Kurs OMSI
im WiSe 2014/15

Objektorientierte Simulation mit ODEMx

Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dr. Markus Scheidgen
Dipl.-Inf. Ingmar Eveslage

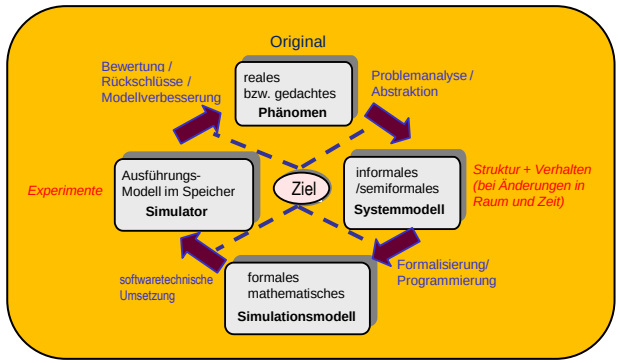
fischer|ahrens|eveslage@informatik.hu-berlin.de

Letzte Vorlesung

1. Systemsimulation – was ist das?
2. Ein Blick zurück in die Anfänge
3. Modelle und Originale
4. Modellierungssprachen, Simulationsumgebungen
5. Bespiele aus der aktuellen Forschung (ODEMx)
6. Paradigma der objektorientierten Modellierung
7. Einordnung von UML
8. Klassifikation dynamischer Systeme
9. Scheduler für zeitdiskrete und zeitkontinuierliche Systemmodelle
10. M&S eines Niedertemperaturofens

Zusammenfassung

Computersimulation: Charakteristischer ‘Scientific Workflow’

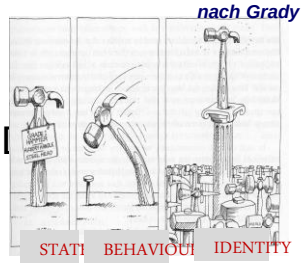


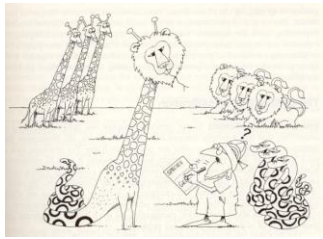
… formuliert als Fragensammlung

- Charakterisieren Sie Computersimulation als Untersuchungsmethode.
- Was versteht man unter einem Simulator?
- Welche Zeitkonzepte lassen sich bei der Computersimulation unterscheiden?
- Was versteht man unter Echtzeitsimulation?
- Wie kommt man zu ausführbaren Simulationsmodellen?
- Erläutern Sie die Bedeutung des Untersuchungsziels für den Scientific Workflow “Computersimulation”.
- Was versteht man unter einem (dynamischen) System?
- Welche Bedeutung haben Strukturäquivalenzen von Original- und Modellsystem für die Computersimulation?
- Wonach werden Modellsysteme klassifiziert?

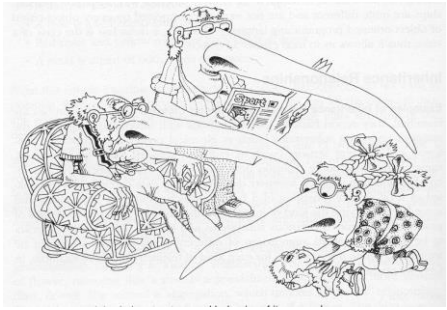
Zusammenfassung (1)

Objektorientiertes Abstraktionskonzepte



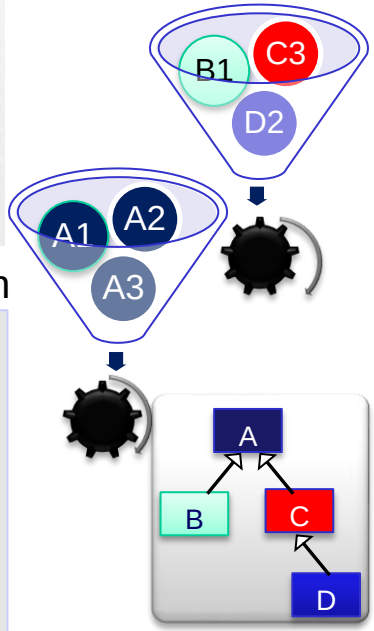


- eigenverantwortlich handelnde, interagierende
  - Zustand (Attribute)
  - individuelles Verhalten (Methoden, Dienste)
  - Identität (Referenz)

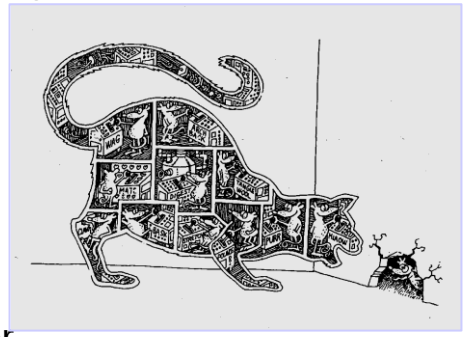


Klassen (Definition von Objekten)

Unterscheidung zw.
- aktiven und
- passive Klassen



Identifikation verschiedenster Beziehungen zwischen Instanzen (bzw. Instanzmengen)
- Navigierbarkeit
- Abhängigkeit
- …



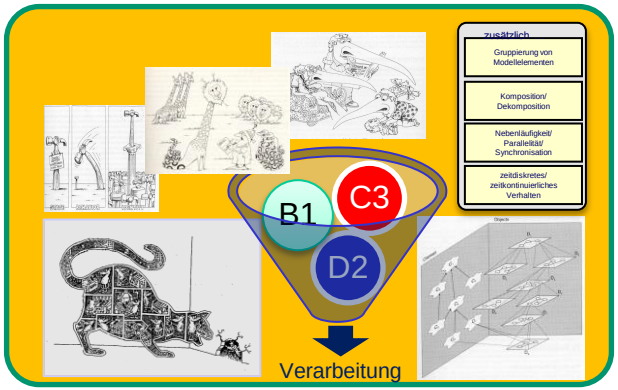
Beziehung zwischen Klassen
- Spezialisierung / Generalisierung
- abstrakte und konkrete Klassifizierer

- Polymorphie von (getypten) Referenzen

zusätzlich
…
- Gruppierung von Modellelementen
- Komposition/ Dekomposition
- Nebenläufigkeit/ Parallelität/ Synchronisation
- zeitdiskretes/ zeitkontinuierliches Verhalten

Zusammenfassung (2)

... formuliert als Fragensammlung



Paradigma der objekt-orientierten Modellierung

- Erläutern Sie das Paradigma der OO-Modellierung.
  - Worin besteht die Natürlichkeit dieses Paradigmas bei der Abstraktion eines Originalsystems zu einem Modellsystem?

    ~ Bedeutung für
    a) Strukturäquivalenz
    b) Verhaltensanalogie

1. Einführung

1. Systemsimulation – was ist das?
2. Ein Blick zurück in die Anfänge
3. Modelle und Originale
4. Modellierungssprachen, Simulationsumgebungen
5. Bespiele aus der aktuellen Forschung
6. Paradigma der objektorientierten Modellierung
7. Einordnung von UML
8. Klassifikation dynamischer Systeme
9. Scheduler für zeitdiskrete und zeitkontinuierliche Systemmodelle
10. M&S eines Niedertemperaturofens

Die UML

551 v. Chr. bis 479 v. Chr.

„Wenn die Sprache nicht stimmt,
ist das was gesagt wird, nicht das, was gemeint ist.“
(Konfuzius)

- UML = Unified Modeling Language
- … ist zunächst Standardsprache (der OMG) zur Visualisierung, Spezifikation, Konstruktion und Dokumentation komplexer Softwaresysteme
- … kombiniert Konzepte der
  - Objektorientierten Modellierung
  - Datenmodellierung (Entity-Relationship-Diagramme)
  - Business-Modellierung (Work Flows)
  - Komponentenmodellierung
  - Verhaltensmodellierung (Erweiterte Zustandsautomaten)

  ...
- UML-Modelle sind in erster Linie graphische Repräsentationen in Form von Diagrammen

UML-Charakterisierung

- OO-Modellierungssprache mit
  - formal definierter **statischer Semantik** in UML und OCL (Meta-Level)
  - informal definierter **dynamischer Semantik** (engl. Text)

- dynamische Semantik, enthält sog. semantische Variationspunkte
  (z.T. alternative semantische oder offene semantische Festlegungen)

  → Offenheit für verschiedene Kombinationen von UML mit realen OO-Programmiersprachen

  → Problem für Compiler (Zielcode, Simulatorcode)

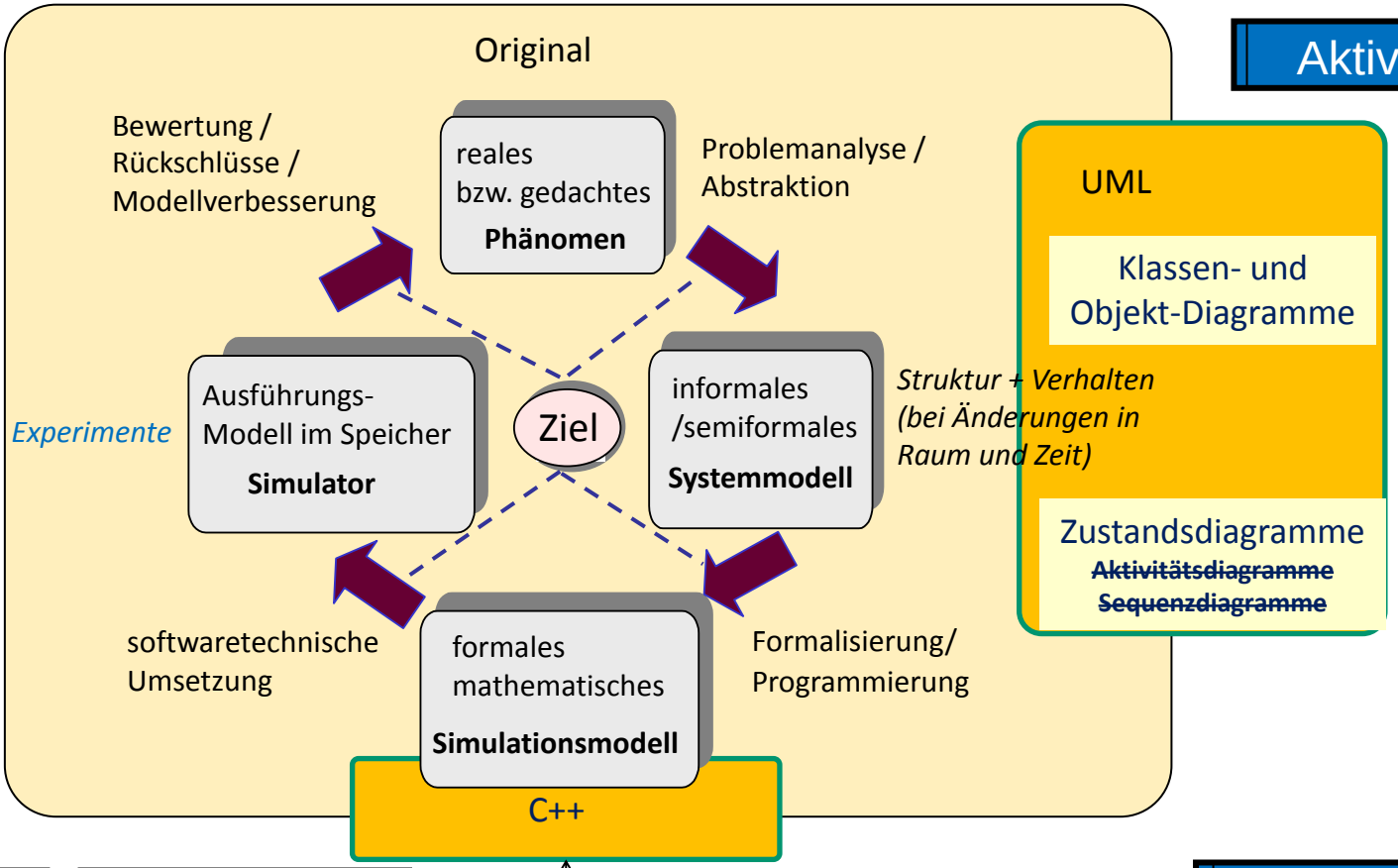
- **Action-Sprache** von UML befindet sich in Entwicklung
  (Einfluss auf Datentypen in UML)
  Referenzsemantik ← →Wertesemantik ?

  UML-Instanzen von Klassen

  UML-Built-IN-Datentypen

  - SDL-ASN.1-UML-Compiler (HU Berlin) benutzt C als Action-Sprache
  - OMSI-Vorlesung benutzt C++ als Actionsprache

Unser Scientific Workflow



PassivKlasse

AktivKlasse

Process

Continuous

Beladung

Ofen

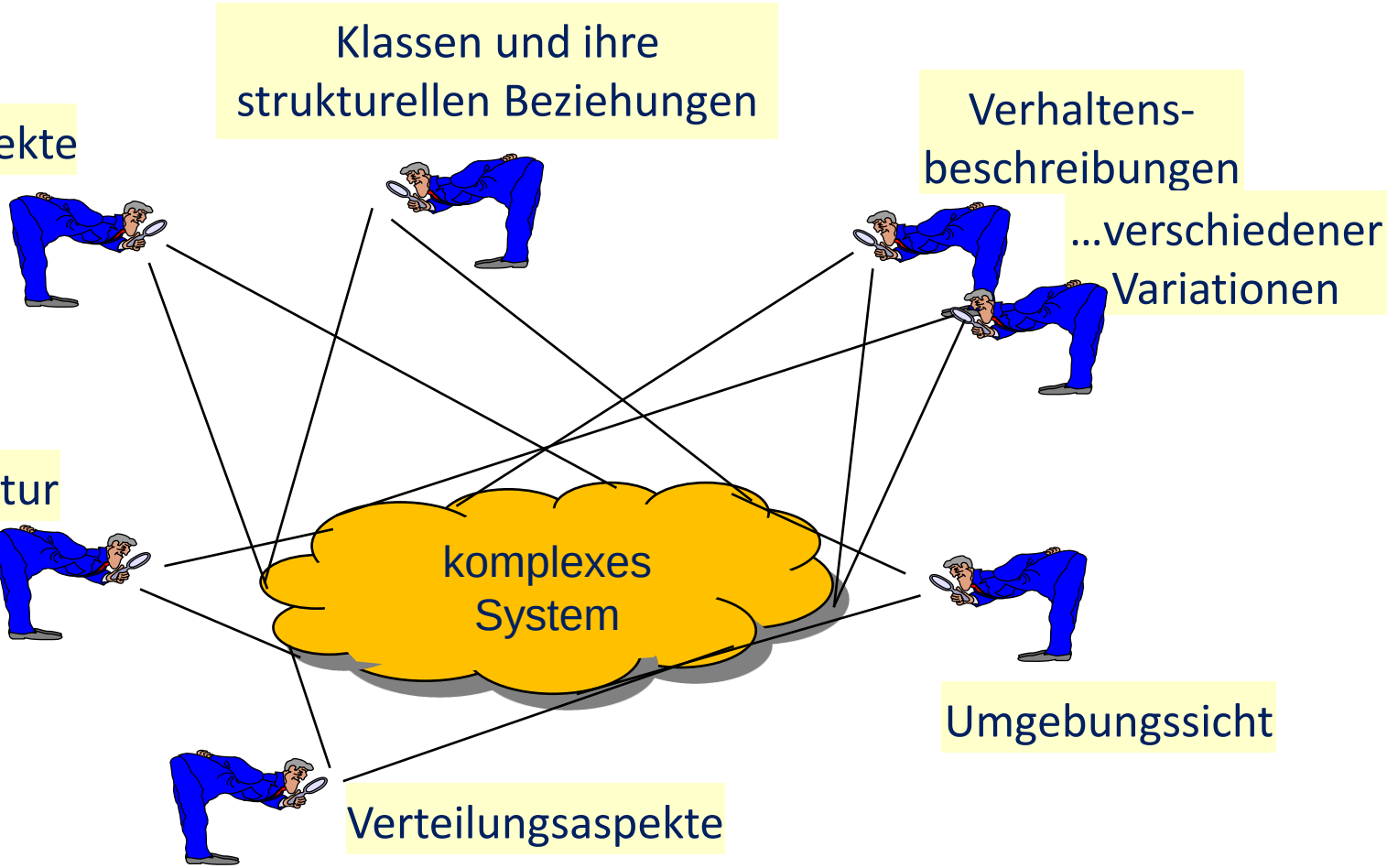
Barren

ODEMx

Process

Continuous

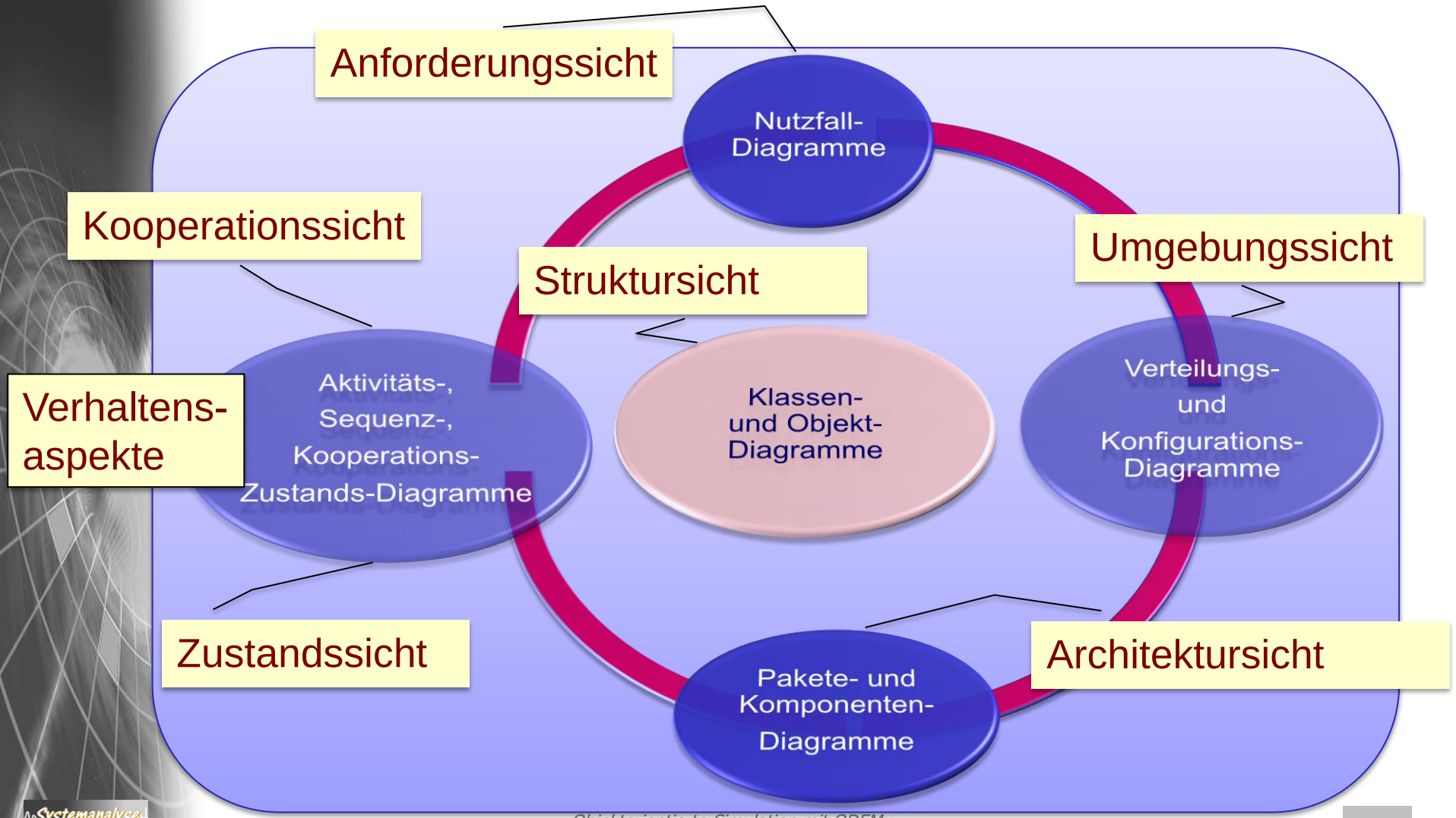
UML-Sichtweisen auf ein komplexes System



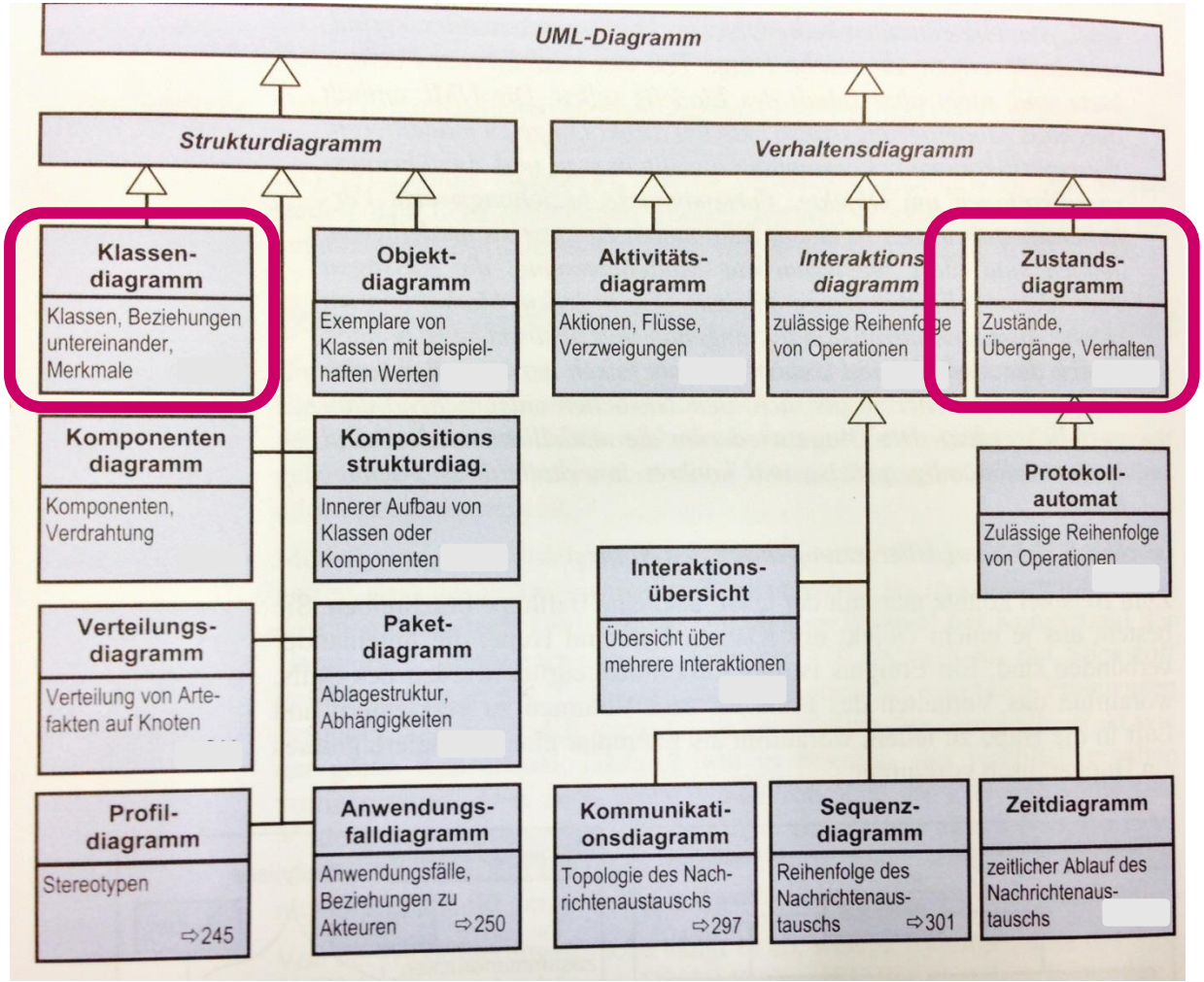
➔ verschiedene UML-Teilsprachen (Diagramme)

Diagrammarten ~ UML-Teilsprachen

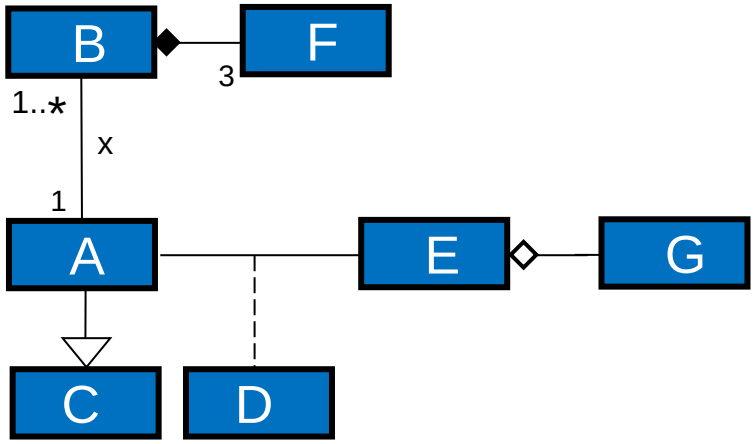
Modellklassen zur Beschreibung von dynamischen Systemen



UML-Diagrammarten



Klassendiagramme



- A steht mit B in einer Beziehung namens x
  jedem A-Objekt sind 1 bis beliebig viele B-Objekte zugeordnet,
  jedem B-Objekt ist genau ein A-Objekt zugeordnet
- A ist eine Spezialisierung von C
- D ist eine Assoziationsklasse, die die Beziehung zwischen A- und E-Objekten beschreibt
  jeder A-E-Link ist durch ein D-Objekt charakterisiert,
  dieses hat zwei Attribute: A-Referenz, E-Referenz
- Jedes E-Objekt besitzt ein G-Objekt (Aggregation)
  G-Objekt ist Teil von A (könnte aber gleichzeitig Teil von etwas anderem sein)
- Jedes B-Objekt besitzt 3 F-Objekte als (Komposition)
  F-Objekt kann (max.) nur Teil vom B-Objekt sein

Klassen und Objekte

passive Klasse

aktive Klasse

Kunde

Kunde

name
titel
geburtstag
geschlecht

Kunde

name: **String**
titel: **String**
geburtstag: **Datum**
geschlecht: **Geschlecht**

alter(): **Integer**

Ticket
{abstract}

KommEinheit

partner: **KommEinheit**

«**signal**»
SigA(**integer**),
SigB(**float**)

op (**int**, **char**)

- beliebige **Detailierungsgrade** in der Darstellung einer Klasse möglich
  Tools sichern, dass in einem Namensraum keine Widersprüche zwischen Darstellungen einer Klasse auftreten
- Sichtbarkeit kann individuell eingeschränkt werden
  \+ public, # protected, - private, ~ package

Attribut-Belegung (Zustand zu einem Zeitpunkt) des Kunden-Objektes K1

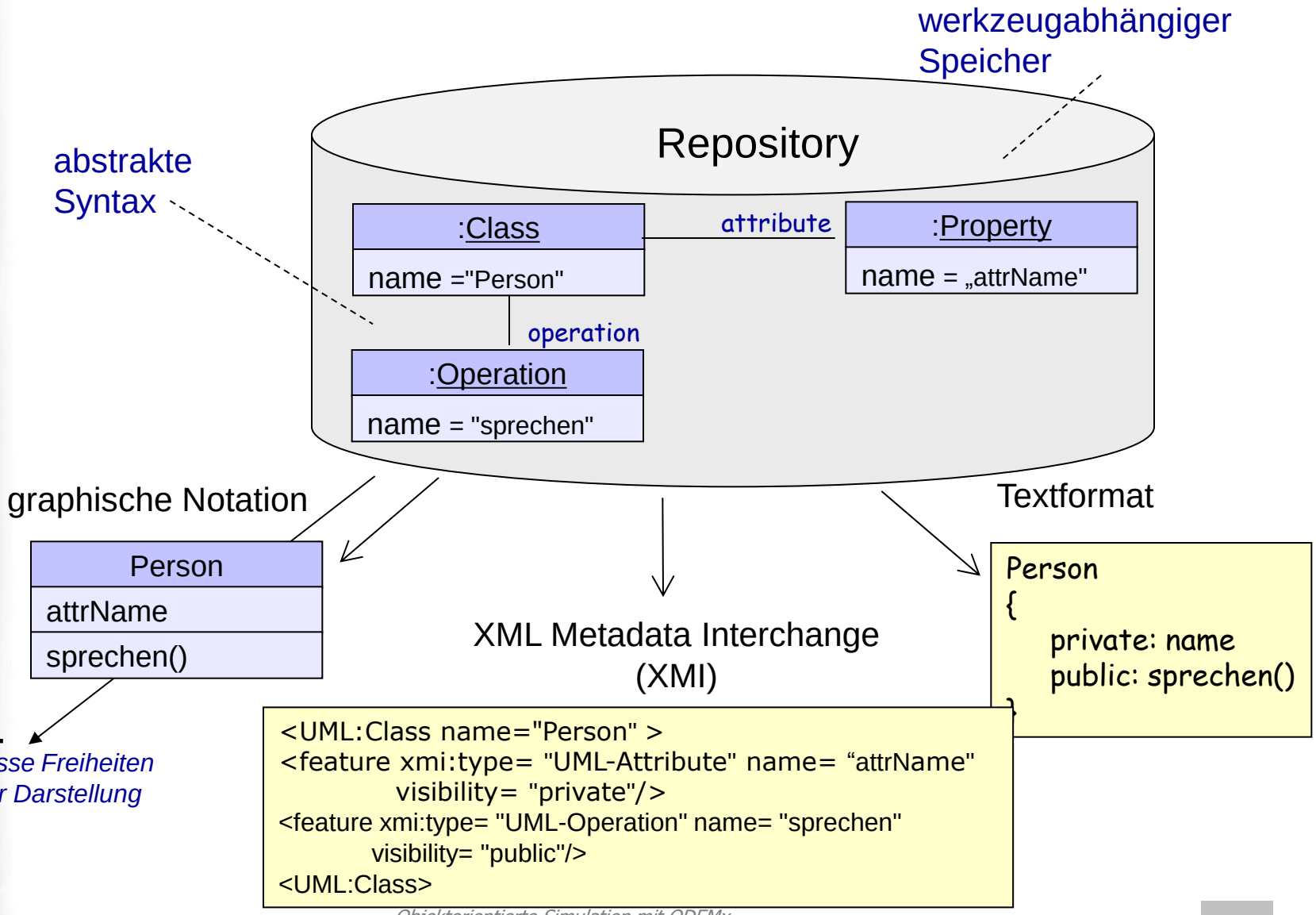
K1:Kunde

name = “Mustermann”
titel = “Dipl-Inf”
geburtstag= 10-09-1992
geschlecht= weiblich

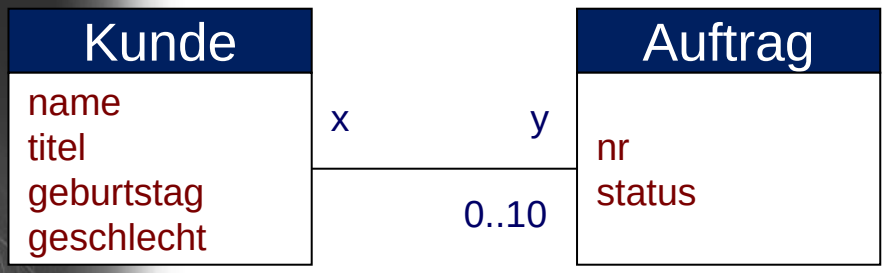
:Kunde

anonymes Objekt

Illustration: UML-Modell- Repräsentation



Dualität von Attributen und Assoziationsenden



Lesart:
jedem Kunde-Objekt ist eine Kollektion y von Aufträgen zugeordnet

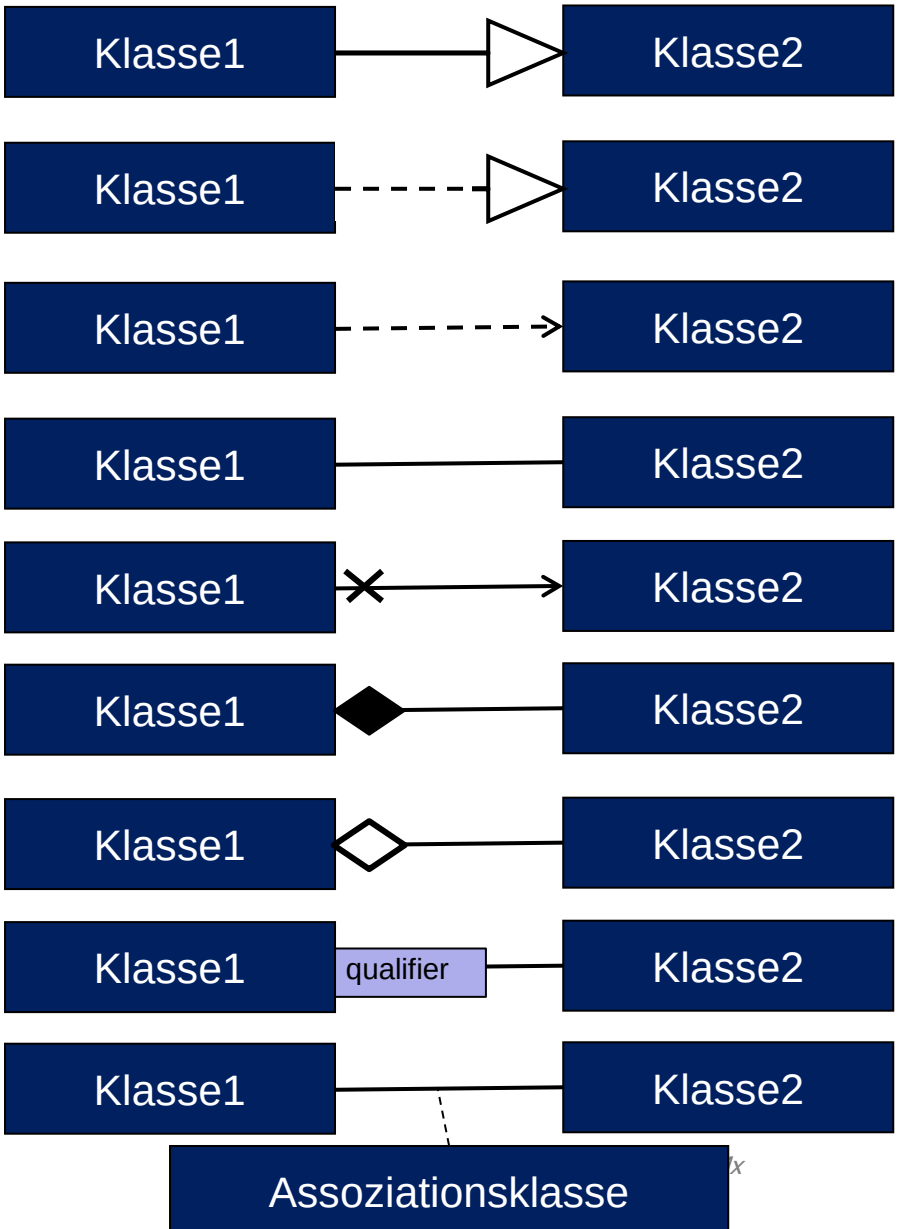
jedem Auftrag-Objekt ist genau ein Kunde-Objekt x zugeordnet

- *Das Assoziationsende x (vom Typ Kunde) dient der Navigation und entspricht implizit einem Attribut der Klasse Auftrag*
- *Das Assoziationsende y (vom Typ Menge über Auftrag) entspricht implizit einem Attribut von Kunde*

Kardinalität als Eigenschaften von Attributen/Assoziationsenden

falls Assoziationsenden nicht benannt worden sind (also wenn Angabe y fehlt)

Beziehungen *zwischen Klassen bzw. Objektmengen*



Spezialisierung

Realisierung
mit Varianten

Abhängigkeit
mit Varianten

Assoziation

gerichtete Assoziation

Komposition

Aggregation

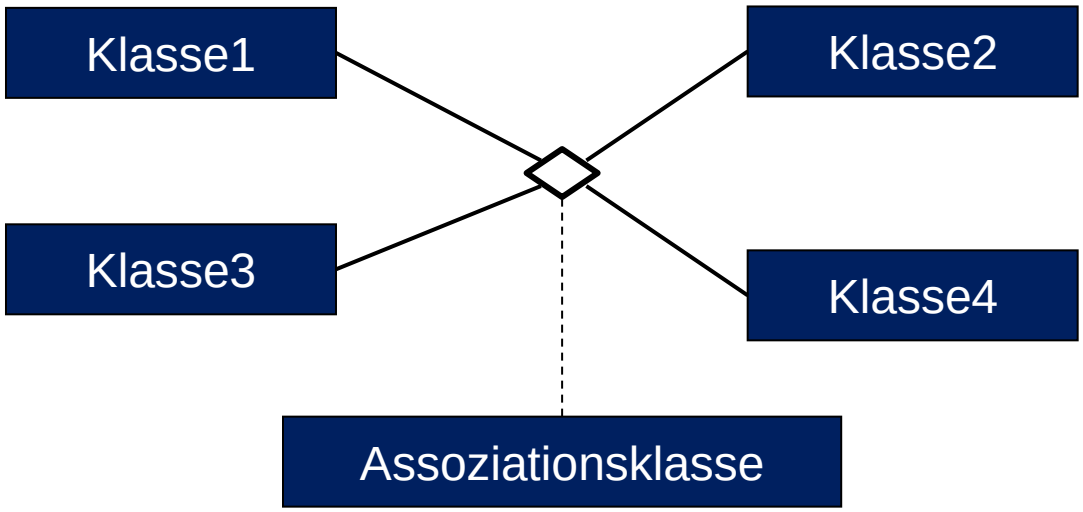
qualifizierte Assoziation

attributierte Assoziation

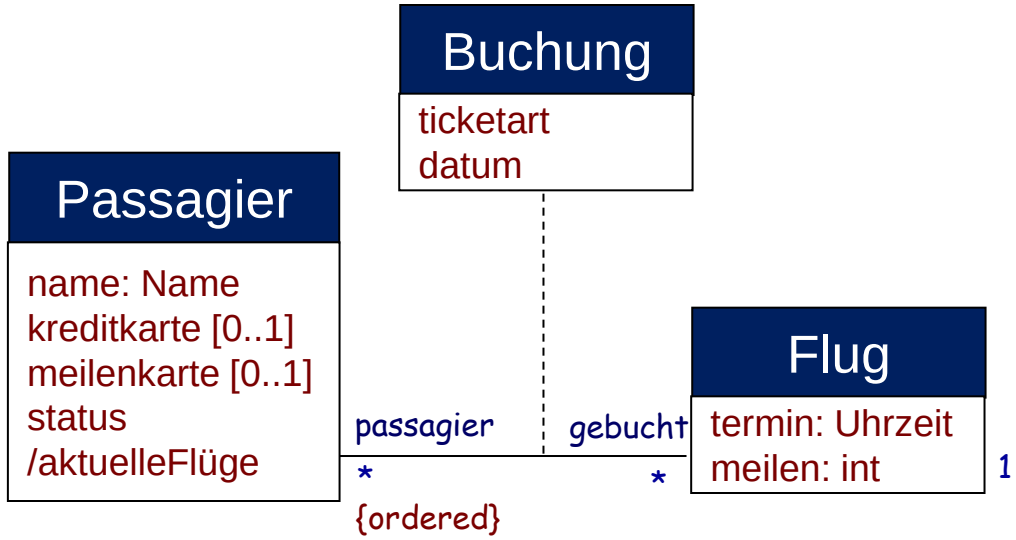
Assoziation

- beschreibt als **Relation** die gemeinsame Struktur und Semantik von Mengen von Objektverbindungen der beteiligten Klassen
- **Links** als Objektverbindungen sind Instanzen einer Assoziation
- Eine Assoziation kann einen **Namen** haben
  (beschreibt worin oder weshalb eine Beziehung besteht)
- Die **Assoziationsenden** tragen immer einen Namen
  (beschreiben die Rollen, die die Objekte der Klassen am jeweiligen Ende spielen)
  Ist kein Name angegeben, wird der kleingeschriebene Klassenname angenommen
- Für jede Rolle kann eine Reihe **zusätzlicher Angaben** gemacht werden:
  - Multiplizität (Standardwert=1),
  - Sichtbarkeit,
  - Eigenschaften (verschiedener Art)
- **Dualität** von Assoziationsende und Attribut (jedes Rollenende ist ein Attribut der gegenüberliegende Klasse oder Attribut einer evtl. Assoziationsklasse)

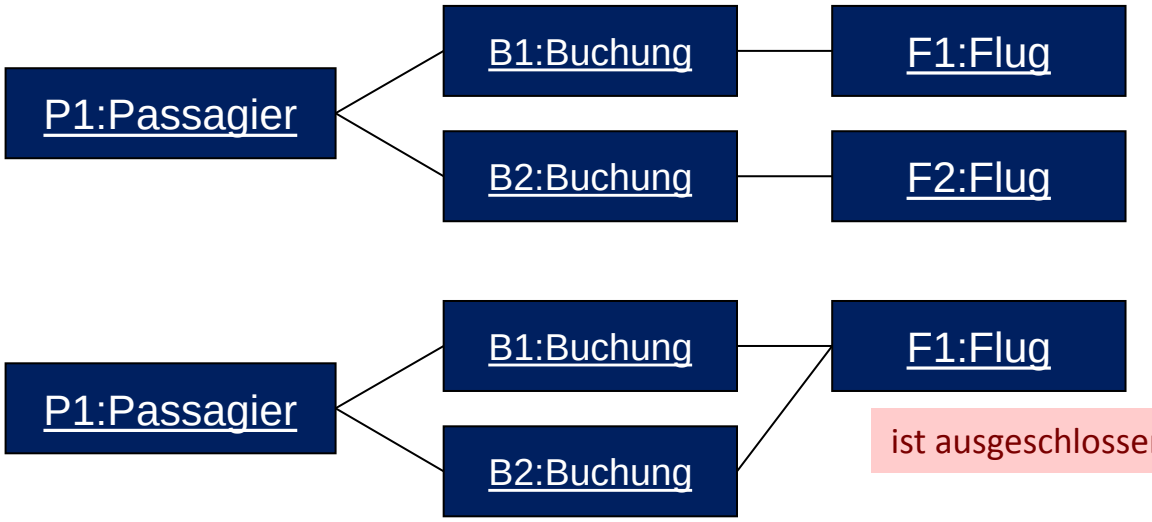
Mehrstellige Assoziation



Assoziationsklasse



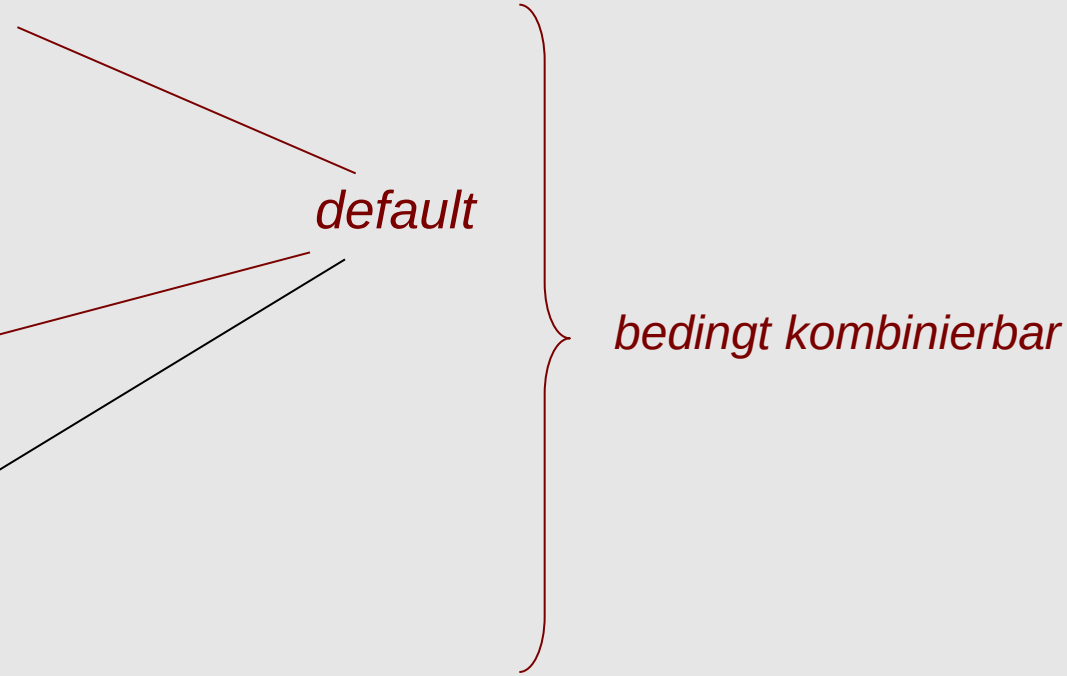
jedes Verbindungspaar ist einmalig



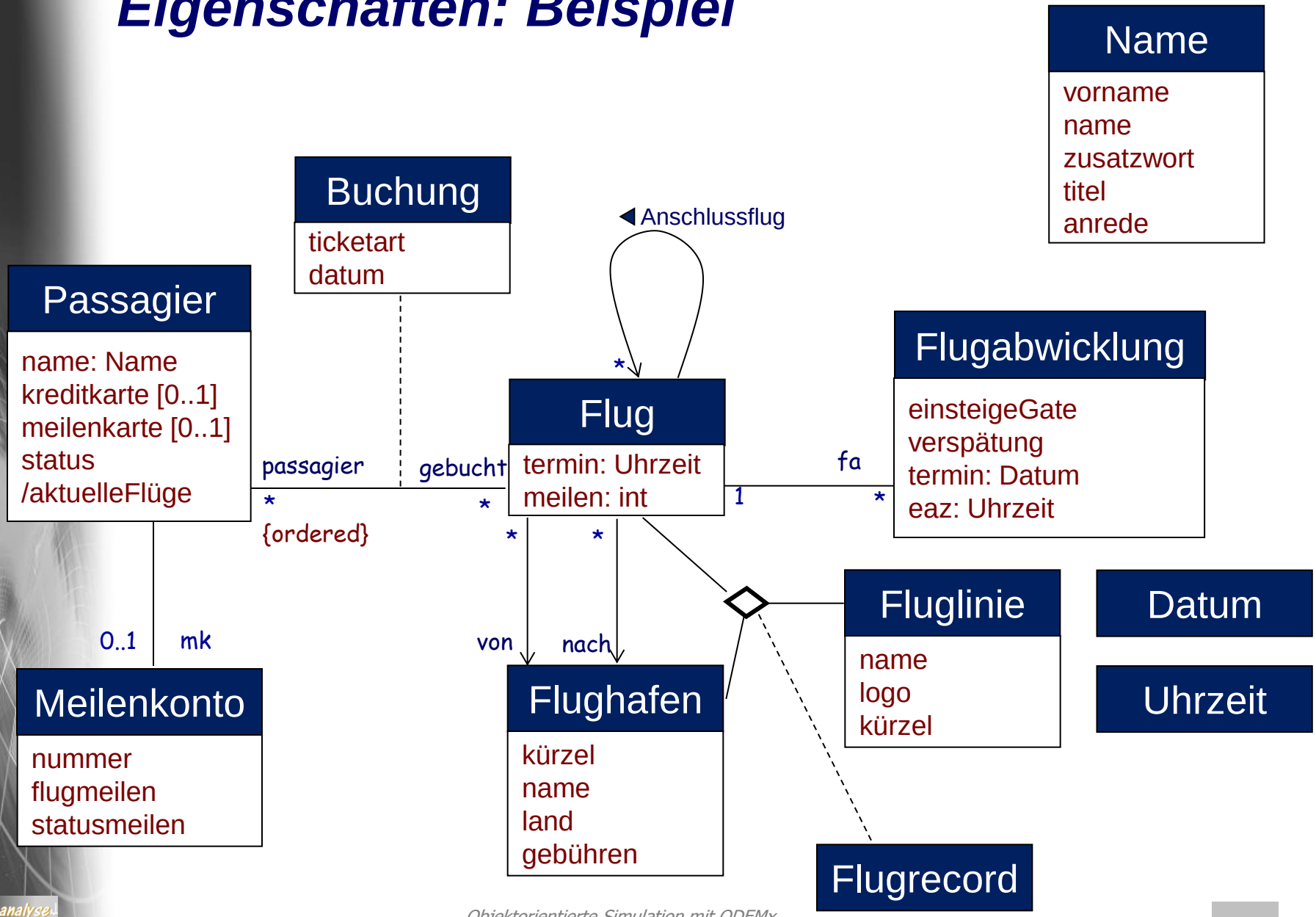
ist ausgeschlossen

Liste möglicher Eigenschaften

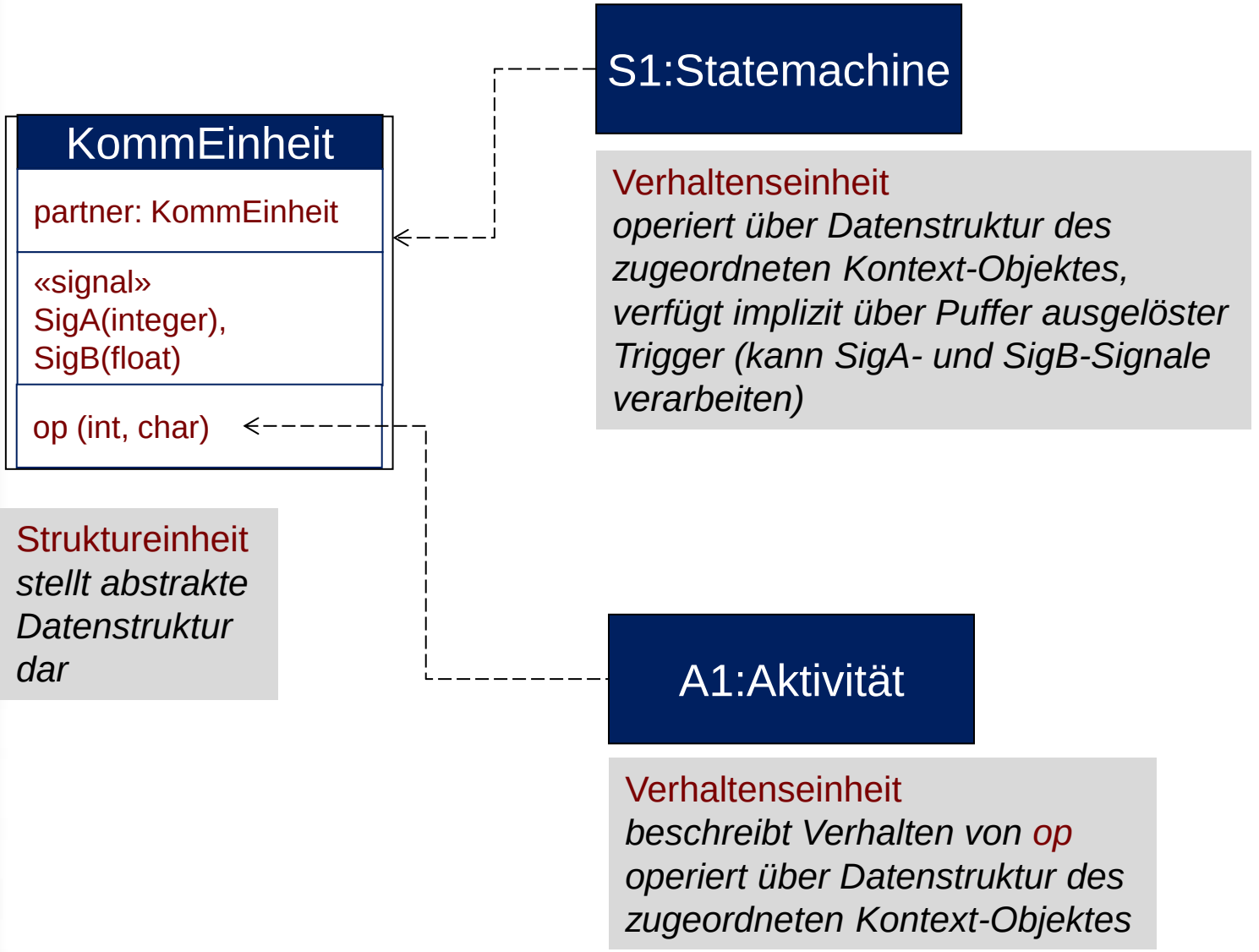
- *… für Assoziationsenden (auch für Attribute) in {…} als Constraint*
  - / (abgeleitete Assoziation)
  - readonly ← → unrestricted
  - composite
  - redefines Attr
  - subsets Attr
  - union
  - unique ← → nonunique
  - ordered ← → unordered
  - seq (Vor.: Multiplizität > 1)
  - bag (Vor.: Multiplizität > 1)



Eigenschaften: Beispiel



Klassen – Aktivitäten - Zustandsmaschinen



Erweiterter Endlicher Zustandsautomat

Endlicher Zustandsautomat
- endliche Anzahl von Grundzuständen
- ein Startzustand
- Zustandsübergang mit möglichen Aktionen

A) Erweiterung eines Endlichen Zustandsautomaten
- Variablen
- Timer
- Pool von Nachrichten (asynchrone Komm.)
- Trigger (Zustandsübergänge)
- Nachrichten/Remote-Op-Rufe
- Guards (als Bedingungen über Zustandsgrößen)

B) Zustandserweiterung
- Eintritts-Aktivität
- Do-Aktivität
- Austritts-Aktivität
- Defer-Aktivität

C) Zustand als Classifier
(Vererbung, Instanziierung, …)

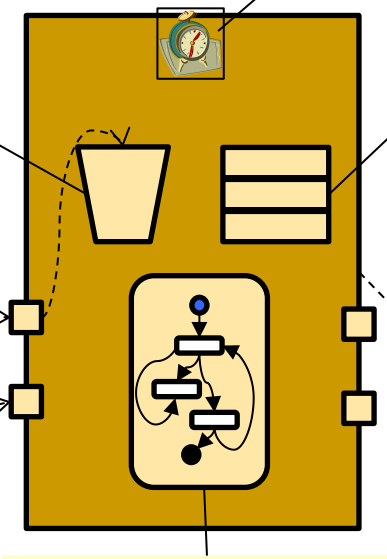
Im Kontext einer aktiven Klasse

Automaten zur Laufzeit

lokaler Timer (Wecker)

Implizites Laufzeitkonzept
Pool ankommender
Signale und
eingehender Operationsrufe

Objekt
einer (aktiven)
Klasse



Signale

sigA

Einheit
~ unklar in UML
unsere Sprechweise:
in SDL: Agent
In ODEMx: Ableitung der
der ODEMx-Klasse
Process

Port
(adressierbare Anschlusspunkte)
mit zugeordnetem **Interface**
(ein- u. ausgehende Signale u. Op-Rufe)

Zustandsmaschine
(Instanz einer Zustands-
maschinenklasse)

Zustand eines SDL-Agenten zu einem Zeitpunkt:

- Stand des gestarteten Timers
- Belegung des Empfang-Pools (inklusive aktueller Parameter der Signale u. Op-Rufe)
- Wertebelegung der Attribute des zugeordneten Objektes
- Aktueller Zustand der Zustandsmaschine
- Befehlsregister (falls im Zustandsübergang)

Achtung (wird hier nicht behandelt):

- Spezialisierung (Vererbung) von Zustandsautomaten)
- hierarchische Zustände
- orthogonale Zerlegung (in parallel) agierende Untermaschinen

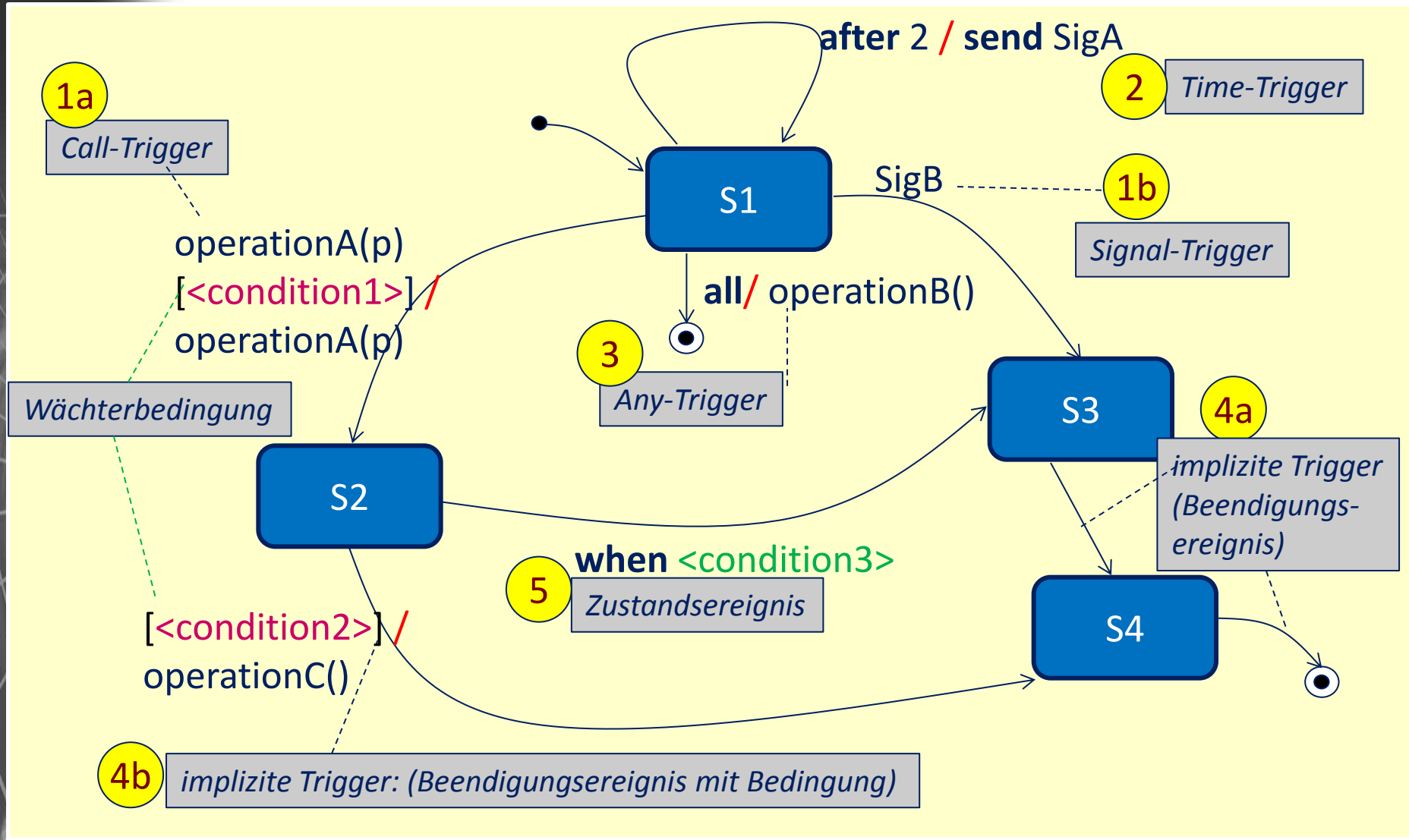
Allgemeine Arbeitsweise

- Automat **verharrt** in seinem Grundzustand bis
  - ein **Signal** oder **RemoteCall** eintrifft (im Pool)
    → Behandlung für den Zustand definiert?
    dann: Behandlung bei Signalkonsumtion
    bzw. Retten (defer)
    sonst: Verwerfen

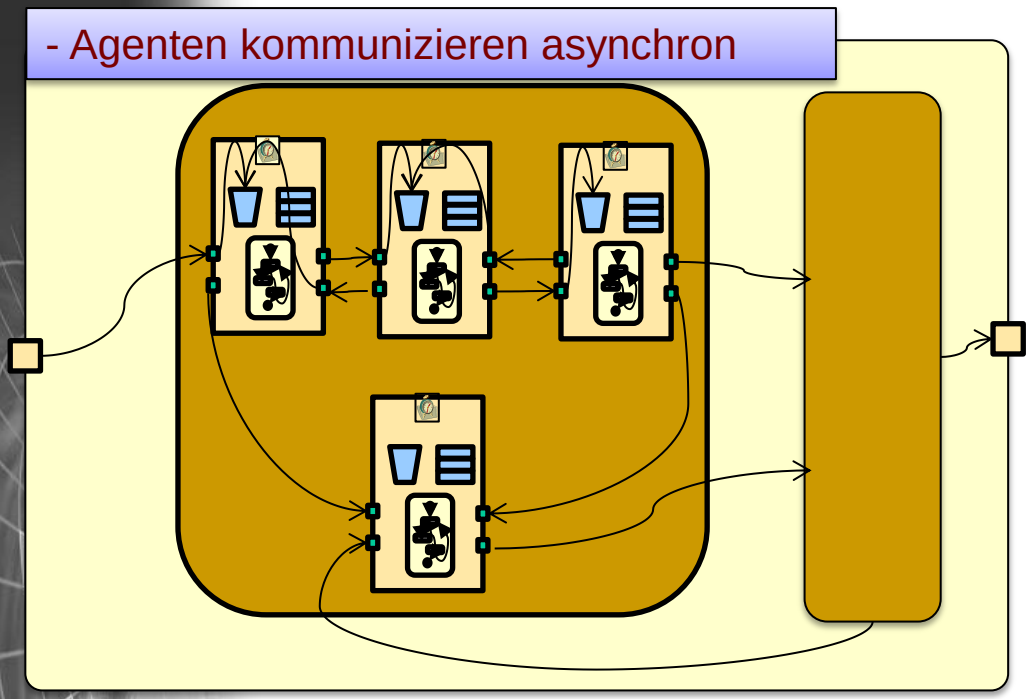
    zusätzl.
    optionale Wächterbedingung
  - der evtl. gestartete **Timer** läuft ab
    dann: vorgesehene Behandlung
  - die evtl. gestartete **Do-Aktivität** wird beendet
    → Auslösung eines impliziten Trigger-Ereignisses
    dann: vorgesehene Behandlung
    (Transition ohne TriggerAngabe)

    zusätzl.
    optionale Wächterbedingung
  - für den Zustand ist ein **Zustandsereignis** definiert (when …)
    und dieses tritt ein
    → Auslösung eines impliziten Trigger-Ereignisses
    dann: vorgesehene Behandlung
- und führt dann i.allg einen **Zustandsübergang** durch (nicht bei defer)

Verschiedene Trigger-Arten (Überblick)



System als Agent-Ensemble



UML-Probleme

- Pool-bearbeitung:
  sog. semantischer Variationspunkt
  (Reihenfolge, …)
- Adressierung von Signalen
  (nicht komplett gelöst,
  Bekanntmachung der Agenten/Ports)
- Übertragung von Signalen
  (ungelöst: Zeitverbrauch, Sicherheit)
- semantischer Variationspunkt
  Offenheit bzgl. Action-Sprache
  Datentypen, Anweisungen

Es gibt kein UML-Werkzeug, welches das Agent-Konzept allgemein unterstützt.
ABER: es gibt praktikable Teillösungen
SDL (Specification and Description Language)