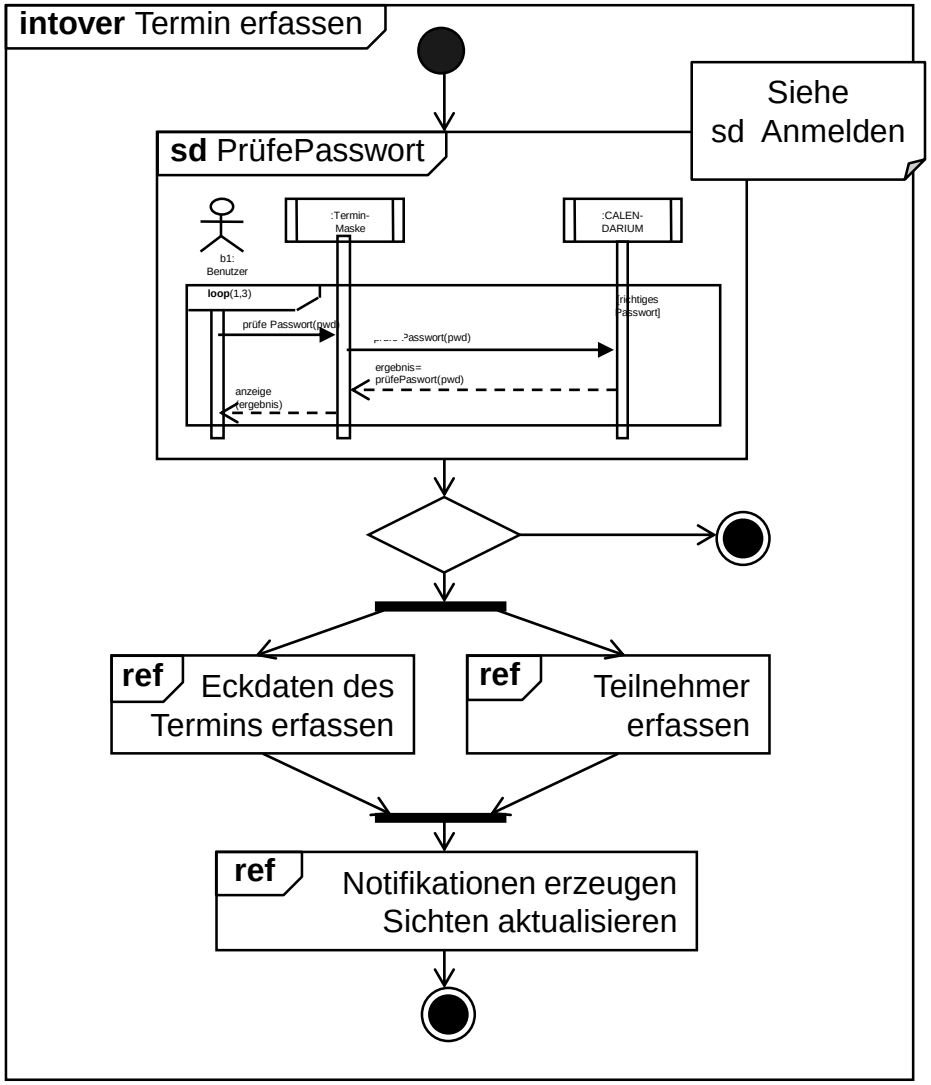
- Partition
  - Gruppierung von Teilen eines Aktivitätsdiagramms
  - Gruppierungskriterium ist beliebig
    - ⇨ Im Beispiel: Zugehörigkeit der Aktionen zu gewissen Klassen
- Notation
  - „Schwimmbahnen“/„Swimlanes“



3.4.4 Interaktionsübersichtsdiagramme

Interaktionsübersichtsdiagramm (‚Interaction overview diagram')

- Modelliert Kontrollfluss zwischen verschiedenen Interaktionsabläufen
  - Reihenfolge und Bedingungen
- Ist im Prinzip ein Aktivitätsdiagramm das statt Aktionen Interaktionsdiagramme als Knoten enthalten kann
  - Auch nur Referenzen auf Interaktionsdiagramme möglich („ref“)



3.4.5 Zustandsdiagramme

Einführung
Unterautomaten
Entry- und Exit
Innere Transitionen
History States
Transitionen im Detail
Notationsüberblick

Beispiel: Eine einfache LCD-Uhr

Struktur

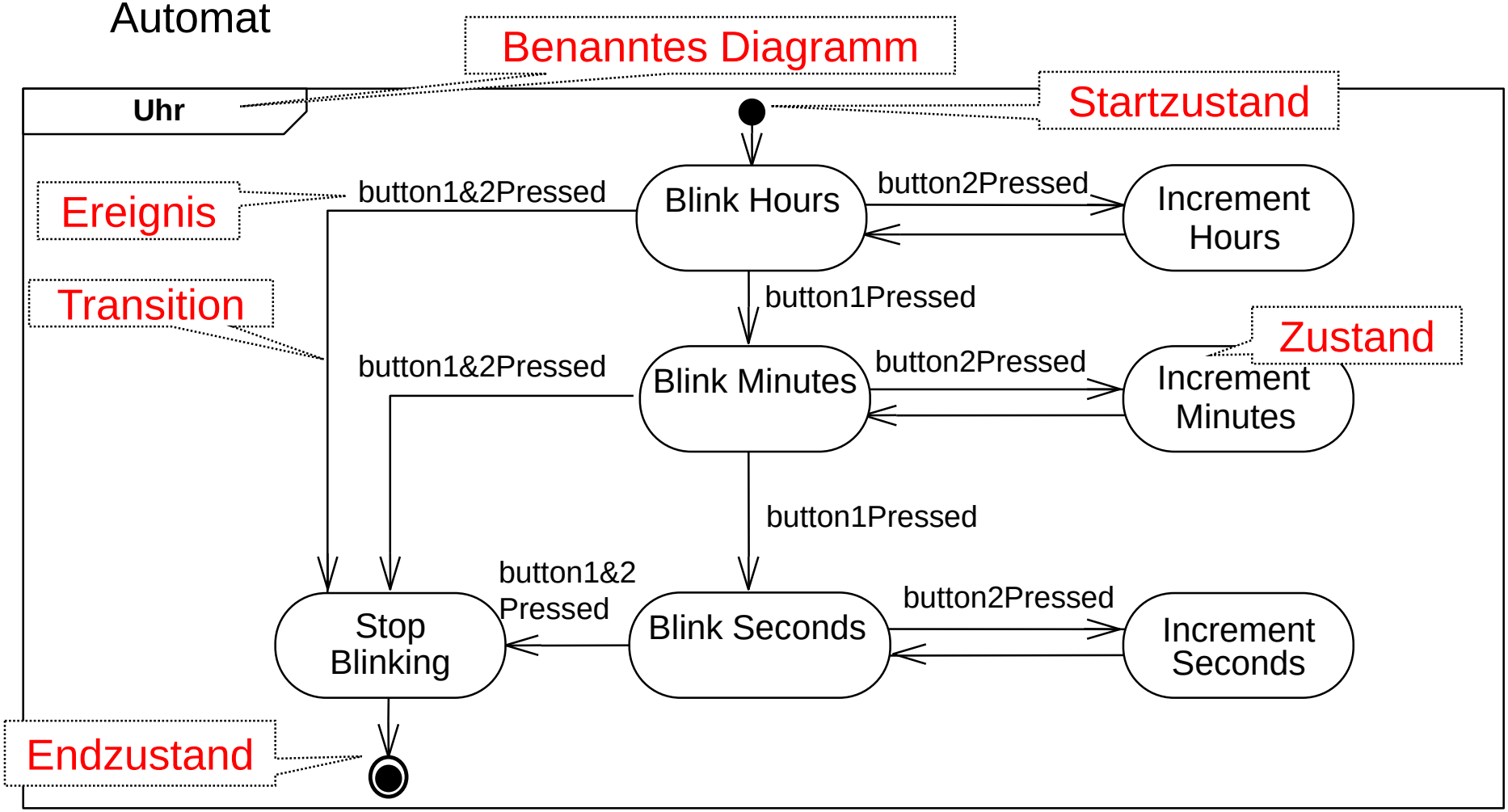
- 1 Display
- 2 Knöpfe
- 1 Batterie
- …

Verhalten

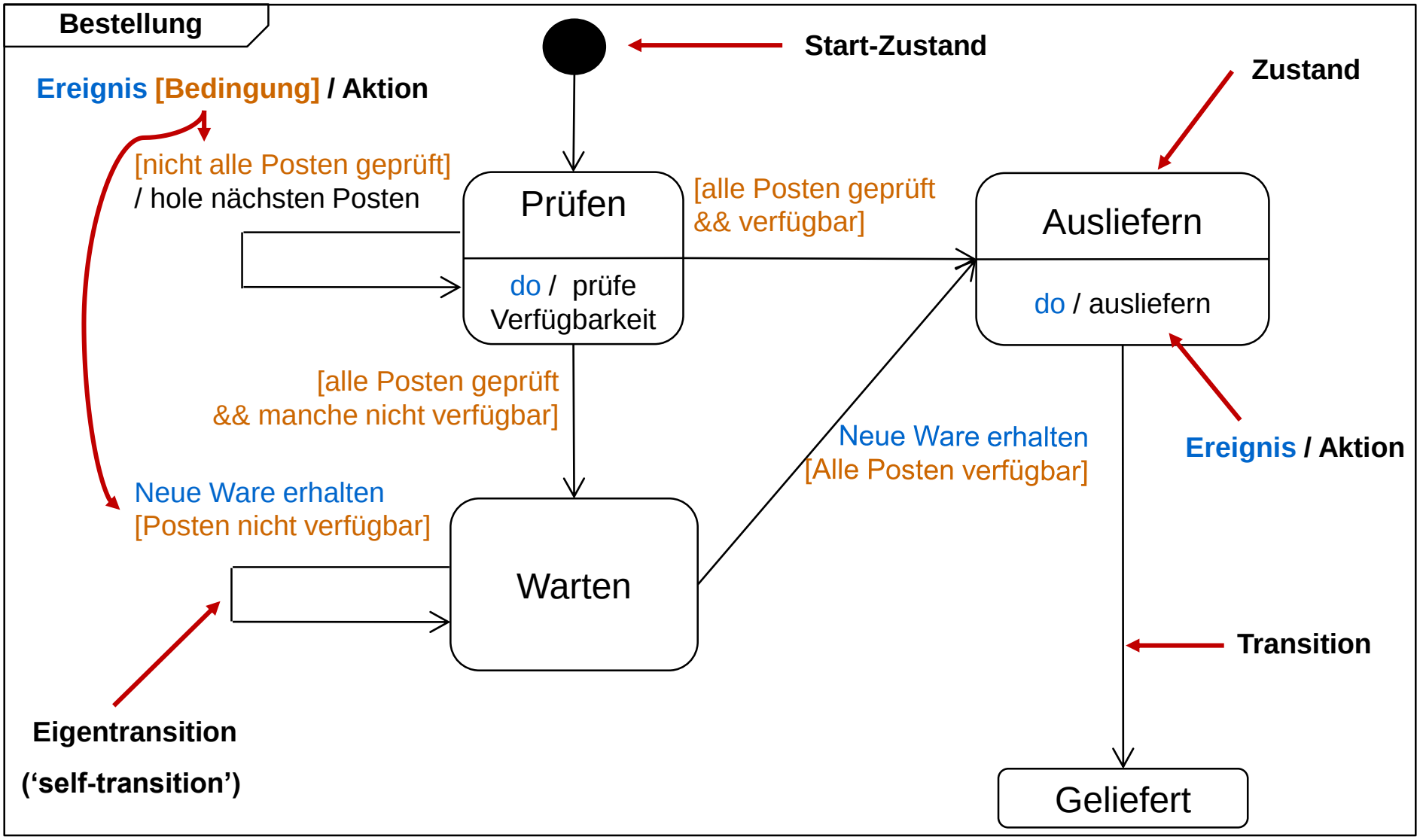
- Knopf 1 drücken
  - startet Stunden-Einstellung
  - Jeder weitere Druck schaltet weiter zu Minuten, Sekunden, Stunden, ….
- Knopf 2 drücken
  - erhöht den Wert des aktuell einzustellenden Elementes.
- Beide Knöpfe drücken
  - beendet die Einstellung

Zustandsdiagramm

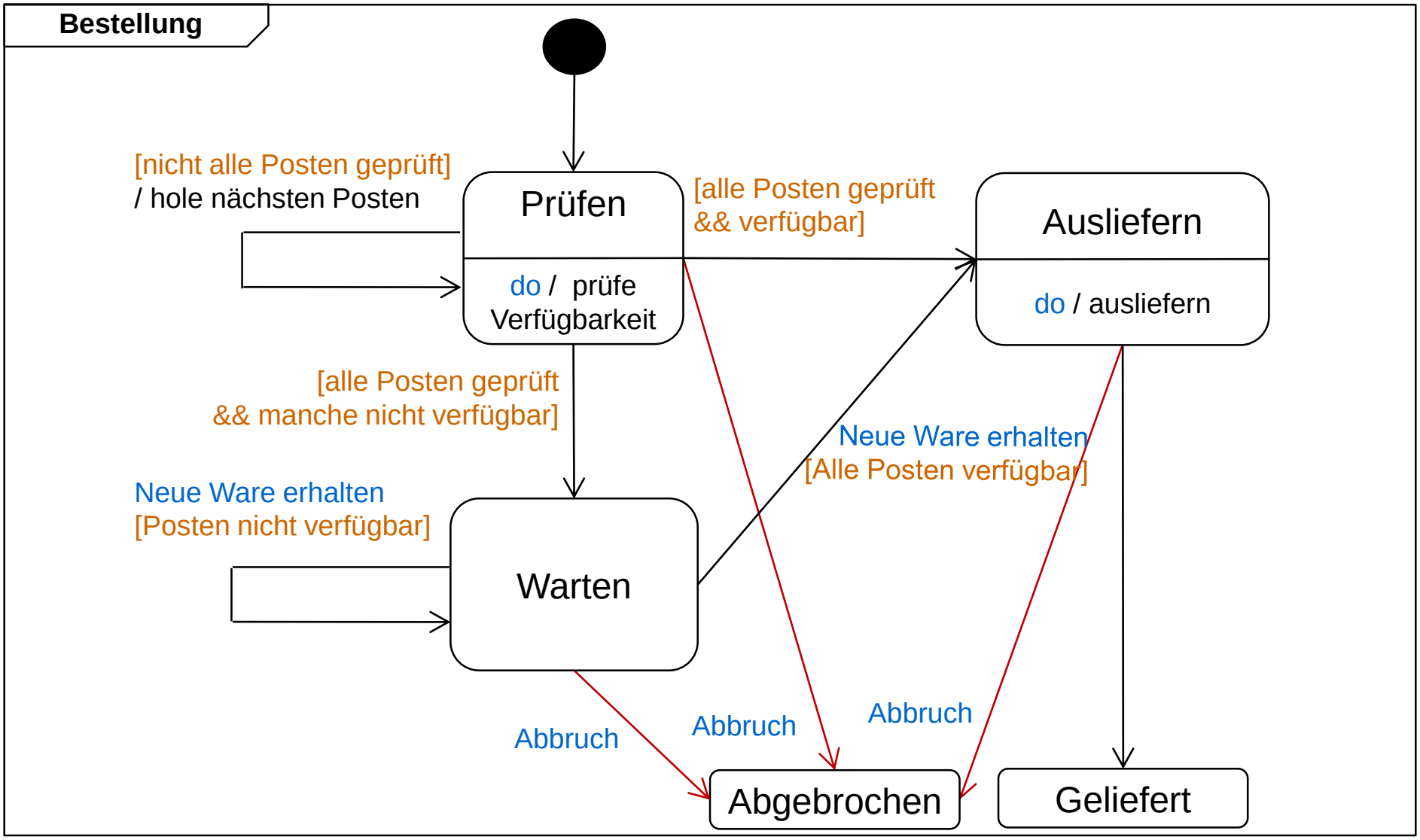
- Beschreiben das dynamische Verhalten eines Objektes als endlicher Automat



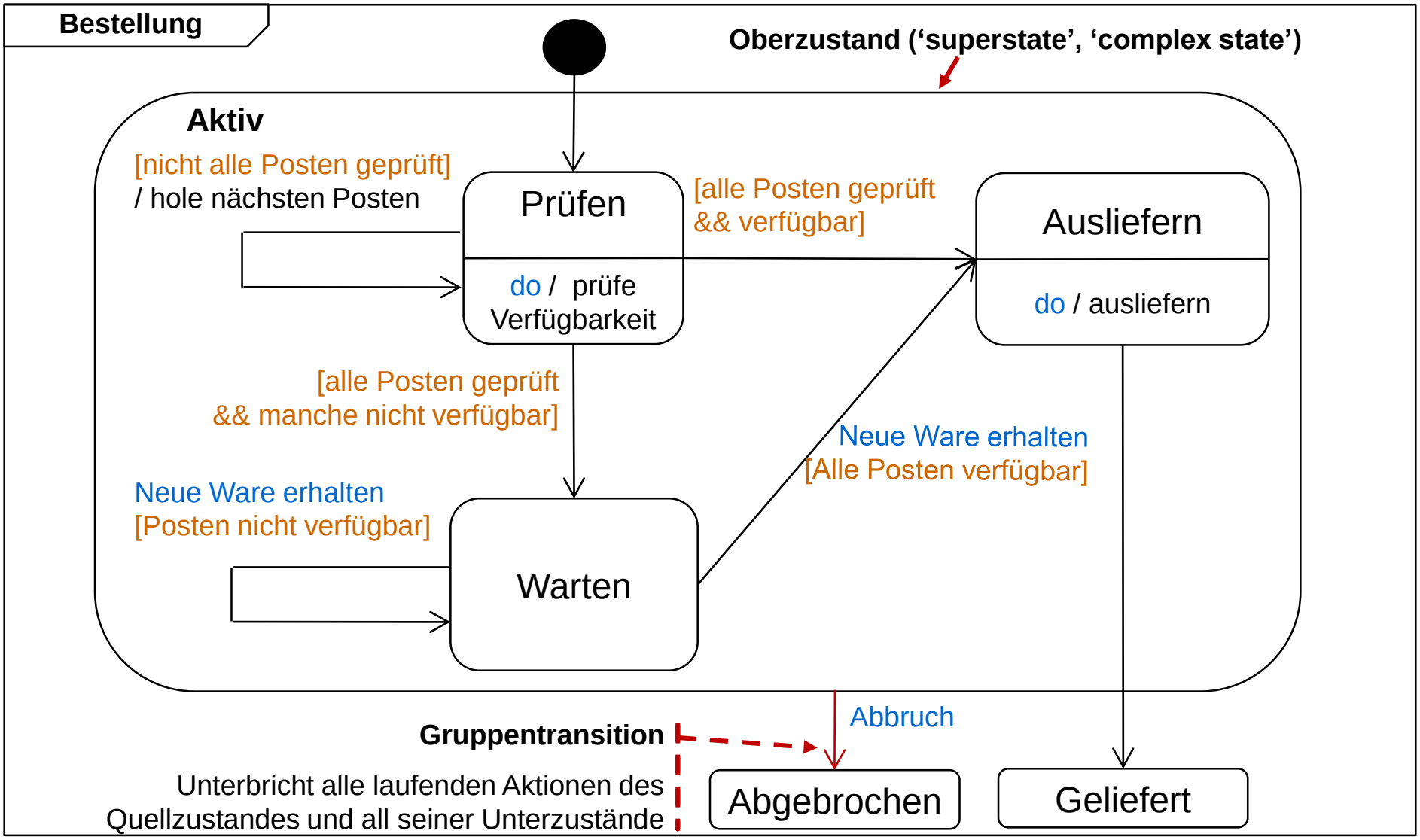
Zustandsdiagramm: Beispiel „Bestellungsbearbeitung“



Zustandsdiagramm: Beispiel „Bestellungsbearbeitung“ mit Abbruch

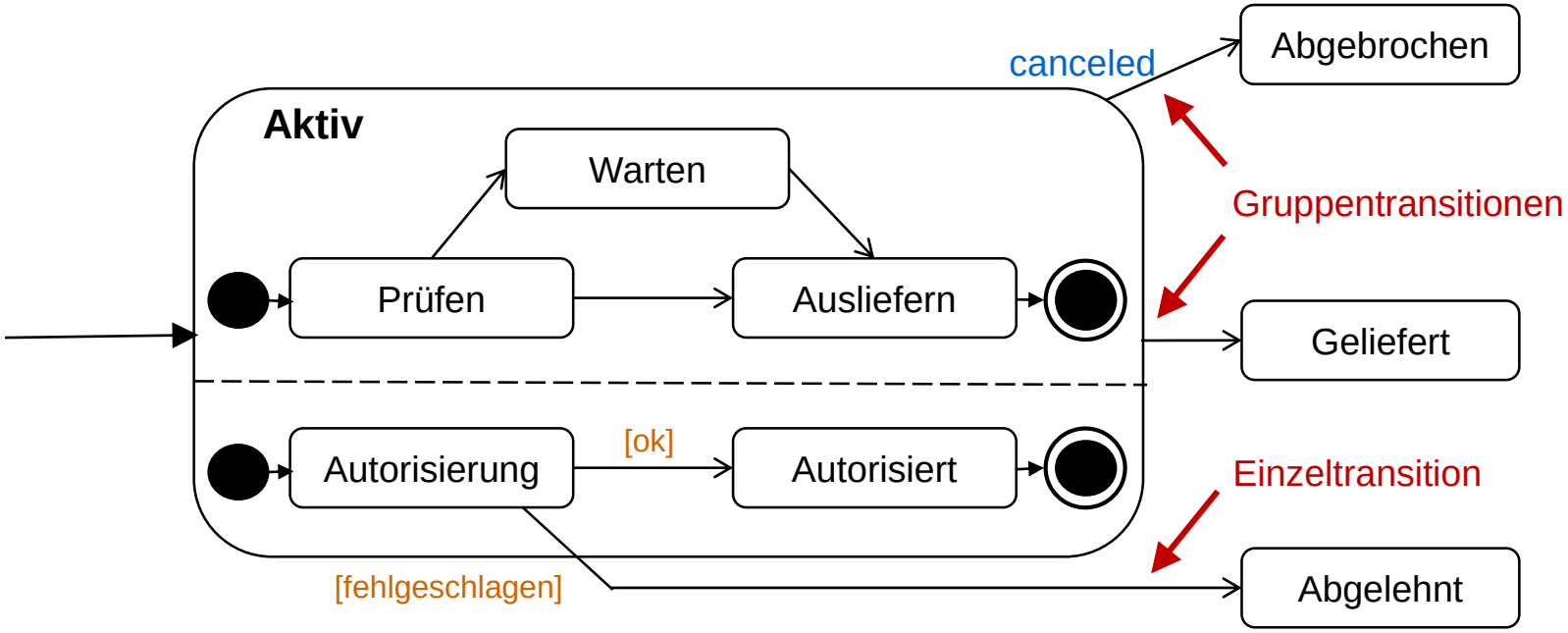


Zustandsdiagramm – Geschachtelte „Bestellungsbearbeitung“ mit Abbruch



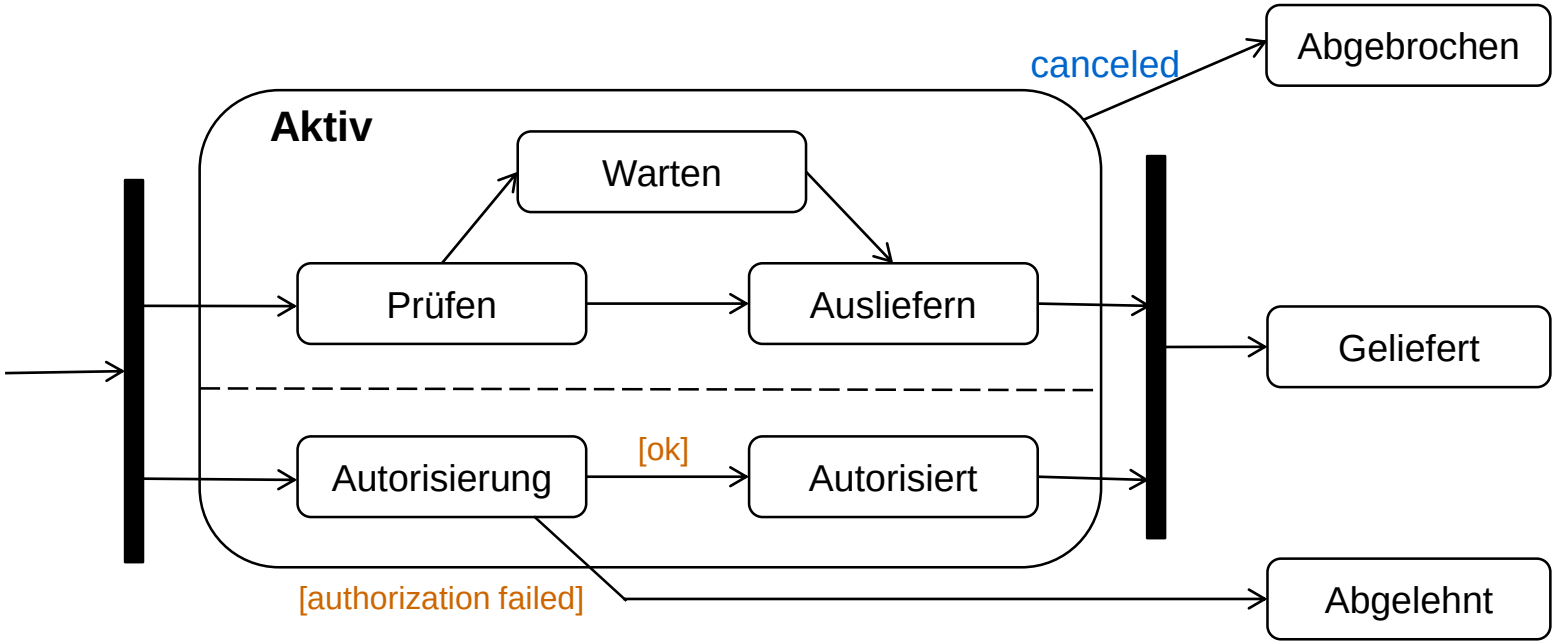
Nebenläufige Zustands-Diagramme: Mit impliziten Übergängen auf ● und von ◉

- Implizite Parallelisierung
  - Implizite Transition von Außen in jeden der parallelen Startzustände
- Implizite Synchronisation
  - Implizite Transition von den parallelen Endzuständen zum nächsten äußeren Zustand
  - … erst wenn alle parallelen Endzustände erreicht sind.



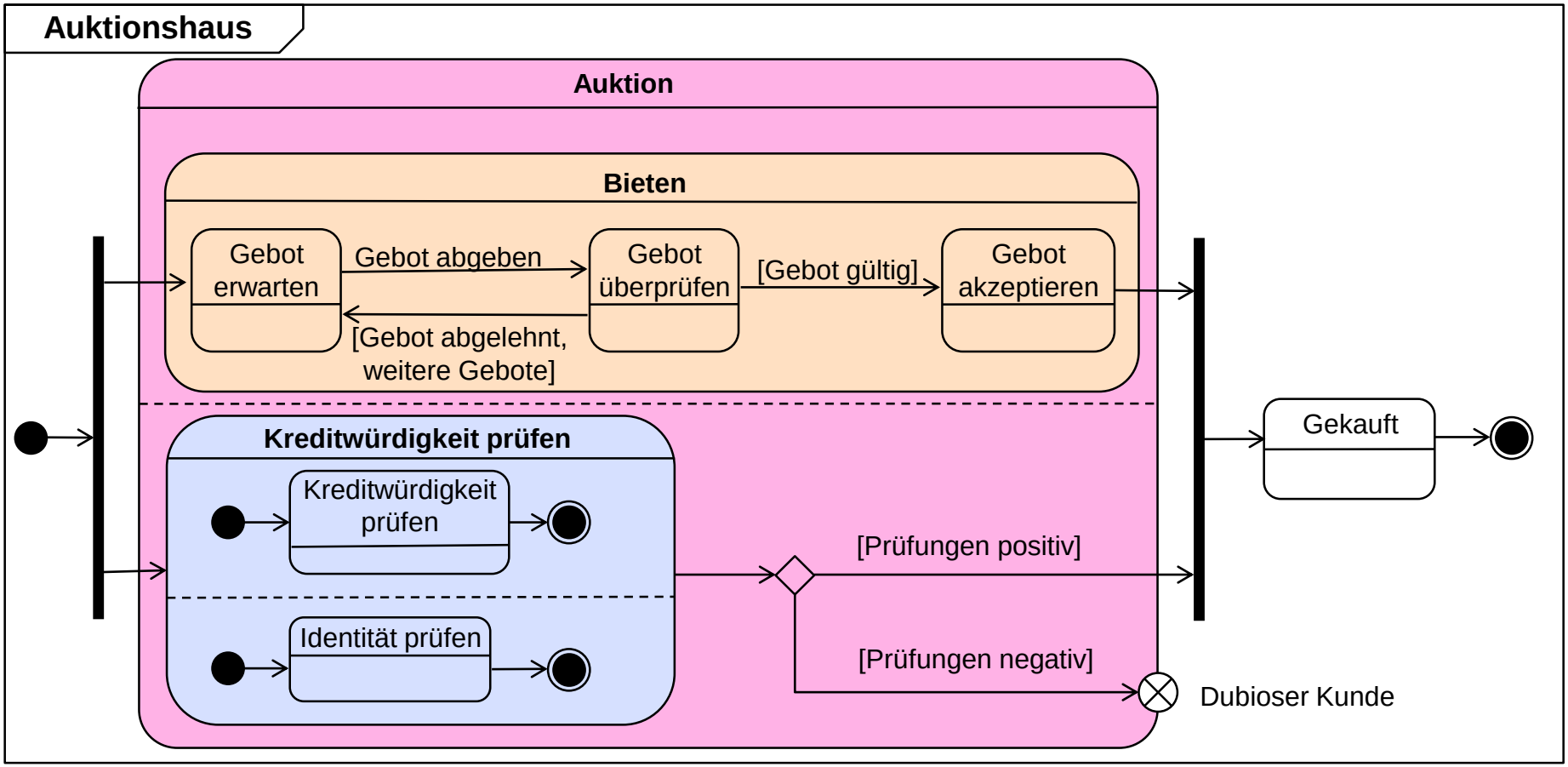
Nebenläufige Zustands-Diagramme: Mit expliziter Parallelisierung / Synchron.

- Explizite Parallelisierung: Quellzustände außerhalb, Zielzustände in verschiedenen parallelen Unterbereichen
  - Zielzustände beliebig (müssen nicht Startzustände sein)
- Explizite Synchronisation: Zielzustände außerhalb, Quellzustände in verschiedenen parallelen Unterbereichen
  - Quellzustände beliebig (müssen nicht Endzustände sein)



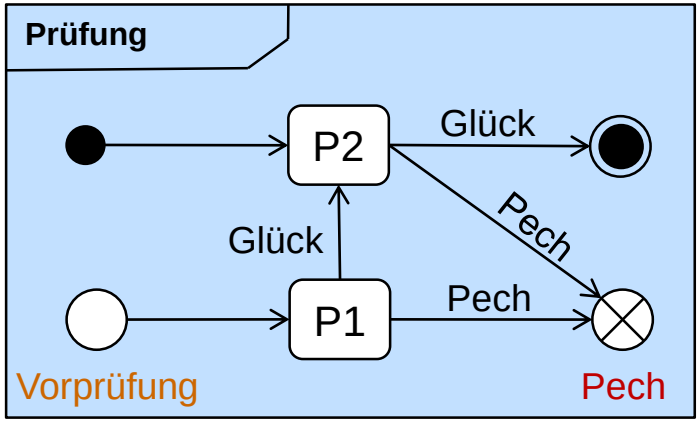
Unterautomaten: Beispiel „Auktion“

- Aus dem Aktivitätsdiagramm bekannte Kontrollflussnotation: Entscheidung und Vereinigung

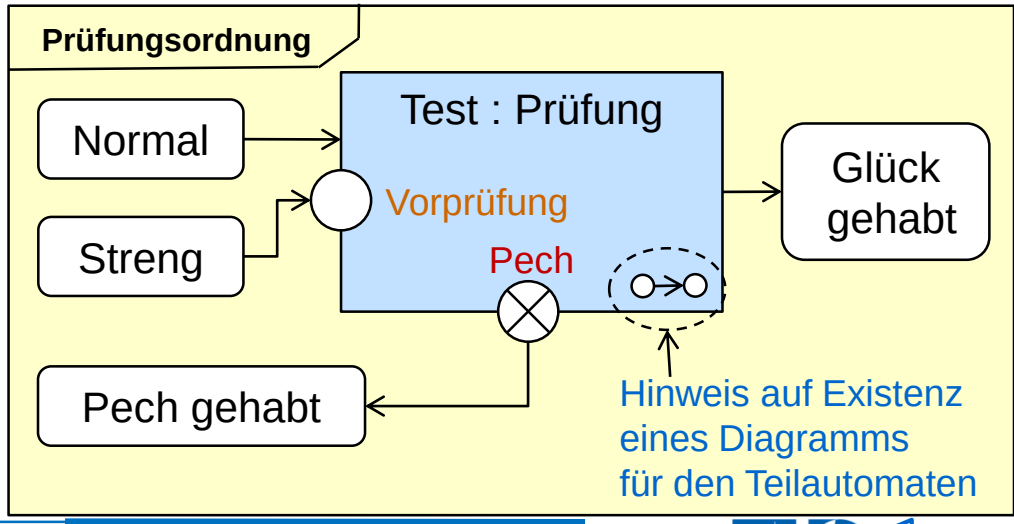


Unterautomaten: Einstiegs- und Austiegspunkte

- Anfangszustand
  - Hierhin findet eine implizite Fortsetzung statt von eingehenden Transitionen des umgebenden Zustands
- Endzustand
  - Von hier findet ein implizite Fortsetzung statt zu ausgehenden Transitionen des umgebenden Zustands, die keine explizite Eventangabe haben



- Einstiegspunkt / Ausstiegspunkt
  - Hierhin / Von hier finden nur Transitionen über explizite Ein- / Ausgangskanten des jeweiligen Punktes statt
    - ⇨ Streng → Vorprüfung → P1
  - Diese Punkte modellieren Abläufe, die vom „Normalfall“ abweichen



Zustand: Ein Zustand ist ein Viertupel

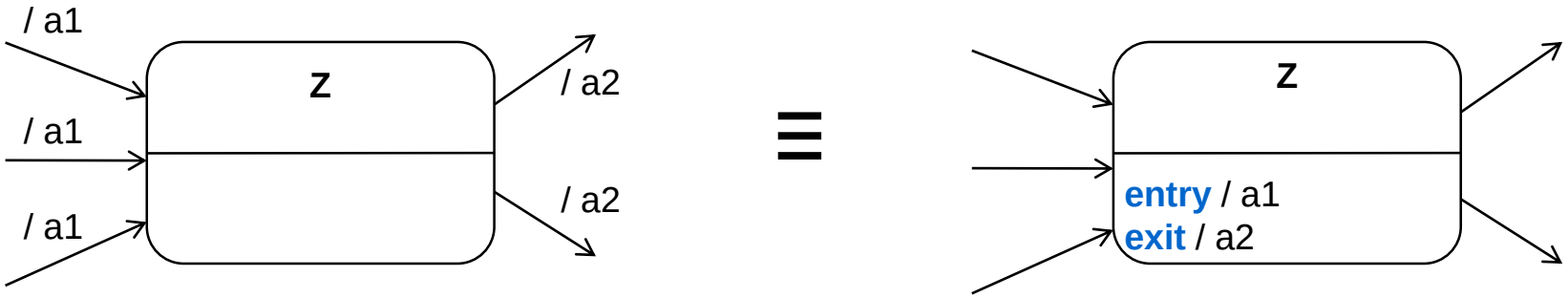
- Name
  - Mehrere unbenannte Zustände gelten als verschieden
- Aktivitäten
  - Was geschieht beim Eintritt in den Zustand (entry),
  - … während des Zustands (do) und
  - … beim Verlassen des Zustands (exit)
- Innere Struktur
  - Geschachtelte Unterzustände und die dazugehörigen Transitionen
- Innere Transitionen
  - Interne Zustandsübergänge, die nicht die entry- und exit-Aktivitäten auslösen

Jede Kategorie ist optional aber mindestens eine muss angegeben sein.
Innere Struktur kann ersetzt werden durch Verweis auf Teilautomat.
Siehe ○→○ auf vorheriger Folie.

Entry- und Exit-Aktivitäten

- Entry-Aktivität
  - Kurzschreibweise für Aktivität die auf jeder eingehenden Kante geschieht
- Exit-Aktivität
  - Kurzschreibweise für Aktivität die auf jeder ausgehenden Kante geschieht

➔ Kapselung und Wartbarkeit
  - Entry und Exit im Zustand statt auf jeder ein- und ausgehenden Kante



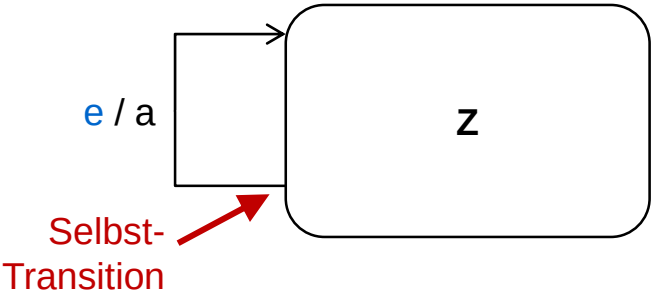
Innere Transitionen

Selbst-Transition

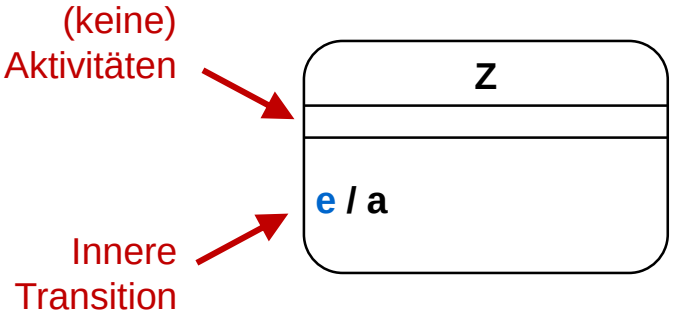
- Externer Übergang in den selben Zustand
- Löst wie jede externe Transition alle Aktivitäten des Zustands aus.

Innere Transition

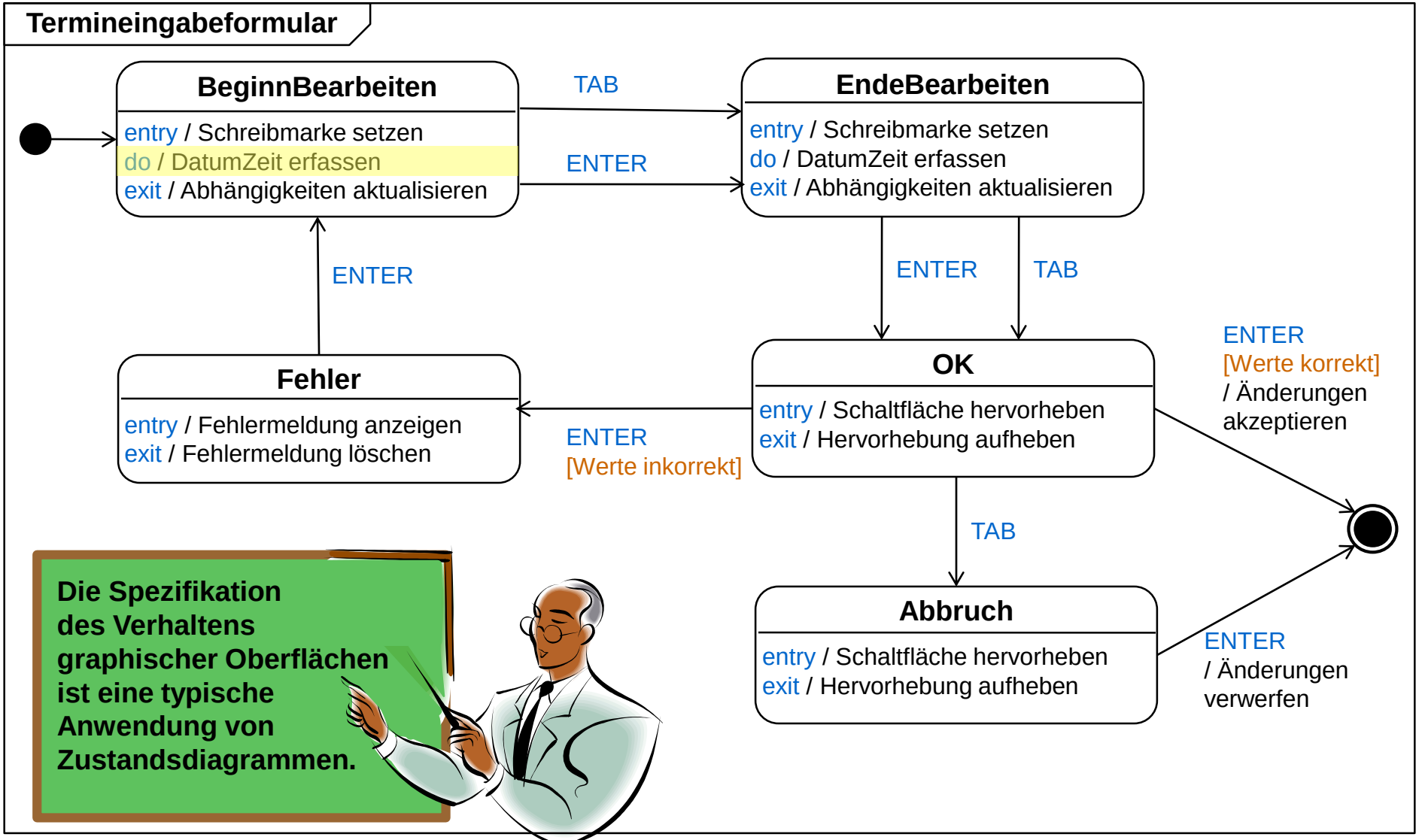
- Interner Übergang in den selben Zustand
- Löst keine entry- und exit-Aktivitäten aus!



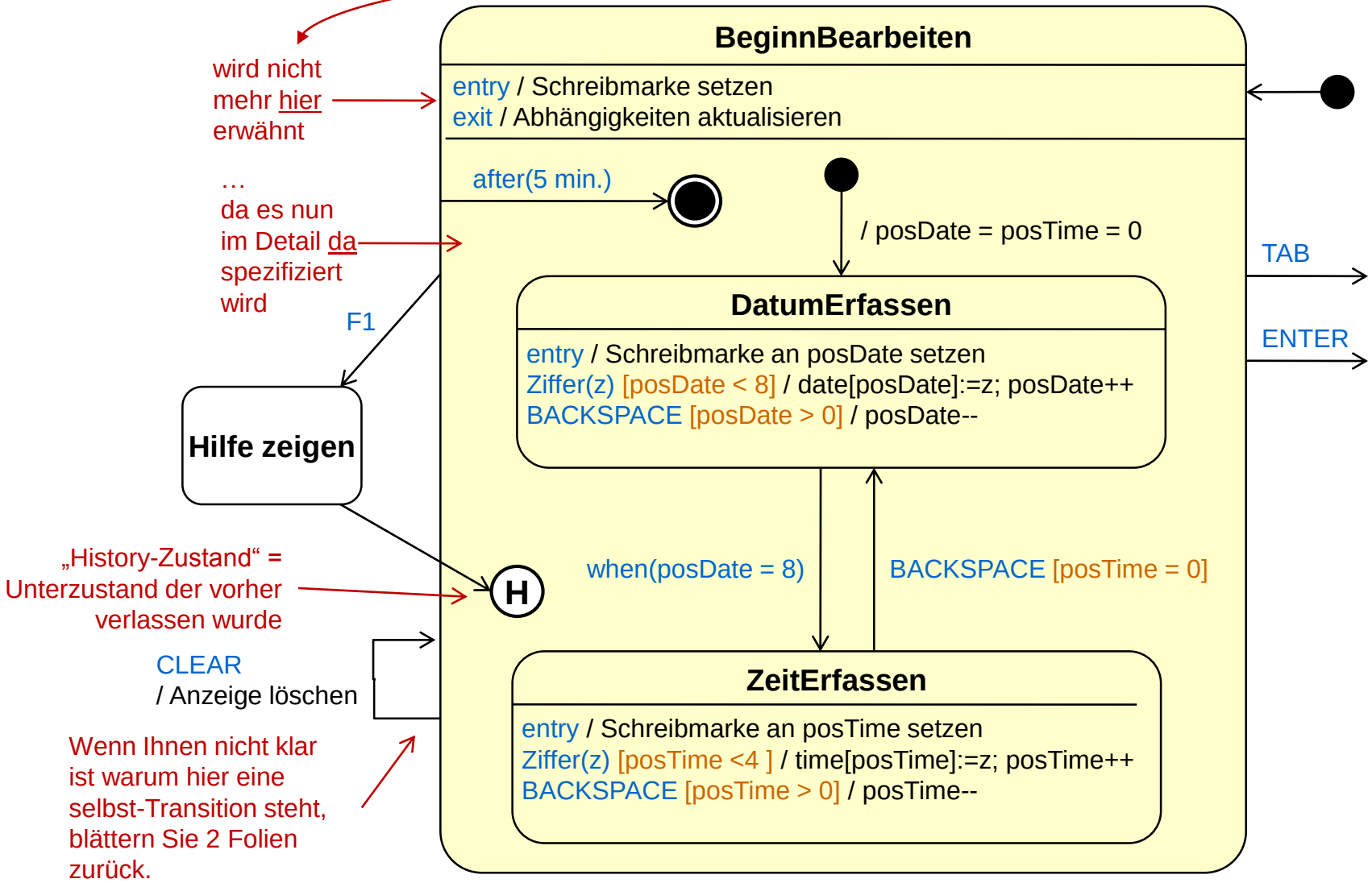
≠



Zustandsautomat mit entry- und exit-Aktivitäten: „Termineingabeformular“

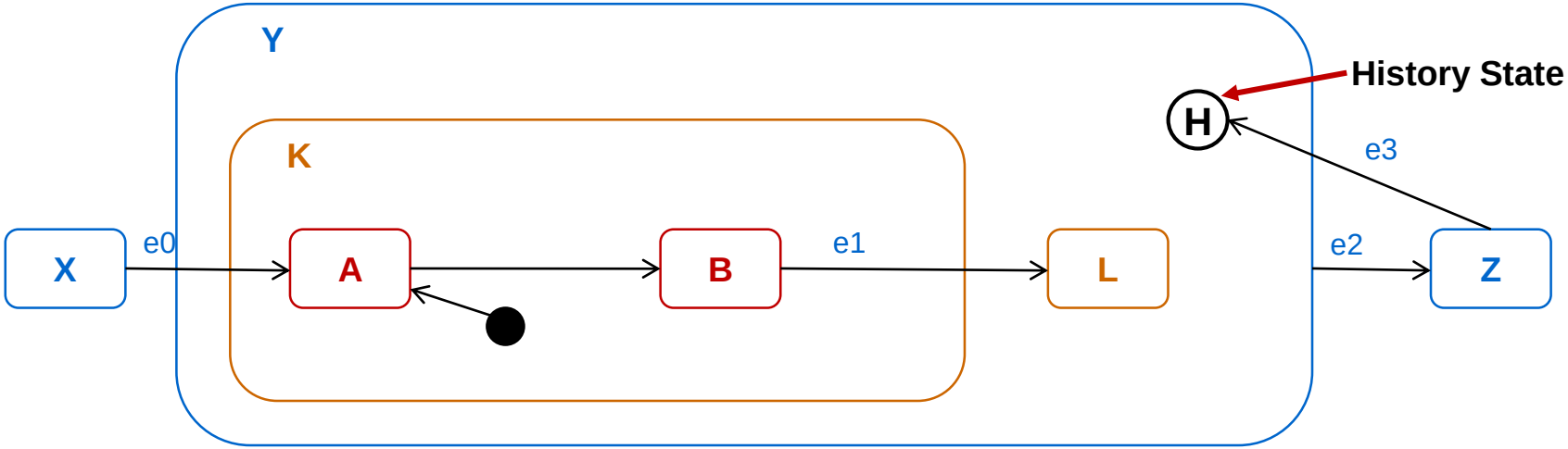


Expansion von „do / DatumZeit erfassen“



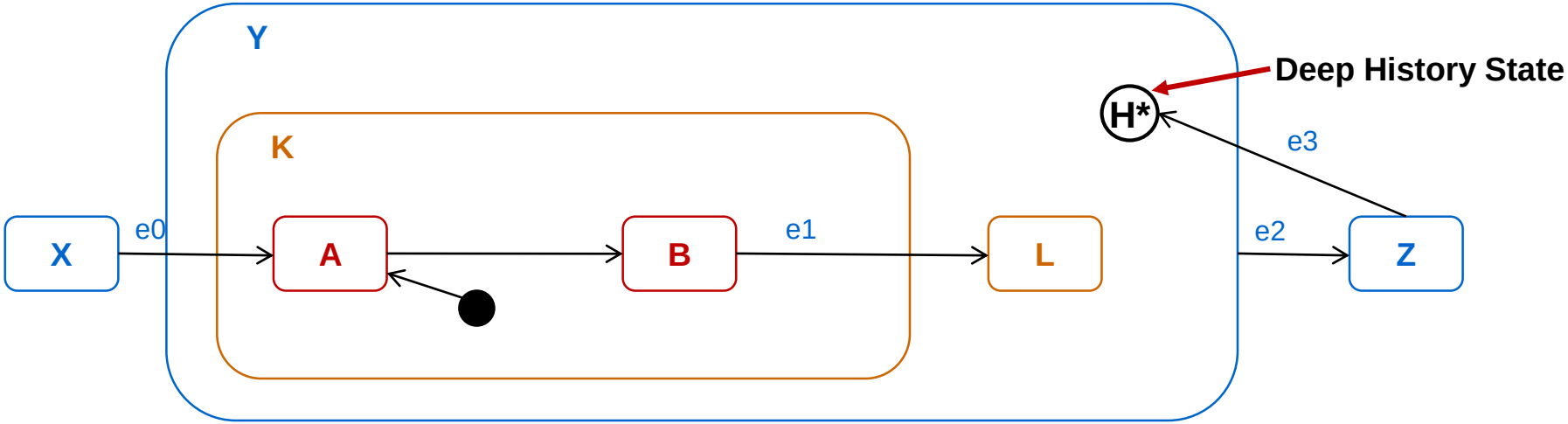
Unmittelbar vorheriger Zustand (‚History State')

- Ⓗ „merkt" sich die Unterbrechungsgeschichte der unmittelbaren Teilzustände
  - Dient dazu in den unmittelbar enthaltenen unterbrochenen Teilzustand zurückzukehren.
  - Im folgenden Beispiel würde ‚e3' zu L oder K zurückkehren.
  - Falls es zu K zurückkehrt, würde es wieder im Startzustand A anfangen
  - Merke: Ⓗ merkt sich nur Teilzustände auf der nächsten Schachtelungs-ebene!



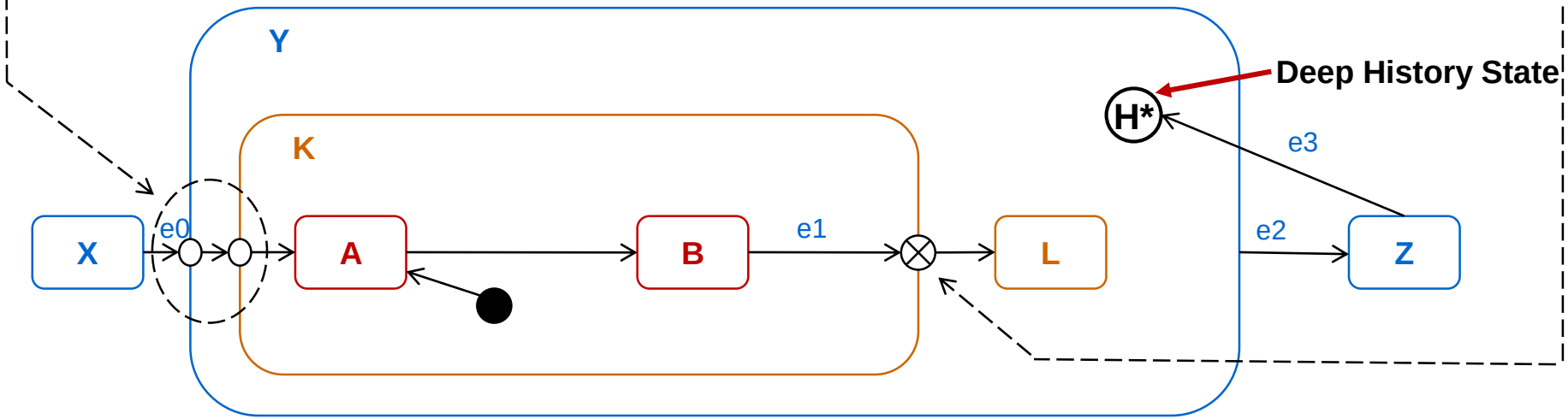
Beliebig tief geschachtelter vorheriger Zustand (‚Deep History State')

- (H*) „merkt" sich die Unterbrechungsgeschichte beliebig tief geschachtelter Teilzustände
  - Um im vorherigen Beipiel gegebenenfalls auch nach B zurückkehren zu können würde man ein (H*) verweden.
  - Referenzen auf geschachtelte Unterzustände mit „Doppelpunkt-Notation": z.B: „Y:K:B"



Nochmal Ein- und Ausstiegspunkte

- Das Diagramm auf der vorherigen Seite hätte man auch mit expliziten Einstiegs- und Ausstiegspunkten modellieren können
- Vorteil: Zustände werden besser gegeneinander gekapselt
  - Äußere Zustände müssen nicht die Interna ihrer inneren Zustände kennen.
  - Z.B. muss „X“ nicht wissen, dass „Y“ „K“ und „K“ „A“ enthält.
- Hier ist das nicht so wichtig, da die Verwendung des „Deep History State“ sowieso wissen über geschachtelte Unterzustände voraussetzt.



Transition = Zustandsübergang

- Notation
  - Spitzer Pfeil mit Beschriftung Ereignis [Bedingung] / Aktion

- Semantik
  - Wenn das Ereignis eintritt und die Bedingung wahr ist, wird die Transition und die Aktion ausgeführt → "die Transition feuert"
  - Die Transition unterbricht evtl. noch laufende Aktionen des Quellzustandes
  - Sind mehrere Transitionen „feuerbereit" wird <u>nichtdeterministisch</u> genau eine ausgewählt.
- Ereignis
  - *Keines angegeben*: Implizit das Ende der Aktivitäten des Quellzustands.
  - *Signal*(*Param*): Empfangenes Signal, evtl. mit Parametern.
  - *Call*(*Param*): Empfangene Nachricht, evtl. mit Parametern.
  - when(*Cond*): Wahr werden einer laufend überprüften Bedingung.
  - after(*TimeIntervall*): Nach Ablauf einer absoluten Zeitspanne (1 Sek) oder eines relativen Zeitintervalls (5 Sekunden seit Verlassen von Zustand Z).
  - *Sonstiges*: Auch recht informelle "Ereignisse" zulässig.

Transition: Semantische Feinheiten

- when(Lager leer) / Aktion
  - Sobald das Lager leer ist passiert Aktion

- Bestellung eingegangen [Lager leer] / Aktion
  - Nur wenn eine Bestellung eingeht und dann das Lager leer ist passiert Aktion

- [Lager leer] / Aktion
  - Wenn bei Ende der Aktivität des Quellzustands das Lager leer ist passiert Aktion

Zustandsdiagramme: Notationsüberblick

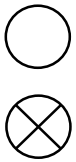
- Zustand

Name des Zustands

- Start- und Endknoten
  - Startzustand
    - Es kann nur einen geben
    - Keine eingehenden, nur ausgehende Transitionen
    - Objekt kann in diesem Zustand <u>nicht</u> verweilen
  - Endzustand
    - Es kann mehrere geben
    - Nur eingehende, keine ausgehenden Transitionen
    - Objekt muss in diesem Zustand verweilen
    - <u>Kann</u> Desktruktion des Objekts bedeuten, <u>muss</u> aber nicht !
  - Terminierungsknoten
    - Objekt, dessen Verhalten modelliert wird, hört auf zu existieren
    - Nicht identisch mit Endzustandsknoten

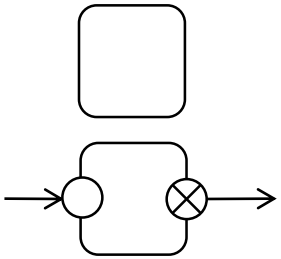
Zustandsdiagramme: Notationsüberblick

- Verbindungspunkt
  - Einstiegspunkt
  - Ausstiegspunkt

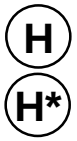


- Verbindungsstelle

- Zustandsautomat
  - Allgemein (Automat oder Subautomat)
  - Subautomat mit Ein- uns Ausstiegspunkt

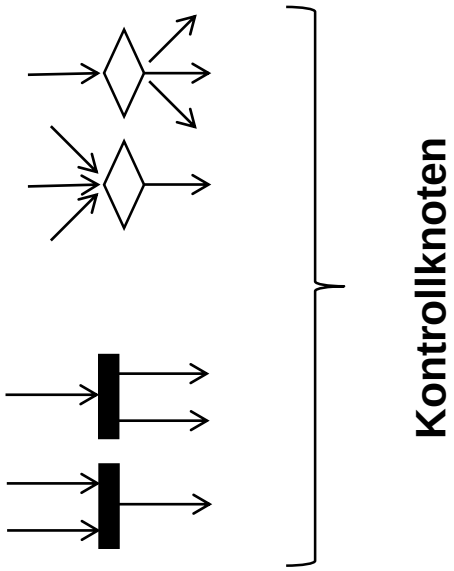


- History
  - History-Zustand
  - Tiefer History-Zustand



Zustandsdiagramme: Notationsüberblick

- Alternative Abläufe
  - Entscheidungsknoten
  - Vereinigungsknoten
- Nebenläufige Abläufe
  - Parallelisierungsknoten
  - Synchronisierungsknoten



3.5 Zusammenfassung und Ausblick

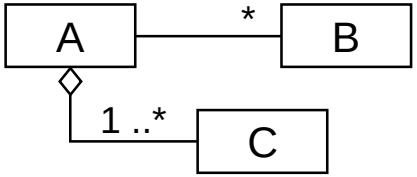
Kurzrückblick der besprochenen Notationen

Kurzübersicht der ausstehenden Notationen

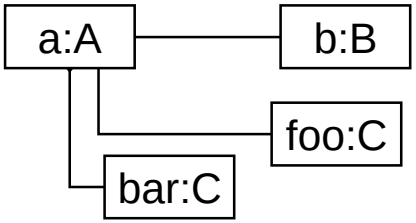
Überblick kommender Kapitel in Bezug auf UML

UML Kurzübersicht: Strukturdiagramme

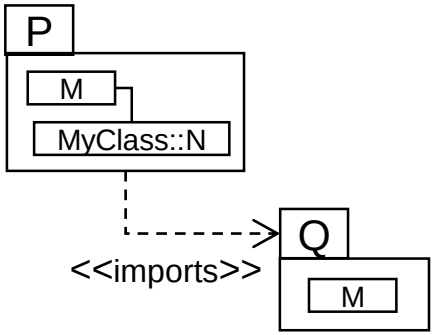
- Klassendiagramm / *Class diagram*
  - Beschreibt die statische Struktur des Systems: Typen, Attribute, Operationen und Assoziationen.



- Objektdiagramm / *Object diagram*
  - Beschreibt Objekte und deren Beziehung

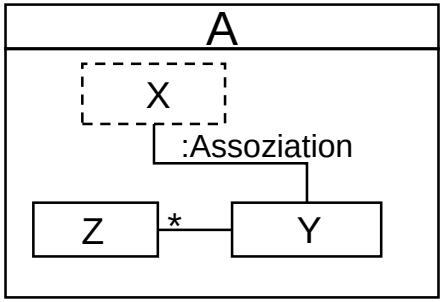


- Paket Diagramm / *Package diagram*
  - Beschreibt die Komposition von Paketen
  - Kann Klassendiagramme enthalten

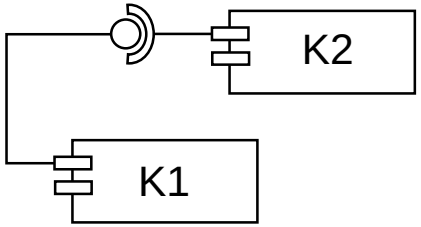


UML Kurzübersicht: Strukturdiagramme 2

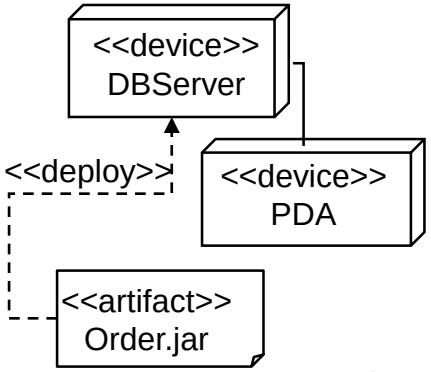
- Kompositionsstrukturdiagramm / *Composite structure diagram*
  - Hierarchische Dekomposition verschiedener Systembestandteile (UML Modell-Elemente)
  - Darstellung der internen Struktur von Klassen, Komponenten, Knoten und Kollaborationen



- Komponentendiagramm / *Component diagram*
  - Zeigt die Implementierungsabhängigkeiten von Komponenten untereinander

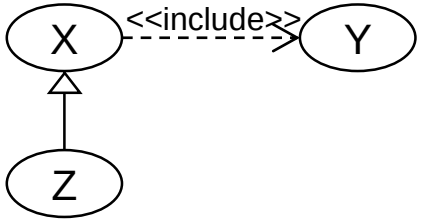


- Verteilungsdiagramm / *Deployment diagram*
  - Zeigt die eingesetzte Hardwaretopologie
  - …und das Laufzeitsystem

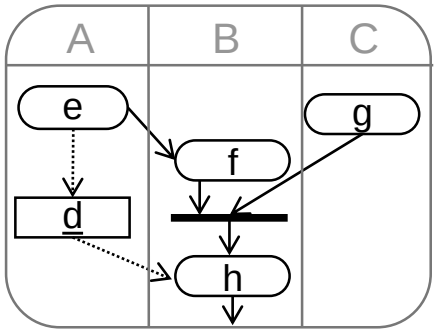


UML Kurzübersicht: Verhaltensdiagramme

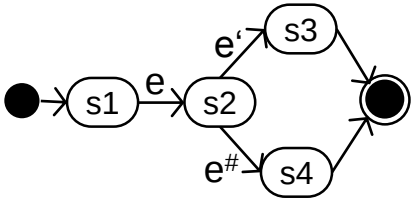
- Anwendungsfalldiagram / *Use case diagram*
  - Beschreibt das funktionelle Verhalten des Systems aus Sicht des Benutzers.



- Aktivitätsdiagramm / *Activity diagram*
  - Modelliert das dynamische Verhalten eines Systems, insbesondere den Workflow, z.B. durch ein Flussdiagramm



- Zustandsdiagramm / *State diagram*
  - Beschreiben das dynamische Verhalten eines individuellen Objektes als endlichen Automat



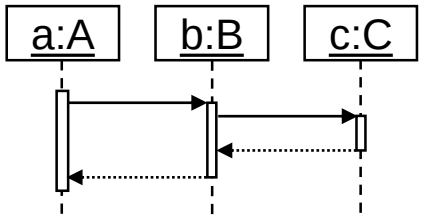
- Interaktionsdiagramm / Interaction diagramm
  - Beschreibt das Zusammenspiel verschiedener Objekte

Vier verschiedene Typen

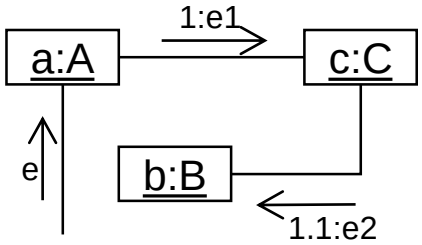
→ s. nächste Folie

UML Kurzübersicht: Verhaltensdiagramme: Interaktion

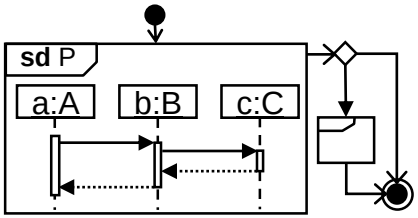
- Sequenzdiagramm / *Sequence diagram*
  - Beschreibt das dynamische Verhalten zwischen Akteuren und System aus *zeitlicher* Sicht



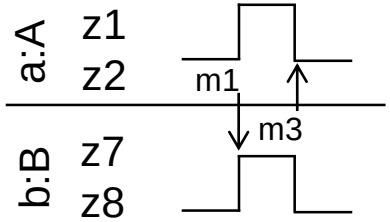
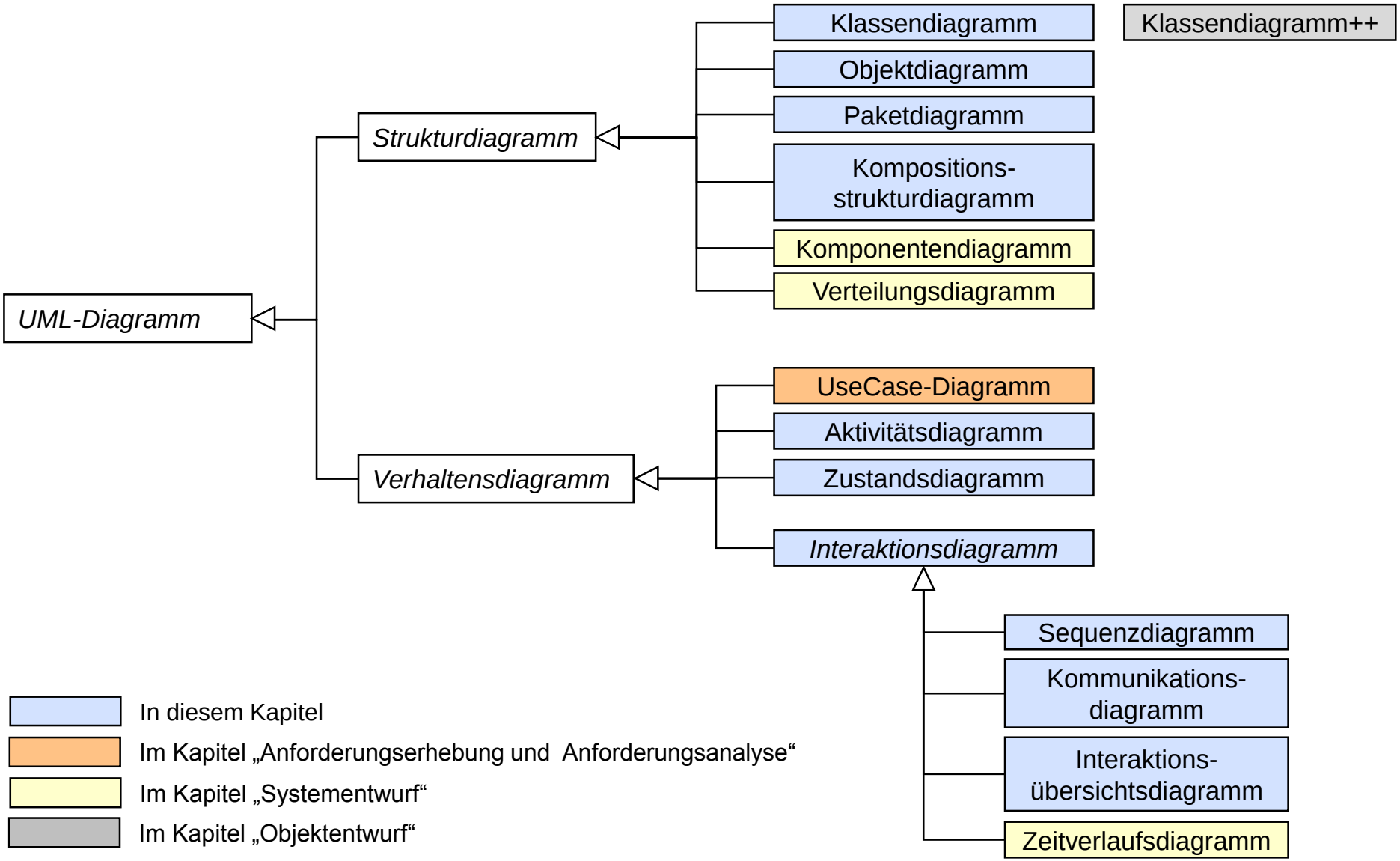
- Kommunikationsdiagram / *Communication diagram*
  - Beschreibt das dynamische Verhalten zwischen Akteuren und System aus *struktureller* Sicht



- Interaktionsübersichtdiagramm / *Interaction overview diagram*
  - Zeigt Interaktionsdiagramme im Kontrollfluss



- Zeitverlaufsdiagramm / *Timing diagram*
  - Darstellung von Zustandsänderungen von Interaktionspartnern
  - Sinvoll bei der Modellierung von zeitkritischem Verhalten, z.B. Echzeitsysteme



UML Diagrammtypen: Was wird wo erläutert?