universitätbonn ROOTS

Ferientutorien Softwaretechnologie 2010

von Eva Stöwe und Jan Nonnen

12.03.2010

Disclaimer

- Die Folien
  - stellen lediglich die Basis der jeweiligen Themen dar
  - erheben keinen Anspruch auf Vollständigkeit
- An mehreren Stellen wurden Sachen vereinfacht oder weggelassen.
- Wir raten dringend dazu, auch andere Quellen zur Klausurvorbereitung zu nutzen.

Klausurrelevant sind die Vorlesungsfolien.

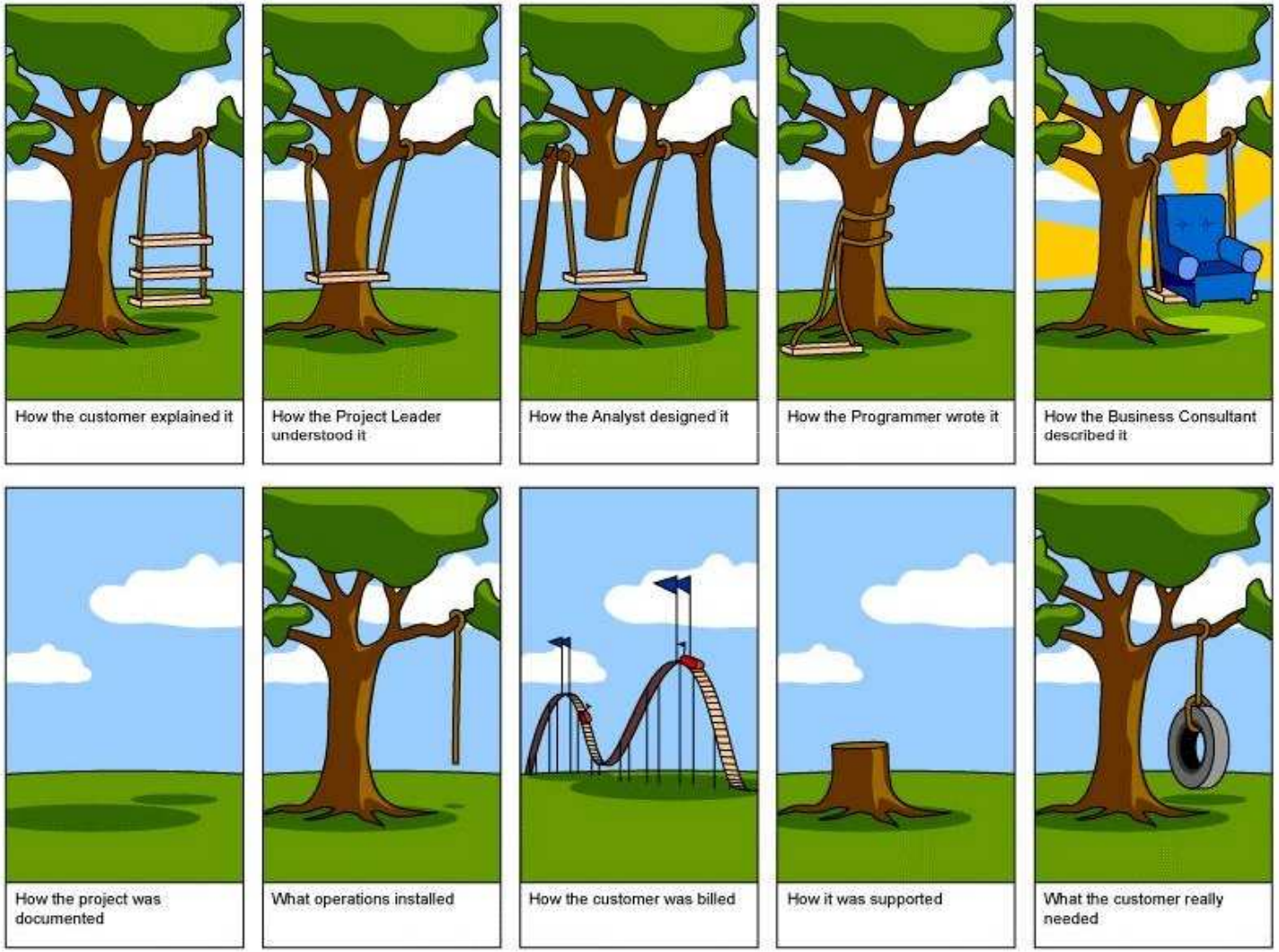
universitätbonn ROOTS

UML-Diagramme

von Eva Stöwe und Jan Nonnen

10.02.2010

UML – Wozu, weshalb, warum?



from: www.jacobsen.no

UML – Wozu, weshalb, warum?

- „Modeling“
  - Idee: Über Software unterhalten
  - Code selbst viel zu komplex, oft noch nicht vorhanden
  - ➔ Modell für die Software (wie Architekten auch)
- „Language“
  - alle Beteiligten müssen Symbole verstehen
  - Programmiersprache meist nicht richtige Ausdrucksstärke
  - ➔ „Meta“-Sprache wird gebraucht
- „Unified“
  - auch ein Jahr später noch Bedeutung verstehen
  - nicht für jedes Projekt „Rad neu erfinden“, nur einmal Sprache lernen
  - ➔ Vereinheitlichte Syntax für Vielzahl von Projekten

universitätbonn ROOTS

Allgemeine Modell-Elemente

Allgemein ▸ Bezeichner

- Typen werden durch ihren Namen repräsentiert
  - Feuerwehrmann
- Objekte werden durch „Rolle : Typ“ und Unterstreichung gekennzeichnet
  - Grisu: Feuerwehrmann
- Operationen: „OperationsName (ParameterName : Typ,…) : Typ“
  - getPrice (zone : Zone) : Price
- Attribute werden durch „AttributName: AttributTyp“ beschrieben
  - name : String

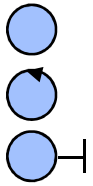
Allgemein ▸ Bedingungen

- Conditions
  - Bedingungen bei Entscheidungen
  - diese sind nötig für das Element
  - werden in eckigen Klammern dargestellt [Alternative1]
- Kommen nur bei Entscheidungs-Elementen vor
  - Aktivitätsdiagramm: Entscheidung, Sequenzdiagramm: Fragmente, …

- Constraints
  - sind zusätzliche einschränkende Bedingungen
  - stehen in geschweiften Klammern {Einschränkung}
  - können umgangssprachlich formuliert sein, oder als Code / Formel
  - können sich auf beliebige Sachverhalte beziehen, nicht nur auf Elemente, die im Diagramm vorkommen
- Können in jedem Diagramm fast überall vorkommen

Allgemein ▸ Stereotypen

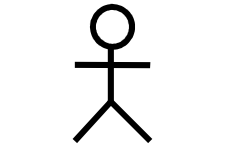
- Stereotyp
  - Hinweis auf semantische Kategorie für die es keine spezifische Notation gibt
  - Ansatzpunkt für anwendungsspezifische Erweiterungen
- Notation:
  - <<stereotyp>> oder graphisches Symbol, z.B.
    - ⇨ <<entity>>
    - ⇨ <<controller>>
    - ⇨ <<boundary>>



    - ⇨ <<interface>>
    - ⇨ <<include>>
    - ⇨ <<selbst definierte Kategorie>>
- Kommen in den meisten Diagrammen vor, reguläres Sprachelement

Allgemein ▸ Akteure

- Stellen aktive Elemente außerhalb des betrachteten Systems dar.
  - Menschen
  - externe Programme
- Können Namen / Rolle haben
  - wird unter dem Symbol angegeben

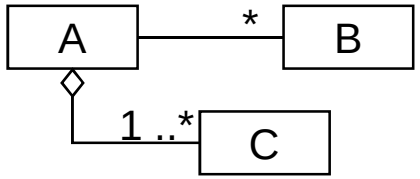


- Nur Relevant in Diagrammen bei denen das Zusammenspiel zwischen dem System und der Außenwelt eine Rolle spielt
  - UseCase-Diagramme
  - Sequenz- und Kommunikationsdiagramme

Aktive Klassen / Objekte

- Aktive Klasse / Objekte: Jede Instanz ist ein eigener Thread.
- Notation
  - Klasse/Objekt mit doppeltem vertikalem Rand oder {active}

- Kann vorkommen in allen objektorientierten Diagrammen
  - Klassen- und Objektdiagrammen
  - Sequenz- und Kommunikationsdiagrammen



Klassendiagramme

Klassendiagramm ▸ Aufgaben

- repräsentieren die **statische Struktur** auf Typebene
- beschreiben Elemente eines objektorientierten Modells
  - Typen = Klassen, Interfaces
    - ⇨ Namen, Attribute und Operationen
  - Beziehungen zwischen Typen
    - ⇨ Assoziationen
    - ⇨ Generalisierungen, Realisierungen
    - ⇨ sonstige Abhängigkeiten
  - Pakete
    - ⇨ Gruppe von Typen, die thematisch zusammengehören

- Klassendiagramme können in unterschiedlichen Detailierungsgraden vorkommen

Klassendiagramm ▸ Verwendung im Entwicklungsprozess

- erzeugt in der Anforderungsanalyse
  - Modellierung der Konzepte der Anwendungsdomäne
- angepasst im Systemdesign
  - Modellierung von Teilsystemen & Diensten
- verfeinert beim Objektdesign
  - Modellierung von Klassen und ihren Beziehungen

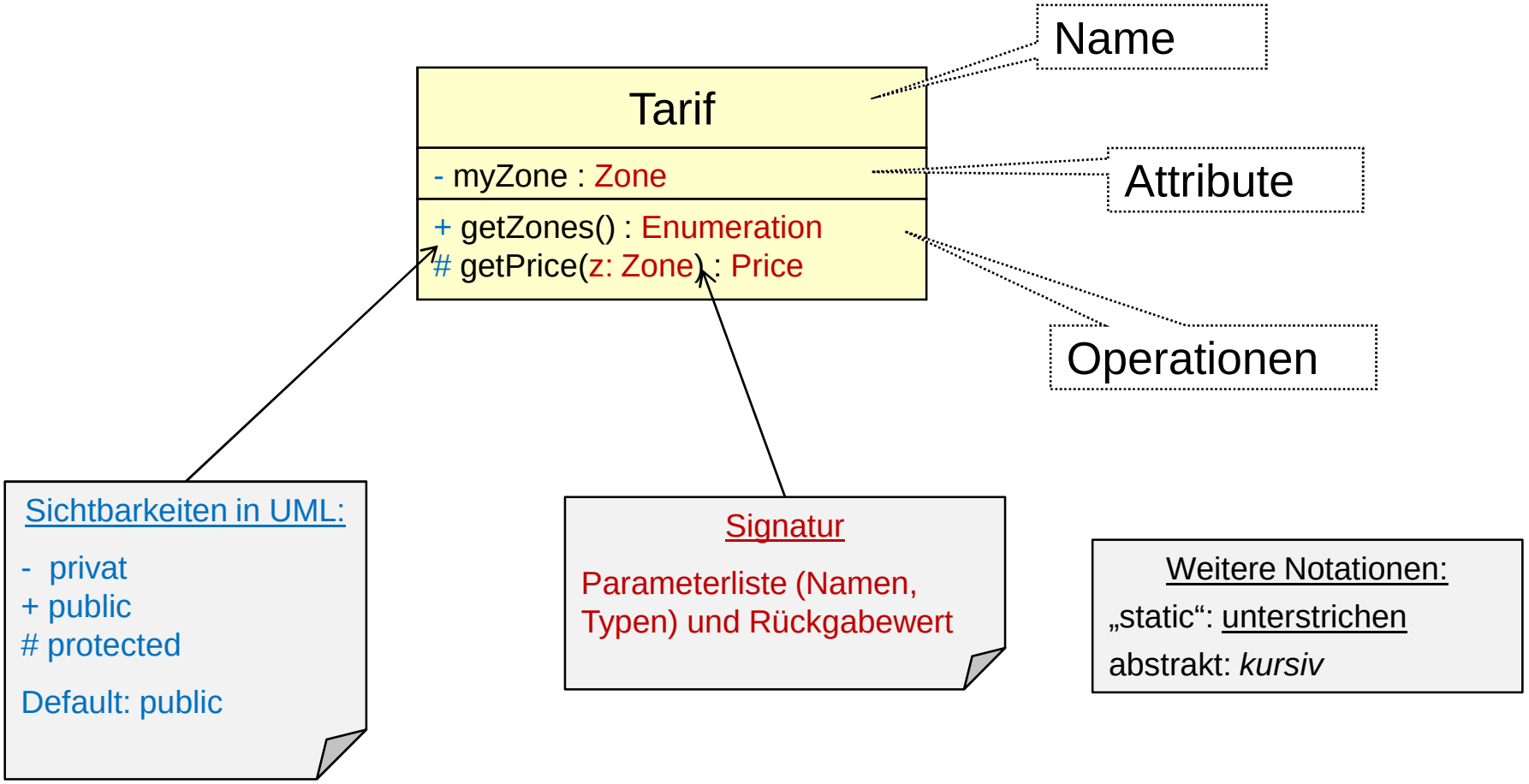
- in allen späteren Entwicklungsphasen werden sie als Referenz verwendet

Klassendiagramm ▸ Typen

- Ein **Typ** repräsentiert ein **Konzept.**
- Ein Typ kapselt
  - Zustand **„Attribute“**
    - ⇨ In Java: Felder
    - ⇨ Jedes Attribut hat eine **Sichtbarkeit**, einen **Namen** und einen **Typ**
  - Verhalten **„Operationen“**
    - ⇨ In Java: Methoden
    - ⇨ jede Operation hat eine **Sichtbarkeit**, einen **Namen** und eine **Signatur**
- Es ist möglich Details (Attribute, Operatoren, Signaturen) zu verstecken, die im aktuellen Modellierungsprozess nicht relevant sind
- Der **Typname** ist die einzige unverzichtbare Information!

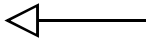
- **Interfaces** werden mit <<interface>> markiert
- **Abstrakte Klassen** durch kursive Schrift oder {abstract}

Klassendiagramm ▸ Beispielklasse

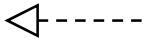


Klassendiagramm ▸ Beziehungen

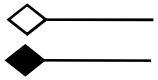
- Spezialisierung
  - Entweder zwischen zwei Klassen oder zwei Interfaces

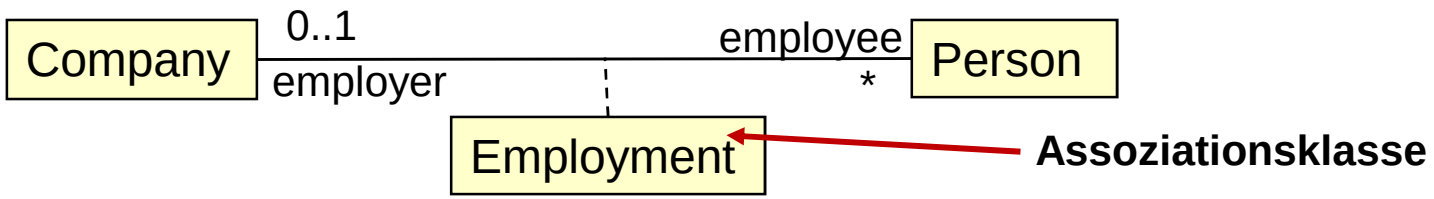


- Implementierung
  - Zwischen einer Klasse und einem Interface

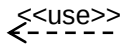


- Assoziationen
  - Beliebig zwischen Klassen und Interfaces
  - haben Kardinalität (default 1) und gegebenenfalls Rollenbezeichner
  - Spezialfälle: Aggregation, Komposition
  - Besonderheit: Assoziationsklasse, [n-äre-Assoziationen]



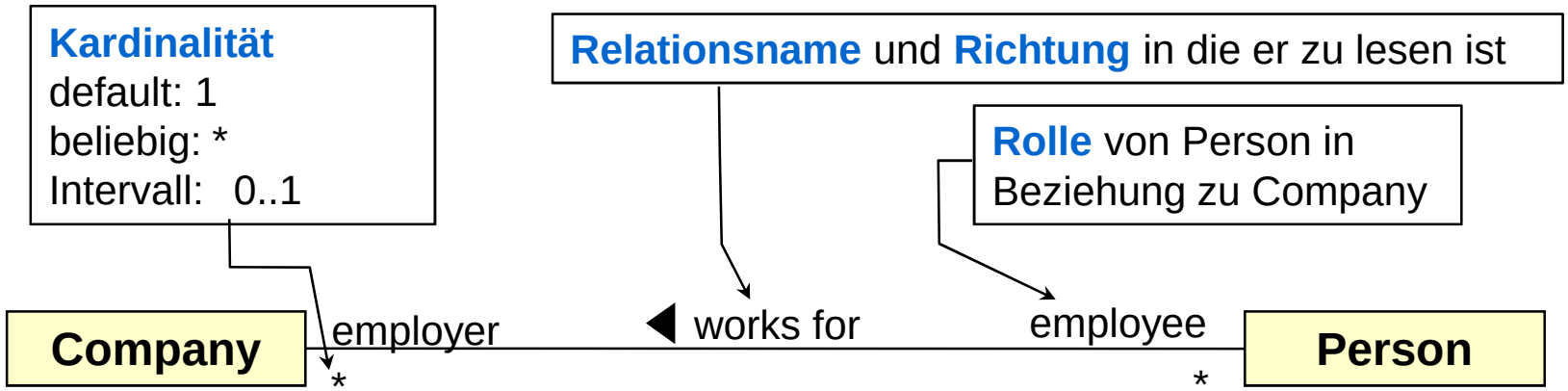


- Sonstige Beziehungen (z.B.: Use)
  - da wo gebraucht



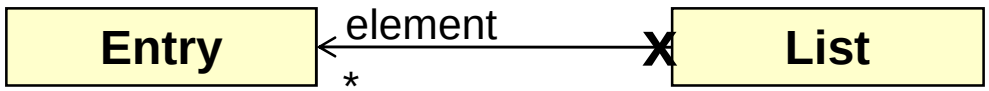
Klassendiagramm ▸ Assoziationen

- Assoziationen definieren Beziehungen zwischen **Instanzen** von Klassen.

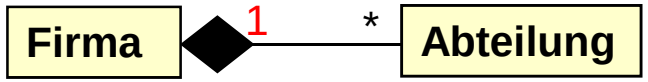
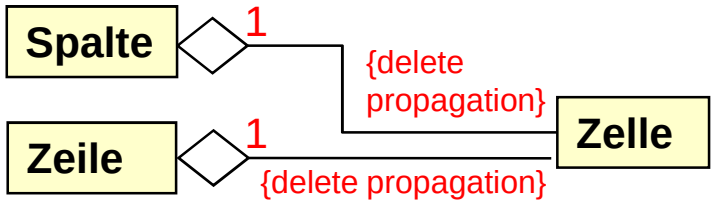
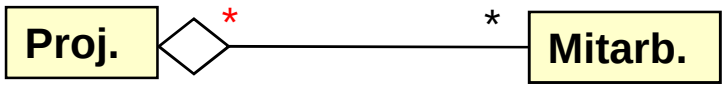


- Die Kardinalität an Zielobjekten: Wie viele können von einem Quellobjekt mit der Relation referenziert werden?

- gerichtete Assoziation:
  - nur eine Seite weiß von der anderen Seite und der Relation



Klassendiagramm ▸ 4 Aggregationsarten

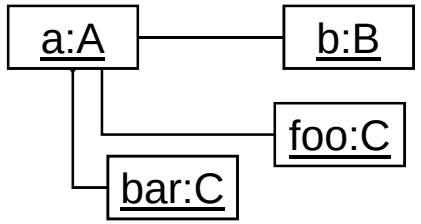
	exklusiv (nur in einem Ganzen)	nicht exklusiv (in mehreren Ganzen)
existenz-abhängig (wird mitgelöscht)		
existenz-unabhängig (besteht weiter)		

Klassendiagramm ▸ Nicht Aufgaben

- Klassendiagramme stellen nicht dar:
  - Zeitliche Abläufe
  - tatsächliche Abhängigkeiten und Kardinalitäten zur Laufzeit
  - System auf dem das Programm läuft
  - Tatsächliche dynamische Typen zur Laufzeit
  - Externe Dinge (Akteure, sonstige Programme)

Klassendiagramm ▸ Typische Fehler

- Komposition zu mehreren Objekten (gleichen oder unterschiedlichen Typs)
- Kardinalität an falschem Ende der Assoziation
- Signaturen der Attribute / Operationen falsch
- Spezialisierung / Implementierung verwechselt
- ein und die selbe Beziehung mit Attribut und Assoziation ausgedrückt



Objektdiagramme

Objektdiagramm ▸ Aufgaben

- spezifizieren einen theoretischen Zustand des Systems
- beschreiben **Objekte** und ihre **Beziehungen**

- Objekte werden durch Rechtecke dargestellt, aber:
  - **Name** einer Instanz ist unterstrichen und besteht aus:
    - ⇨ Namen für die **Rolle** dieser Instanz
    - ⇨ **Doppelpunkt** gefolgt von
    - ⇨ (dynamischem) **Typ** der Instanz
  - **Attribute** werden mit ihren Werten angegeben
- Die Rolle im Namen ist optional, der Doppelpunkt nicht!

Sommertarif : Tarif
prices = { {'1', 1.80}, {'2', 2.40} }

- Die einzige Beziehung ist der **Link**
  - bei Bedarf gerichtet, Komposition oder Aggregation
    - ⇨ nur wenn darauf Fokus liegt dies modellieren
  - immer zwischen genau zwei Objekten

Objektdiagramm ▸ Verwendung im Entwicklungsprozess

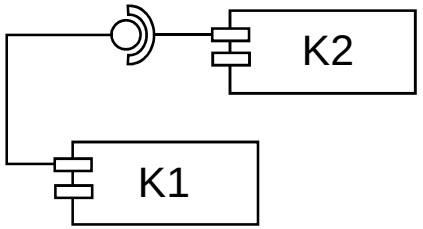
- Objektdiagramme können parallel zu Klassendiagrammen erstellt und verwendet werden
  - zur Veranschaulichung der Instanziierungs-Beziehungen
  - Ergänzung und Präzisierung

Objektdiagramm ▸ Nicht Aufgaben

- Es zeigt keinen zeitlichen Verlauf, es ist lediglich eine Momentaufnahme des Systemzustands
- Jedes Objekt ist eine Instanz, sie hat einen klar definierten (dynamischen) Typ, daher gibt es keine Vererbung
- Links sind immer zwischen zwei Objekten, es gibt keine Multiplizitäten

- Typische Probleme:
  - Konsistenz mit anderen Diagrammen (nur Klausur / Übung)

universitätbonn ROOTS



Komponentendiagramm

Komponentendiagramm ▸ Aufgaben

- zeigt Implementierungs-Abhängigkeiten von Komponenten untereinander

- Komponenten sind
  - gekapselte Teile eines Systems mit
  - nach außen wohldefinierten Schnittstellen

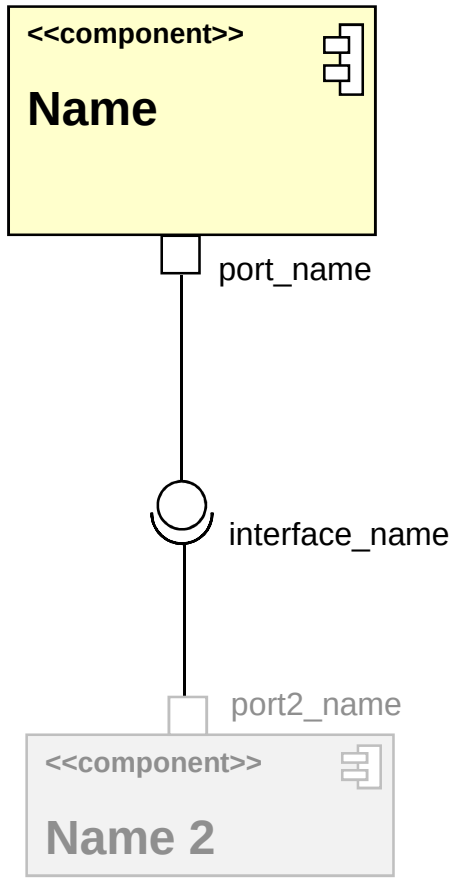
- Gekapselt werden beliebig komplexe Teilstrukturen
  - Klassen, Objekte, Beziehungen oder ganze Verbünde von Teilkomponenten (→ hierarchische Komposition)
  - Quellcode, Laufzeitbibliotheken, ausführbare Dateien, …

Komponentendiagramm ▸ Verwendung im Entwicklungsprozess

- Komponentendiagramme werden im Systementwurf erzeugt und gegebenenfalls verfeinert
- Mit ihrer Hilfe können einzelne Komponenten „unabhängig“ von den restlichen entwickelt werden

Komponentendiagramm ▸ Elemente

- Komponente („component“)
- Port
- angebotenes Interface
- benötigtes Interface

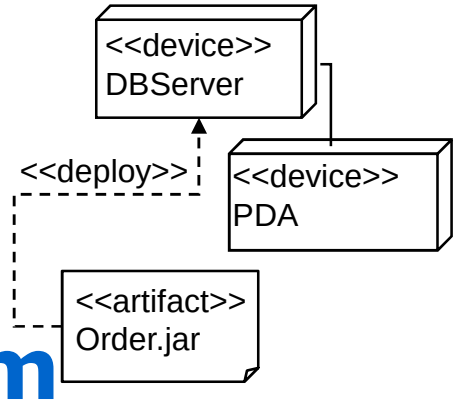


Komponentendiagramm ▸ Nicht Aufgaben

- Komponentendiagramme kennen nicht:
  - Konkrete innere Struktur der Komponenten
  - Zeitliche Abläufe
  - Hardware, Programmiersprachen, sonstige Techniken mit deren Hilfe die Komponenten realisiert werden
  - Dokumente, Daten, Datenbanken
  - Akteure, Programme außerhalb des Systems

Komponentendiagramm ▸ Typische Fehler

- Akteure
- Versuch konkret DBs etc. darzustellen
- Vertauschen von Interfaces und Komponenten
- Interfaces die unangesprochen im Raum hängen (ist zwar möglich aber entspricht selten der Intention)



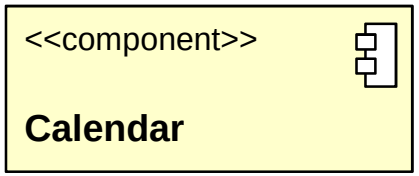
Verteilungsdiagramm

Verteilungsdiagramm ▸ Aufgaben

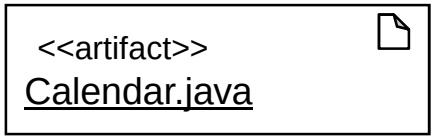
- Zeigt
  - eingesetzte Hardwaretopologie
  - Laufzeitsystem
- Spezifikation der
  - Hardware-Software Zuordnung
  - Subsystemdekomposition
  - Verteilung im Netzwerk
  - Einsatz zur Laufzeit

Verteilungsdiagramm ▸ Elemente

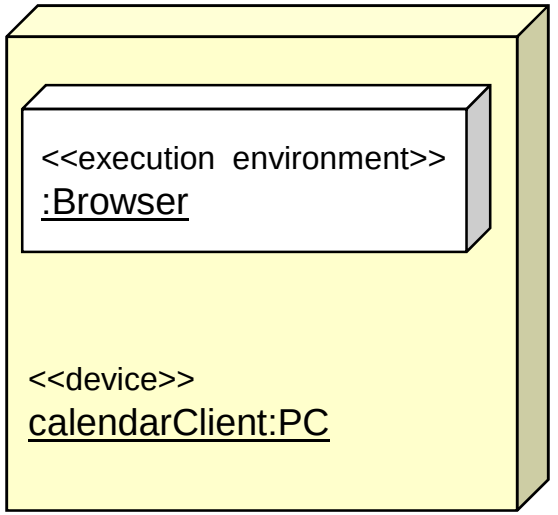
- Komponente
  - Logische Einheit mit expliziten Abhängigkeiten
  - wie im Komponentendiagramm



- Artefakt
  - Physische Realisierung einer Komponente
  - z.B. Quellcode



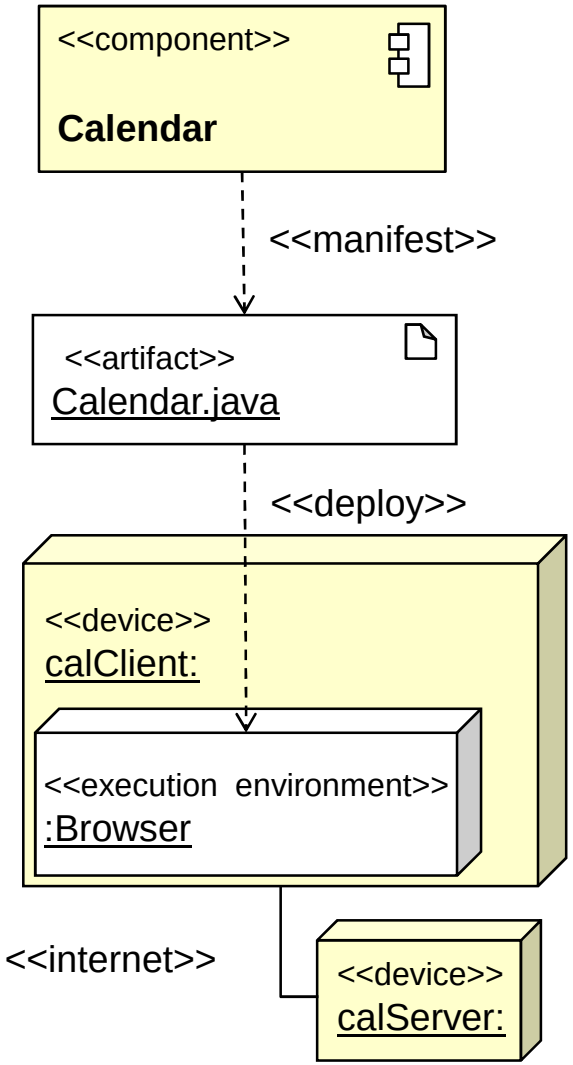
- Laufzeitumgebung
  - Softwaresystem in dem Artefakte zum Einsatz kommen
  - z.B. Java Virtual Machine, Applikationsserver, …



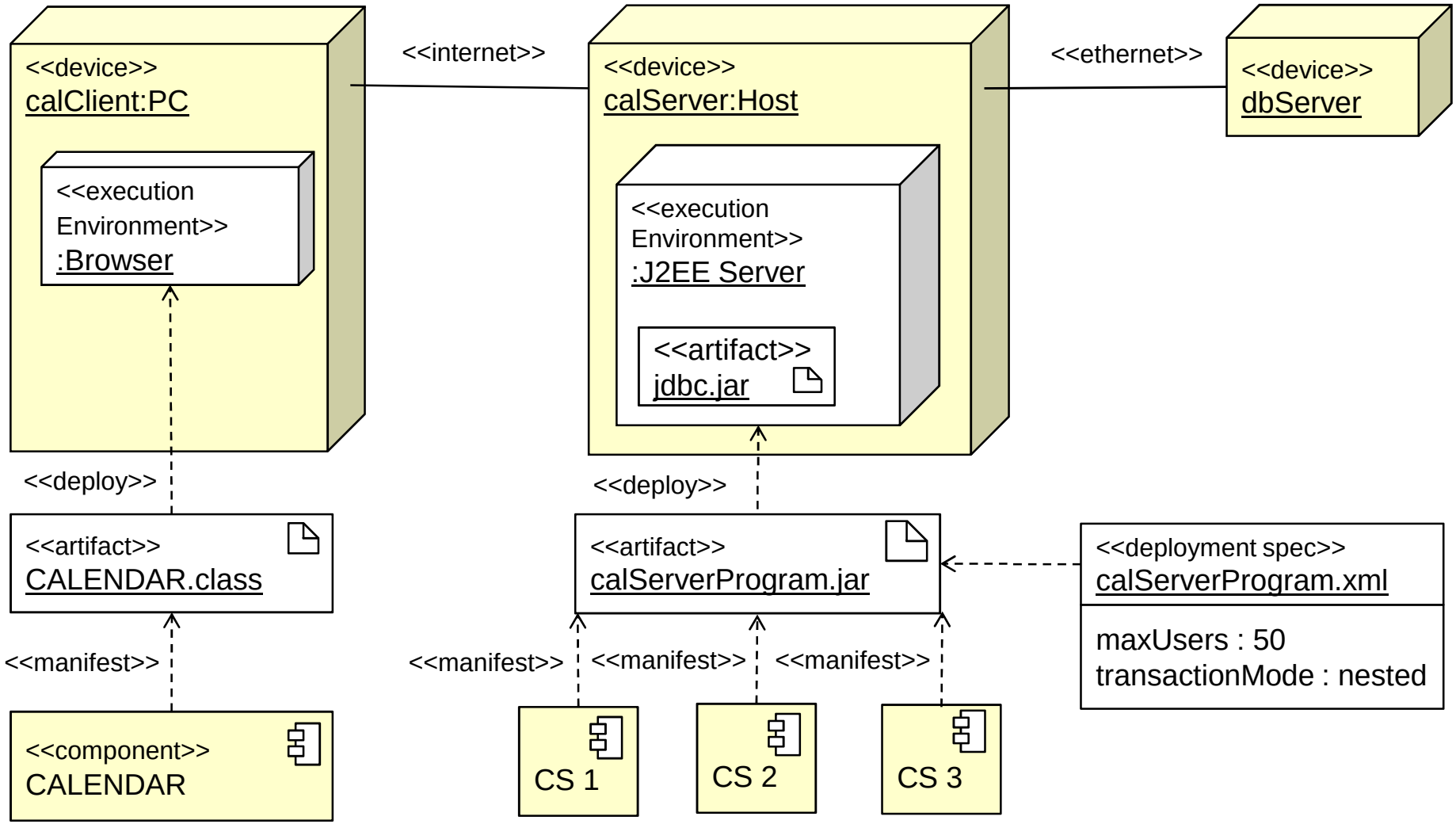
- Gerät
  - Physikalisches Gerät auf dem Artefakte zum Einsatz kommen (Rechner)

Verteilungsdiagramm ▸ Beziehungen

- Manifestation
  - Komponente ist durch Artefakt realisiert
- Einsatzbeziehung
  - Artefakt wird auf Ausführungsumgebung oder Gerät eingesetzt
- Kommunikationsbeziehung
  - Physische Verbindung über die Ausführungsumgebungen kommunizieren
  - kann als Stereotyp angegeben werden
    - ⇨ z.B. `<<internet>>`, `<<ethernet>>`, ...



Verteilungsdiagramm ▸ Beispiel



Verteilungsdiagramm ▸ Verwendung im Entwicklungsprozess

- Verteilungsdiagramme werden im Systementwurf erzeugt

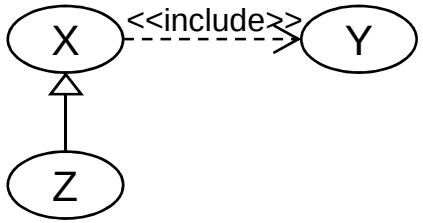
Verteilungsdiagramm ▸ Nicht Aufgaben

- Verteilungsdiagramme wissen nichts über
  - Zeitliche Abläufe
  - Implementierungsdetails der Artefakte und Komponenten
  - Akteure

Verteilungsdiagramm ▸ Typische Fehler

- Inkonsistenz mit Komponentendiagramm
- Verwechslung von deploy und manifest
- falsche Pfeilrichtungen
- Unklarheit was Dokument und was Laufzeitumgebung ist

universitätbonn ROOTS



Use Case-Diagramme

Use Case-Diagramm ▸ Aufgaben

- Ein Use Case-Diagramm beschreibt
  - einen Aspekt der Funktionalität des Systems
  - aus Anwendersicht.
- Die Menge aller Use Case-Diagramme eines System beschreibt das komplette System aus Anwendersicht.
- Use Case-Diagramme dienen insbesondere zur Kommunikation zwischen Entwicklern und Kunden.

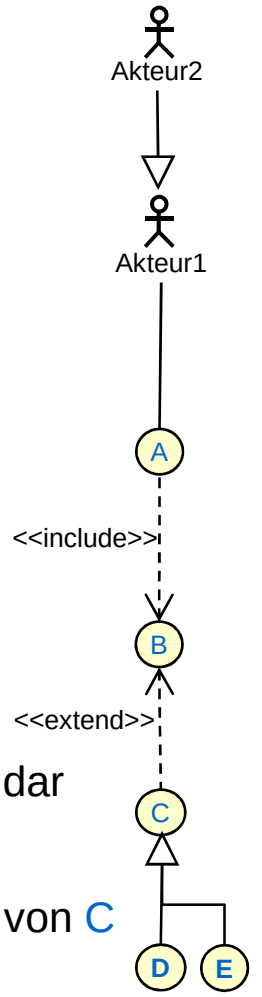
Use Case-Diagramm ▸ Verwendung im Entwicklungsprozess

- UseCase-Diagramme werden
  - erstellt in der Analysephase
  - dienen als Grundlage für diverse weitere Phasen
    - ⇨ insbesondere bei der Kommunikation mit dem Kunden

- Zu jedem UseCase gehört zusätzlich eine UseCase-Beschreibung

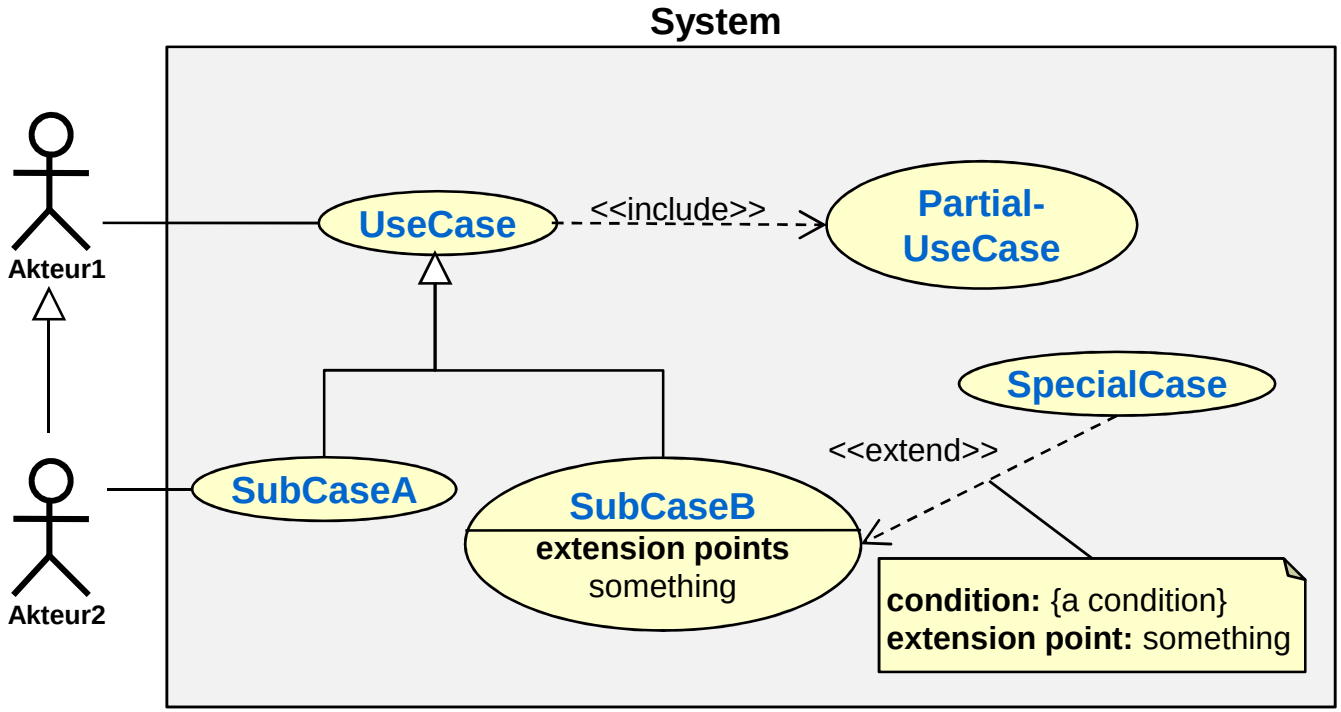
Use Case-Diagramm ▸ Beziehungen

- Zwischen Akteuren
  - Spezialisierung
    - ⇨ Alles was Akteur1 kann, kann auch Akteur2
- Zwischen Akteuren und Use-Cases
  - Link
    - ⇨ Akteur1 kann A ausführen
- Zwischen Use-Cases
  - Include
    - ⇨ Immer wenn A ausgeführt wird, muss auch B ausgeführt werden
  - Extend
    - ⇨ C ist nicht zwingend nötig, um B auszuführen
    - ⇨ C stellt Sonder- oder Fehlerfälle, optionales oder seltenes Verhalten dar
  - Spezialisierung
    - ⇨ D und E sind jeweils spezielle Ausprägungen des gesamten Ablaufs von C
    - ⇨ Nur eine der Spezialisierungen wird durchgeführt, die aber komplett



Use Case-Diagramm ▸ Beispiel

- UseCases sind Ellipsen mit dem jeweiligen Namen
  - Möglichst Verben verwenden (da Tätigkeiten)

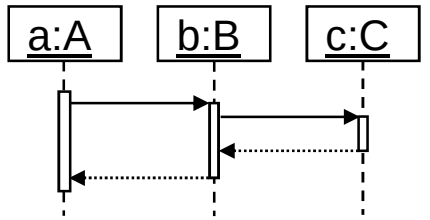


Use Case-Diagramm ▸ Nicht Aufgaben

- UseCase-Diagramme kennen keine
  - Strukturinformationen über das System
  - zeitlichen Abhängigkeiten
  - Information über die Aufteilung der Software-Elemente
  - spezifische Information über alles außerhalb des Systems
  - Information über Daten, Datenflüsse oder Zustände (außer abstrakt in Bedingungen)

Use Case-Diagramm ▸ Typische Fehler

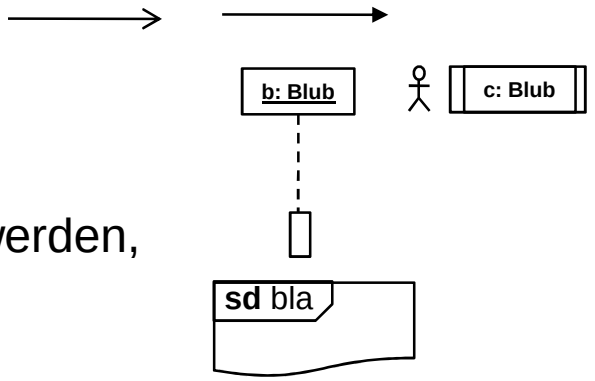
- Versuch „Reihenfolgen“ darzustellen
- Verwechslung der Pfeiltypen
- Verwechslung der Pfeilrichtungen
- Versuch weitere Beziehungen zwischen UseCases darzustellen
  - Insbesondere direkte Beziehung zwischen UseCases
- Systemgrenze vergessen



Sequenzdiagramm

Sequenzdiagramm ▸ Aufgaben

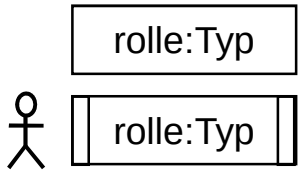
- Sequenzdiagramme spezifizieren wie
  - Nachrichten und Daten
  - zwischen verschiedenen Interaktionspartnern
  - über die Zeit hinweg
  - in einem bestimmten Kontext ausgetauscht werden,
  - um eine bestimmte Aufgabe zu erfüllen



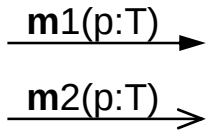
- Darstellbar sind dabei
  - Nebenläufigkeit
  - Verzweigungen / Schleifen

Sequenzdiagramm ▸ (wichtige) Elemente

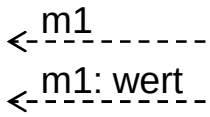
- Interaktionspartner
  - passives Objekt (aktiv nur als Reaktion auf Nachrichten)
  - aktives Objekt (Prozess/Thread)



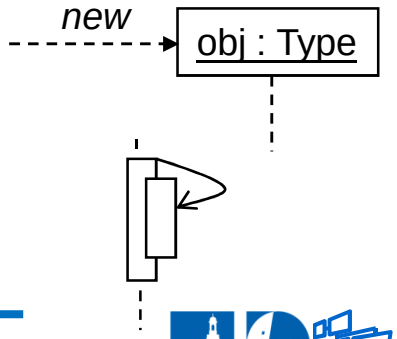
- Nachricht
  - synchron (Aufrufender wartet auf Ende der Aktivierung)
  - asynchron (Aufrufender sendet Nachricht und macht sofort weiter)



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

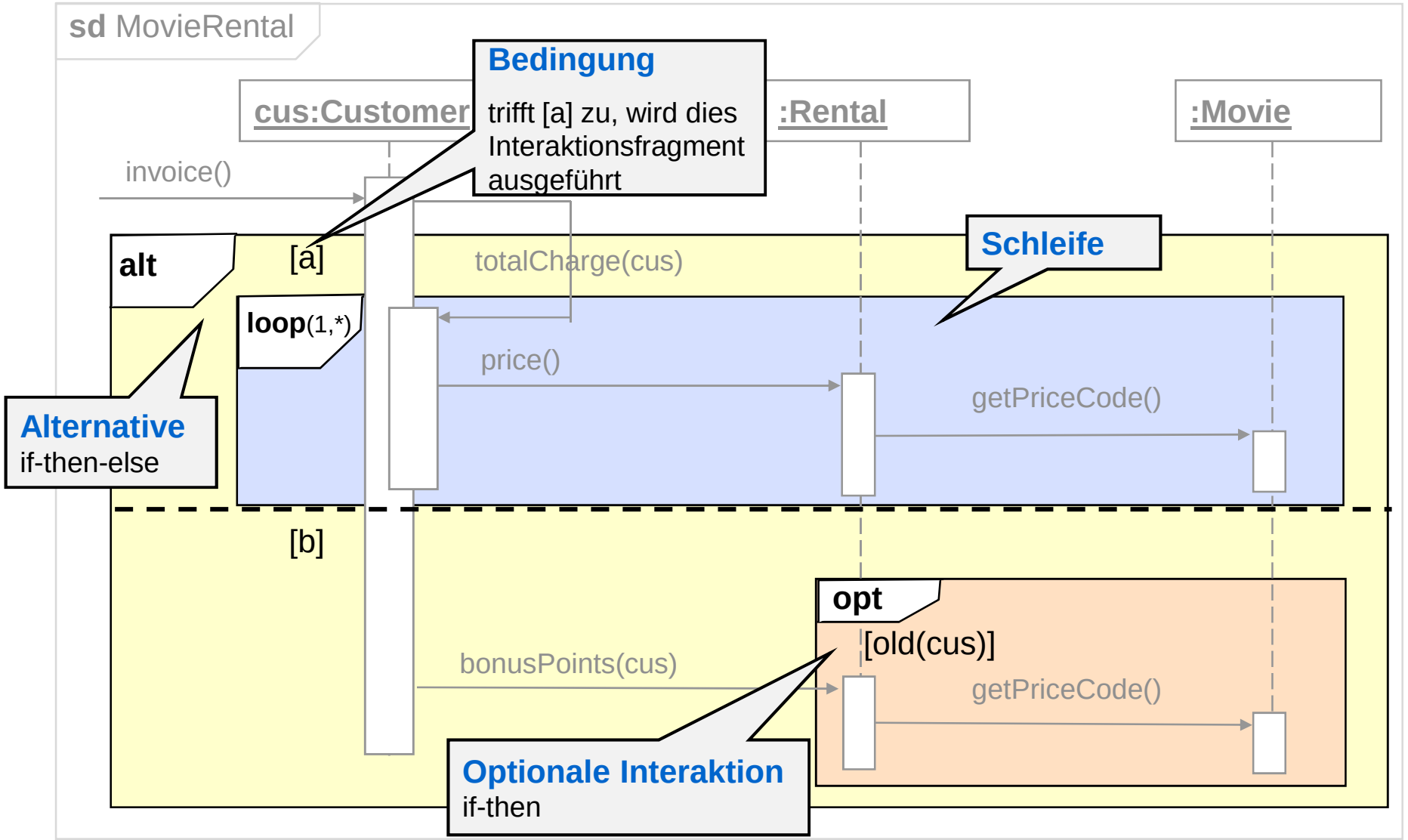


- Objekt wird durch Nachricht erzeugt



- Nachricht an sich selbst

Sequenzdiagramm ▸ Fragmente

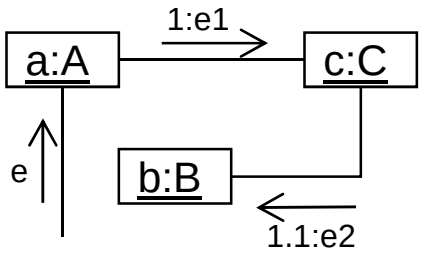


Sequenzdiagramm ▸ Nicht Aufgaben

- Sequenzdiagramme sagen nichts aus über
  - statische Strukturinformationen
  - Kommunikation außerhalb des betrachteten Systems
  - Zustand des Systems

Sequenzdiagramm ▸ Typische Fehler

- Akteure vergessen
- Verwechseln von synchron / asynchron
- Keine / fehlerhafte / inkonsistente Beschriftung der Nachrichten
- Schleifenbedingungen falsch → selbst nachschauen!

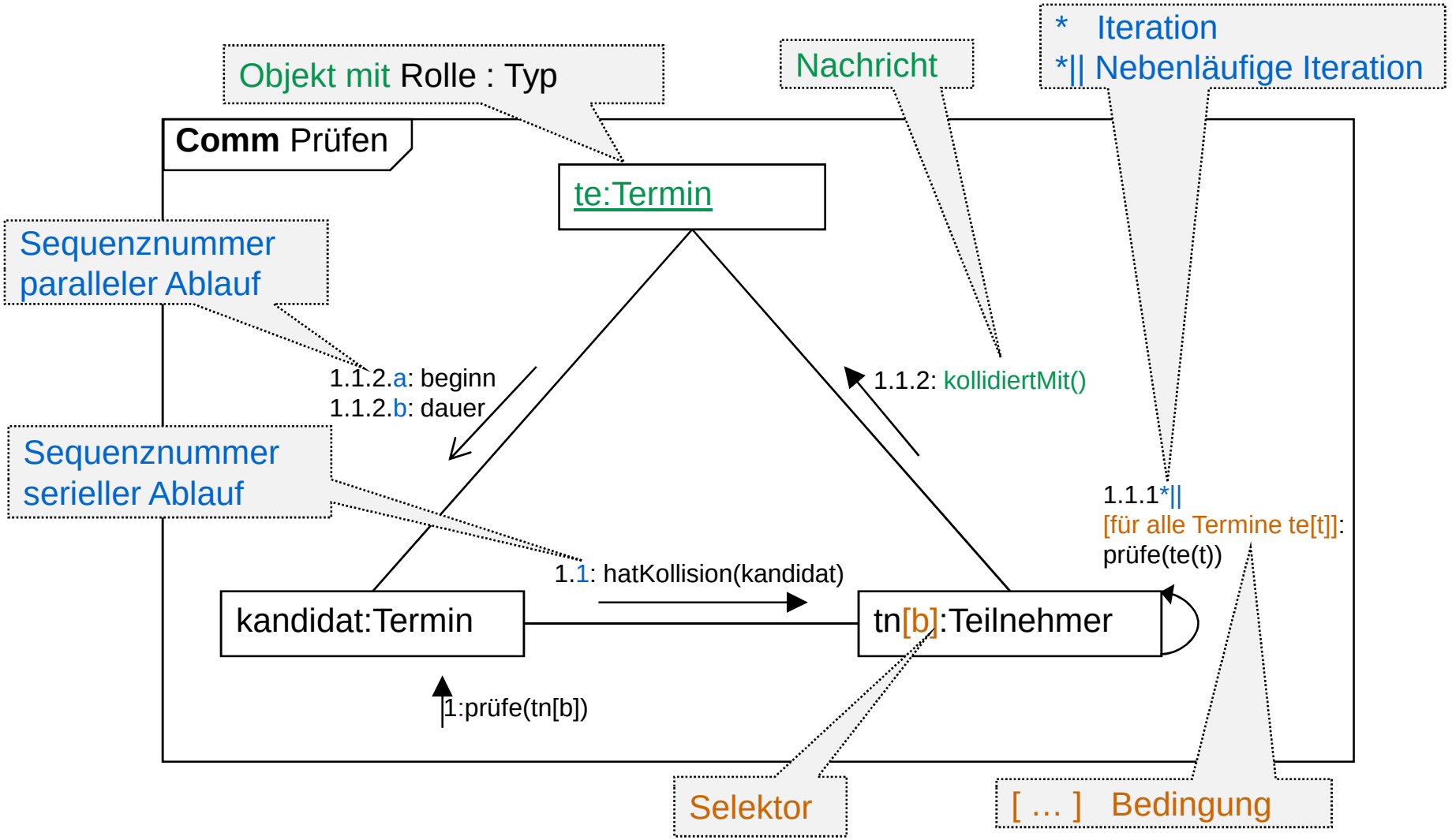


Kommunikationsdiagramm

Kommunikationsdiagramm ▸ Aufgaben

- Im Prinzip eine Kombination aus
  - Sequenzdiagramm und
  - Objektdiagramm / Klassendiagramm
- Der Fokus liegt auf
  - Kommunikation zwischen mehreren Objekten durch Nachrichten
  - zeitlichen Ablauf
    - ⇨ Strukturinformationen nur als „Ergänzung“
- Die zusätzlichen Strukturinformationen werden erkauft durch
  - geringere Ausdrucksfähigkeit der Abläufe (Schleifen, Entscheidungen)
  - Gefahr schnell unübersichtlich zu werden

Kommunikationsdiagramm ▸ Beispiel

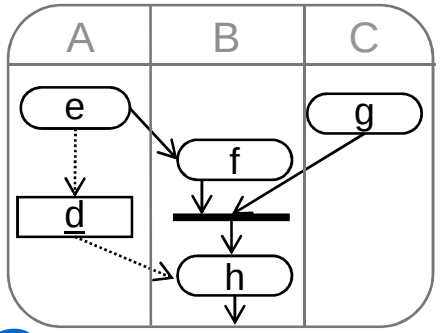


Kommunikationsdiagramm ▸ Nicht Aufgaben

- Keine Wiedergabe der kompletten statischen Struktur
- Keine Wiedergabe aller möglichen Abläufe zwischen den dargestellten Elementen
- Dargestellt wird die Reihenfolge (bzw. parallele oder alternative Ausführung) der Nachrichten, keine konkreten Zeitpunkte.
- Der Versuch komplexen Verschachtelungen mit Kommunikationsdiagrammen führen in den meisten Fällen zu Chaos

Kommunikationsdiagramm ▸ Typische Fehler

- Nummerierung durcheinander
- Inkonsistenz mit Klassen- / Objektdiagramm oder Sequenzdiagramm
- synchron / asynchron verwechselt
- „spontane“ Nachrichten zwischen Objekten (ohne dass von Initial-Nachricht erreicht)
- Schleife falsch



Aktivitätsdiagramme

Aktivitätsdiagramm ▸ Aufgaben

- Beschreibt Kontrollfluss (und Datenfluss) in einem System
  - Insbesondere nebenläufiges Verhalten
    - ⇨ Parallele Abläufe
    - ⇨ Synchronisationsbedarf aufzeigen
  - auch anwendbar für nicht objektorientierte Systeme
    - ⇨ Aktivitäten sind unabhängig von Objekten
    - ⇨ Anwendbar auch auf Geschäftsprozesse, Funktionsbibliotheken, …
- Basiert auf
  - BPEL (Business Process Engineering Language) und
  - Petri-Netzen (Token)

- Elementare Konzepte:
  - Aktivität = benanntes Verhalten
  - Aktion = atomare Aktivität (einzelner Arbeitsschritt)
  - Kontrollknoten

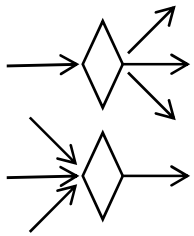
Aktivitätsdiagramm ▸ (wichtige) Elemente

- Aktivität

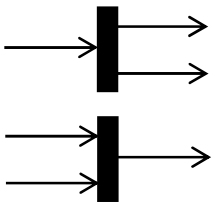
- Start- und Endknoten
  - Start
  - Aktivitätsende (Ende aller Abläufe)
  - Ablaufende (Ende eines bestimmten Ablaufs)

Aktionsknoten

- Alternative Abläufe
  - Entscheidungsknoten (nur ein Pfad wird verfolgt)
  - Vereinigungsknoten (weiter, wenn ein Pfad fertig)

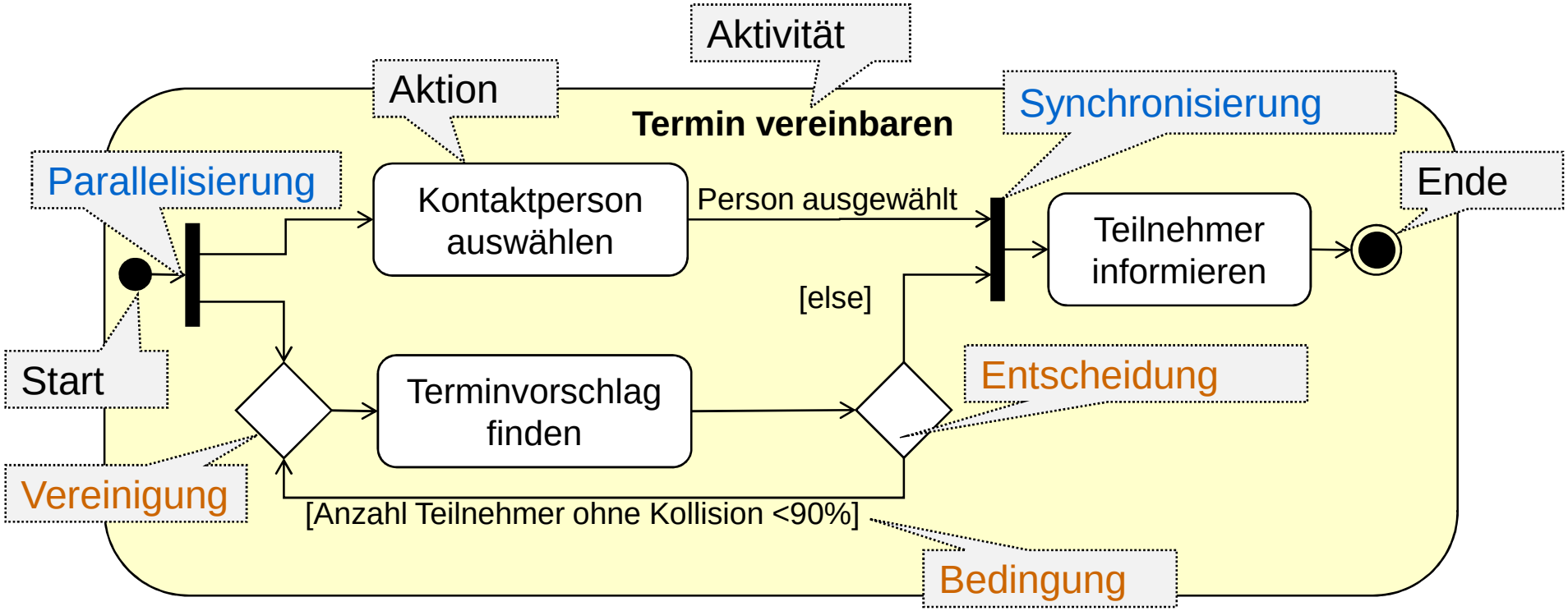


- Nebenläufige Abläufe
  - Parallelisierungsknoten (alle in belieb. Reihenfolge)
  - Synchonisierungsknoten (weiter, wenn alle fertig)



Kontrollknoten

Aktivitätsdiagramm ▸ Beispiel



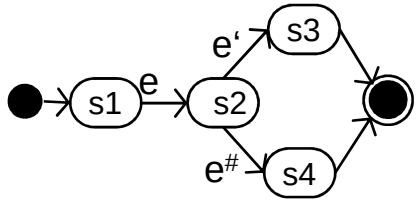
- Parallelisierung + Synchronisation und Bedingung + Vereinigung kommen in den allermeisten Fällen paarweise vor
  - siehe Token-Modell aus Vorlesung

Aktivitätsdiagramm ▸ Nicht Aufgaben

- Keine Strukturellen Informationen
- Aktivitätsdiagramme sind nicht OO-orientiert
  - Interessieren sich nicht für „Objekte“
  - Interessieren sich nicht für „Nachrichten“
- Daher:
  - Keine Abläufe innerhalb eines „Objekts“
  - Keine Abläufe zwischen „Objekten“
  - Keine Information über Darstellung von Date

Aktivitätsdiagramm ▸ Typische Fehler

- Linien statt Pfeilen / falsche Pfeilenden
- Fehlende / falsche End- und Anfangsknoten
- Falsche Kombination von Entscheidungen und Synchronisationen:
  - Mehrere Kanten einer Entscheidung führen in eine Synchronisation
    - ⇨ ewig auf ein Token warten
  - Nach einer Parallelisierung eine Vereinigung zweier paralleler Pfade:
    - ⇨ ein Token wird „vergessen“



Zustandsdiagramme

Zustandsdiagramm ▸ Aufgaben

- Beschreiben das dynamische Verhalten
  - eines Objektes (bzw. Teil des Systems)
  - als endlicher Automat
- Alle möglichen Folgen von Zuständen
  - während ihres „Lebenslaufes“ (von Erzeugung bis Destruktion) oder
  - während der Ausführung einer Operation

Zustandsdiagramm ▸ Zustand

- Ein Zustand kann enthalten:
  - Name
  - Aktivitäten
    - ⇨ Was geschieht beim Eintritt in den Zustand (entry),
    - ⇨ … während des Zustands (do) und
    - ⇨ … beim Verlassen des Zustands (exit)
  - Innere Transitionen
    - ⇨ Interne Zustandsübergänge
    - ⇨ lösen nicht eigene entry- und exit-Aktivitäten aus
  - Innere Struktur
    - ⇨ Geschachtelte Unterzustände und ihre Transitionen

Mindestens eins muss vorhanden sein.

Zustandsdiagramm ▸ Transitionen

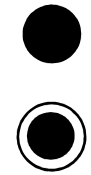
- Notation
  - Spitzer Pfeil mit Beschriftung Ereignis [Bedingung] / Aktion

- Semantik
  - Wenn das Ereignis eintritt und die Bedingung wahr ist, wird die Transition und die Aktion ausgeführt
  - Die Transition unterbricht laufende Aktionen des Quellzustandes
  - Sind mehrere Transitionen bereit wird nichtdeterministisch eine ausgewählt

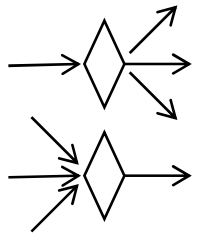
- Ereignis
  - *Keines angegeben*: Implizit das Ende der Aktivitäten des Quellzustands.
  - after(*TimeIntervall*): Nach Ablauf einer absoluten Zeitspanne (1 Sek) oder eines relativen Zeitintervalls (5 Sekunden seit Verlassen von Zustand Z).
  - *Sonstiges*: Auch recht informelle “Ereignisse” zulässig.

Zustandsdiagramm ▸ weitere Elemente

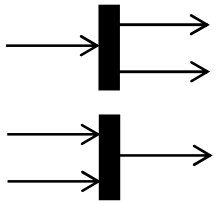
- Startzustand
- Endzustand
  - Es kann mehrere geben
  - Kann Destruktion des Objekts bedeuten, muss aber nicht !
- Terminierungsknoten
  - Objekt, dessen Verhalten modelliert wird, hört auf zu existieren



- Entscheidungsknoten
- Vereinigungsknoten

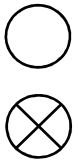


- Parallelisierungsknoten
- Synchronisierungsknoten



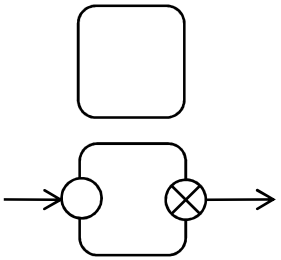
Zustandsdiagramme: Notationsüberblick

- Verbindungspunkt
  - Einstiegspunkt
  - Ausstiegspunkt

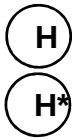


- Verbindungsstelle

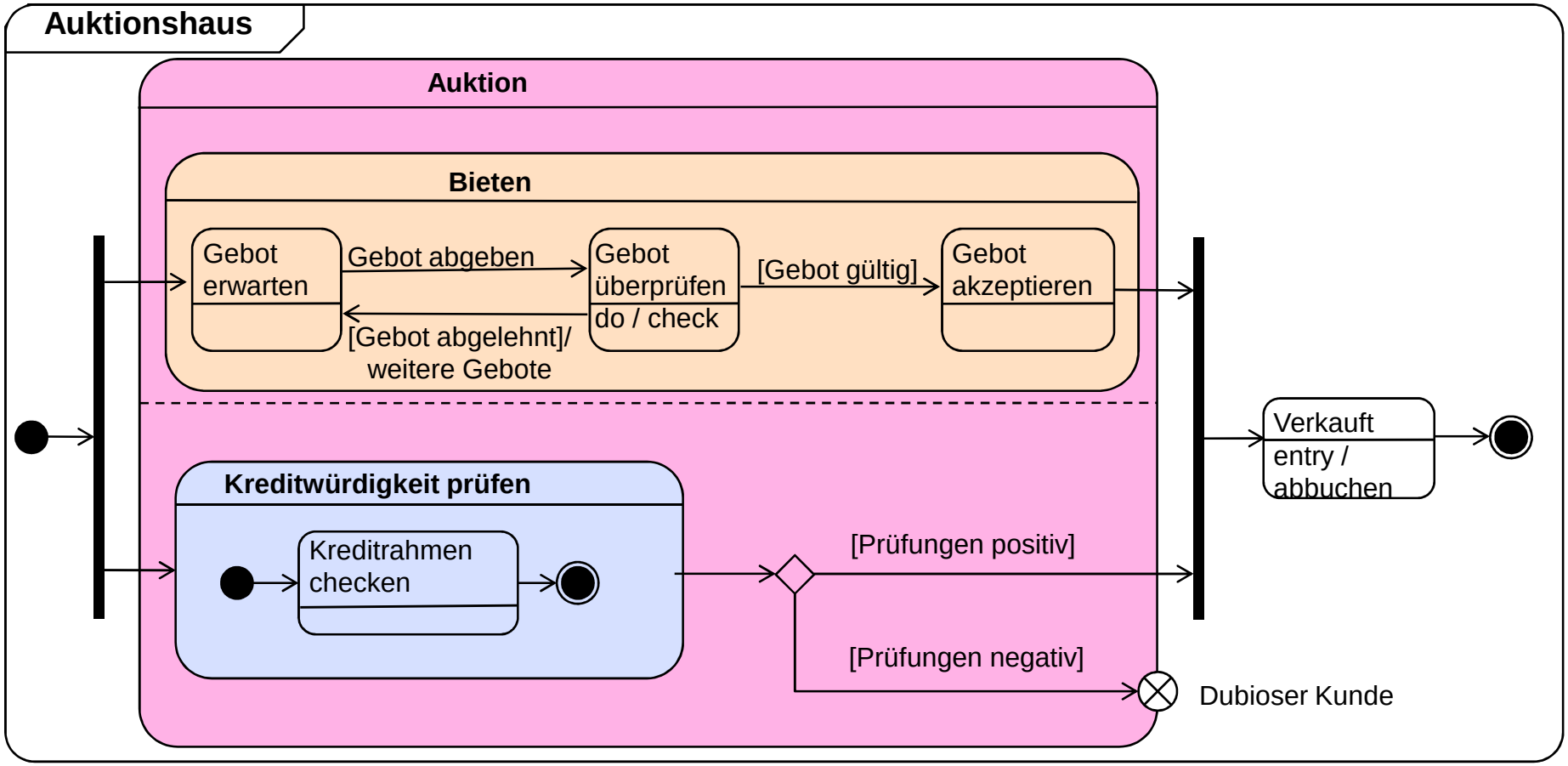
- Zustandsautomat
  - Allgemein (Automat oder Subautomat)
  - Subautomat mit Ein- uns Ausstiegspunkt



- History
  - History-Zustand
  - Tiefer History-Zustand



Zustandsdiagramm ▸ Beispiel



Zustandsdiagramm ▸ Nicht Aufgaben

- Nicht darstellbar ist
  - das Verhalten
    - ⇨ zwischen Objekten
    - ⇨ bzw. zu anderen Teilen des Systems (falls nicht objektorientiert)
  - Strukturelle Abbildung des Zustandes auf das reale System (keine Strukturinformationen)
  - wie die Zustandsübergänge stattfinden, also
    - ⇨ Keine „Nachrichten“
    - ⇨ Keine Datenflüsse

Zustandsdiagramm ▸ Typische Fehler

- Unklar wovon Zustand modelliert wird
- Verwechslung Zustand – Aktivität
- Einbeziehen externer Sachverhalte in die Zustände
  - außer in Bedingungen
- Fehlende Anfangs- oder Endzustände (wenn benötigt)

universitätbonn ROOTS

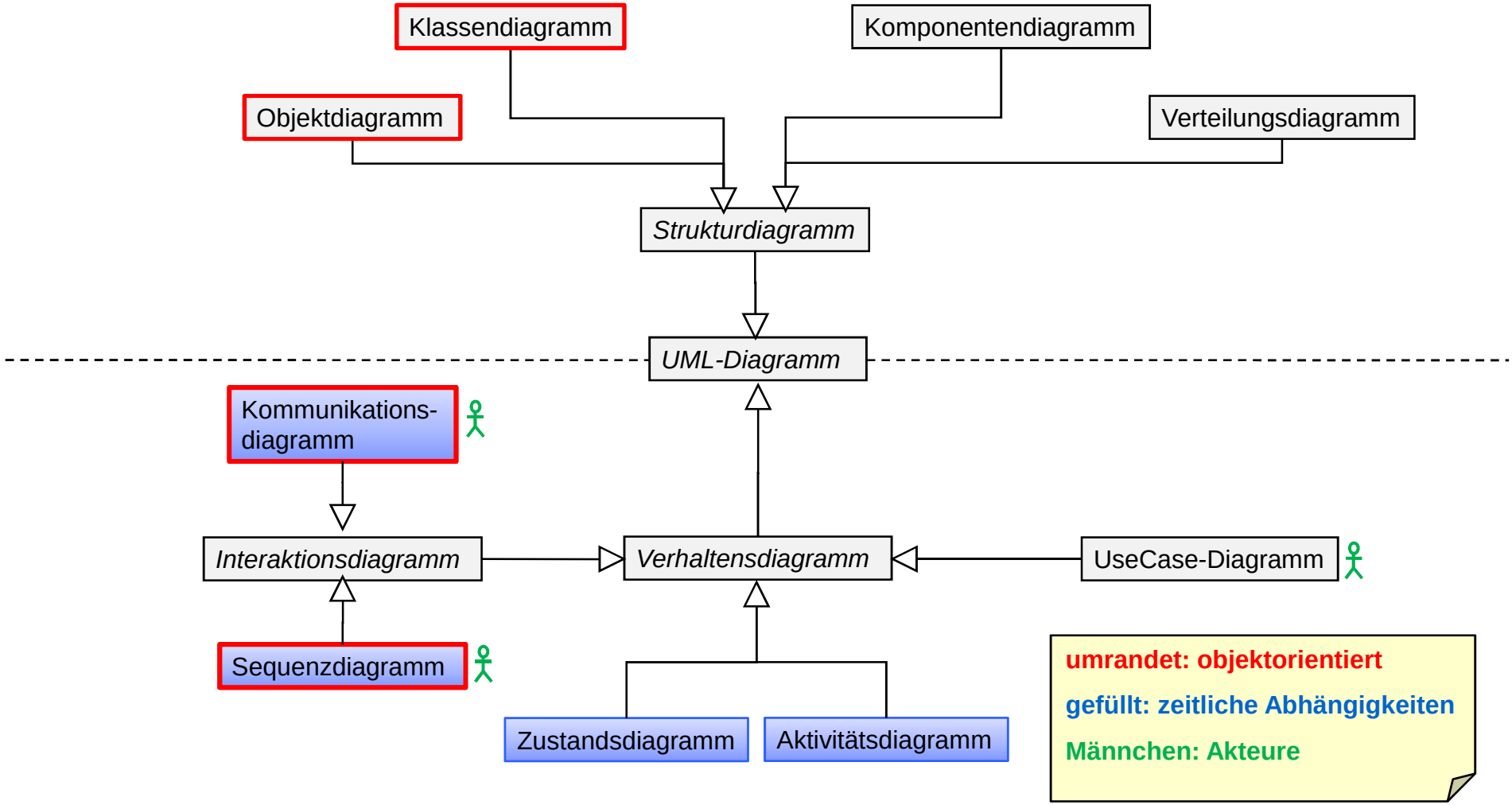
Schlussbemerkungen

Unser UML-Mantra

- Gibt es Akteure?
- Gehört ein Kasten um die Elemente?
- Welche Pfeiltypen gibt es? Wo zwischen?
- Was muss wie beschriftet werden?

- Was kann / will ich ausdrücken?
- An wen richtet sich das Diagramm?
- Welche Infos sind relevant?

UML-Diagramme



Literaturempfehlungen

- Vorlesungsfolien zur SWT 2009/2010
  - http://sewiki.iai.uni-bonn.de/teaching/lectures/se/2009/folien
  - Kapitel 3, 4, 5

- Überblicksdiagramm von OOSE
  - www.oose.de/fileadmin/Dateien/uml-2-Notationsuebersicht-oose.de.pdf

- M. Hitz, G. Kappel, E. Kapsammer, W. Retschitzegger:

  „UML @ Work“

- Bernd Brügge, Allen H. Dutroit:

  „Object-Oriented Software Engineering“