ModSoft

Modellbasierte Software-Entwicklung mit UML 2 im WS 2014/15

Teil IIa: Strukturmodellierung

Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

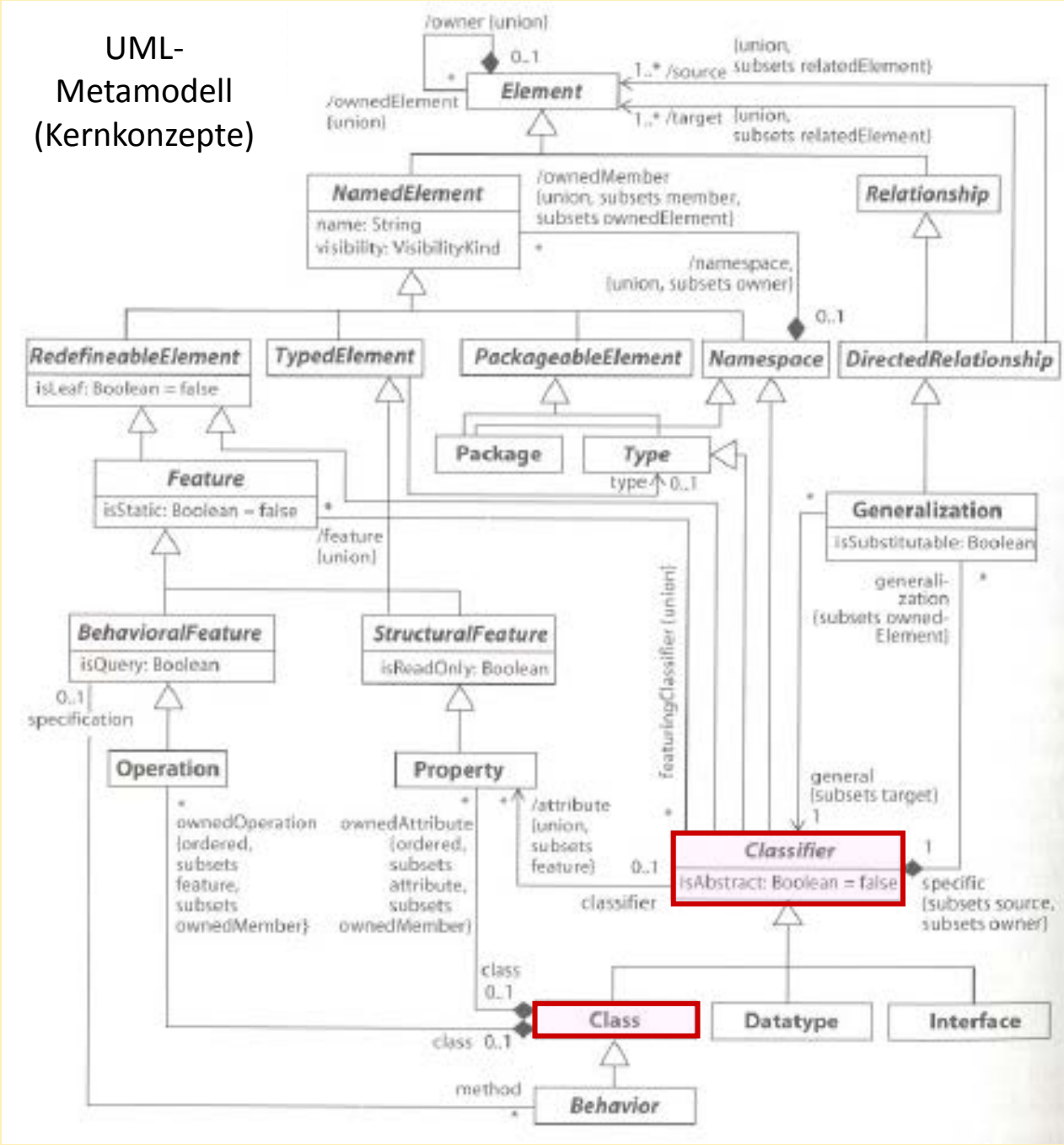
M3
MOF 2.0

M2
UML 2.0

M1
Benutzer-modell

M0
Objekte der Realität

UML-Metamodell (Kernkonzepte)



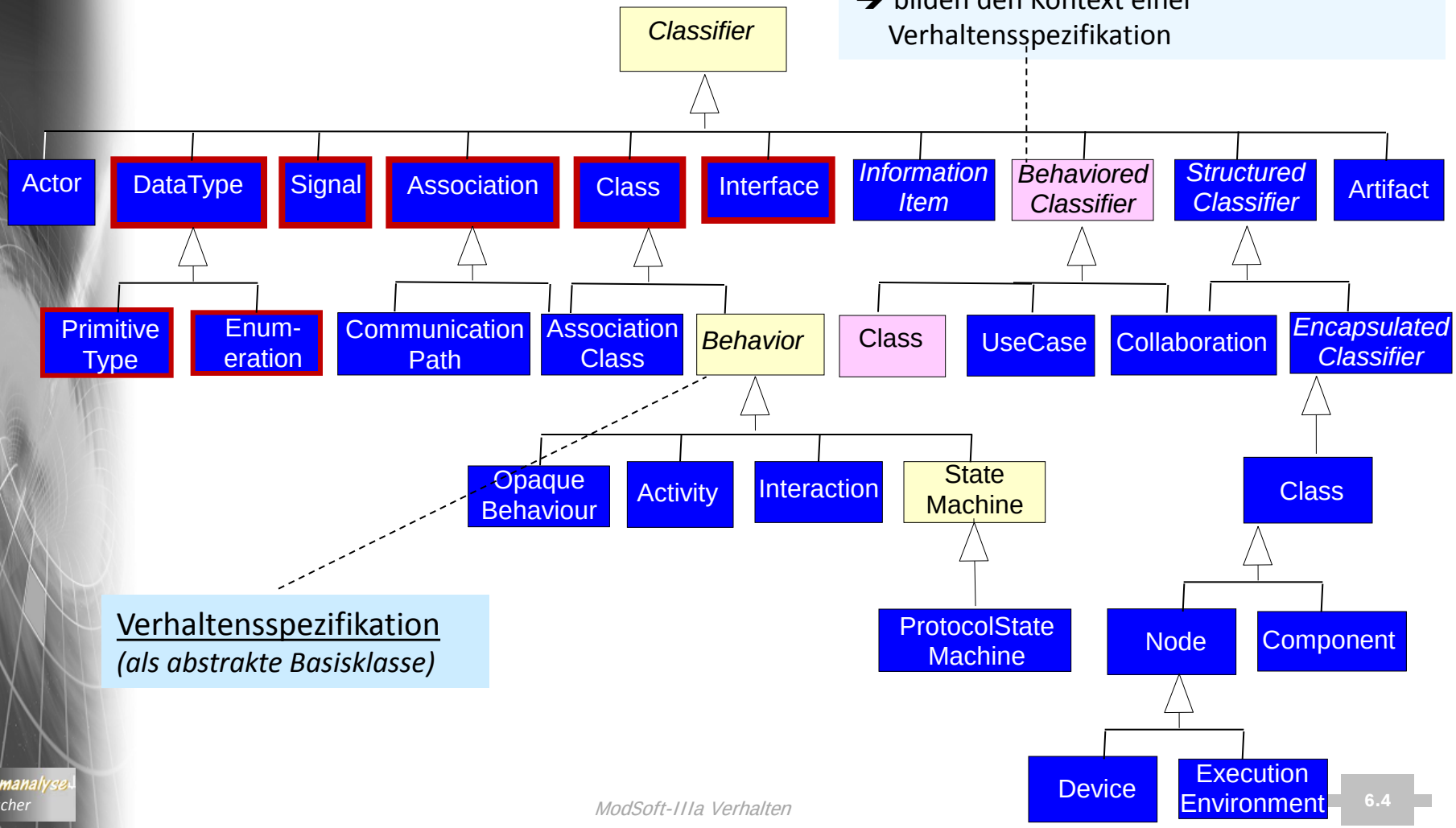
Inhalt (letzte Vorlesung)

1. Klassen und Assoziationen (zweiter Eindruck)
2. Interface
    a. Konzept und Anwendung
    b. Vertiefung von Abhängigkeitsbeziehung
    c. Vertiefung von Erweiterungsbeziehungen
       Spezialisierung - Stereotypisierung
3. Datentyp, Signal, Signalempfangsspezifikation
4. Klassen und Verhaltensbeschreibungen (erster Eindruck)

Übersicht von Classifier (UML-Metamodell)

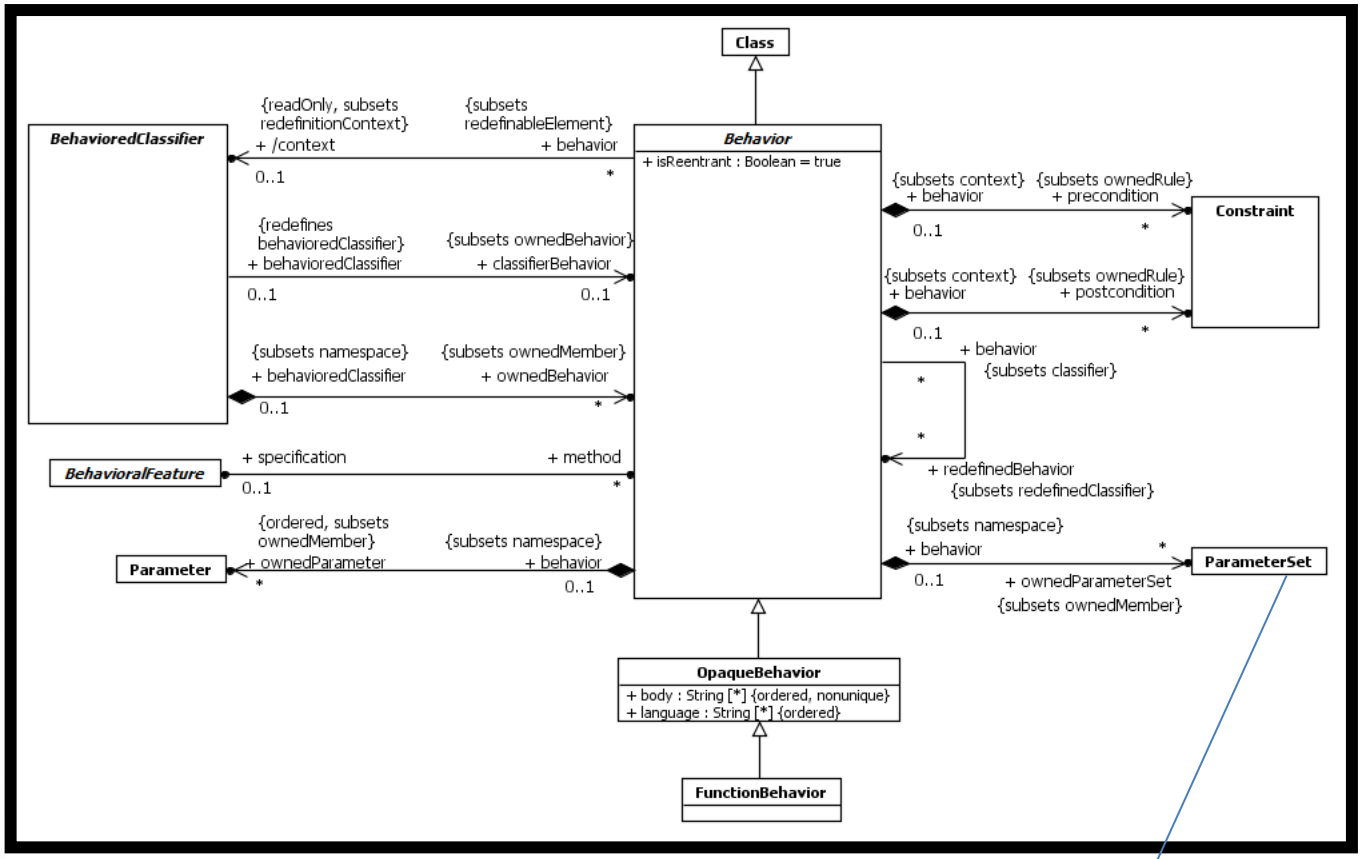
A **BehavioredClassifier** may have InterfaceRealizations, and owns a set of Behaviors one of which may specify the behavior of the BehavioredClassifier itself.

verhaltensspezifischer Classifier:
beschreibt alle Classifier, die eine Verhaltensspezifikation besitzen dürfen
➔ bilden den Kontext einer Verhaltensspezifikation



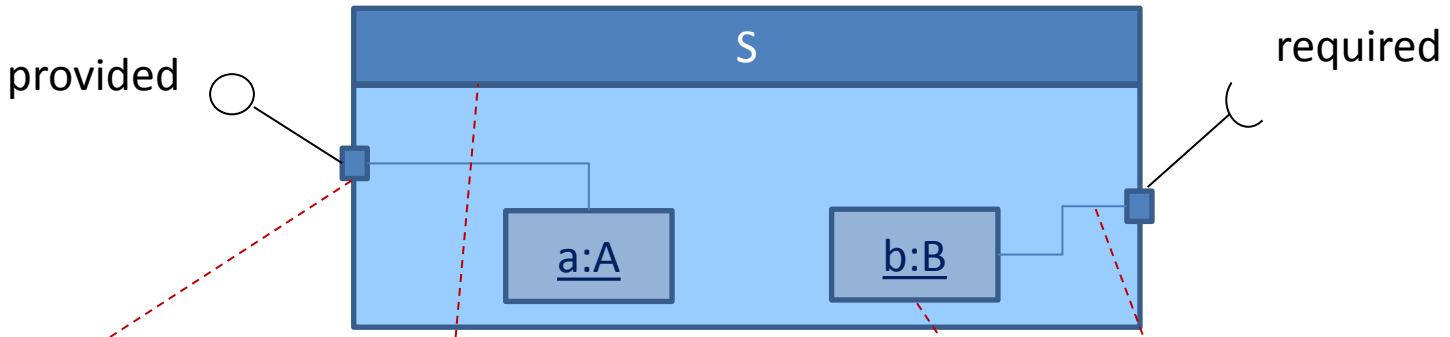
Verhaltensspezifikation
(als abstrakte Basisklasse)

Behaviour (UML-Metamodell)



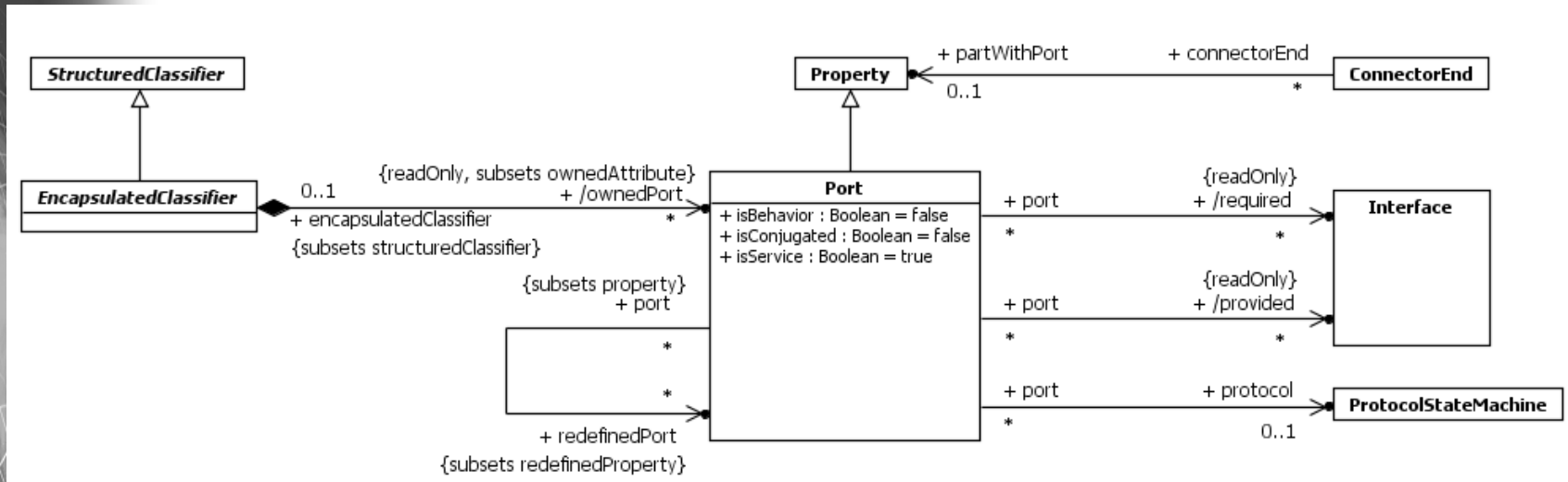
- kann parametrisiert werden (Operation, Zustandsautomat, ...)
- kann instanziiert werden (Instanz entspricht Ausführung von Behavior)
- isReentrant==true: erlaubt weitere Instanziierung (auch wenn erste noch nicht komplett ausgeführt sein sollte)
- Vererbung möglich

Strukturierte Klasse, Port (Interaktionspunkt), Konnektor

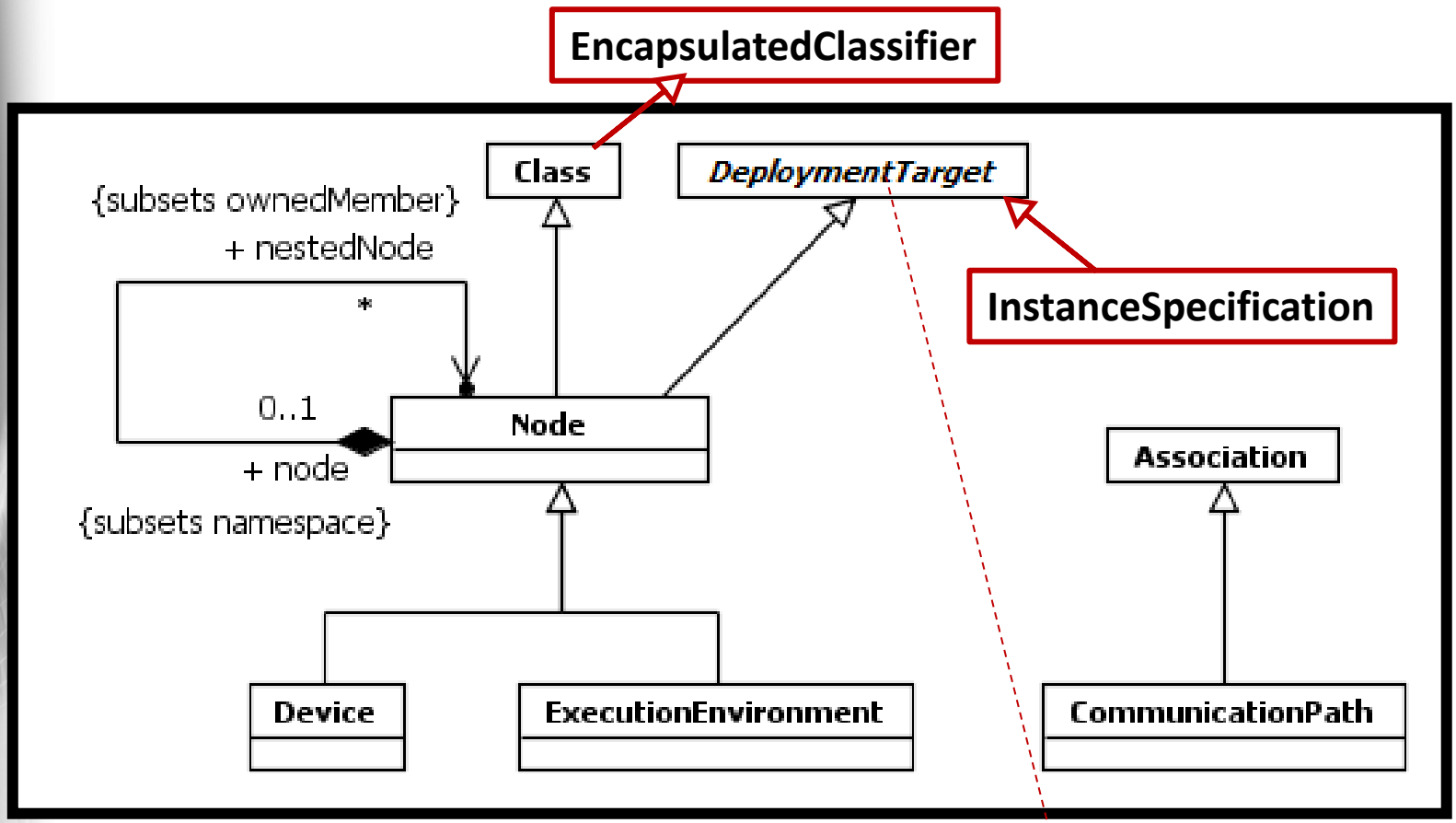


- A **Port** is a property of an **EncapsulatedClassifier** that specifies a distinct interaction point between that EncapsulatedClassifier and its environment or between the (behavior of the) EncapsulatedClassifier and its **internal parts**.
- Ports are connected to Properties of the EncapsulatedClassifier by **Connectors** through which requests can be made to invoke BehavioralFeatures.
  - Konnektoren spezifizieren Links (Instanzen von Assoziationen)
- A Port may specify the services an EncapsulatedClassifier provides (offers) to its environment as well as the services that an EncapsulatedClassifier expects (requires) of its environment.
- A Port may have an associated **ProtocolStateMachine**.

Port (UML-Metamodell)



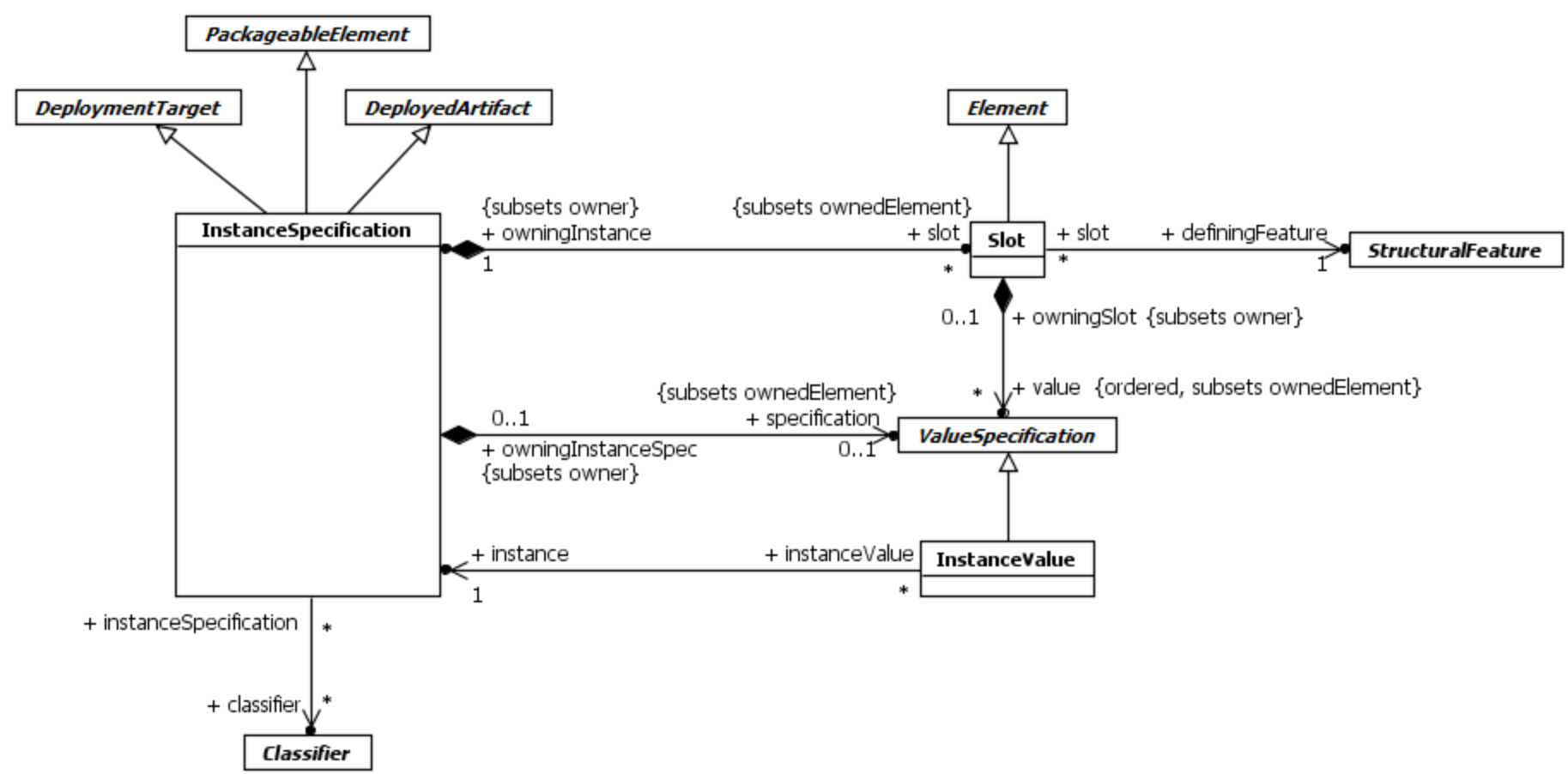
Kommunikationskanal



A **communication path** is an association between two deployment targets, through which they are able to exchange signals and messages

- Känale zwischen strukturierten Klassen nicht möglich (klar, Typdefinitionsniveau)
- Instanzspezifikationen sind aber erlaubt (z.B. Objekte strukturierter Klassen) !!!

Instanzen (Objekte , Links, …)



An **InstanceSpecification** represents the possible or actual existence of instances in a modeled system and completely or partially describes those instances.

Weitere Vorgehensweise

Ausgangspunkt:

- strukturierte Klassen,
- Objekte aktiver Klassen als Bestandteile
- asynchroner Austausch von Signalen (Grundverständnis Zustandsautomat)
- Instanzen der strukturierte Klasse und ihrer Bestandteile bilden ein System
- sie sind über Kommunikationspfade verbunden

Ziel:

- uns interessiert die Verhaltensbeschreibung solcher Art von Systemen (als Ensemble von Zustandsautomaten)

Probleme für Modellierung im aktuellen UML-Standard:

- offene Semantische Variationspunkte
- Signaladressierung, Empfänger- bzw. Port-Identifikation nicht definiert

  Lösung

Vorgehensweise:

- Zustandsautomat in UML
- Beispielbetrachtung mit ergänzenden Konzepten aus SDL

ModSoft

Modellbasierte Software-Entwicklung mit UML 2
im WS 2014/15

Teil III α: Verhaltensmodellierung

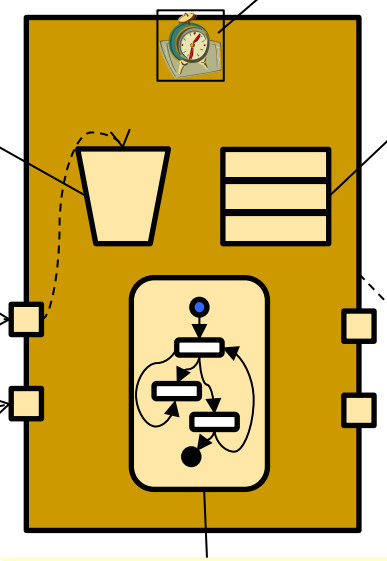
Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

Inhalt

1. Zuordnung: Strukturmodellentität – Verhaltensmodellentität
2. Semantik des UML-Zustandsautomaten (noch nicht vollständig)
3. Alternative Syntax (Ähnlichkeit zu SDL)
4. Dämonenspiel als „UML“-Modell (System)
   a) Informale Beschreibung
   b) SDL: instanz-basierte Definition
   c) Systemerweiterung
   d) UML: typ-basierte Definition
   e) Annahmen für eine mögliche Interpretation von UML
5. UML-SDL-Tool (PragmaDev)

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

Zustandsautomat (State Machine)

- Ausprägungen
  - Behavioral State Machines (internes Verhalten)
    - Spezifikation des Verhaltens individueller Classifier
    - objektorientierte Variante von State Charts von Harel
  - Protocol State Machines (externes Verhalten)
    - Spezifikation von Protokollen
    - Lifecycle als legale Aufrufszenerien bereitgestellter Operationen oder Ereignis-Trigger
      (geeignet für Interfaces und Ports)

Für konkrete Zuordnung von Strukturdiagramm (Kontext-Classifier) und Zustandsmaschine

gibt es **keine** UML-Notation (tool-abhängig)

ABER …

ALF

October 2013

OBJECT MANAGEMENT GROUP

Action Language for Foundational UML (Alf)

Concrete Syntax for a UML Action Language

Version 1.0.1

OMG Document Number: formal/2013-09-01

Standard document URL: http://www.omg.org/spec/ALF/1.0.1

Associated File(s):

Normative: http://www.omg.org/spec/ALF/20120827/Alf-Syntax.xmi
http://www.omg.org/spec/ALF/20120827/ActionLanguage-Profile.xmi
http://www.omg.org/spec/ALF/20130315/Alf-Library.xmi

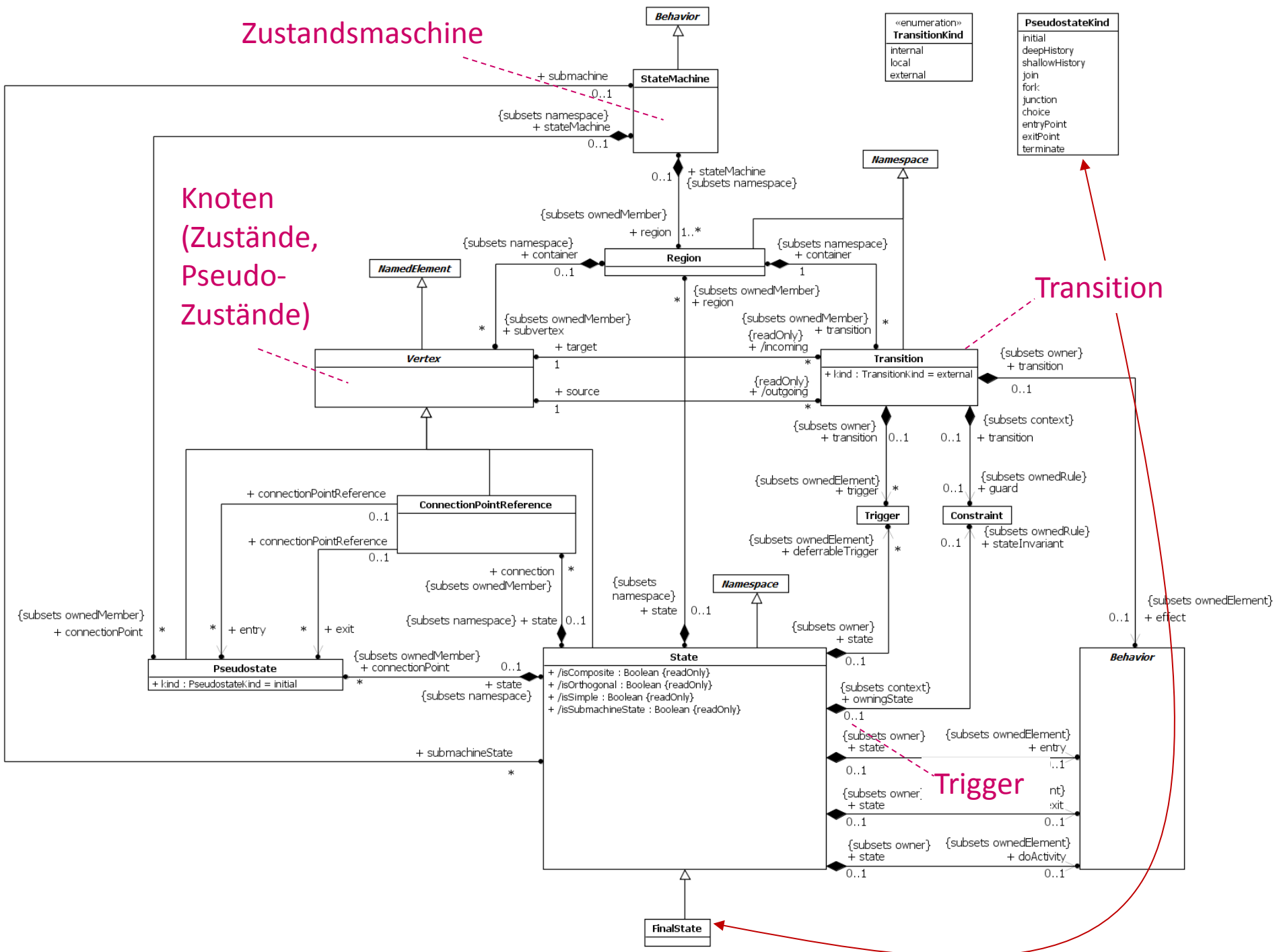
Beispiel aus dem ALF-Standard

```
active class Order { // An active class
        public dateOrderPlaced: Date; // Attribute definitions
        public totalValue: Money;
        public deliveryAddress: MailingAddress;
        public contactPhone: TelephoneNumber;

        public receive CheckOut; // Reception definitions
        public receive SubmitCharge;
        public receive PaymentDeclined;
        public receive PaymentApproved;
        public receive OrderDelivered;

} do Order_Behavior // Classifier behavior stub
```

… demonstriert die fehlende Verbindung


Zustandsmaschine
Knoten (Zustände, Pseudo-Zustände)
Transition
Trigger
Behavior
StateMachine
«enumeration» TransitionKind
internal
local
external
PseudostateKind
initial
deepHistory
shallowHistory
join
fork
junction
choice
entryPoint
exitPoint
terminate
Namespace
NamedElement
Region
Vertex
Transition
+ kind : TransitionKind = external
ConnectionPointReference
Pseudostate
+ kind : PseudostateKind = initial
State
+ /isComposite : Boolean {readOnly}
+ /isOrthogonal : Boolean {readOnly}
+ /isSimple : Boolean {readOnly}
+ /isSubmachineState : Boolean {readOnly}
Trigger
Constraint
FinalState
+ submachine
0..1
{subsets namespace}
+ stateMachine
0..1
+ stateMachine
{subsets namespace}
{subsets ownedMember}
+ region
1..*
{subsets namespace}
+ container
0..1
1
{subsets ownedMember}
+ region
{subsets ownedMember}
+ subvertex
{subsets ownedMember}
+ transition
{readOnly}
+ /incoming
+ target
1
+ source
1
{readOnly}
+ /outgoing
{subsets owner}
+ transition
0..1
{subsets context}
+ transition
{subsets ownedElement}
+ trigger
{subsets ownedRule}
+ guard
{subsets ownedElement}
+ deferrableTrigger
{subsets ownedRule}
+ stateInvariant
+ connectionPointReference
0..1
+ connection
{subsets ownedMember}
{subsets ownedMember}
+ connectionPoint
+ entry
+ exit
{subsets namespace}
+ state
0..1
{subsets namespace}
+ state
0..1
{subsets owner}
+ state
{subsets context}
+ owningState
{subsets ownedElement}
+ effect
{subsets owner}
+ state
{subsets ownedElement}
+ entry
{subsets owner}
+ state
{subsets ownedElement}
+ doActivity
+ submachineState

Inhalt

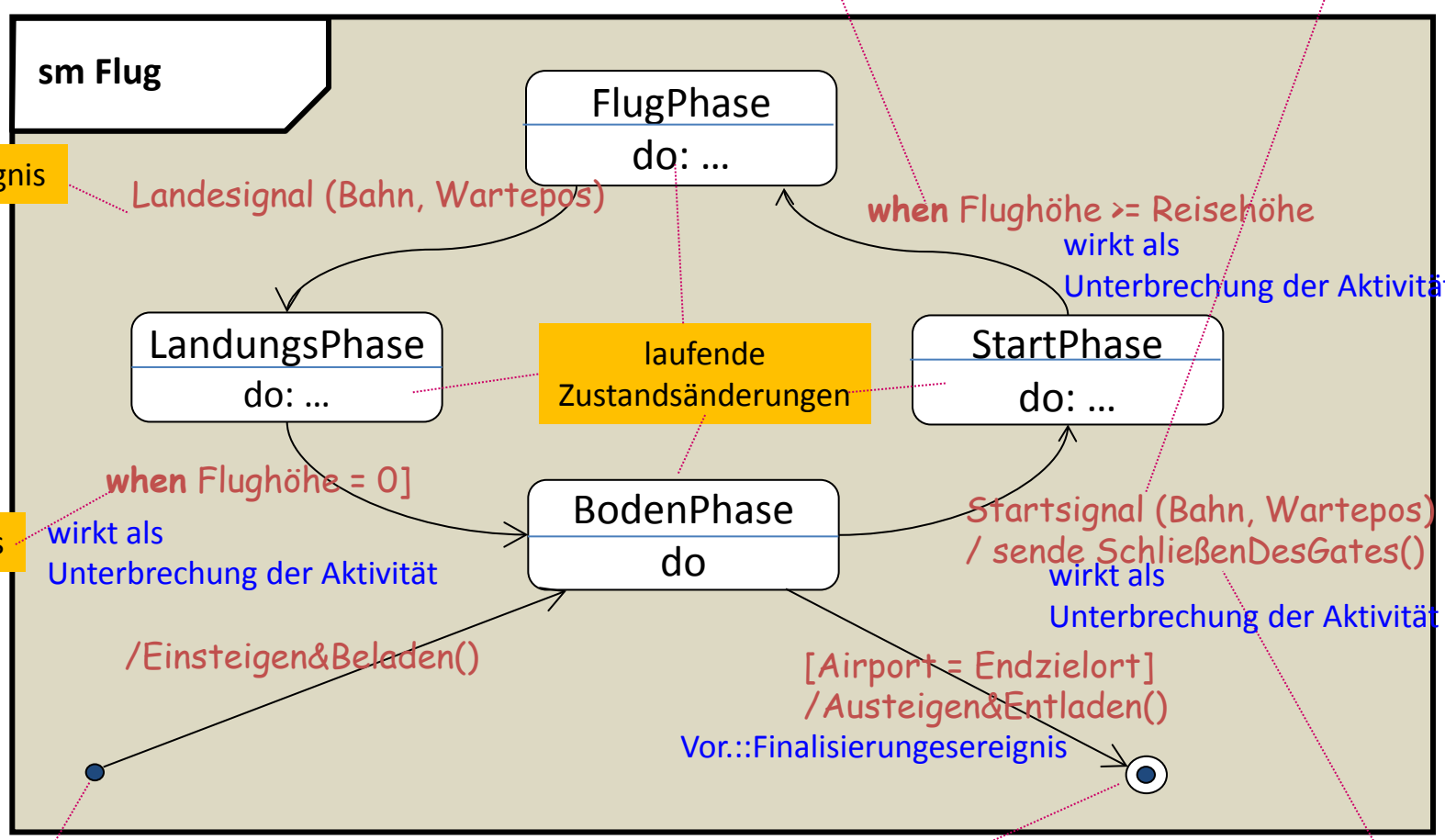
1. Zuordnung: Strukturmodellentität – Verhaltensmodellentität
2. Semantik des UML-Zustandsautomaten (noch nicht vollständig)
3. Alternative Syntax (Ähnlichkeit zu SDL)
4. Dämonenspiel als „UML“-Modell (System)
   a) Informale Beschreibung
   b) SDL: instanz-basierte Definition
   c) Systemerweiterung
   d) UML: typ-basierte Definition
   e) Annahmen für eine mögliche Interpretation von UML
5. UML-SDL-Tool (PragmaDev)

Zustandsautomat: *Semantik*

- **Besitzer (Kontext-Objekt)**: vom Typ Behaviored Classifier
  - legt erlaubte Empfangsmenge von Signal-Trigger und Call-Trigger für den Automaten fest
  - verfügt über Attribute und Operationen, die in den Aktivitäten des Automaten genutzt werden können
  - Automat kann auch nur Verhalten für ein einzelnes „Behavioral Feature“ (z.B. Operation) spezifizieren
    - in diesem Fall wird der Automat entsprechend dem Merkmal des Behaviored Classifier parametrisiert

- **Ausführung**:
  - erfolgt getriggert durch Ereignisse (Events)
  - Ereignisse bestimmen die auszuführenden Zustandsübergänge und die damit verbundenen Aktivitäten (Zustandsaktivitäten, Übergangsaktivitäten)
  - Ereignis: Signalankunft, Beendigung einer Operation, bestimmte Zustandsänderung

Beispiel



Zustandsereignis

auslösendes Ereignis (Trigger)

auslösendes Ereignis

laufende Zustandsänderungen

Zustandsereignis

Einstiegspunkt
Startzustand
(mit Aktivierung des Classifier-Objektes)

Finalzustand
(verbunden mit Terminierung des Zustandsautomaten, **nicht** des Kontext-Objektes)

Auslösen
Eines
Sende-Ereignisses
Syntax???

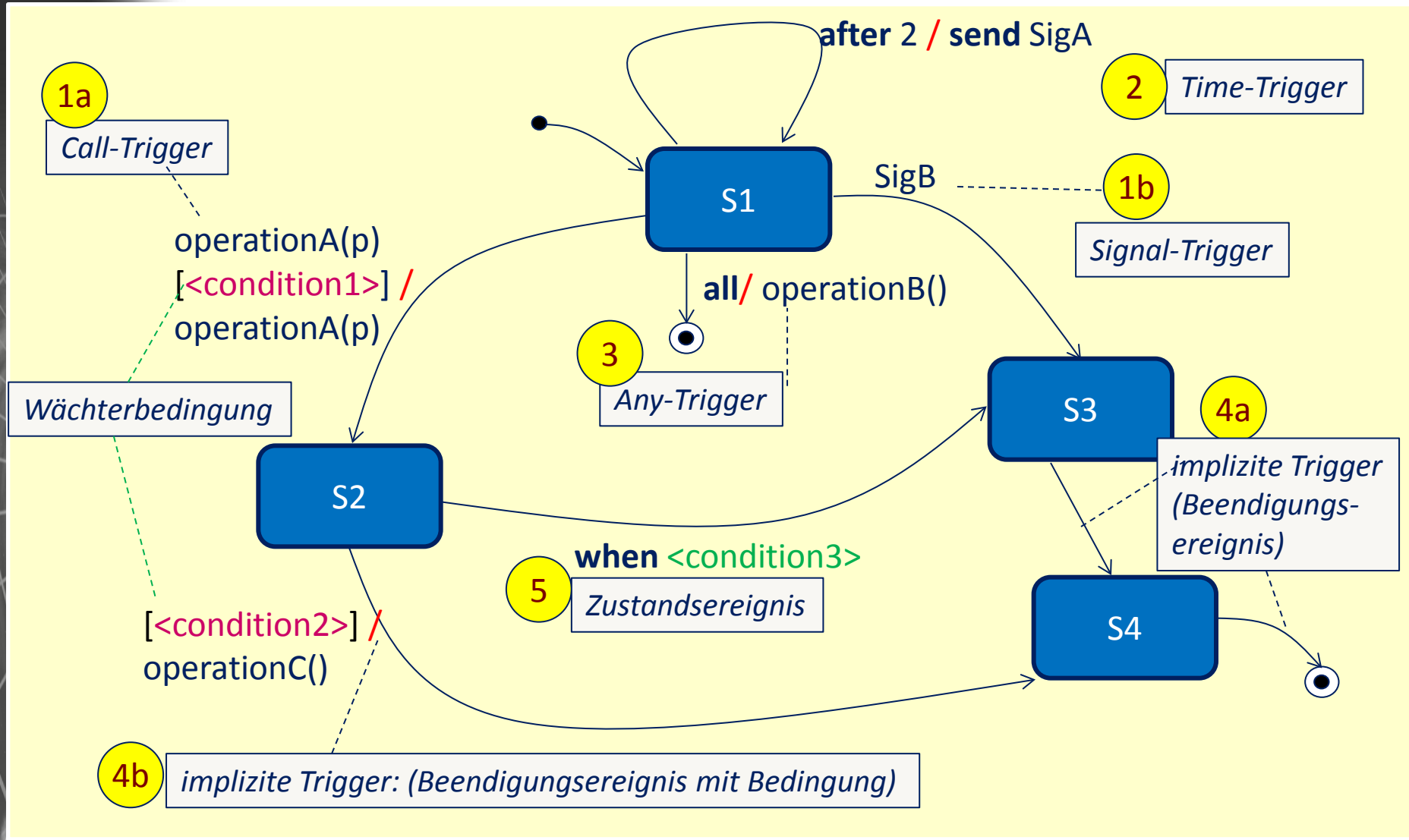
Allgemeine Arbeitsweise

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

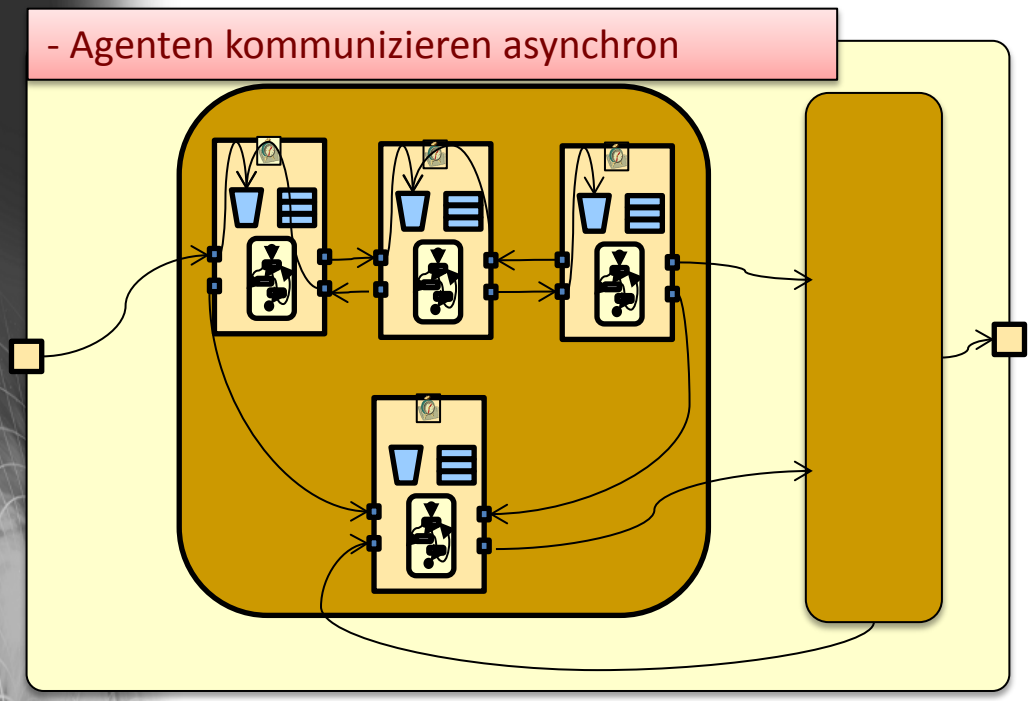
    zusätzl.
    optionale Wächterbedingung
  - der evtl. gestartete **Timer** läuft ab
    dann: vorgesehene Behandlung
  - die evtl. gestartete **do-Aktivität** wird beendet
    → Auslösung eines FinalisierungsEreignisses
    dann: vorgesehene Behandlung
    (Transition ohne TriggerAngabe)

    zusätzl.
    optionale Wächterbedingung
  - für den Zustand ist ein **Zustandsereignis** definiert (**when** …)
    und dieses tritt ein
    → Auslösung eines impliziten Trigger-Ereignisses
    dann: Unterbrechung der evtl. Do-Aktivität, 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

AutomateninstanzReferenztyp: **Pid**
NULL (als einziges Literal)
==, =/= (Operatoren)

Vereinbarungen

Jede Automateninstanz verfügt über

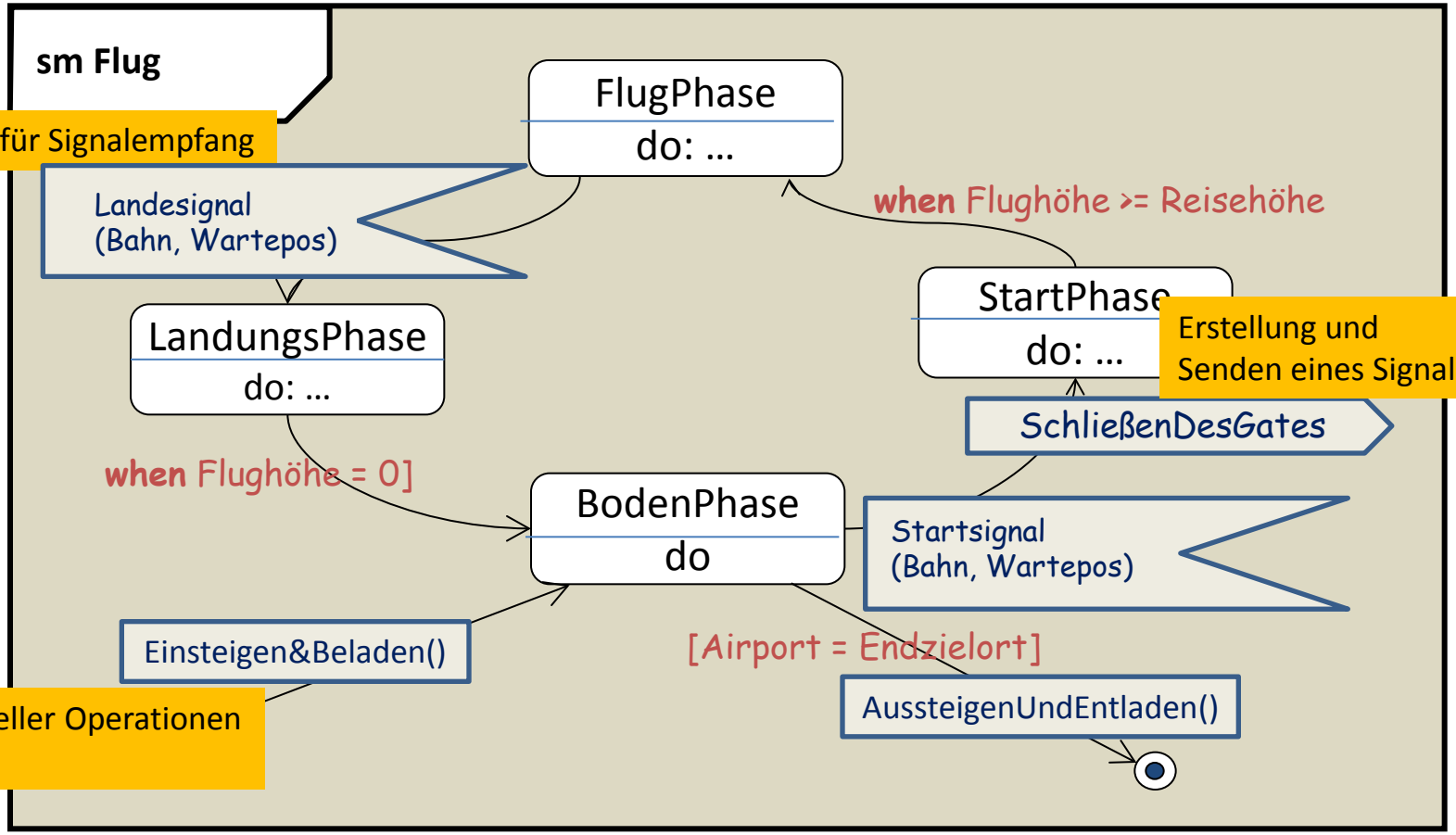
- **myself** ~ Referenz zum zugehörigen Objekt einer aktiven Klasse
- **self** ~ Automateninstanzreferenz auf sich selbst
- **sender** ~ Automateninstanzreferenz zum Sender des aktuellen Signals
- **parent** ~ Automateninstanzreferenz zum Erzeuger
- **newprocess** ~ Erzeugung eines Objekts und Aktivierung der Automateninstanz
- **offspring** ~ Automateninstanzreferenz zur letzten erzeugten Automateninstanz

in SDL identisch

Inhalt

1. Zuordnung: Strukturmodellentität – Verhaltensmodellentität
2. Semantik des UML-Zustandsautomaten (noch nicht vollständig)
3. Alternative Syntax (Ähnlichkeit zu SDL)
4. Dämonenspiel als „UML“-Modell (System)
   a) Informale Beschreibung
   b) SDL: instanz-basierte Definition
   c) Systemerweiterung
   d) UML: typ-basierte Definition
   e) Annahmen für eine mögliche Interpretation von UML
5. UML-SDL-Tool (PragmaDev)

Beispiel mit SDL-ähnlicher Syntax



Inhalt

1. Zuordnung: Strukturmodellentität – Verhaltensmodellentität
2. Semantik des UML-Zustandsautomaten (noch nicht vollständig)
3. Alternative Syntax (Ähnlichkeit zu SDL)
4. Dämonenspiel als „UML“-Modell (System)
   a) Informale Beschreibung
   b) SDL: instanz-basierte Definition
   c) Systemerweiterung
   d) UML: typ-basierte Definition
   e) Annahmen für eine mögliche Interpretation von UML
5. UML-SDL-Tool (PragmaDev)

Daemon Game: Anforderungen

Idee

- man hat als Spieler (Systemumgebung) zu erraten,
  ob eine Black-Box-Variable ungerade ist
- die Variable wird von einem Dämon geändert: gerade (even)←→ungerade(odd)
  (nicht vorhersagbar)
- liegt man mit seiner Vermutung richtig, erhält der Spieler einen Punkt,
  wenn nicht, wird ihm ein Punkt entzogen
- zulässig ist eine sich dynamisch ändernde Anzahl von Spielern,
  die sich an- bzw. abzumelden haben
- Ein Spieler kann zu einem Zeitpunkt nur maximal an einem Spiel angemeldet sein

Aufgabe

- Spiel ist als reaktives System zu beschreiben, jede Spielanforderung muss unabhängig von den anderen bearbeitet werden
- Spiel und Spieler kommunizieren per Signalaustausch

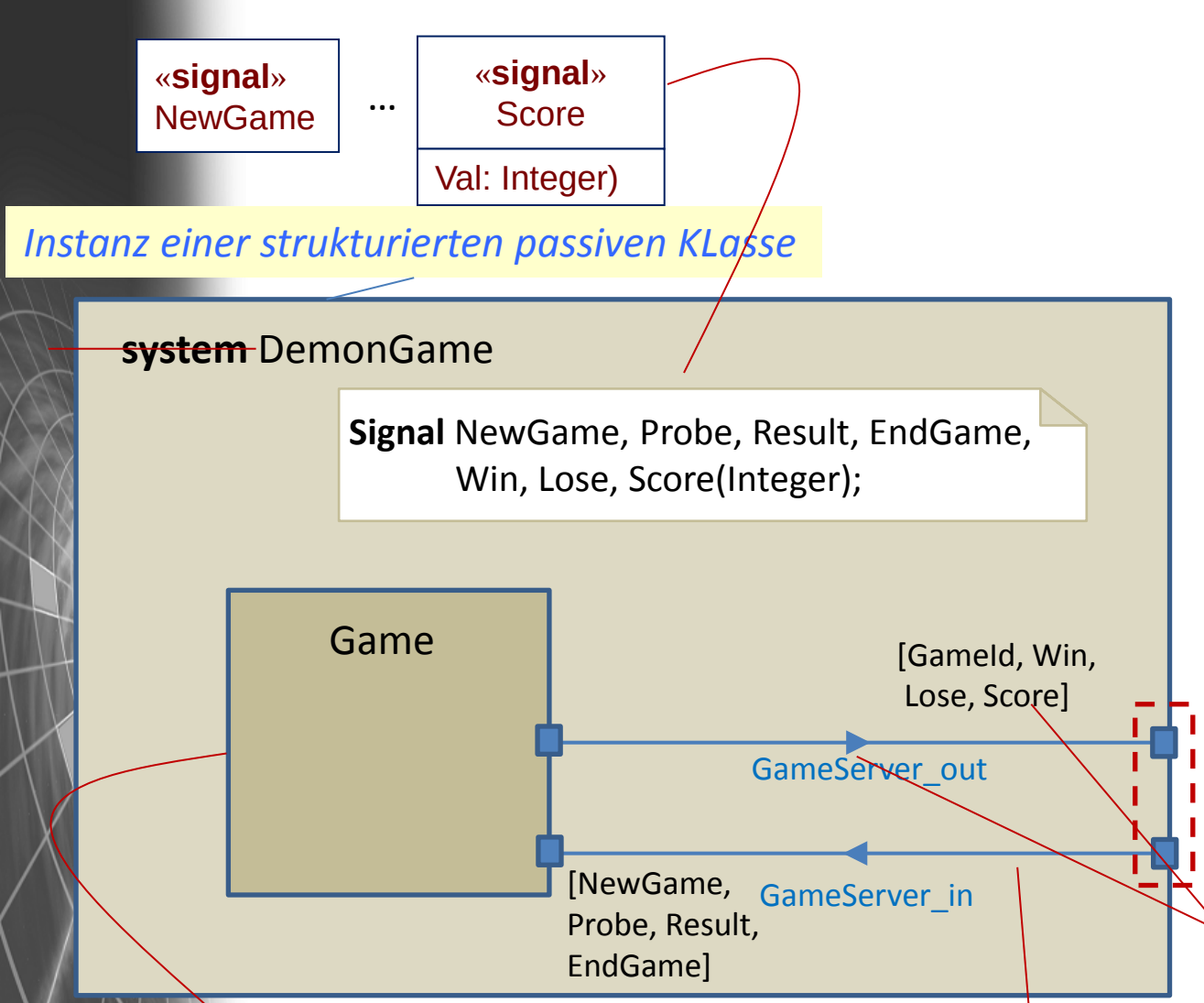
Probleme

- Reaktivität, dynamische Änderung der SYstemstruktur, Verteilung

Inhalt

1. Zuordnung: Strukturmodellentität – Verhaltensmodellentität
2. Semantik des UML-Zustandsautomaten (noch nicht vollständig)
3. Alternative Syntax (Ähnlichkeit zu SDL)
4. Dämonenspiel als „UML“-Modell (System)
   a) Informale Beschreibung
   b) SDL: instanz-basierte Definition
   c) Systemerweiterung
   d) UML: typ-basierte Definition
   e) Annahmen für eine mögliche Interpretation von UML
5. UML-SDL-Tool (PragmaDev)

Demon Game: als System



Systemfunktionalität
Spieler aus der Umgebung
- melden sich an
- spielen
  (raten richtig/falsch
  und bekommen
  Punkte)
- fragen Punktestand ab
- melden sich an

nur angemeldete Spieler dürfen spielen

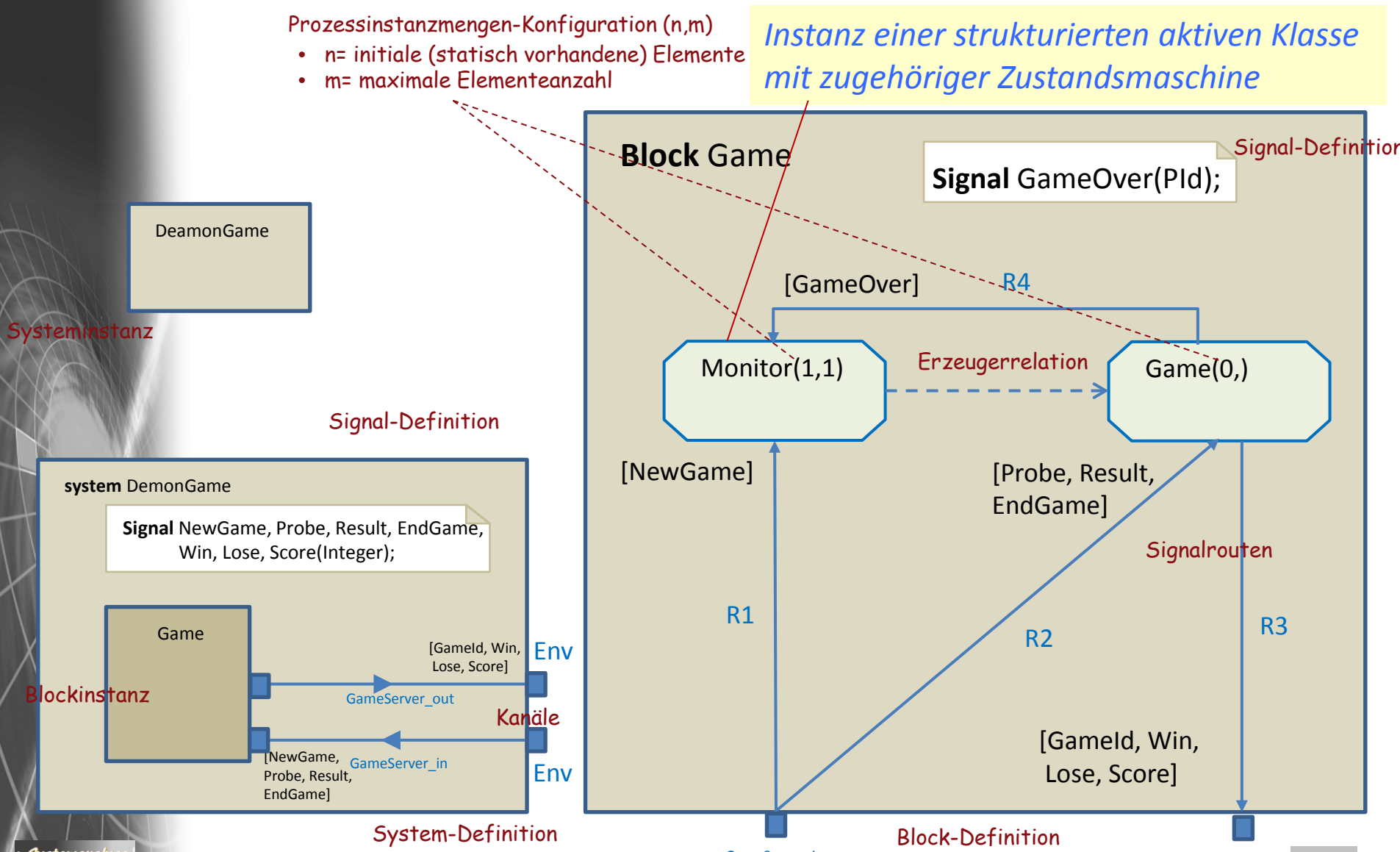
Ein Spieler kann zu einem Zeitpunkt nicht mehr als einmal angemeldet sein

Port mit zwei Interfaces

Richtung und Signalmenge bestimmen Interface

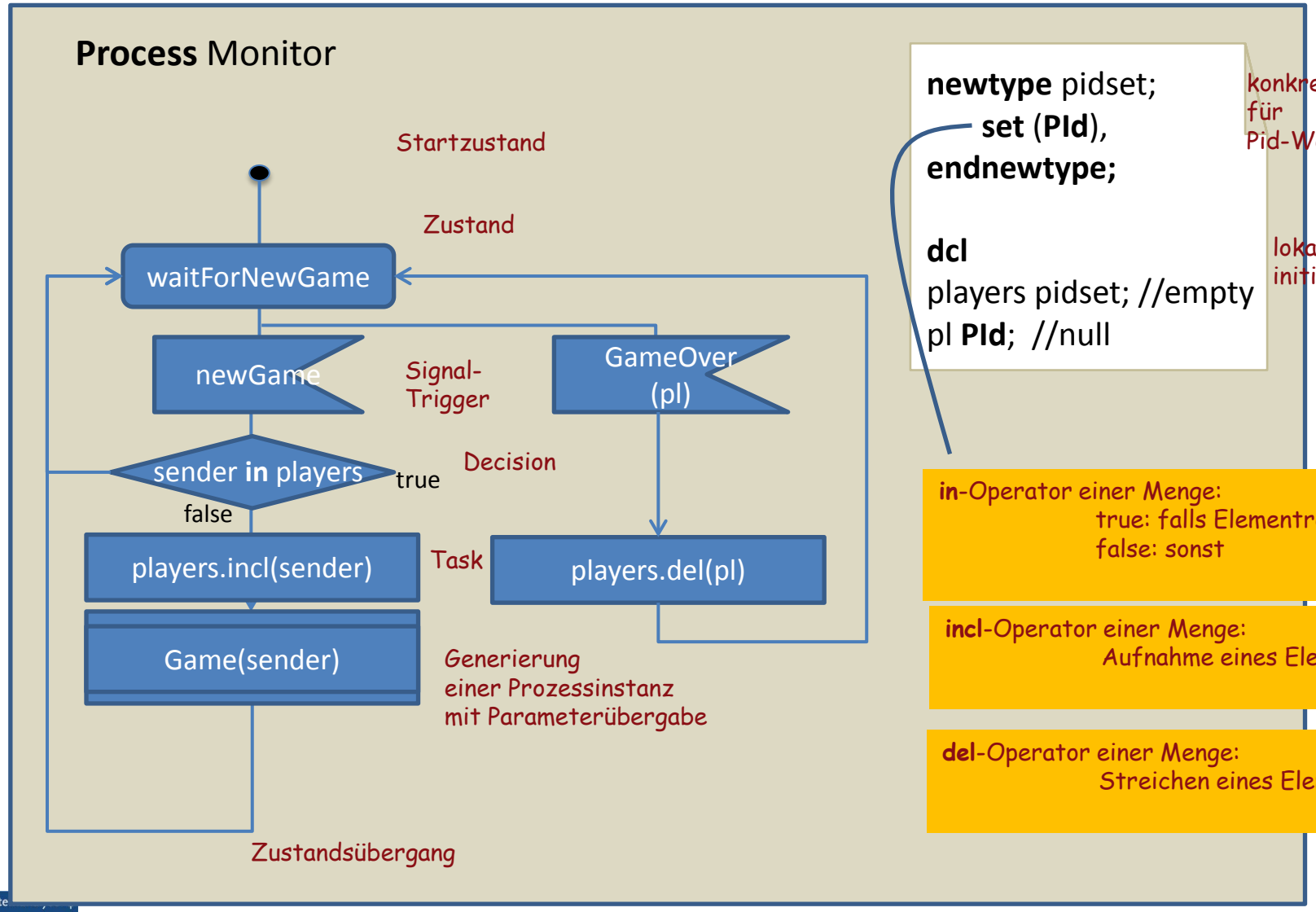
Konnektoren als Kommunikationspfade

Demon-Game: System-Block-Struktur



Demon-Game: Verhalten

Repräsentantendefinition der Menge Monitor



Demon-Game: Verhalten

Repräsentantendefinition der Menge Game