

### Arbeitsgruppe Software Engineering Prof. Elke Pulvermüller

Universität Osnabrück  
Institut für Informatik, Fachbereich Mathematik / Informatik  
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

[elke.pulvermueller@informatik.uni-osnabrueck.de](mailto:elke.pulvermueller@informatik.uni-osnabrueck.de)

<http://www.inf.uos.de/se>

Sprechstunde: mittwochs 14 – 15 und n.V.



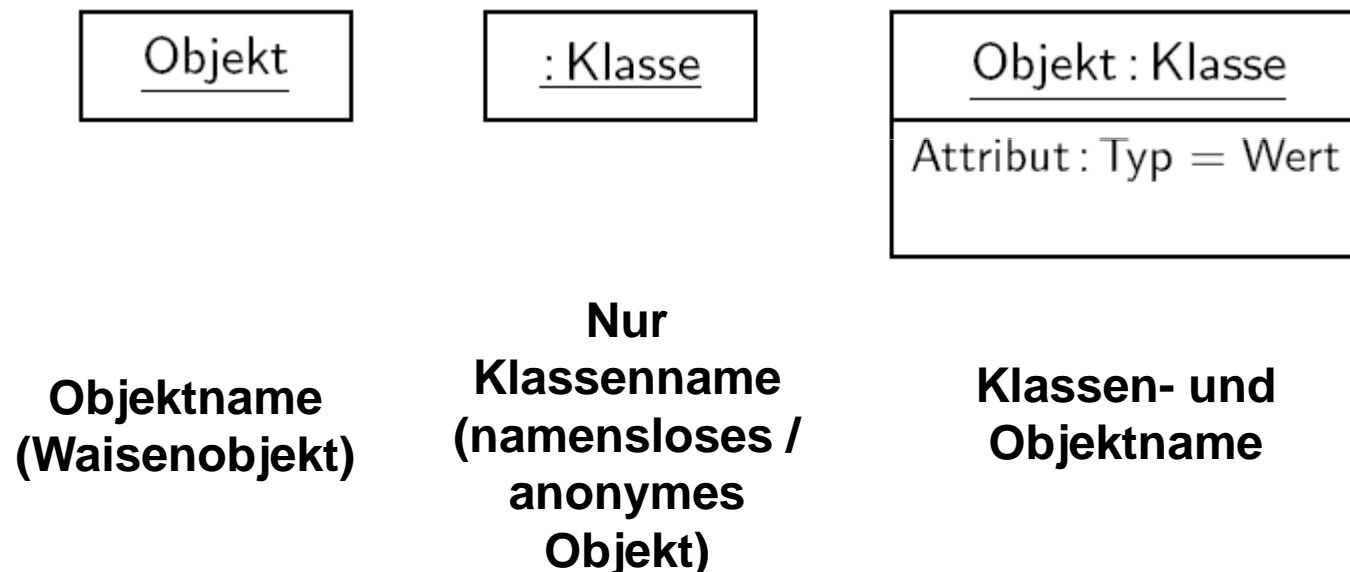
- 1 Software-Krise und Software Engineering**
- 2 Grundlagen des Software Engineering**
- 3 Projektmanagement**
- 4 Konfigurationsmanagement**
- 5 Software-Modelle**
- 6 Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**
- 7 Qualität**
- 8 ... Fortgeschrittene Techniken**

- 5.1 Grundlagen und Modelltypen**
- 5.2 Programmablaufplan**
- 5.3 Struktogramm**
- 5.4 Funktionsbaum**
- 5.5 Strukturierte Analyse (SA)**
- 5.6 EBNF und Syntaxdiagramm**
- 5.7 Entity-Relationship-Modell (ERM)**
- 5.8 Objektorientierte Modellierung mit UML**

## 5.8 OO Modellierung mit UML: Objektdiagramm

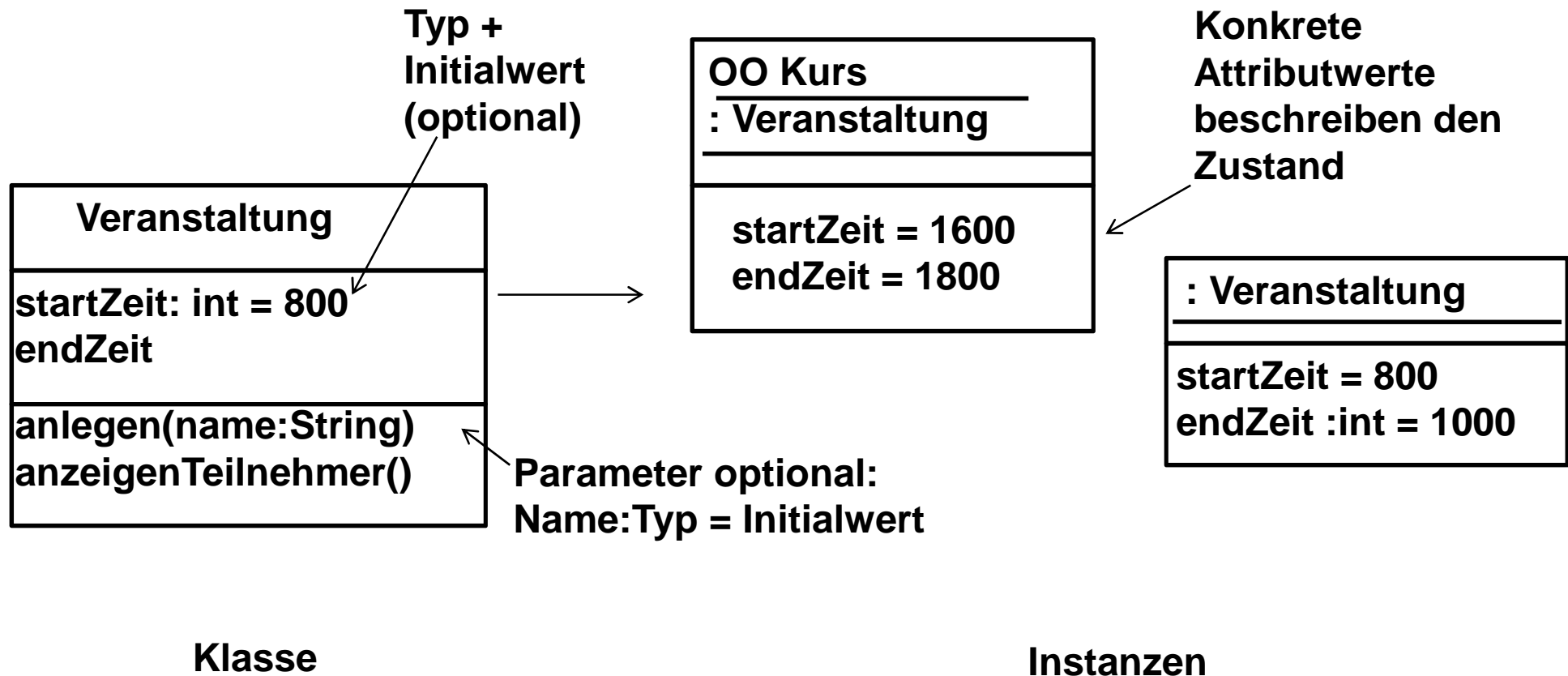
### Objektdiagramme

- zeigen eine Momentaufnahme des Systems
- beschreiben zu einem bestimmten Zeitpunkt die Menge der existierenden Objekte mit ihren augenblicklichen Attributwerten und ihrer Beziehungen untereinander
- Beziehungen heißen Links; sie können auch Namen und Rollen tragen



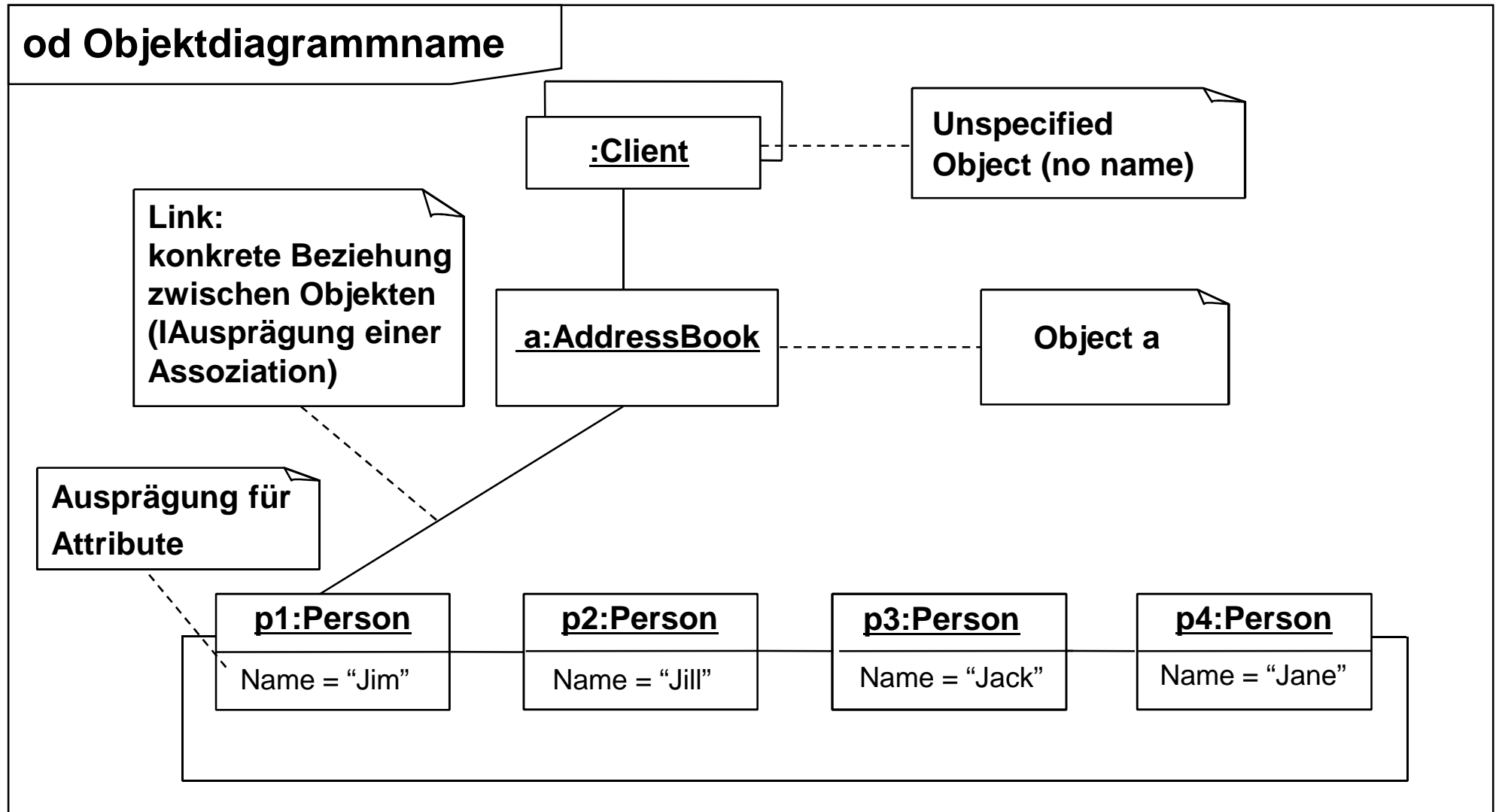
## 5.8 OO Modellierung mit UML: Objektdiagramm

### Repräsentation von Instanzen/Objekten in UML

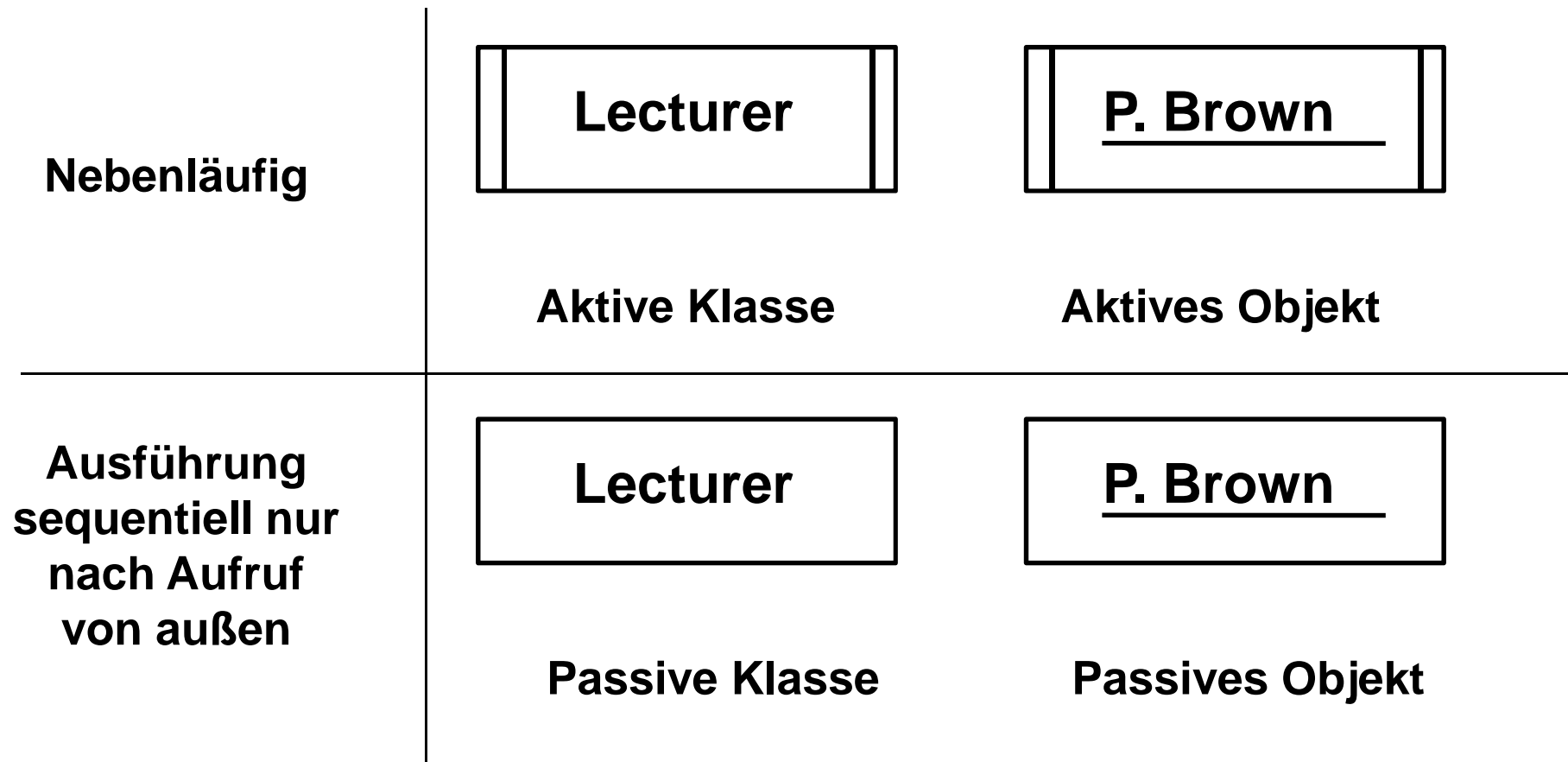


## 5.8 OO Modellierung mit UML: Objektdiagramm

### Beispiel für ein UML Objektdiagramm (Object Diagram):



### Aktive/passive Klassen und Objekte



### UML Kommunikationsdiagramm

- Gehört in die Gruppe der Interaktionsdiagramme (Interaction Diagrams)
- Vorgänger:

Object Diagram von G. Booch

→ Collaboration Diagram (UML 1.x)

→ Communication Diagram (UML 2.x)

**Achtung: Diagrammnamensverwirrung!**

- Modelliert einen Sachverhalt ähnlich zum Sequenzdiagramm, allerdings aus einer anderen Perspektive
- Modelliert für einen begrenzten Systemauszug die Menge der existierenden Objekte/Rollen, deren Attributwerte und Interaktionen
- Geeignet zur Erklärung und Dokumentation spezieller Ablaufsituationen



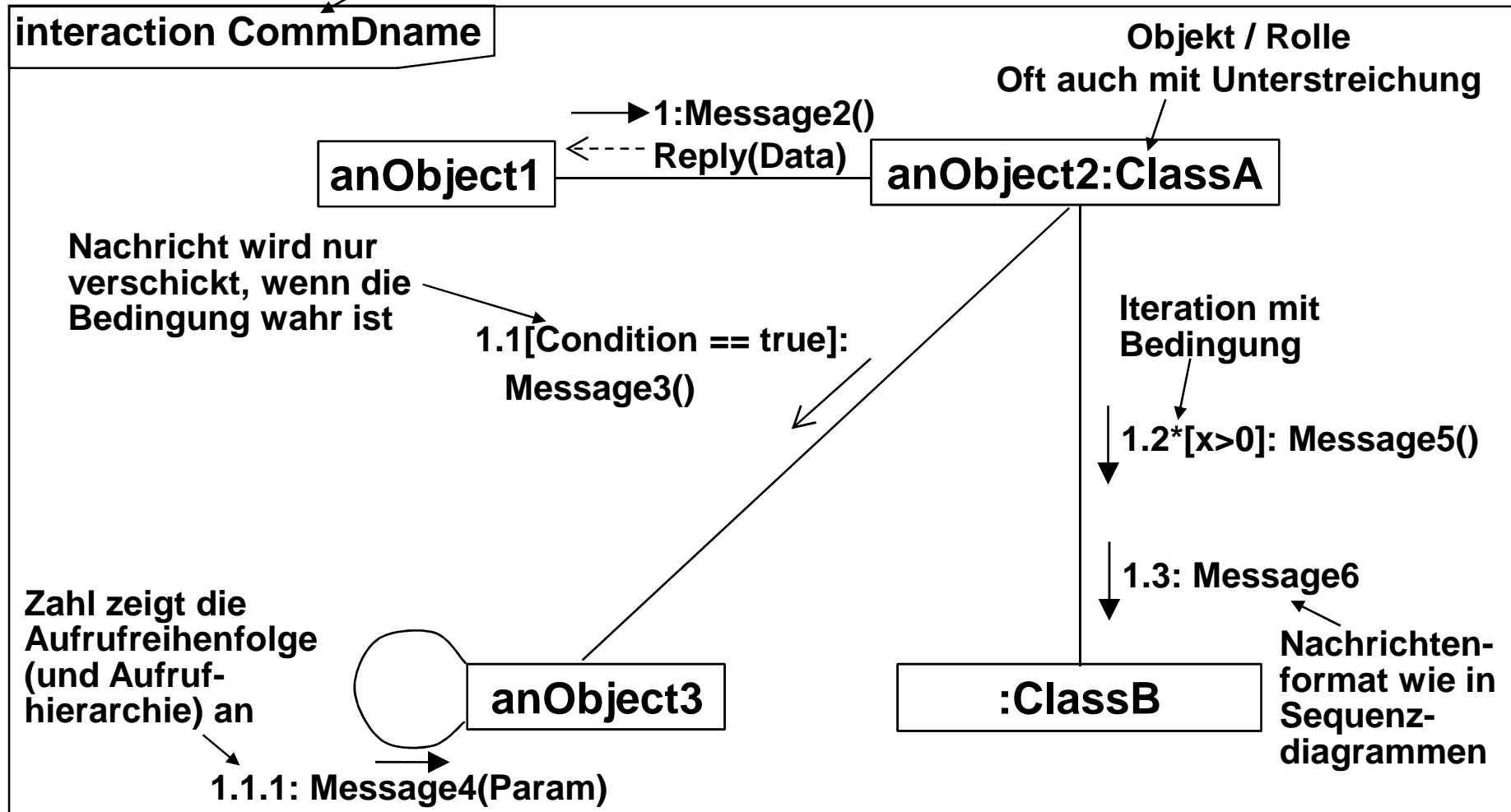
Nachrichten zwischen Objekten / Rollen:



- Wie in einem Sequenzdiagramm zeigen Pfeile die Nachrichten an:
  - ▶ Synchroner Nachricht
  - > Asynchroner Nachricht
  - <---- Rückantwort
- Der Pfeil zeigt vom Sender in Richtung des Empfängers.
- Die Reihenfolge der Nachrichten werden durch eine Nummerierung angegeben.
- Die Nachrichten können
  - sequentiell (Nummerierung: 1, 2, 3)
  - geschachtelt (Nummerierung: 2.3.1, 2.3.2, 2.1.1, 2.1.2)
  - nebenläufig (Nummerierung: 7a,7b,7c)modelliert werden.
- Die erste, aktionsauslösende Nachricht wird ohne Nummer dargestellt.

### Prinzipaufbau eines Kommunikationsdiagramms:

Schlüsselwort interaction oder sd  
gefolgt vom Namen des Diagramms

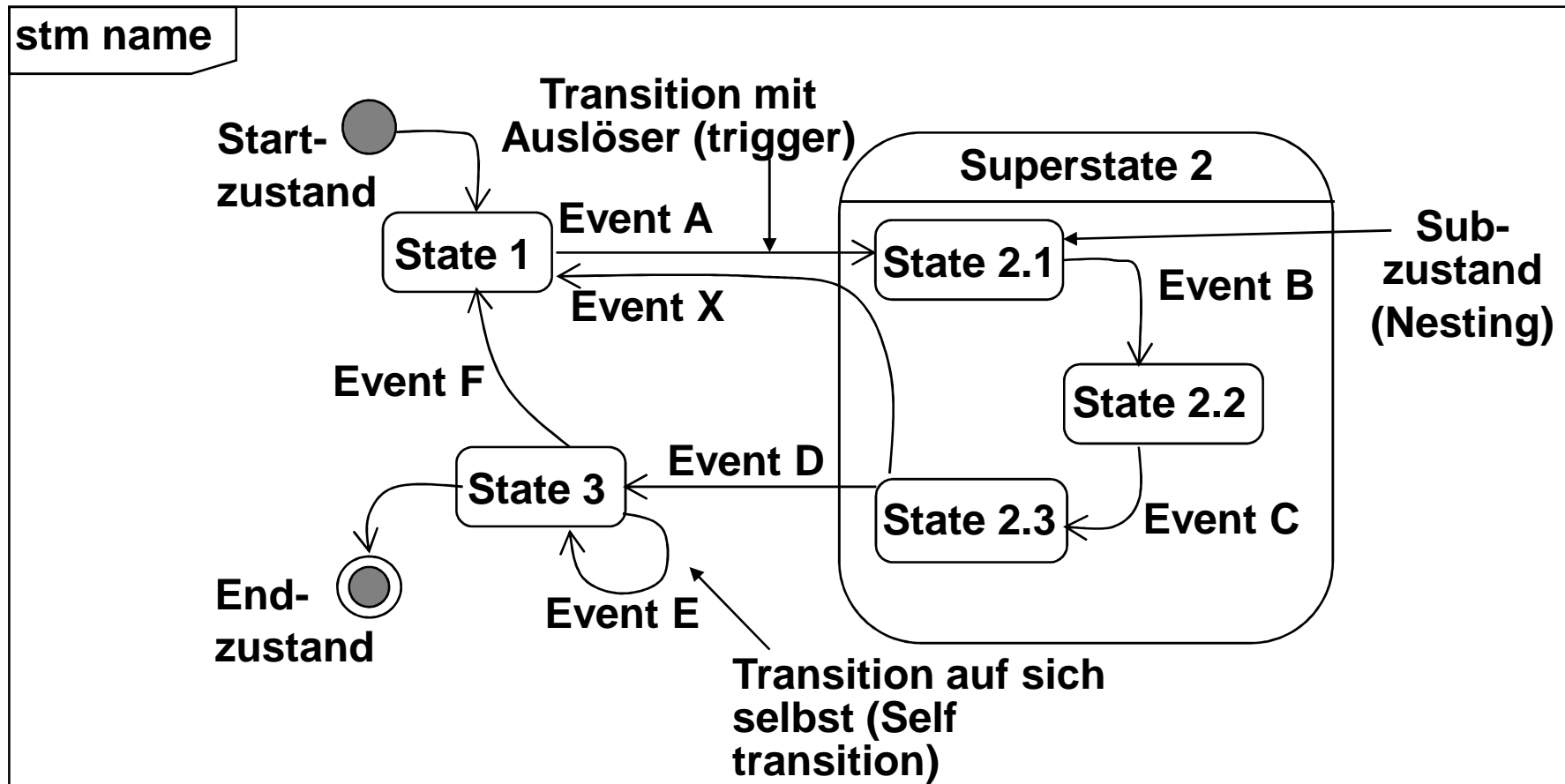


## 5.8 OO Modellierung mit UML: Zustandsdiagramm

- **Spezifikation des Verhaltens z.B.**
  - eines Objekts (mit den Zuständen innerhalb des Objekts)
  - eine Anwendungsfalls (Use Case)
  - eine Kommunikationsprotokolls / einer Schnittstellennutzung
  - einer GUI
- **NICHT geeignet zur Beschreibung von Interaktionen zwischen Objekten**
- **Ursprung: State Charts, David Harrel, 1987**  
**State Charts = extended state automata**
- **Endliche Zustandsautomaten beschreiben für ein Objekt die möglichen Zustände und deren Übergänge zusammen mit den Ereignissen, die die Übergänge initiieren.**
- **Endlicher Zustandsautomat (Finite state machine):**
  - Zustand (state):** Zeitspanne, in der das Objekt auf ein Ereignis wartet (besondere Zustände: Start- und Endzustände)
  - Transition (transition):** Zustandsübergang, verbindet zwei Zustände, wird durch ein Ereignis ausgelöst, hat keine Dauer, kann nicht unterbrochen werden
  - Ereignis (event):** tritt zu einem Zeitpunkt ein (ohne Dauer)

## 5.8 OO Modellierung mit UML: Zustandsdiagramm

### Prinzipaufbau von UML Zustandsübergangsdiagrammen (State Machine Diagram, State Transition Diagram / State Chart)

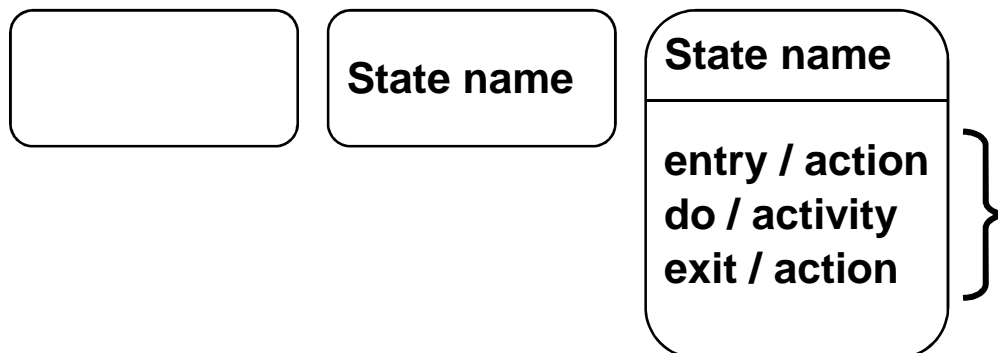
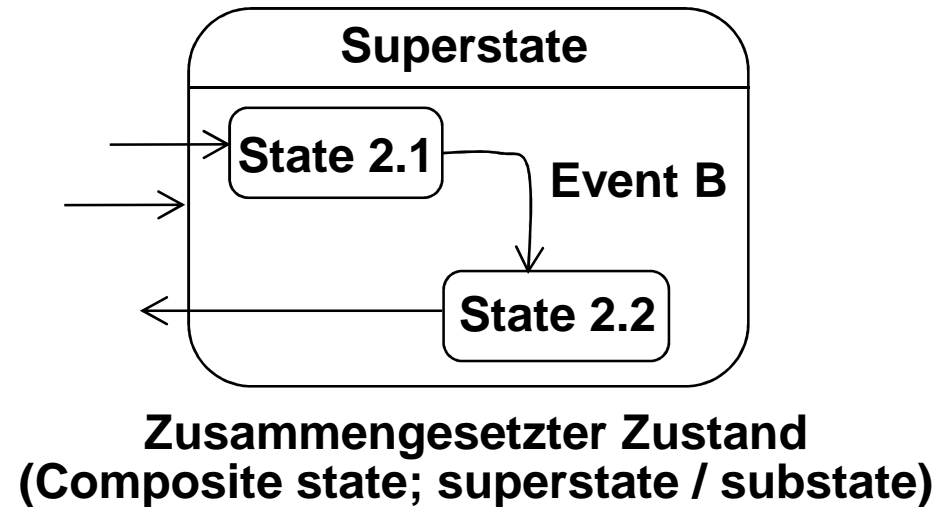


#### Verschachtelung (Nesting):

State 2 ist ein Superzustand, der die Subzustände State 2.1, State 2.2 und State 2.3 enthält.

## 5.8 OO Modellierung mit UML: Zustandsdiagramm

### ▪ Zustände:



**Besondere interne Aktionen (Aktionen ohne Zustandsänderung):**

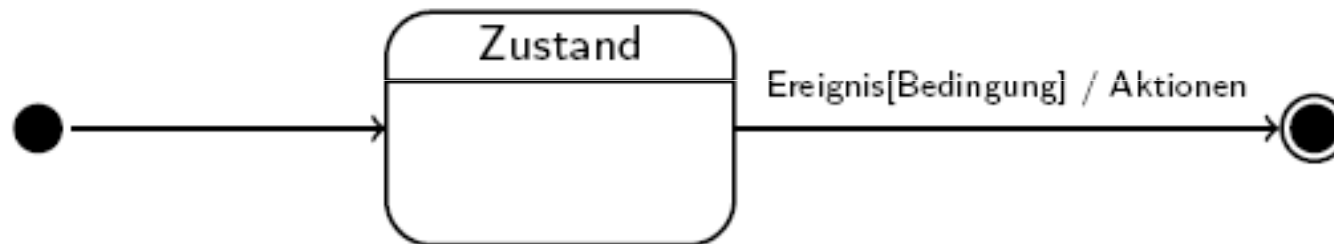
- Aktion wird automatisch ausgeführt, wenn der Zustand betreten wird (von kurzer Dauer, z.B. Versenden einer Nachricht)
- Aktivität wird ausgeführt bis der Zustand verlassen wird (mit Dauer)
- Aktion wird beim Verlassen des automatisch Zustands ausgeführt.

**Anonymer  
Zustand**

**Gewöhnlicher  
Zustand**

**Zustand mit internen Aktionen**  
Format: Event [Guard] / Action

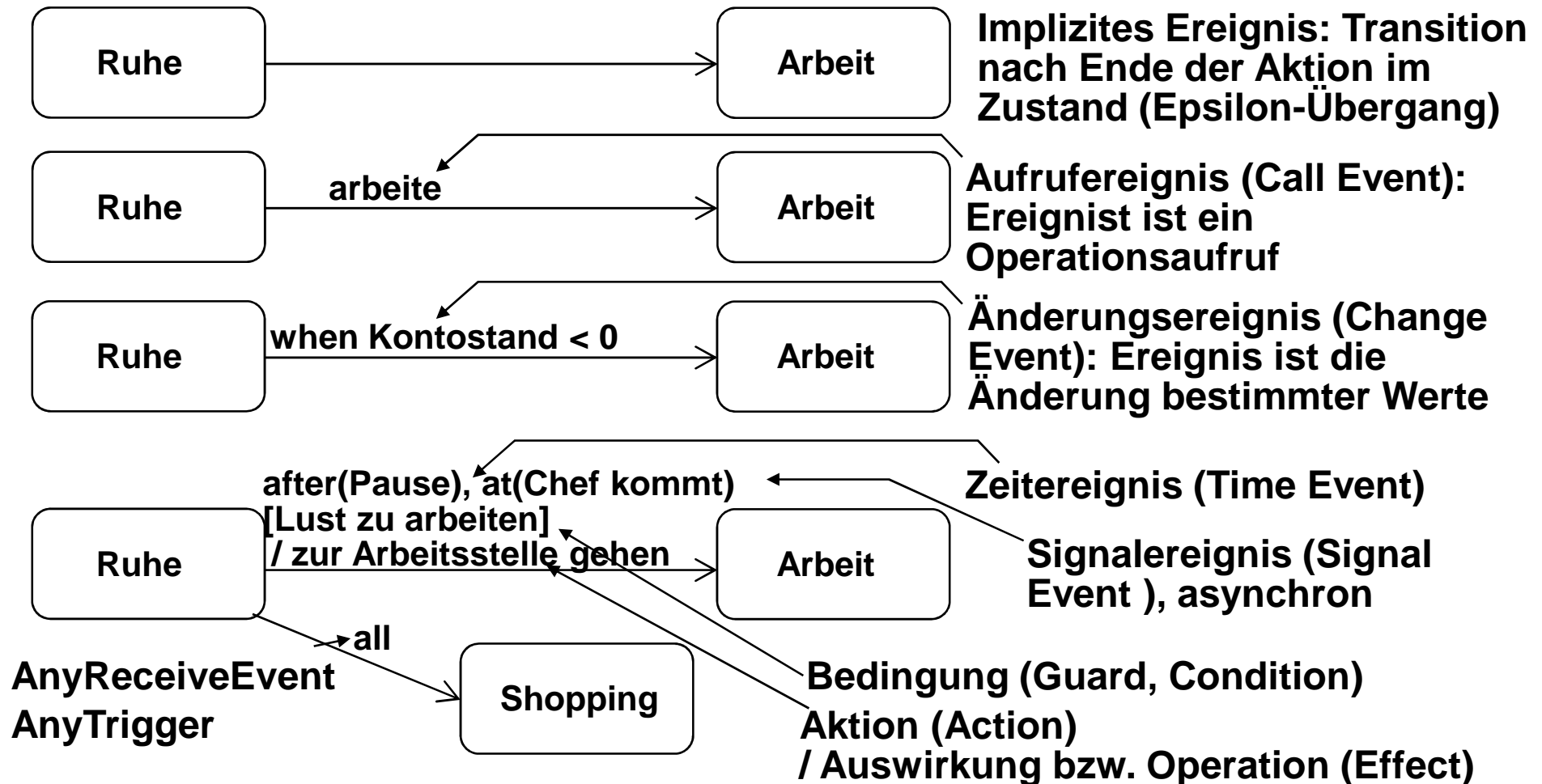
Transitionen: (bedingte) Zustandsübergänge (Event, Transition, Trigger)



- Zustandsübergänge werden durch Ereignisse ausgelöst.  
Ausnahme: Übergang vom Startzustand.
- Ereignisse können um Bedingungen ergänzt werden.
- Zustandsübergänge können Aktionen anstoßen.

## 5.8 OO Modellierung mit UML: Zustandsdiagramm

- Transitionen: - Allgemeines Format für Zustandsübergänge:  
Event, Event, ... [Guard] / Effect
- Eine Transition wird durch ein Event ausgelöst (trigger)

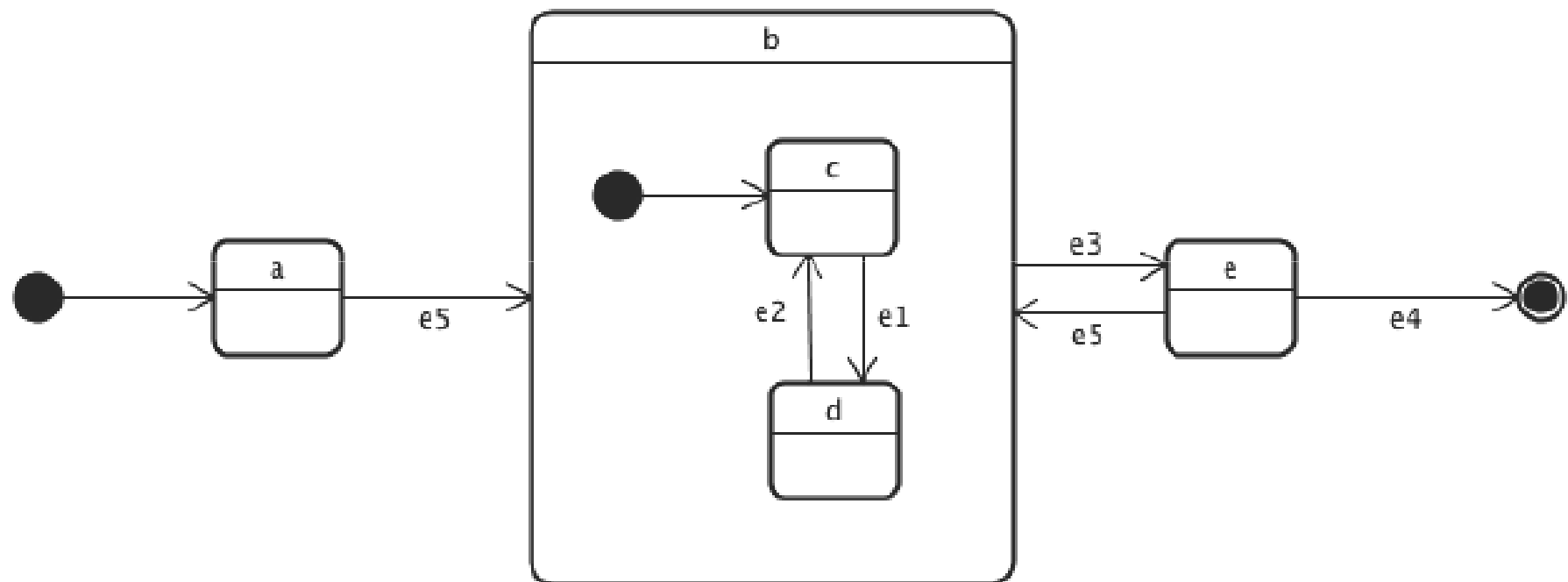


## 5.8 OO Modellierung mit UML: Zustandsdiagramm

- **Transitionen:**
  - **Alle Ereignisse werden ignoriert (und konsumiert), wenn es keinen passenden Übergang für den aktuell aktiven Zustand definiert ist.**
  - **Die Auswahl der Transition ist nicht deterministisch, falls mehrere Transitionen gleichermaßen möglich sind.**
  - **Die Transition, die vom Startzustand ausgeht, wird sofort ausgeführt (ohne Bedingung und in der Regel auch ohne besonderes Ereignis an dieser ersten Transition).**
  - **Es gibt nur einen Startzustand im Diagramm.**
  
- **... weitere Modellelemente, z.B. Senden und Empfangen von Ereignissen im Rahmen einer Transition, Zusammenführung, Teilung und Verzweigung von Transitionen (analoge Symbole wie im Aktivitätsdiagramm)**



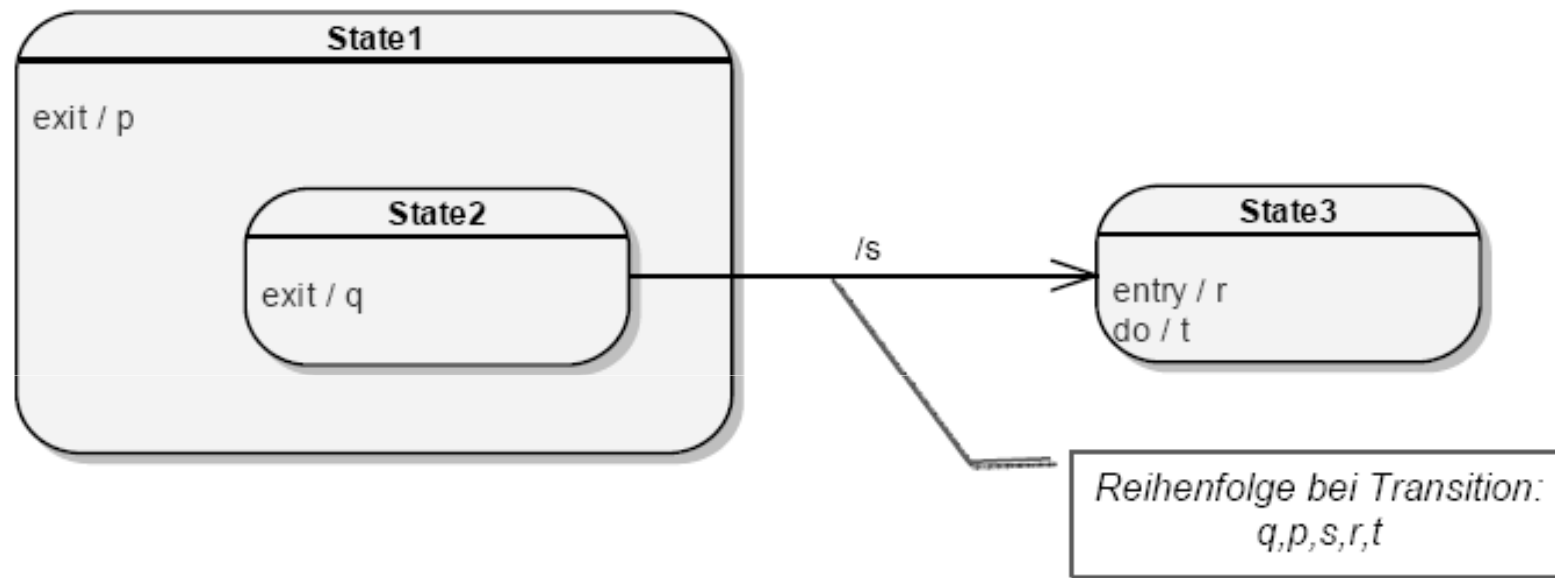
- Hierarchische Zustände / Zusammengesetzte Zustände)
  - Bieten die Möglichkeit, das Zustandsdiagramm besser zu strukturieren und Eigenschaften zu abstrahieren.
  - Können zur Top-down-Entwicklung eines Zustandsmodells verwendet werden.



- Hierarchische / zusammengesetzte Zustände

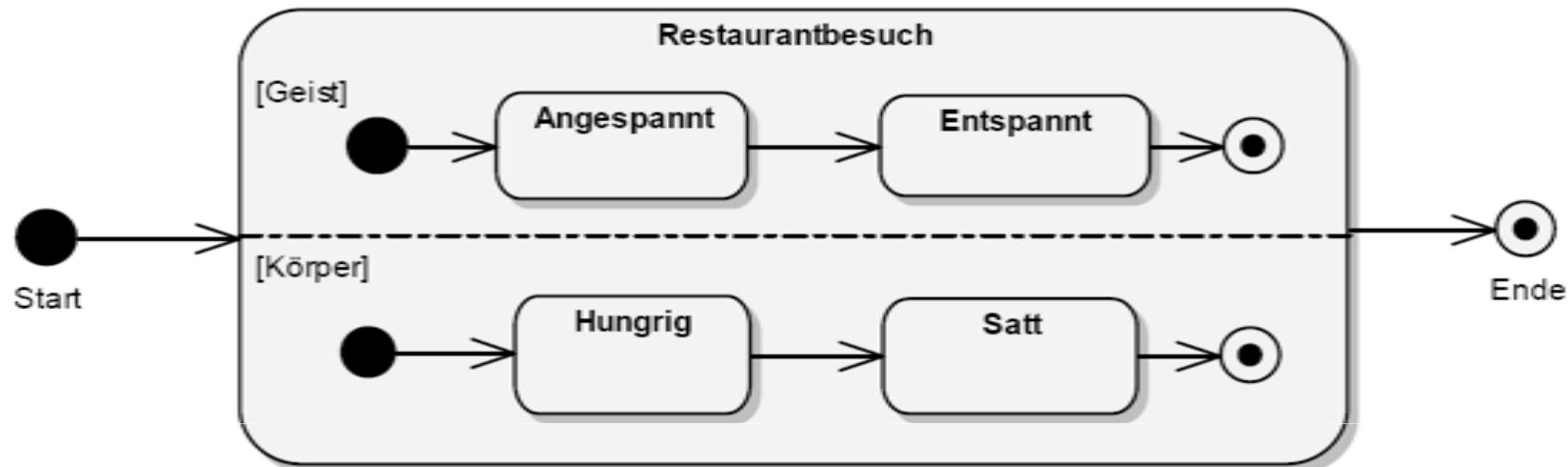
Reihenfolge der Aktionsausführung:

“exit” von “innen” nach “außen”



- **Regionen (Regions): nebenläufige Zustände (concurrent substates)**

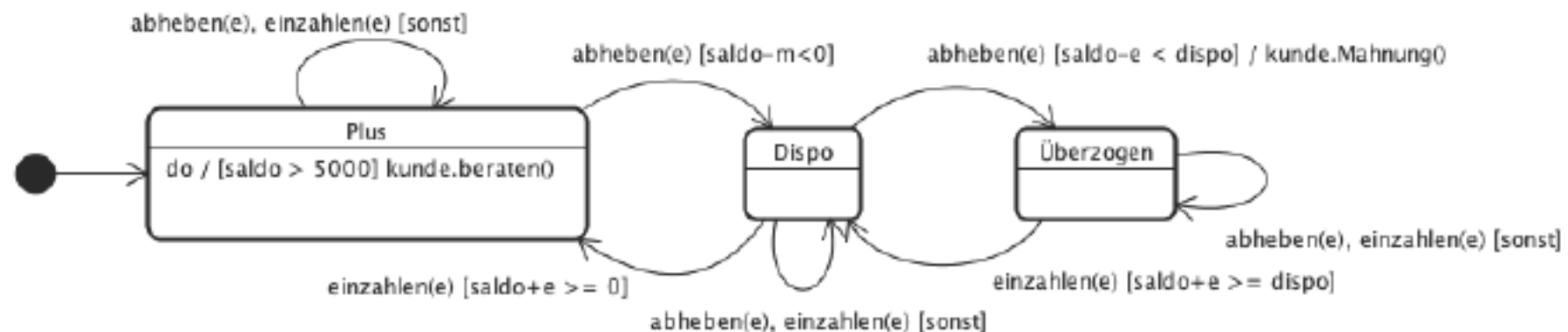
**Regionen teilen Zustände in disjunkte Bereiche auf:  
Zustände, die aus mehreren Unterzuständen bestehen, in denen  
sich das System gleichzeitig befinden kann.**



- Die Regionen werden bei Betreten des (umgebenden) Zustands gleichzeitig aktiviert.
- Sie verhindert “Explosion” der Anzahl von Zuständen.
- Der Zustand wird verlassen, wenn alle Regionen terminiert haben.

### ▪ Beispiel

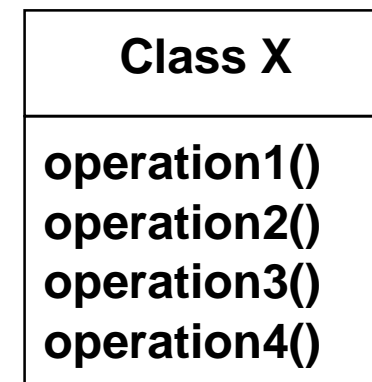
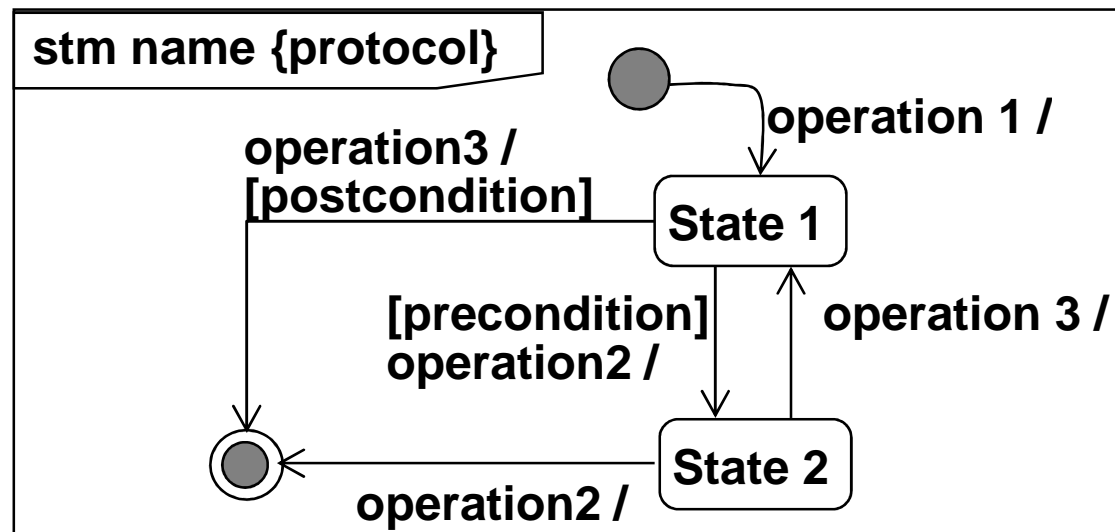
#### Objekt Lebenszyklus für ein Girokonto



Weitere Transitionen sind der direkte Übergang vom Zustand Plus durch Abheben eines großen Betrags in den Zustand Überzogen sowie umgekehrt der direkte Übergang vom Zustand Überzogen durch Einzahlen eines grossen Betrags in den Zustand Plus.

## 5.8 OO Modellierung mit UML: Zustandsdiagramm

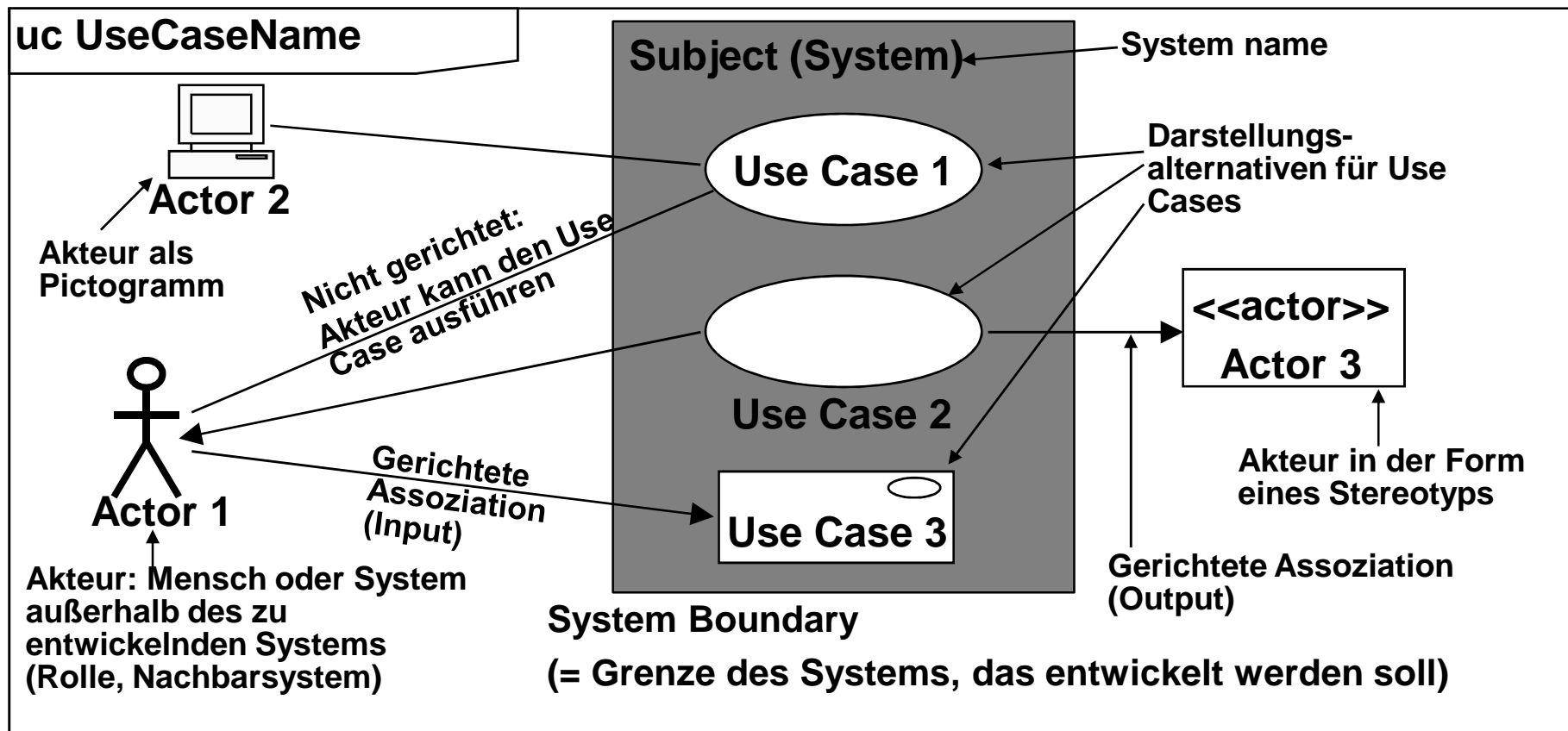
- **Protokollautomat (Protocol state machine)**
  - Spezielle Form eines eindlichen Zustandsautomaten, spezielle Form eines UML Zustandsdiagramms.
  - Beschreibt für ein Objekt einer Klasse, welche Operationen in welchem Objektzustand unter welchen Bedingungen gerufen werden kann.
  - Es gibt keine Aktionen.
  - Transition Syntax:  
[precondition] operation / [postcondition]



### UML Use Case Diagram (Anwendungsfalldiagramm)

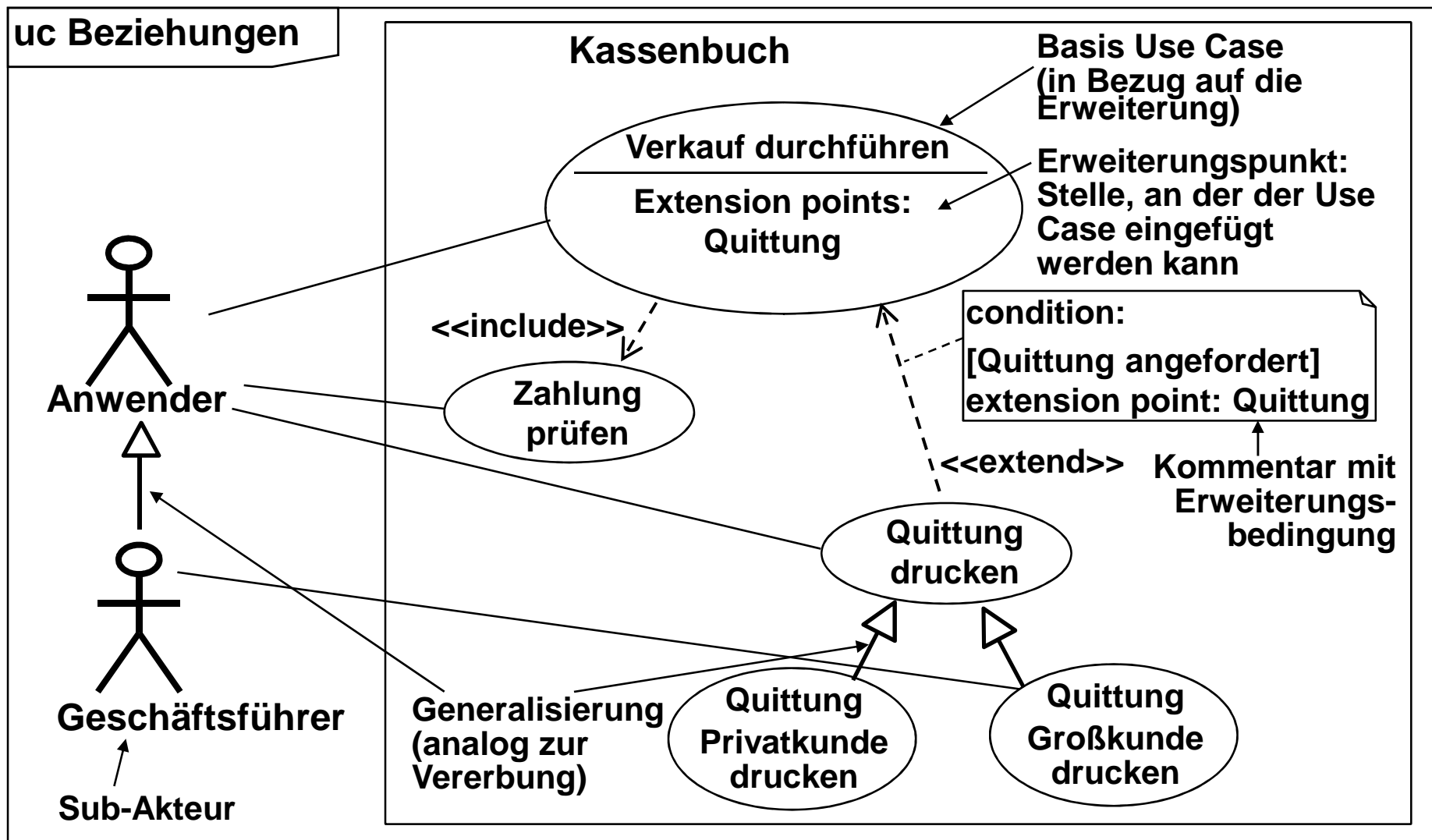
- **Grafische Representation der wichtigsten Anwendungsfälle (Use Cases), Akteure und ihrer Struktur.**
- **Textuelle Beschreibung der Anwendungsfälle (notwendig für das Verständnis der Funktionalität!).**
- **Bestandteile der grafischen Representation:**
  - **Akteure/Aktoren (actors, roles, neighbor system)**
  - **System, Systemname, Systemgrenze (system boundary)**
  - **Anwendungsfälle (Use Cases)**
  - **Assoziationen (Associations; communication between actors and use cases)**
  - **Use Case Beziehungen:**
    - <<include>>-dependency, <<extends>>-dependency, generalization

- **Haupteinsatz: Spezifikation der Anforderungen (Requirements) aus Sicht der Systembenutzung**



## 5.8 OO Modellierung mit UML: Anwendungsfalldiagramm

### Beispiel und Prinzip eines Anwendungsfalldiagramms (Use Case)

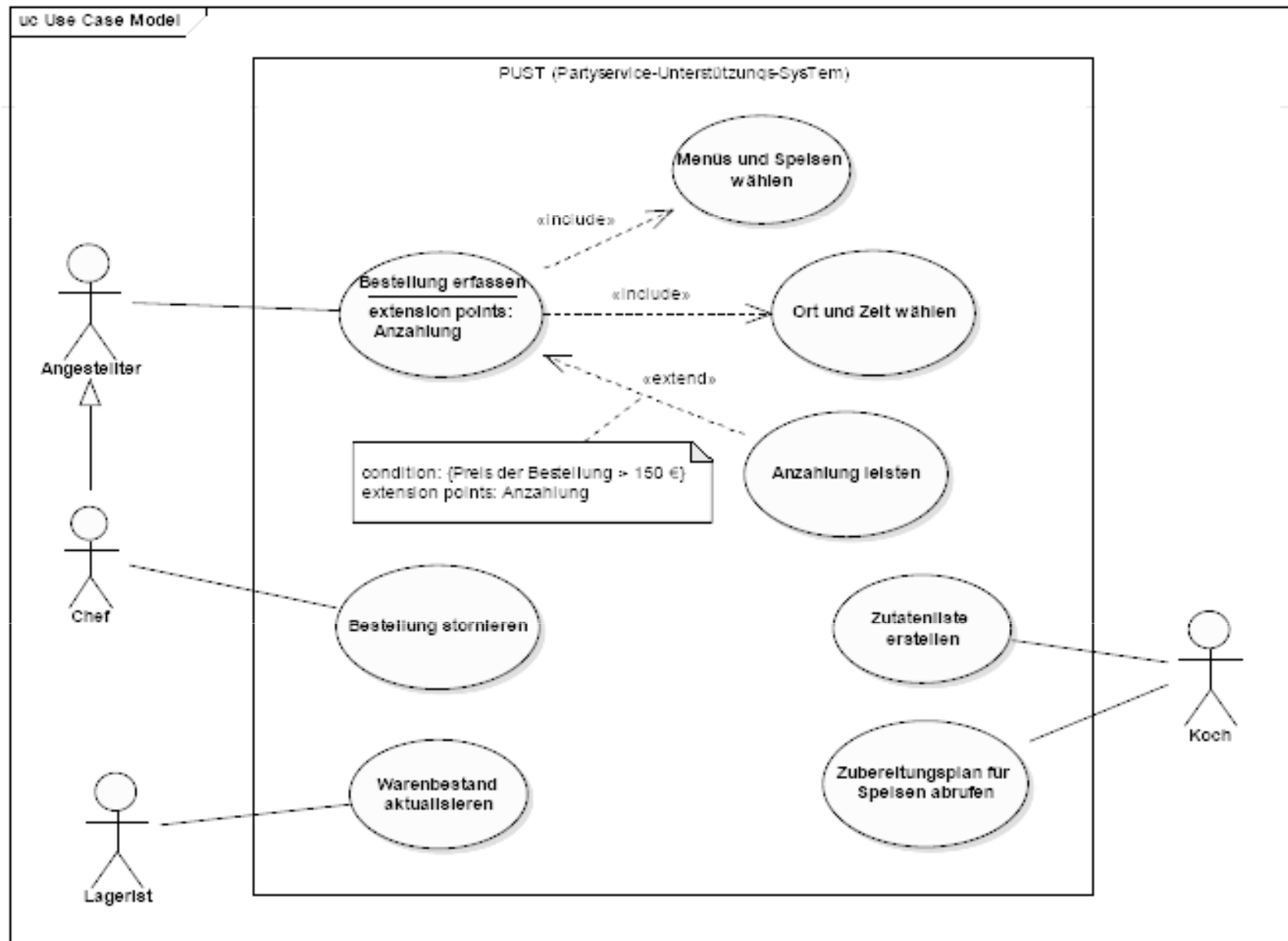




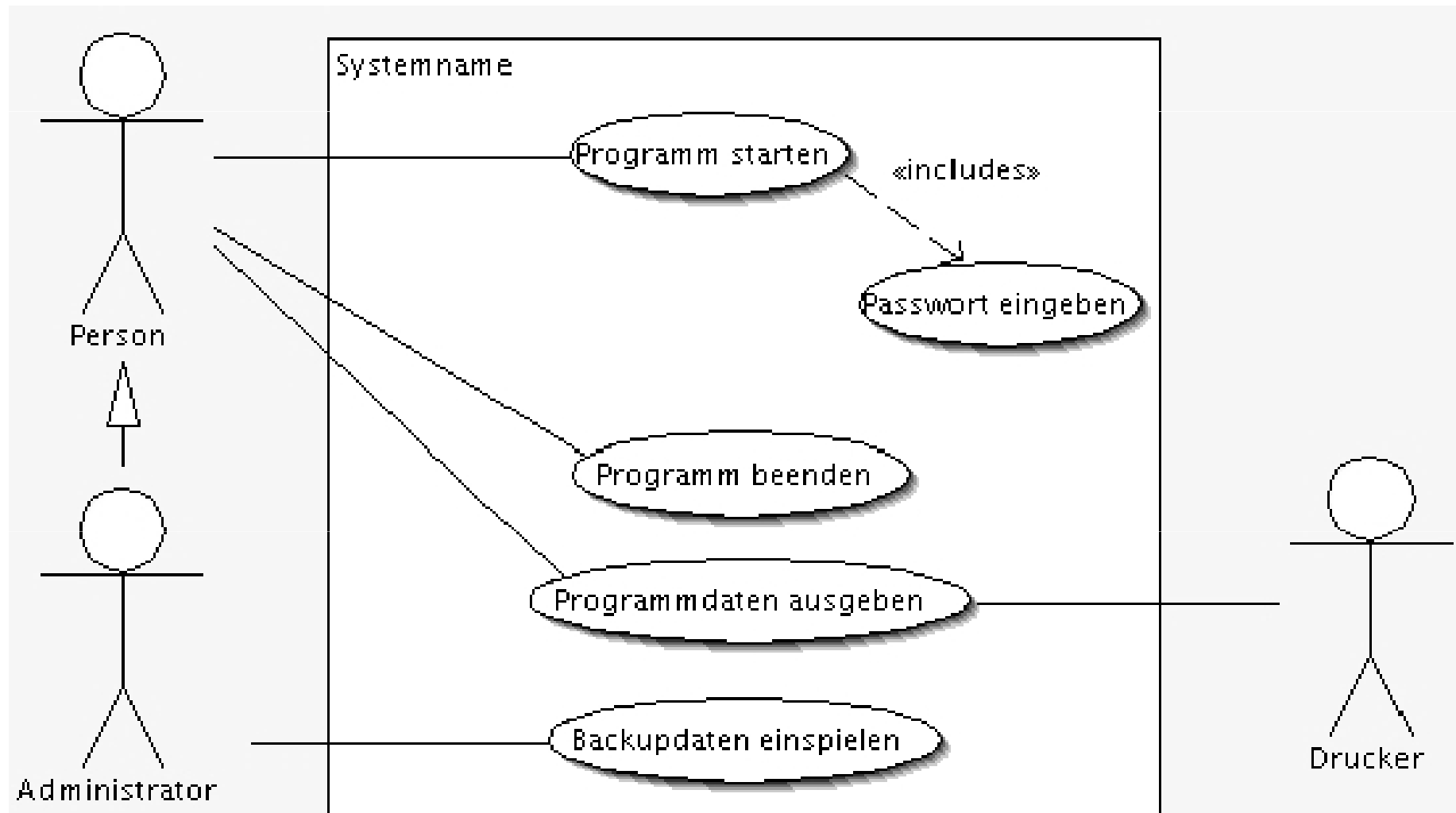
## 5.8 OO Modellierung mit UML: Anwendungsfalldiagramm

- **Dependency <<include>> (Enthältbeziehung):**
  - Use case A inkludiert immer Use Case B. B wird immer ausgeführt, wenn A ausgeführt wird.
  - Wiederverwendung von Teilen, die mehreren Use Cases gemeinsam sind.
  - Beispiel: Die Ausführung von „Verkauf durchführen“ erfordert immer auch die Ausführung von „Zahlung prüfen“
- **Dependency <<extends>> (Erweiterungsbeziehung):**
  - Use Case B ist optional Teil des Use Case A und kann im definierten Erweiterungspunkt (Extension Point) von A eingefügt werden.
  - Optional (aber empfohlen): Zusätzlicher Kommentar mit einer Definition der Bedingung. Use case B wird in Use Case A nur eingefügt, wenn die Bedingung wahr ist.
  - Es kann viele Erweiterungspunkte geben.
  - Beispiel: Wenn „Verkauf durchführen“ ausgeführt wird, kann der Use Case „Quittung drucken“ ausgeführt werden (muss aber nicht).
- **Generalisierungsbeziehung (zwischen Akteuren oder Use Cases):**
  - Beispiel: Der spezialisierte Use Case „Quittung Privatkunde drucken“ erbt alle Eigenschaften von Use Case „Quittung drucken“.

### • Beispiel:



### • Beispiel:



## 5.8 OO Modellierung mit UML: Anwendungsfalldiagramm

- **WICHTIG:** Neben der grafischen Darstellung ist eine textuelle Beschreibung der Anwendungsfälle (in natürlicher Sprache) notwendig, um die Funktionalität und den Ablauf ganz verständlich zu machen, z.B. in Tabellenform:

Beschreibung Anwendungsfall	
Name	
Kurzbeschreibung	
Akteure	
Auslöser	
Ergebnis(se)	
Eingehende Daten	
Vorbedingungen	
Nachbedingungen	
Essenzielle Schritte	
Offene Punkte	
Änderungshistorie	
Sonstige Anmerkungen	

- **Möglich:** Einsatz von Aktivitätsdiagrammen zur Beschreibung der Abläufe in einem Use Case

- Eine Anwendungsfallbezeichnung besteht aus einem Substantiv und einem aktiven Verb (z.B. „Bestellung stornieren“).
- An jedem Anwendungsfall ist mindestens ein Akteur beteiligt.
- Jeder Anwendungsfall hat einen fachlichen Auslöser und ein fachlich relevantes und wertvolles Ergebnis.
- Die Beschreibung wird aus Sicht des zu entwickelnden Systems formuliert, inhaltlich wird jedoch das Verhalten beschrieben, das für den (außenstehenden) Akteur wahrnehmbar ist.
- Die Beschreibung sollte so kurz und abstrakt wie möglich sein und so ausführlich und konkret, wie zum eindeutigen Verständnis nötig ist.

- **Leicht zu verstehen:** Kunden (Experten der Problemdomäne, Fachexperten) können bei der Modellierung kooperieren.
- **Use Cases** werden in der Praxis häufig zur Definition der Grundfunktionalität (Anforderungen) eines Systems eingesetzt.
- Können in initiale Klassendiagramme und andere Diagramme zur Modellierung des Verhaltens transformiert werden.

**Aber:**

- **Keine grafische Modellierung** der Ordnung/Reihenfolge der Use Cases bzw. von Abläufen.
- **Nichtfunktionale Anforderungen** können nicht direkt modelliert werden.

### **Nichtfunktionale Anforderungen (non-functional Requirements):**

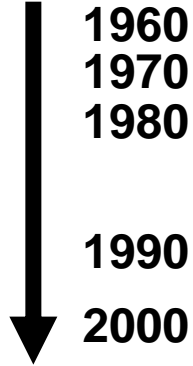
- **Nichtfunktionale Anforderungen sind nicht direkt mit einer Systemfunktionalität verknüpft.**
- **Beispiele:**
  - **Zeitschranken (Time Limits), z.B. falls ein System innerhalb n Millisekunden antworten muss.**
  - **Einsatz spezifischer Kommunikationskonzepte, z.B. Konzepte zur verteilten Programmierung.**
- **Allerdings: Nichtfunktionale Anforderungen finden sich am Ende auch in der Implementierung als Funktionalität:**
  - z.B. Zeitschranken → Watch Dog Funktion**
  - z.B. Verteilung → Einsatz von Threads oder Webservice-Technologie**

### Szenarien (Scenarios):

- Eine detailliertere Spezifikation von Interaktionen zwischen Akteuren und Use Cases wird durch Szenarien möglich.
- **Szenario = spezielle Ausprägung eines Use Case, ein Anwendungsfall mit konkreten Werten**
- Allerdings: Die Notation von Use Case Diagrammen genügt nicht, um Szenarien auszudrücken.
- Szenarien werden mit Sequenzdiagrammen (oder Aktivitätsdiagrammen) modelliert, um zeitliche Details von Interaktionen darzustellen.



## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

- Describes the order and dependencies between actions
  - In UML, activity diagrams are assigned to a class/object, an operation or a use case
  - Main Purposes: business process modeling (Geschäftsprozessmodellierung), Use Case scenarios, algorithms
  - Origins:
    - Petri Nets
    - Flow Charts
    - Nassi-Shneidermann Diagrams
    - Dataflow Diagrams
    - Event Process Chains (EPCs)
    - UML 1.x Activity Diagrams
- 
- |  |      |
|--|------|
|  | 1960 |
|  | 1970 |
|  | 1980 |
|  | 1990 |
|  | 2000 |
- Attention: semantics differs between UML 1.x and UML 2.x
  - Description of control flow (order of functions) and data flow (data exchanged between the functions); token flows are similar to Petri nets

### Ingredients of UML 2.x Activity Diagrams

- **Activity:**
  - One diagram per activity (functionality, behavior)
  - Activities are performed by actors
  - An activity specifies the coordination of executions using a control and data flow of nodes (e.g. branching and merging of the control flow)

## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

### Ingredients of UML 2.x Activity Diagrams

- **Nodes:**

- Action node**

- Performing some action

- Control node**

- Controlling the control flow (control flow token)

- Object node**

- Describing the data flow (control the data token)

- **Transitions: Activity edge**

- Control flow

- Object flow

- **Diagram structuring**

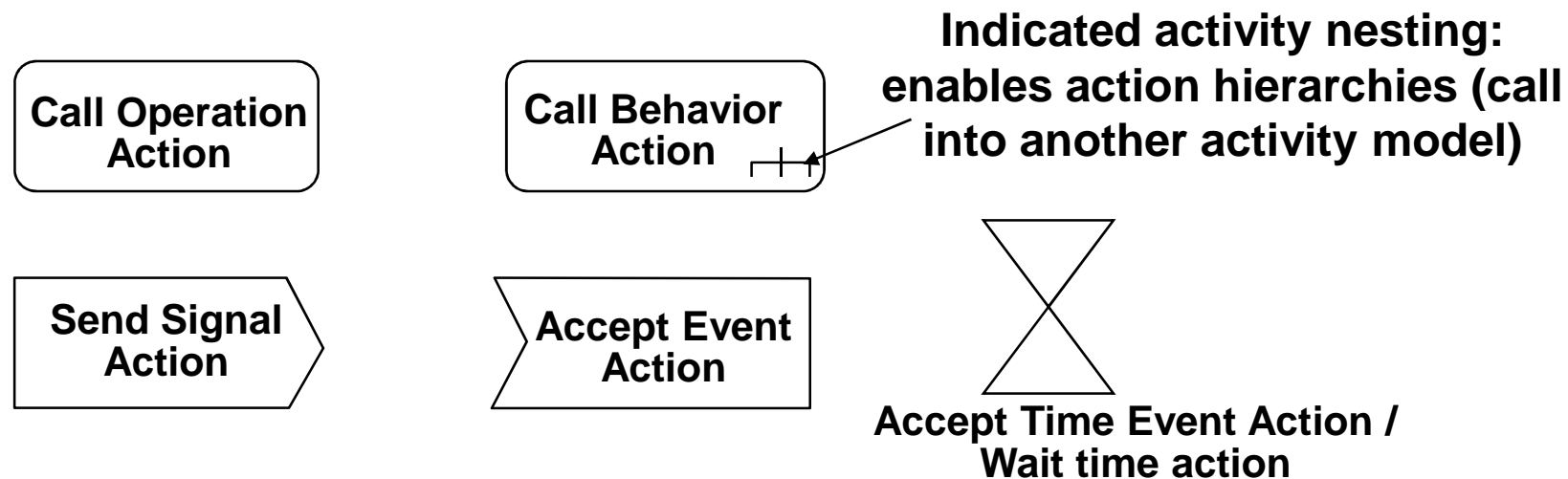
- Connectors

- Activity partitions

- Activity nesting

- Interruptible activity region

**Action, action nodes:** smallest executable functionality



- Actions may have one or several parameters (→ Object Nodes)
- In the control flow signals are sent or received (purpose: reaction to external events or process synchronization; signals are asynchronous)
- Several parallel activity processing flows are possible (independent to each other)

## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

**Control nodes:** control/define the control and data flow (means: control flow tokens)

- **Initial / Start node (Startknoten):**

- control flow only outgoing;
- one or many initial nodes (if there are parameter nodes initial nodes are not needed)
- at the start of an activity tokens are placed at the initial nodes



- **Final node (Endknoten):**

- at least one incoming control flow, no outgoing control flow
- one or many final nodes (if there are parameter nodes final nodes are not needed)
- stops all flows in the activity



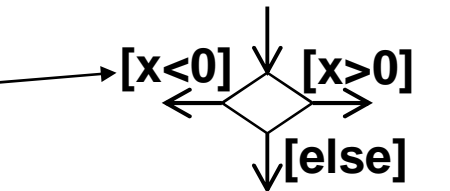
- **Flow final node (Ablaufende):** as with final node but only terminates a single control flow and has no effect on the other flows in the activity



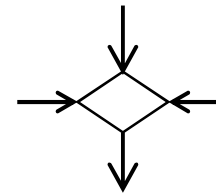
**Control nodes:** control/define the control and data flow (means: control flow tokens)

**Guard:** predicate controlling the token forwarding

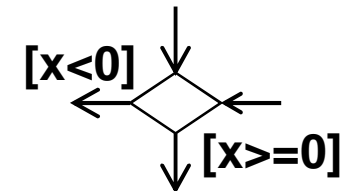
- Overlapping predicates lead to non-determinism (should be avoided)
- Syntax not defined



**Decision**  
**(Entscheidung, Verzweigung):**  
branch in the control and data flow  
One branch is taken depending on the condition

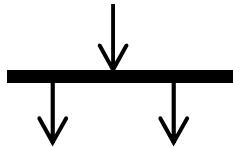


**Merge**  
**(Zusammenführung, Disjunktion, Oder):**  
brings together multiple alternate flows (no synchronization)  
OR semantics

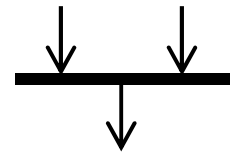


**Decision and Merge**

**Control nodes:** control/define the control and data flow (means: control flow tokens)



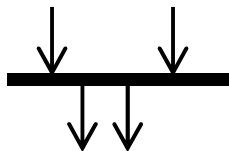
**Fork / Splitting (Teilung):**  
unconditional split of the control/data flow into several outgoing parallel control flows



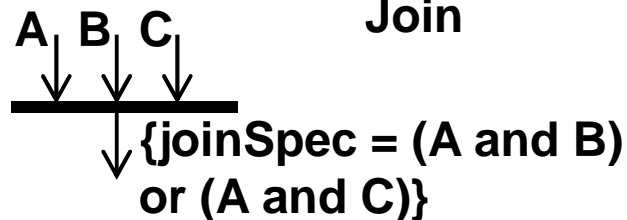
**Join (Synchronisation, Und, Konjunktion):**  
waiting for all incoming control and data flows before continuing on the outgoing control flow;  
AND semantics

- Merge of the control tokens
- Individual forwarding of the data tokens

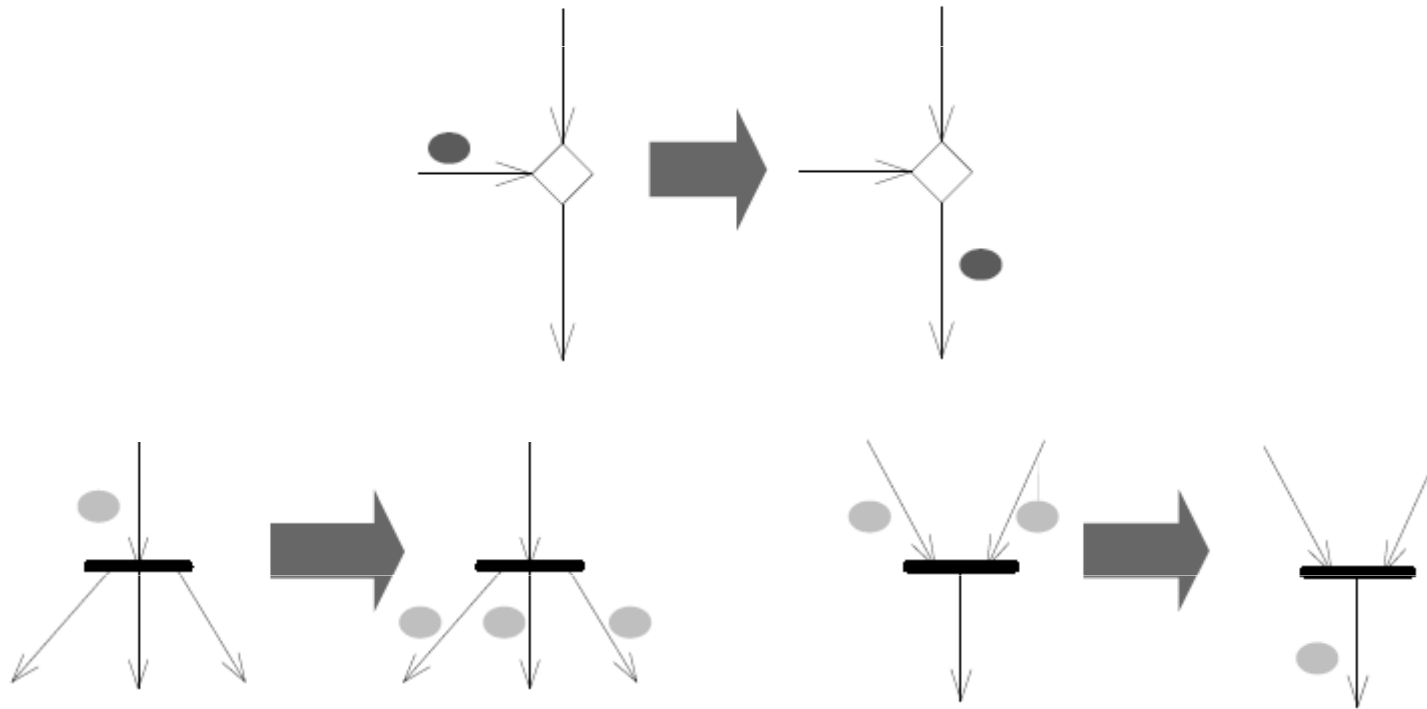
**Fork and Join**



**Specialized Join**



### Control nodes and their semantics:



Remark: The circles next to the arrows show tokens in the sense of Petri-Nets; they are not part of the UML notation!

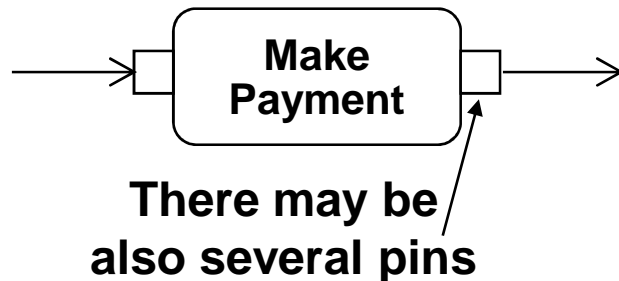


## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

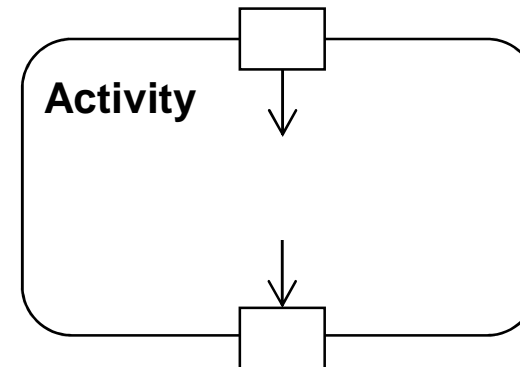
**Object nodes:** define one or many existing objects which are transported in the object flow (as object tokens)

Objects and data cannot pass along a control flow edge.

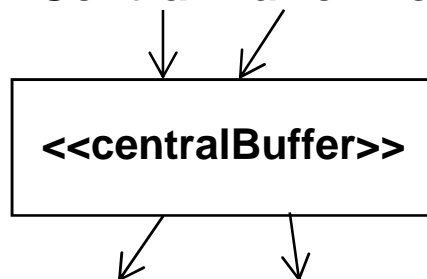
**Parameter of actions:  
Input pin / Output pin**



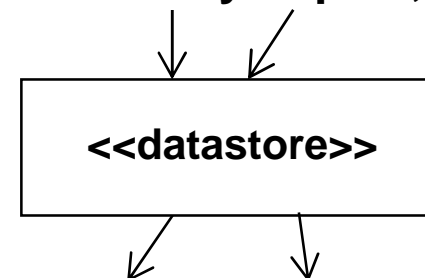
**Parameter activities:  
Activity Parameter Node**



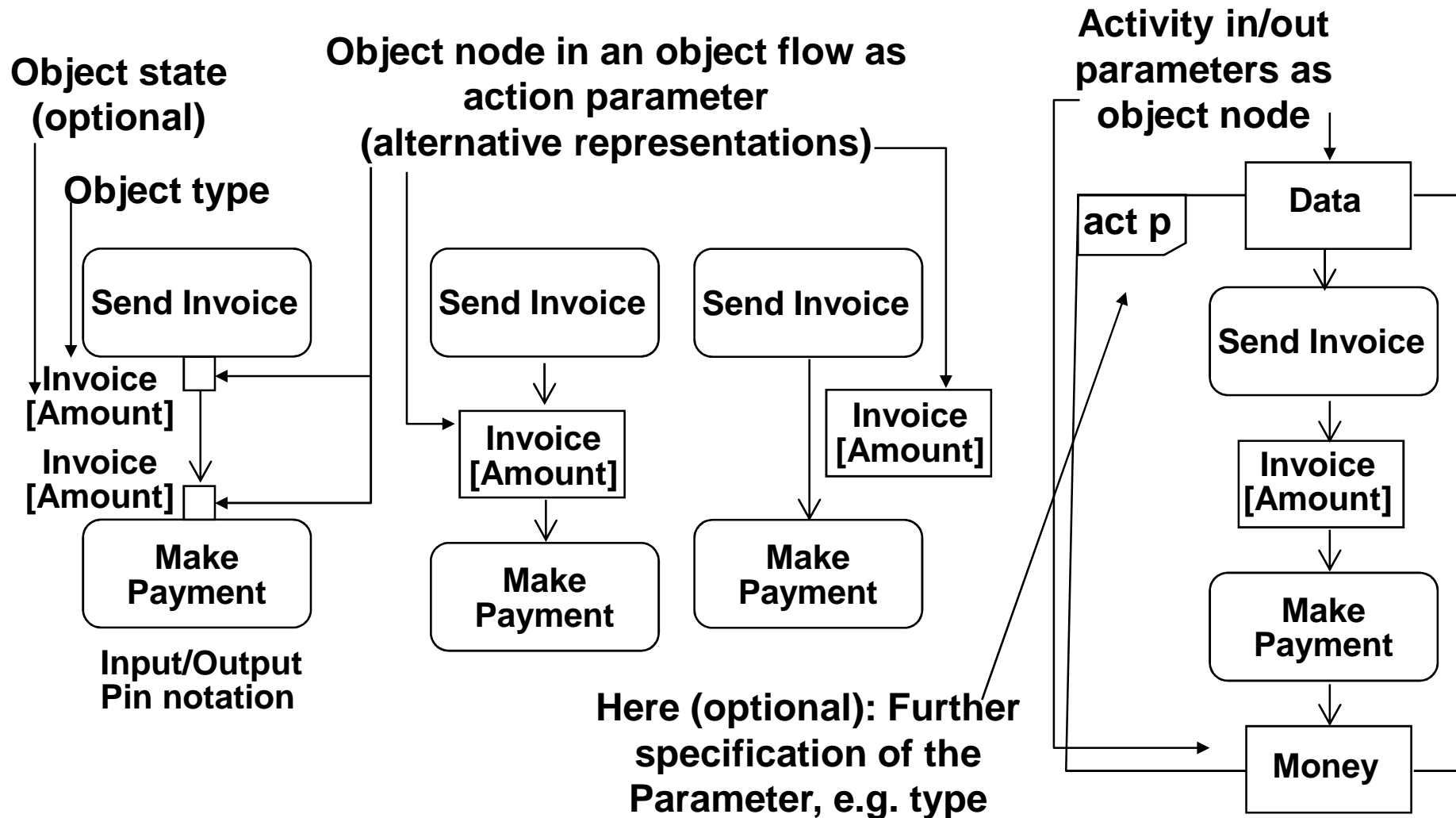
**Temporary buffer storage for data  
tokens: Central Buffer Node**



**Persistent buffer storage for data  
tokens: Data Store Node  
(returns only copies, no delete)**

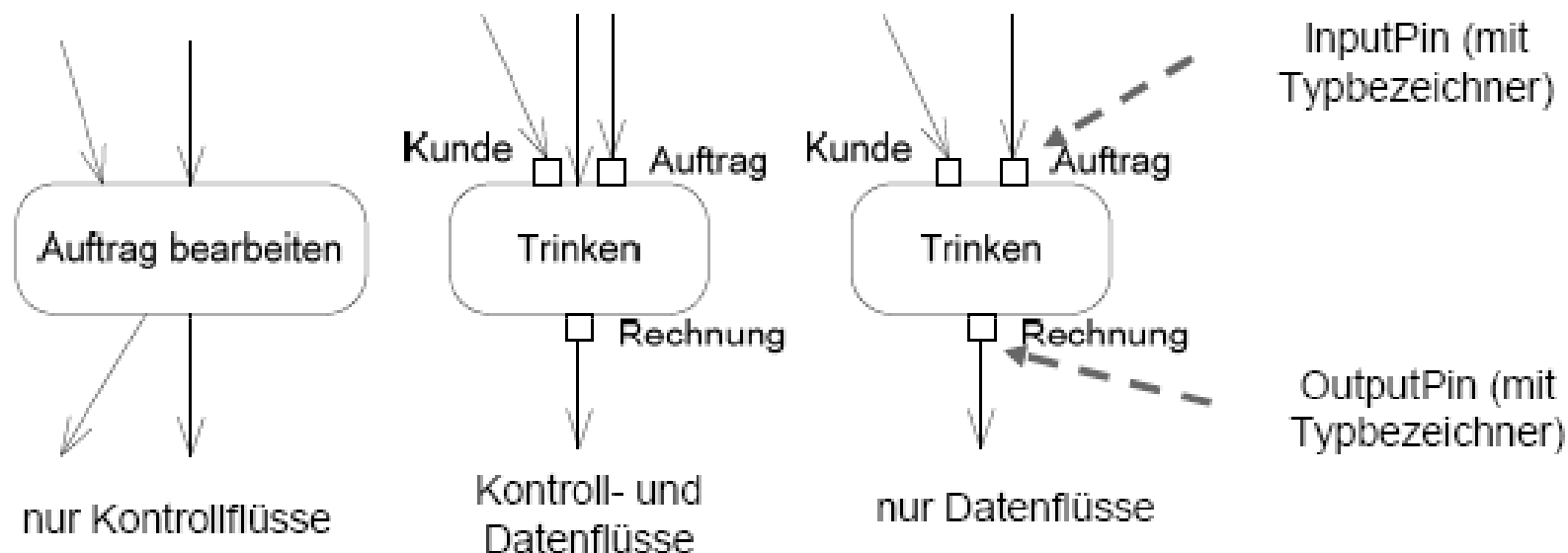


## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm



### Object and Control flow, semantics:

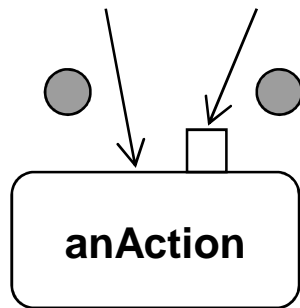
- Object flow edges have the same notation as control flow edges (but have objects in addition!)
- Data tokens may contain arbitrary values (but with type)
- Data flow is sufficient to initiate an action



## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

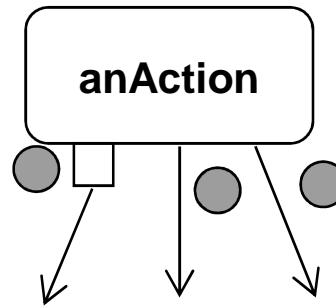
Object and Control flow, semantics:

- Action semantics:



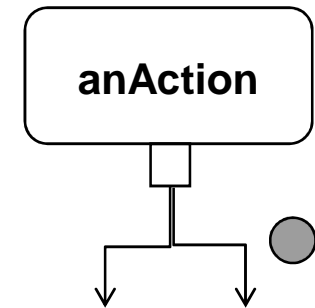
**AND semantics**

Execution in the action can start as soon as the tokens are available on all incoming control flow and data flow edges are available (implicit synchronization)



**Execution terminates**

New control tokens on all outgoing control flow edges, data tokens in all output pins  
⇒ The following actions can start

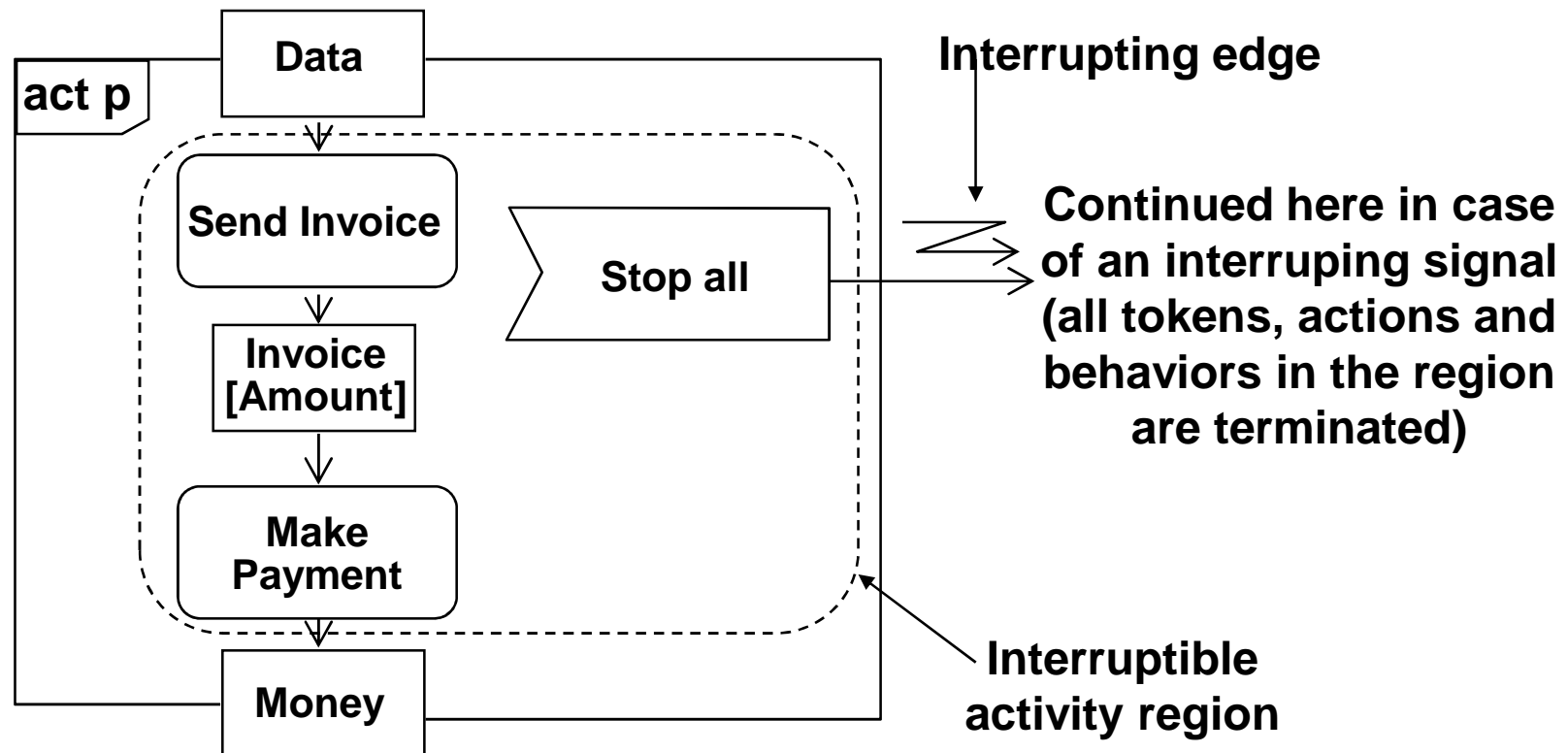


**Exclusive OR**

Only the flow continues which is the first ready to take the token

## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

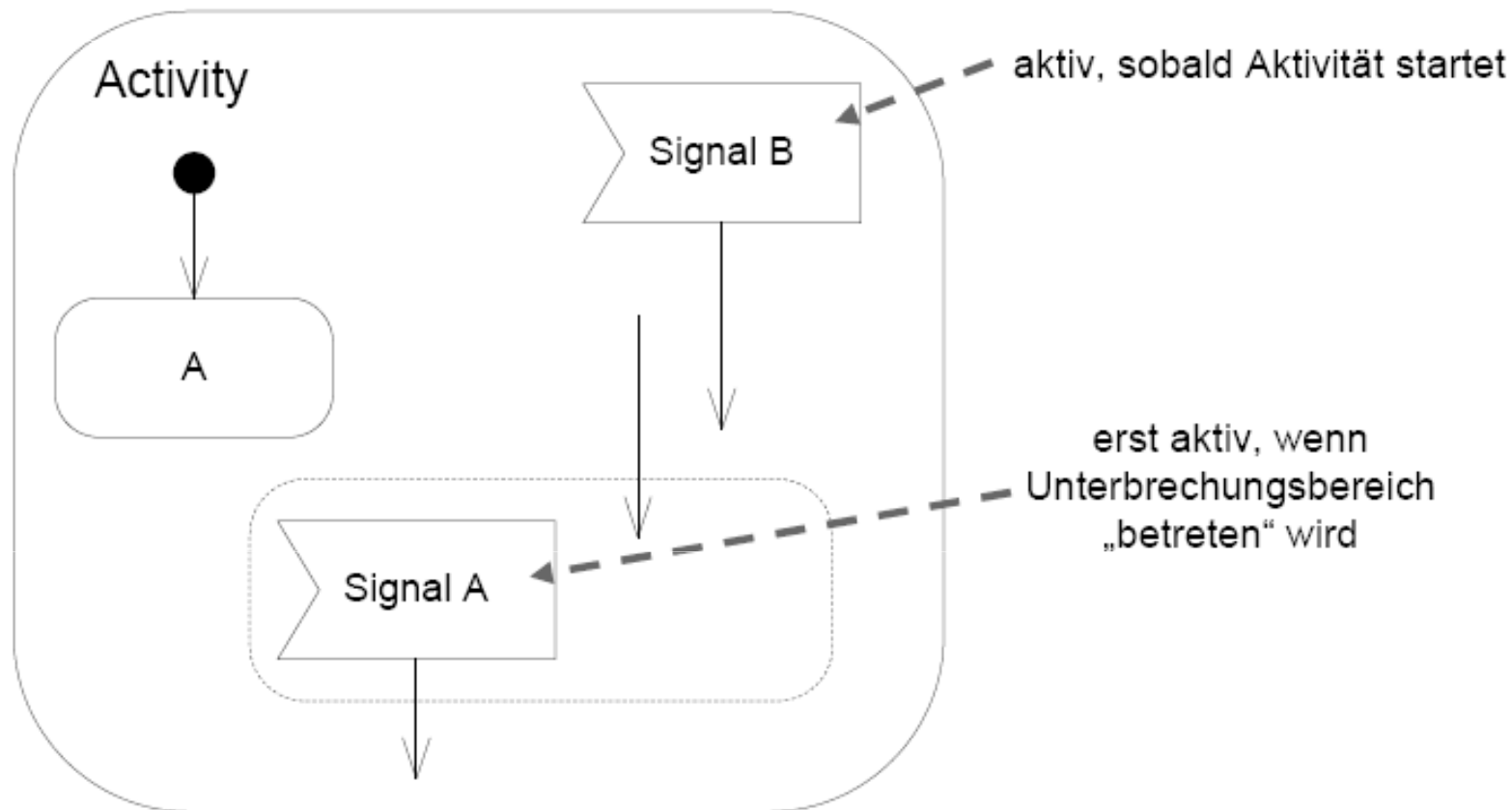
**Interruptible Activity Region:** set of nodes which will be terminated if the interruption edge (a control or data flow edge) is traversed



- If a signal action has no incoming edge it is activated at the start of the activity
- Exception: actions in the interruptible activity region are activated when the region is entered for the first time in the flow
- An activated signal action is ready to receive the respective signal (asynchronous)

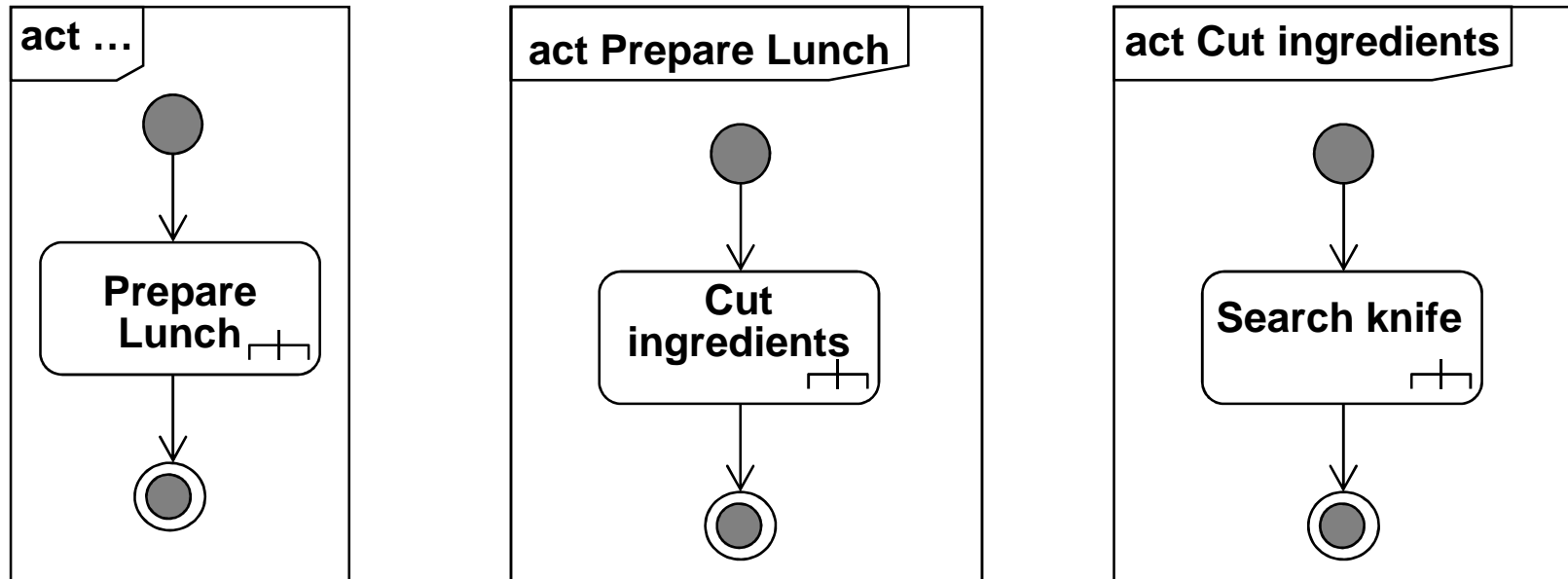
## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

**Interruptible Activity Region:** set of node which will be terminated if the interruption edge (a control or data flow edge) is traversed



## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

**Action/Activity nesting (Call Behavior Action):** an action in an activity diagram calls another activity diagram

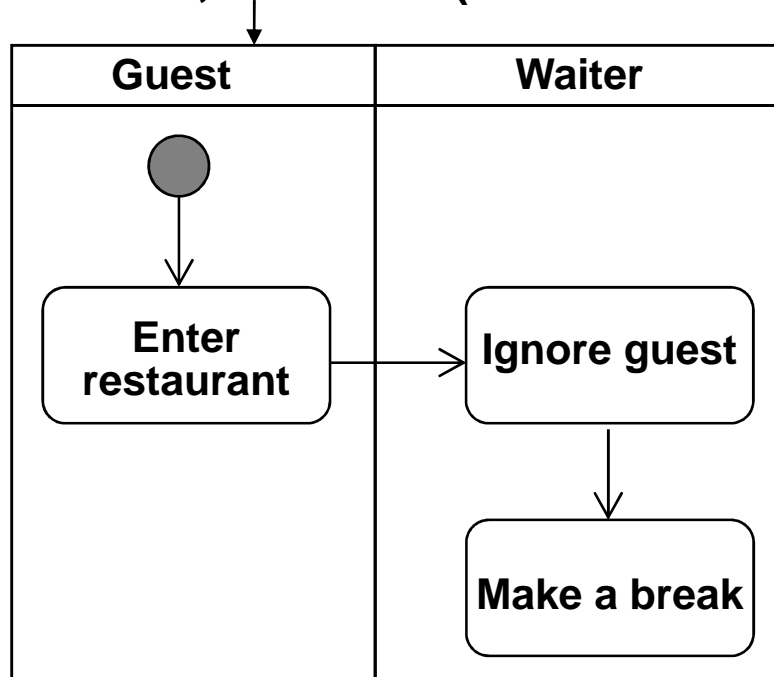


- Nesting is an implicit call from an action to another activity diagram (typically the action and the called activity diagram have the same name)
- Input/Output pins of the call behavior action are mapped to activity parameters

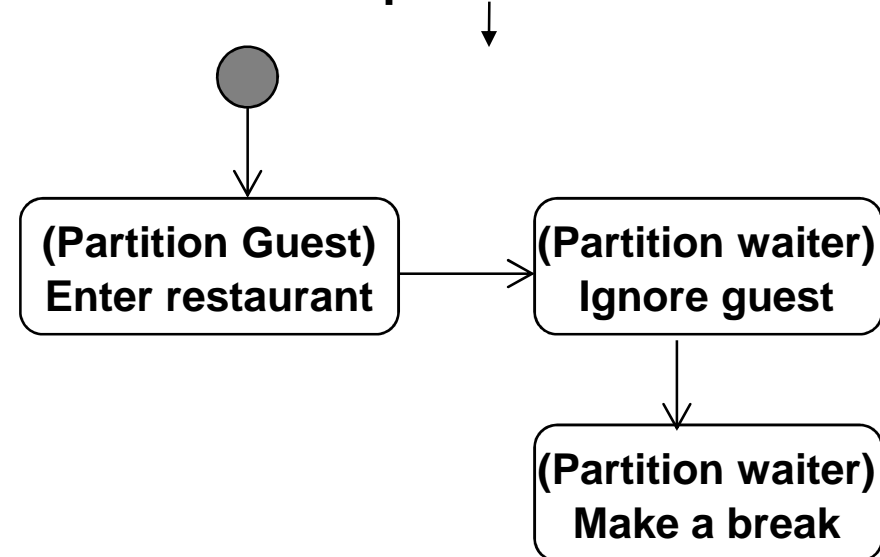
## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

**Activity Partitions (swim lane, Verantwortlichkeitsbereiche):** grouping of nodes and edges based on some common properties to improve the readability (logic view without effect on the activity flow)

**Partition, swim line (Schwimmbahnen)**



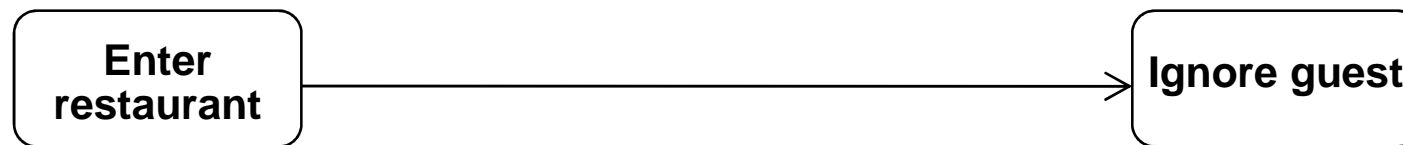
**Alternative partition representation**



- **Also possible: Multidimensional partitions, nesting, overlapping**  
→ readability check



**Connectors:** identifier to improve the diagram structure



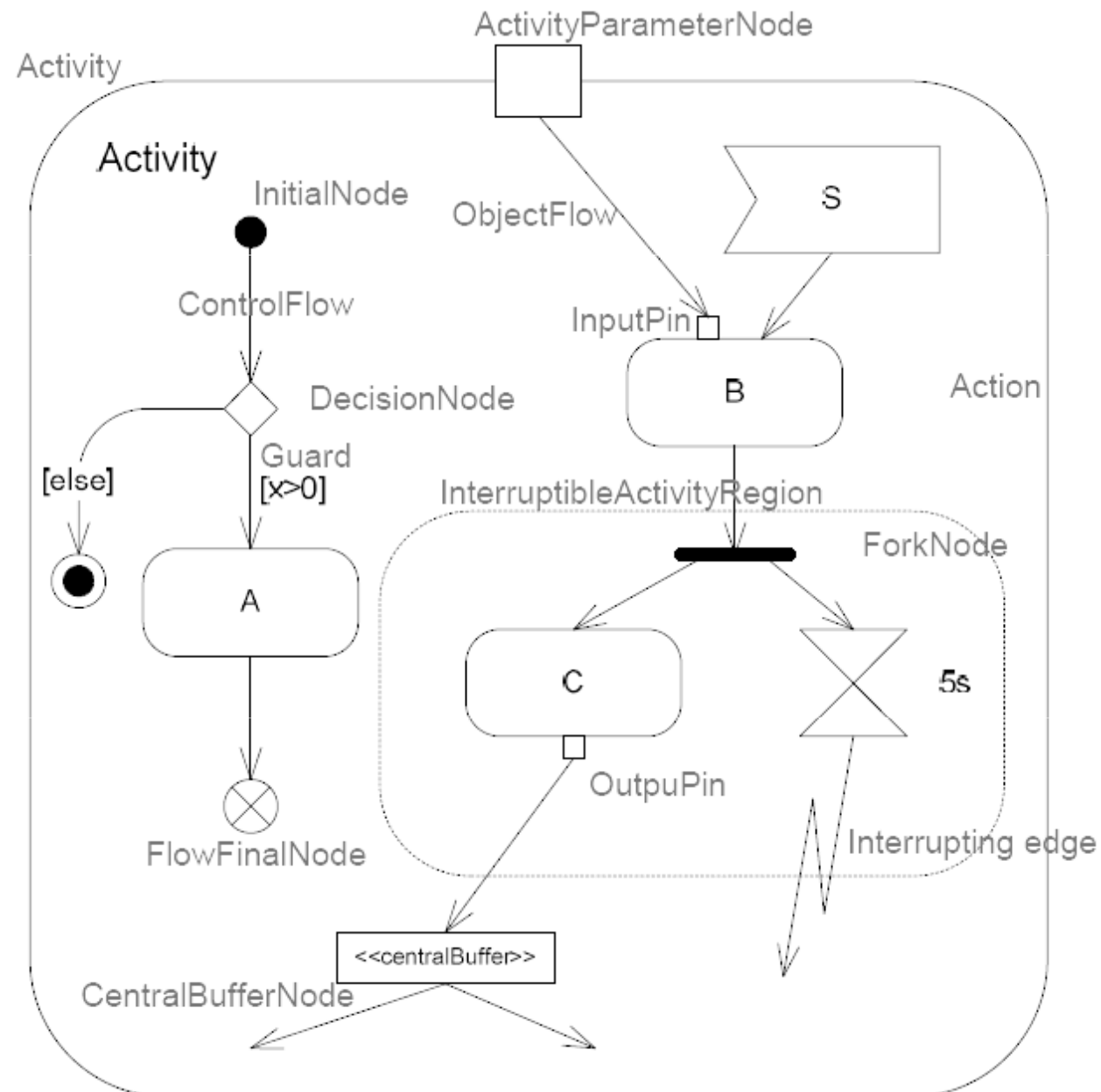
**Alternative  
representation:**



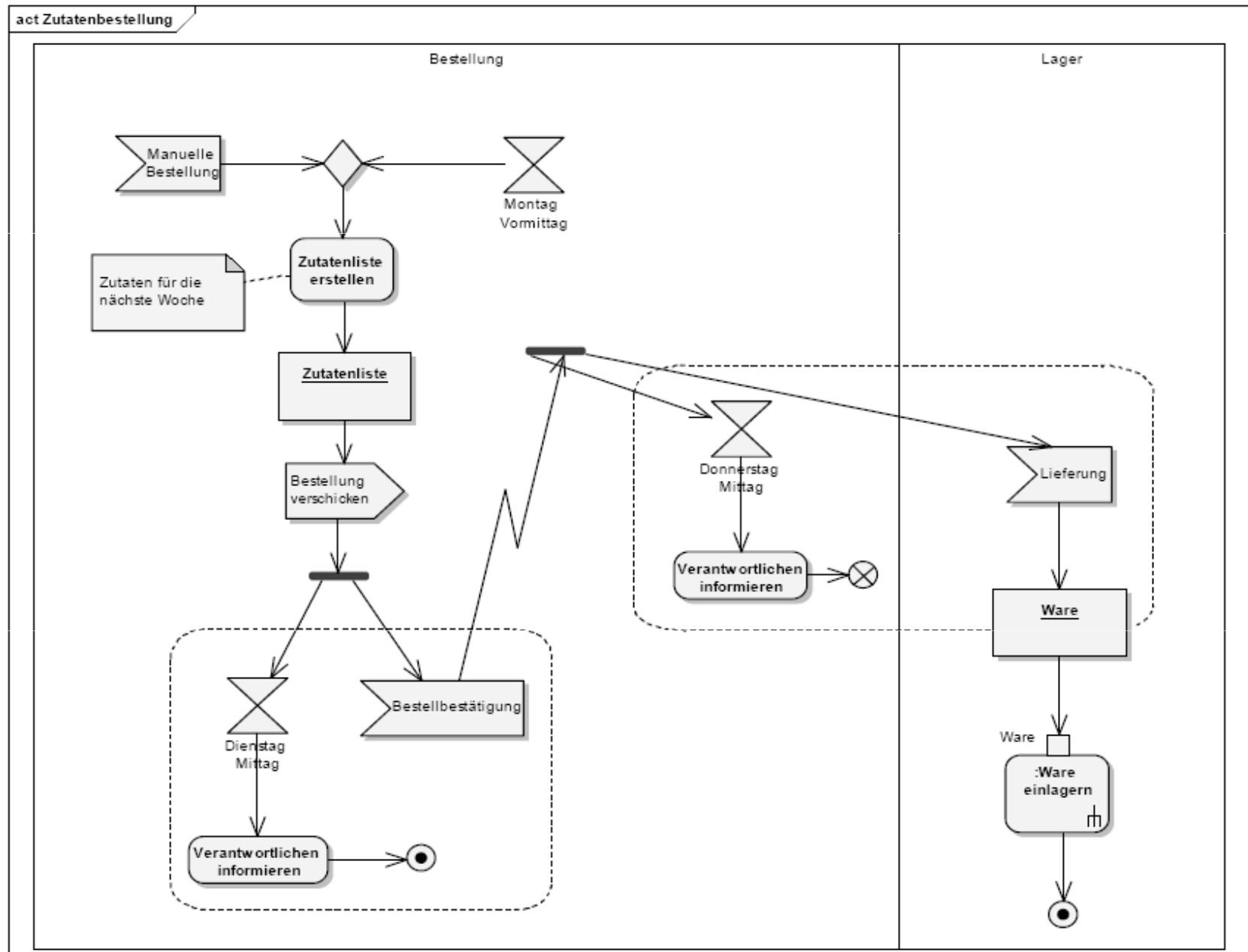
**Connector: Identifier to indicate  
the implicit continuation of the  
transition**

- To avoid long transition edges
- To continue transition edges on another document

- **Example (abstract):**

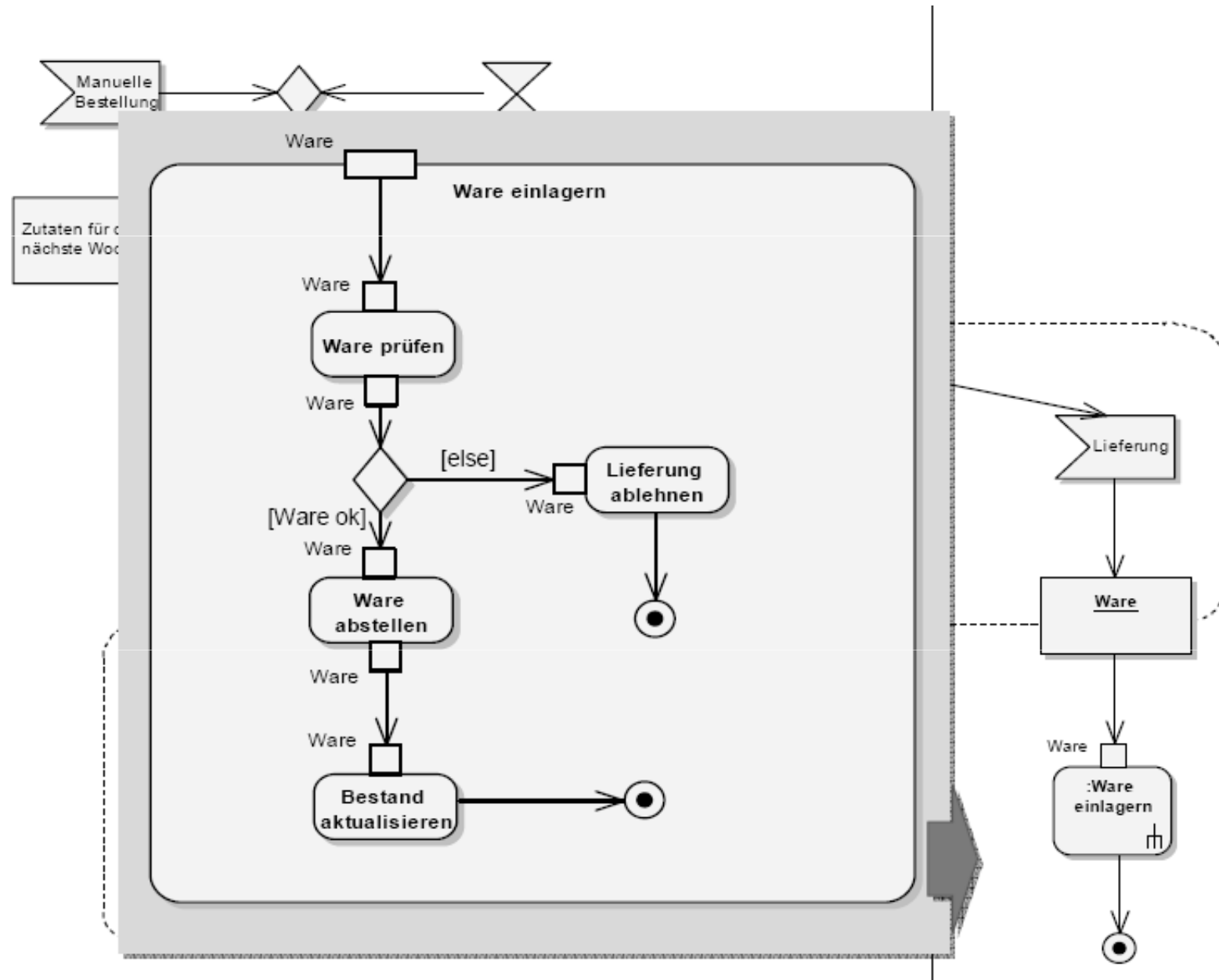


### • Example:



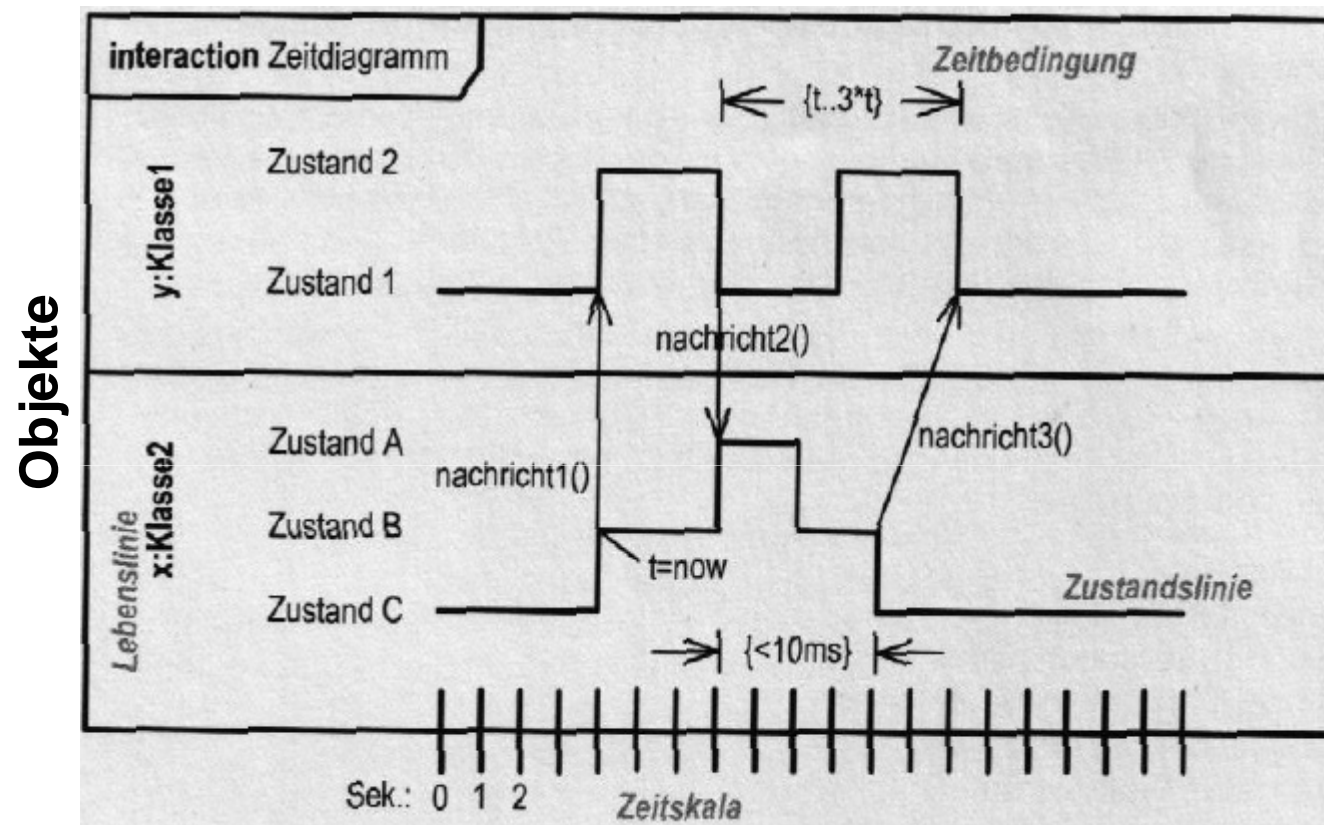
## 5.8 OO Modellierung mit UML: Aktivitätsdiagramm

- Example (showing nesting):



## 5.8 OO Modellierung mit UML: Zeitdiagramm

- Alternativ auch: Zeitverlaufdiagramm, Timing Diagram
- Gehört zur Kategorie der Interaktionsdiagramme
- Beschreibt die zeitlichen Bedingungen von Zustandswechseln mehrerer beteiligter Objekte (zeitliche Abhängigkeiten)
- Einsatz v.a. für Echtzeitsysteme



**Zeitachse mit Zeitpunkten, Zeiträumen und zu erfüllenden Bedingungen**

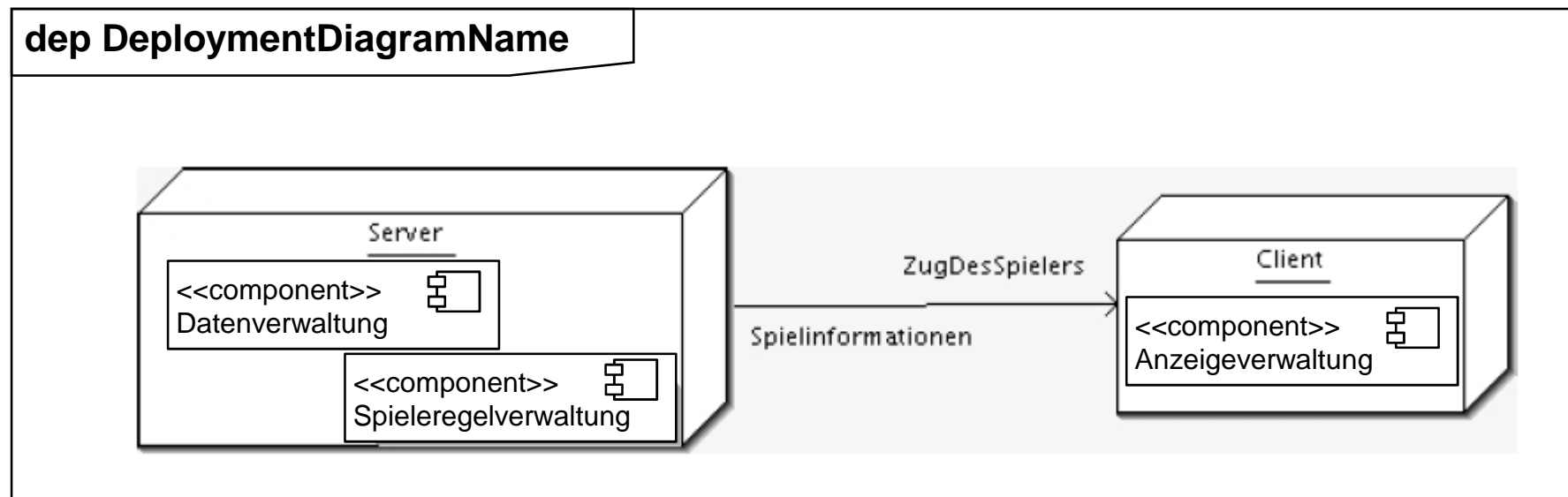
## 5.8 OO Modellierung mit UML: Verteilungsdiagramm

### Verteilungsdiagramme:

- Werden für Verteilungsaspekte im zu modellierenden System verwendet.
- Stellen die physikalische Aufteilung der Rechner und Netze/Knoten dar.
- Helfen bei der Analyse von Performance-Engpässen.

### Knoten:

- Enthalten zur Laufzeit die Komponenten, Objekte und sonstige Artefakte.
- Repräsentieren physikalische Einheiten.



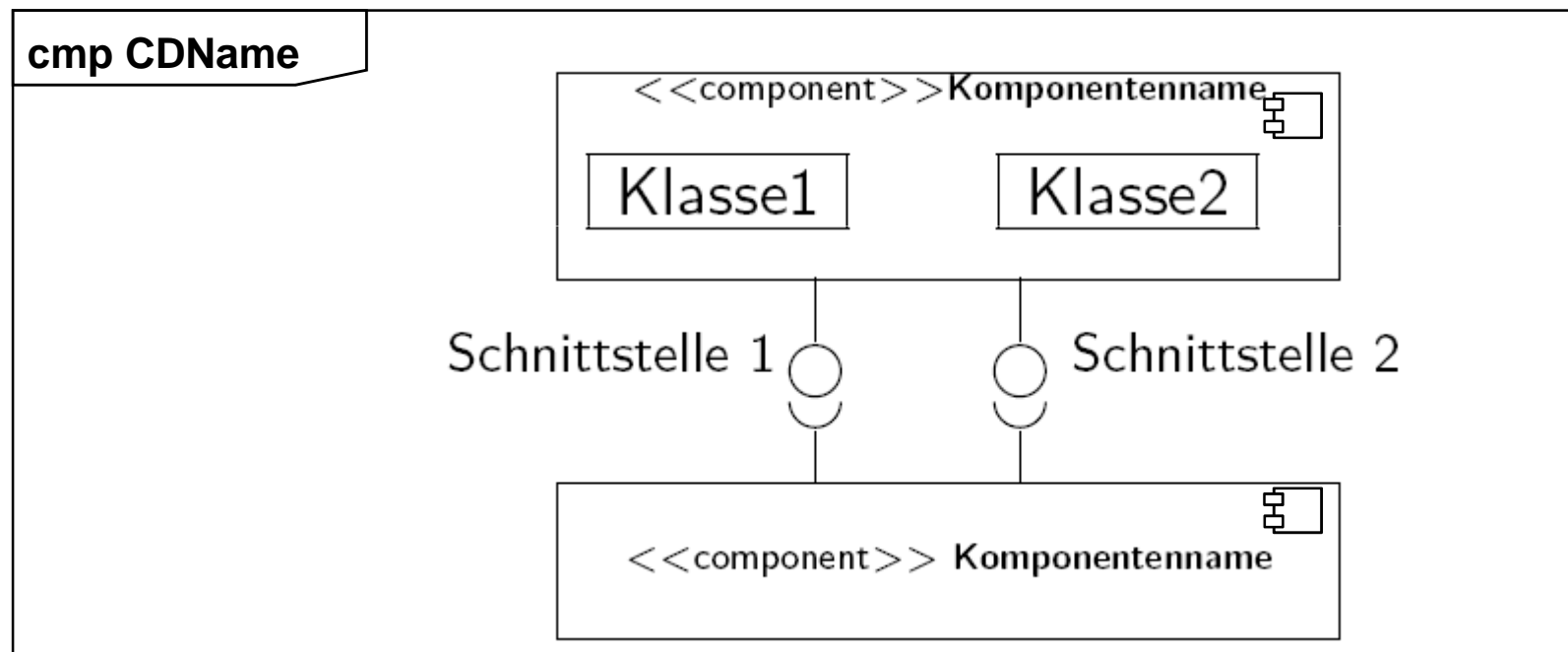
## 5.8 OO Modellierung mit UML: Komponentendiagramm

### Komponentendiagramme:

- Legen die Komponentenarchitektur des Systems fest.
- Beschreiben Abhängigkeiten und Schnittstellen zwischen Komponenten.

### UML-Komponenten:

- Sind einzeln entwickelbare Einheiten, die zu einem Ganzen kombiniert werden können.
- Zeichnen sich durch definierte Schnittstellen aus.



## 5.8 OO Modellierung mit UML: Bewertung

### **Vorteile:**

- **UML hat sich zum Standard für die objektorientierte Modellierung entwickelt.**
- **Eine Vielzahl von Werkzeugen unterstützt die Modellierung mit UML.**
- **UML besitzt gute Erweiterungsmöglichkeiten (→ Kommentare, Stereotypen, Profile).**

### **Nachteile:**

- **Auf Grund der sich teilweise überschneidenden Sichten muss auf die Konsistenz der Diagramme geachtet werden.**
- **Es existiert zur Zeit keine umfassende Semantik für UML.**
- **UML besitzt einen sehr großen Sprachumfang.**



## 5.8 OO Modellierung mit UML: Ursprünge

### Herkunft der Diagrammtypen

- **Use Case Diagram: Jacobson [92]**
- **Klassendiagramm: Coad/Jourdon [91] , Rumbaugh [91] , Booch [94] , ...**
- **Sequenzdiagramm: Rumbaugh [91] , Booch [94] , ...**
- **Objektdiagramm / Kollaborationsdiagramm (heute: Kommunikationsdiagramm): Booch [94]**
- **Zustandsübergangsdigramm: Harel [87]**
- **Deployment Diagram: Booch [94]**
- **Zeitdiagramme: Modellierung in der Elektronik, Elektrotechnik**

D. Harel. Visual Formalism for Complex Systems. Science of Computer Programming, 8:231 – 274, 1987

## 5.8 OO Modellierung mit UML: Werkzeuge

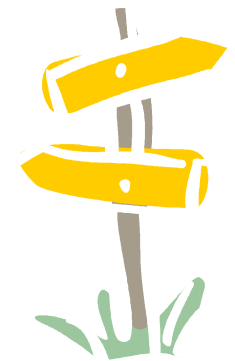
- Jude Community Edition  
<http://jude.change-vision.com/jude-web/product/community.html>
- Quick Sequence Diagram Editor <http://sdedit.sourceforge.net>
- Fujaba <http://wwwcs.uni-paderborn.de/cs/fujaba/>
- Sequenzdiagramme mit LATEX und TikZ <http://fauskes.net>
- Omondo EclipseUML <http://www.omondo.com/>
- Poseidon von Gentleware <http://www.gentleware.com/>
- Rational Rose Developer von IBM  
<http://www-01.ibm.com/software/awdtools/developer/rose/>

## Zusammenfassung und Ausblick

- 1 **Software-Krise und Software Engineering**
- 2 **Grundlagen des Software Engineering**
- 3 **Projektmanagement**
- 4 **Konfigurationsmanagement**
- 5 **Software-Modelle**
- 6 **Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**
- 7 **Qualität**
- 8 **... Fortgeschrittene Techniken**

- 5.1 Grundlagen und Modelltypen  
(Modellbegriff, Modellarten/Sichten, Einsatz, Modellvielfalt, Abstraktionsebenen)
- 5.2 Programmablaufplan
- 5.3 Struktogramm
- 5.4 Funktionsbaum
- 5.5 Structured Analysis
- 5.6 EBNF, Syntaxdiagramm
- 5.7 ERM
- 5.8 OO-Modelle mit UML

**bekannte  
Modelle bzw.  
Modellierungs-  
sprachen**



→ **Wege im Umgang mit der Software-Krise und Umsetzung der Grundlagen und Prinzipien:  
Einsatz von Modellen: weitere bekannte Modelle für verschiedene Sichten und Einsatzzwecke**