Keller (Stapel, Stack, LIFO)

Liste K=[$K(1)$, $K(2)$, ..., $K(n)$] mit beschränktem Zugriff

Operationen:

- pop: liefert oberstes Element $K(1)$
  entfernt oberstes Element: $K' = [\ K(2), ..., K(n)\ ]$
  (Fehler bei leerem Keller)

- push(x): speichert Element x als oberstes Element:
  $K' = [\ x, K(1), K(2), ..., K(n)\ ]$ (Fehler bei vollem Keller)

- isEmpty: "true", falls Keller leer
- isFull: "true", falls Keller voll
- clear: Keller leeren

Keller

Anwendungen

- Suchverfahren in Graphen:
  Tiefe-Zuerst-Suche, Back-Tracking:
  Speicherung der offenen Knoten (Liste *OPEN*)
- Textverarbeitung: *„rückgängig"-Befehle*
- Syntax-Analyse von Programmen
- Auswertung von Ausdrücken
- Laufzeitkeller

Auswertung von Ausdrücken

Darstellung in
umgekehrt polnischer Notation
(postfix-Darstellung)

$x + y * (z-x) + 12 * x$

$= x\ y\ z\ x\ -\ *\ +\ 12\ x\ *\ +$

```
push(x)
push(y)
push(z)
push(x)
push(pop - pop)
push(pop * pop)
push(pop + pop)
push(12)
push(x)
push(pop * pop)
push(pop + pop)
```


Prozedur-Keller

Prozedur-Segmente enthalten:

- Resultat
- Parameter
- Lokale Variable
- Rücksprung-Adresse

```
int fak(int n)
{  if (n==0) { return 1;}
   else { return n * fak(n-1); }
}
```

- Push bei Aufruf einer Prozedur
- Pop bei Beendigung einer Prozedur

Keller: Einsatz für Handlungsplanung

Push:

- Planschritte in umgekehrter Reihenfolge ablegen.

Pop:

- Nächsten Planschritt bearbeiten:
  - Einfache Aktion ausführen bzw.
  - (Verfeinerten) Teilplan zur Umsetzung des Planschritts einkellern

Warteschlangen (Queue, FIFO)

Liste $Q = [\ Q(1), Q(2), ..., Q(n)\ ]$ mit beschränktem Zugriff

Operationen:

- dequeue: liefert erstes Element $Q(1)$
  entfernt erstes Element: $Q' = [\ Q(2), ..., Q(n)\ ]$
  (Fehler bei leerer Schlange)

- enqueue(x) :
  speichert Element x als letztes Element:
  $Q' = [\ Q(1), Q(2), ..., Q(n), x\ ]$
  (Fehler bei voller Schlange)
- isEmpty, isFull, clear ...

Warteschlangen (Queue, FIFO)

Implementation:

-- verkettete Liste
-- "Ring-Array" mit Positionsreferenz
`front`: aktuell erstes Element
`rear`: aktuell letztes Element

Anwendung:

- Suchverfahren in Graphen
  Breite-Zuerst-Suche:
  Speicherung der offenen Knoten (Liste *OPEN*)

Graphen

Definitionen

1. Ein *(gerichteter) Graph* ist ein Paar G = [V, E] mit einer Menge V von *Knoten* ("vertex", "node") und einer Menge $E \subseteq V \times V$ von Kanten ("edge"):

   "Die Kante $[v_1,v_2]$ führt von v_1 nach v_2."

Allgemeiner (Mehfachkanten):
G = [V, E , f] mit f: f: $E \rightarrow V \times V$ (Inzidenz-Funktion)

2. Bei einem *ungerichteten Graphen* G = [V, E] ist die Relation E symmetrisch:
   $[v_1,v_2] \in E \leftrightarrow [v_2,v_1] \in E$

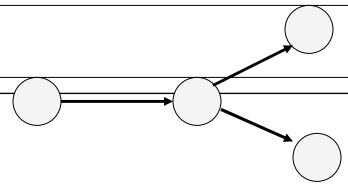
Graphen

3. *Knoten-beschrifteter Graph*: 4-Tupel G = [V, E, A, α] mit
   [V, E] ist ein Graph,
   A ist eine Menge (von Beschriftungen).
   α ist ein Funktion $\alpha: V \rightarrow A$

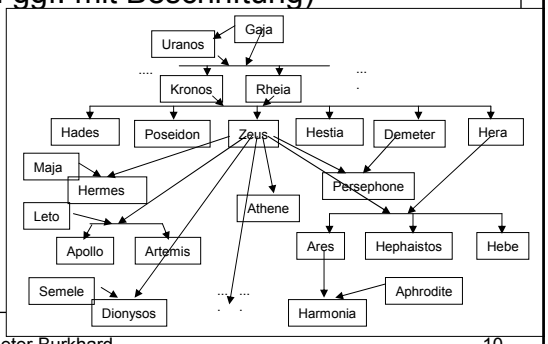
4. *Kanten-beschrifteter Graph*: 4-Tupel G = [V, E, B, β] mit
   [V, E] ist ein Graph,
   B ist eine Menge (von Beschriftungen).
   β ist ein Funktion $\beta: E \rightarrow B$

5. *Beschrifteter Graph:* 6-Tupel G = [V, E, A, α, B, β] mit
   [V, E, A, α] ist ein Knoten-beschrifteter Graph,
   [V, E, B, β] ist ein Kanten-beschrifteter Graph.

Darstellungsformen

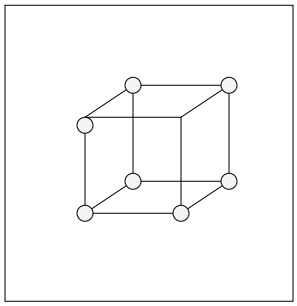


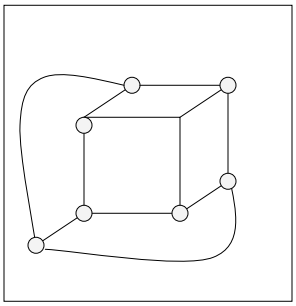
- Graphisch:
  Knoten (z.B.) als Kreise
  (mit Bezeichner und ggf. mit Beschriftung)
  Kanten als Bögen bzw. Pfeile
  (mit Bezeichner und ggf. mit Beschriftung)



Darstellungsformen

Planarer Graph: 2-D-Darstellung ohne Kreuzung von Kanten

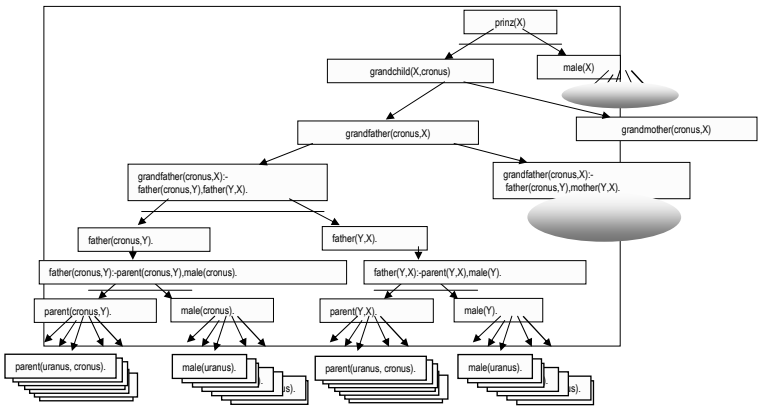




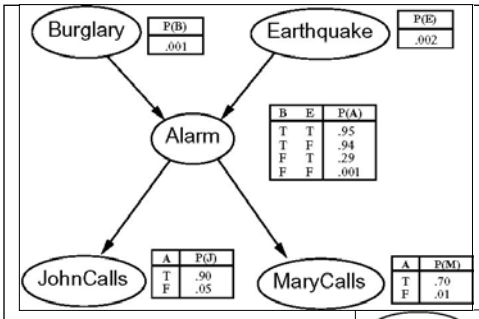
Beweisbaum

2 Sorten von Knoten: „bipartiter Graph“

für Und-Verzweigungen bzw. Oder-Verzweigungen

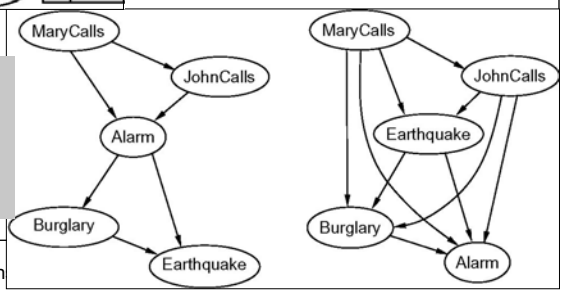


Kausale Abhängigkeiten: Belief-Netze

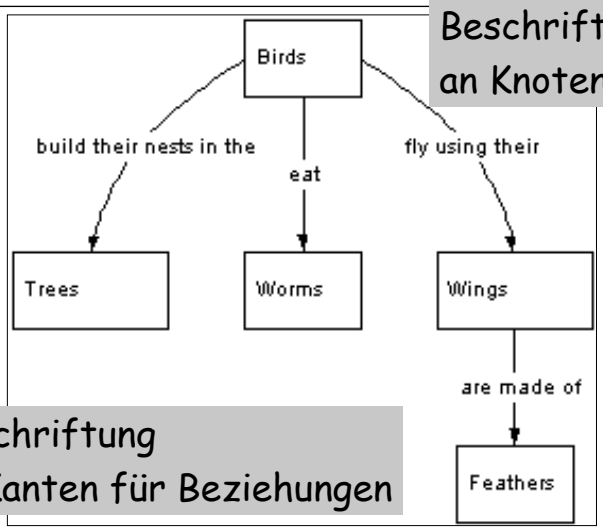


Beschriftung
an Knoten für bedingte
Wahrscheinlichkeiten

Unterschiedliche
Ansätze für
Abhängigkeiten



Sprachliche Zusammenhänge: Semantische Netze

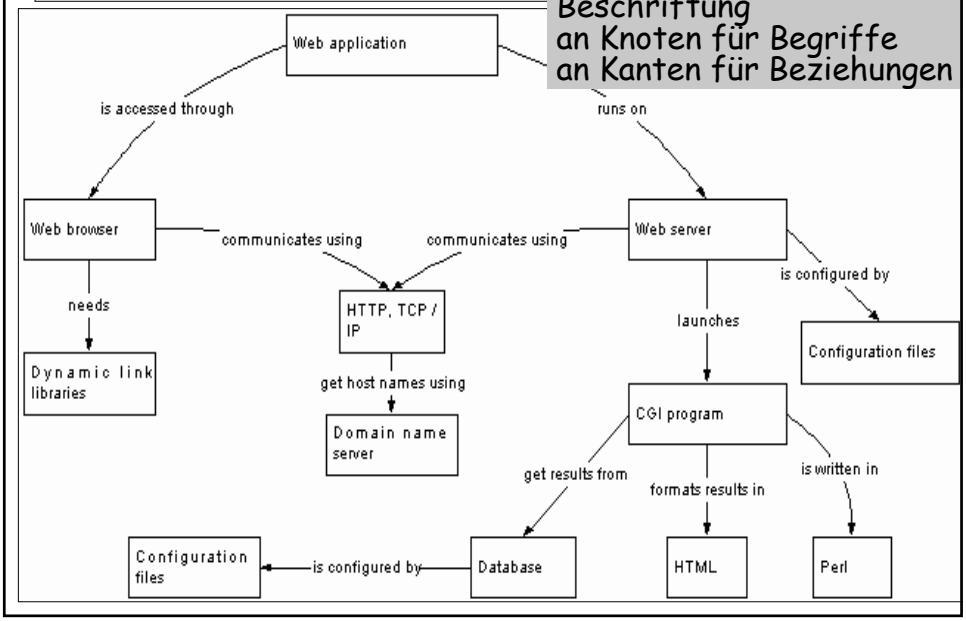


Beschriftung
an Knoten für Begriffe

Beschriftung
an Kanten für Beziehungen

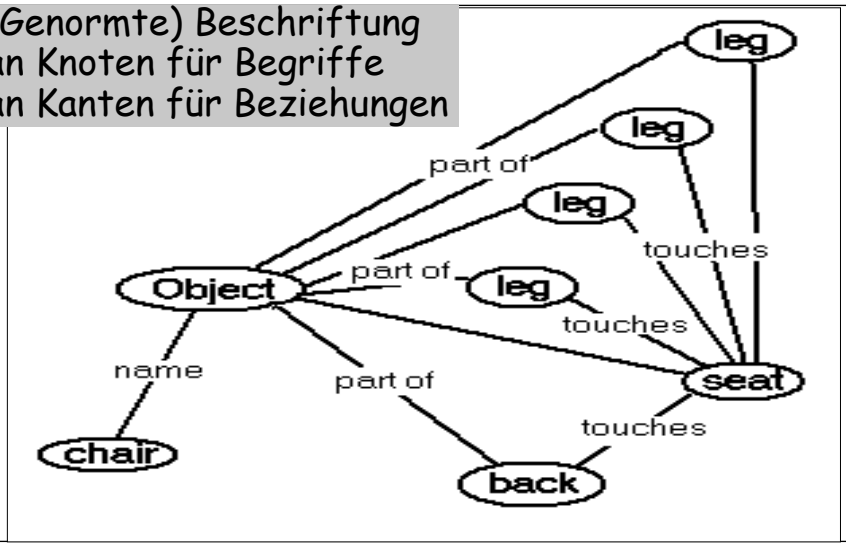
Inhaltliche Zusammenhänge

Beschriftung
an Knoten für Begriffe
an Kanten für Beziehungen



Inhaltliche Zusammenhänge

(Genormte) Beschriftung
an Knoten für Begriffe
an Kanten für Beziehungen



UML: Unified Modeling Language

- Graphische Notationen für Objektorientierte Modellierung
- Hilfsmittel für Erzeugung von Code
- Hilfsmittel für Dokumentation und Test

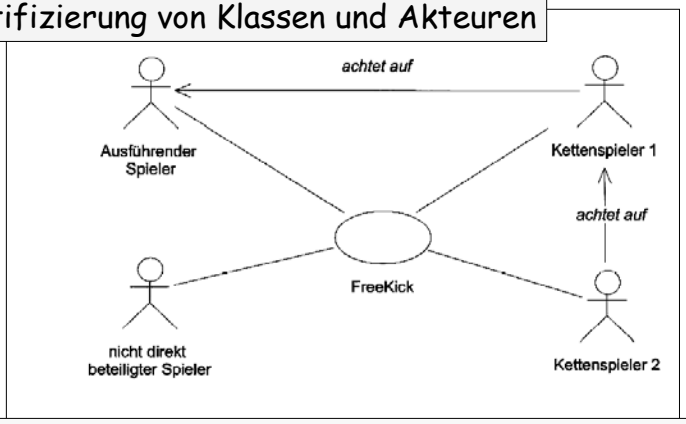
Graphische Darstellungen für ...

- Anwendungsfall-Beschreibung (use-case)
- Klassen/Objekte (Struktur)
  - –Vererbung
  - –Assoziationen: Beziehungen
  - –Aggregationen: Bestandteile („Container“)
- Sequenzdiagramm (Ablauf von Interaktionen)
- Zustandsdiagramm (Ablauf innerhalb eines Objekts)
- Aktivitätsdiagramm (Zusammenarbeit)

... und vieles mehr

Anwendungsfall-Beschreibung (use-case)

Funktionale Anforderungen
Erste Identifizierung von Klassen und Akteuren

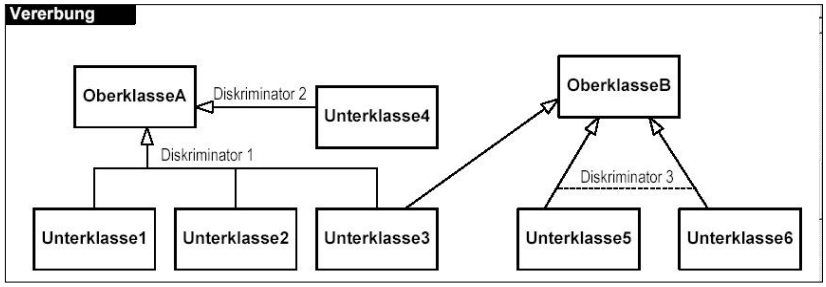


Beispiele:
Diplomarbeit Meinert/Sander: Der Intentionagent (HU, 2001)

Beschreibung von Strukturen: Klassen/Objekte

Diagramme zu den Beispielen (Stand 2003):

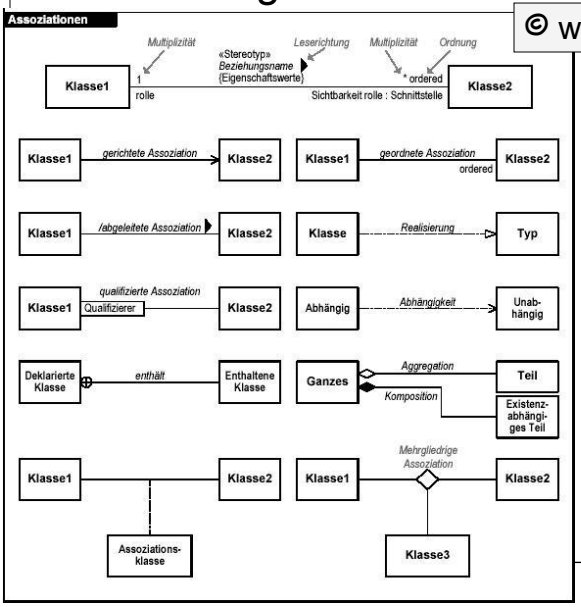
© http://www.oose.de/downloads/uml-notationsuebersicht.pdf



Klassendiagramm: Klassen und ihre Beziehungen

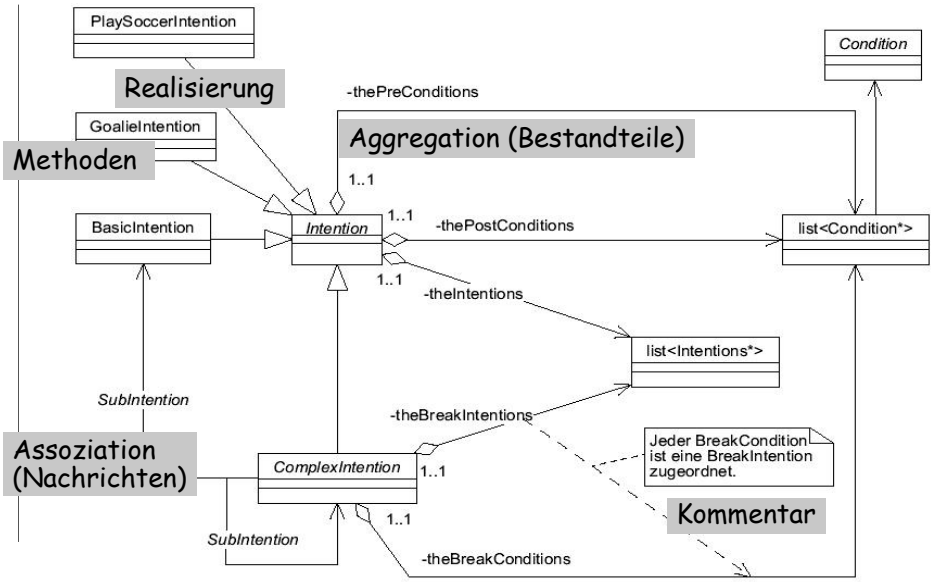
Beschreibung von Strukturen: Klassen/Objekte

© www.oose.de/uml

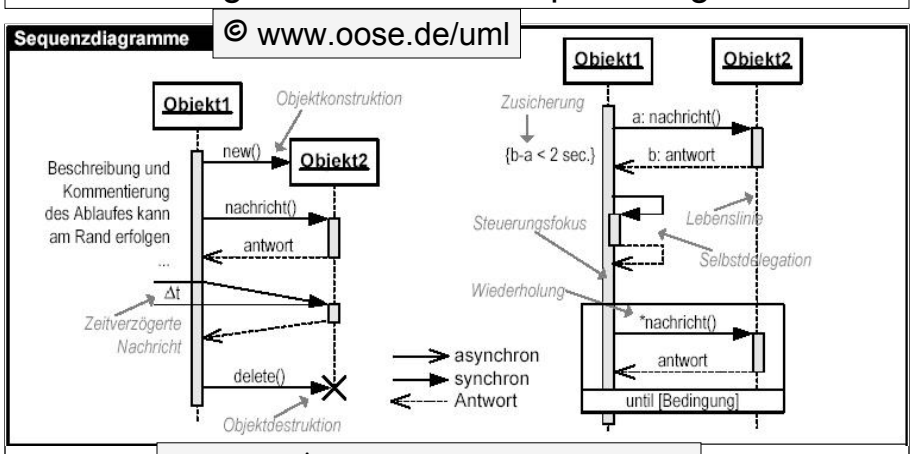


Objektdiagramm: Objekte und ihre Beziehungen

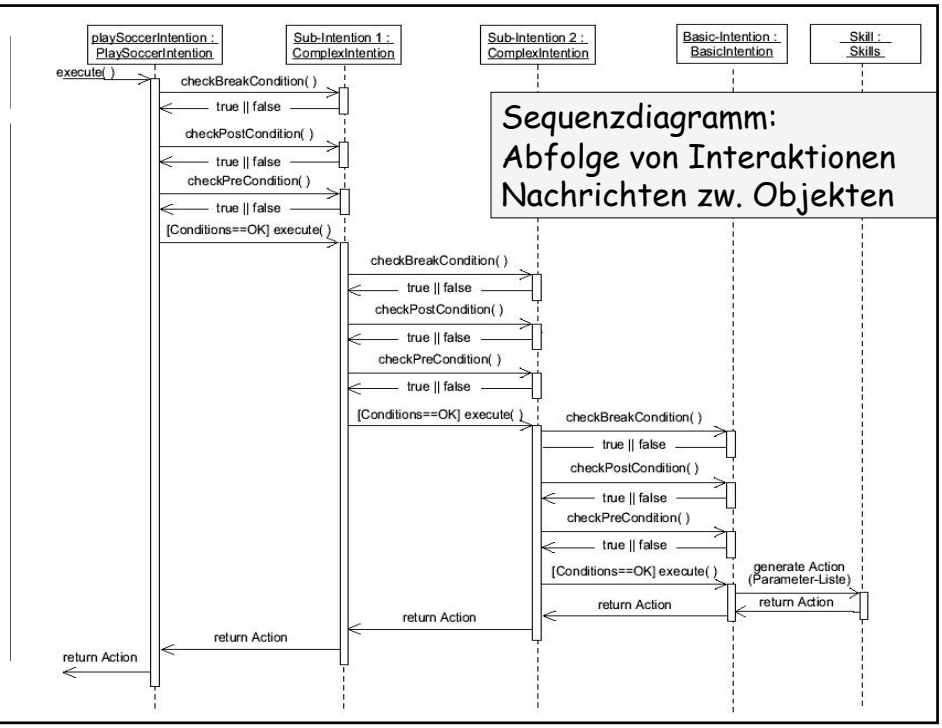
Beschreibung von Strukturen: Klassen/Objekte



Beschreibung von Abläufen: Sequenzdiagramme



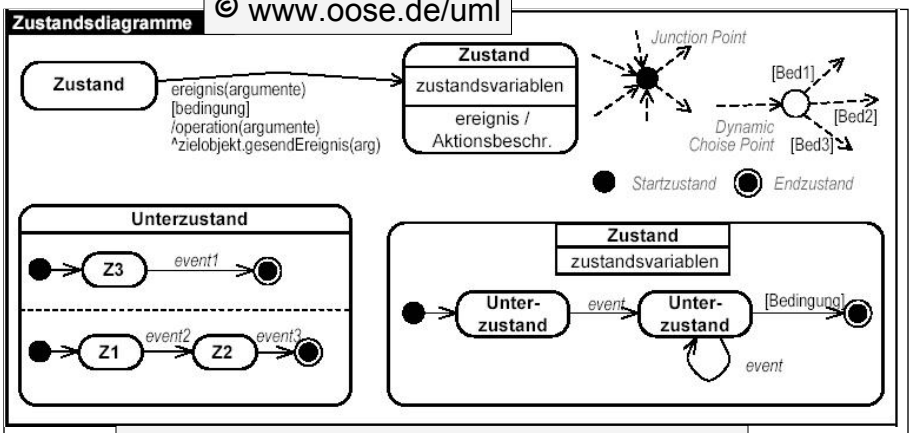
Sequenzdiagramm:
Abfolge von Interaktionen
Nachrichten zwischen Objekten



Sequenzdiagramm:
Abfolge von Interaktionen
Nachrichten zw. Objekten

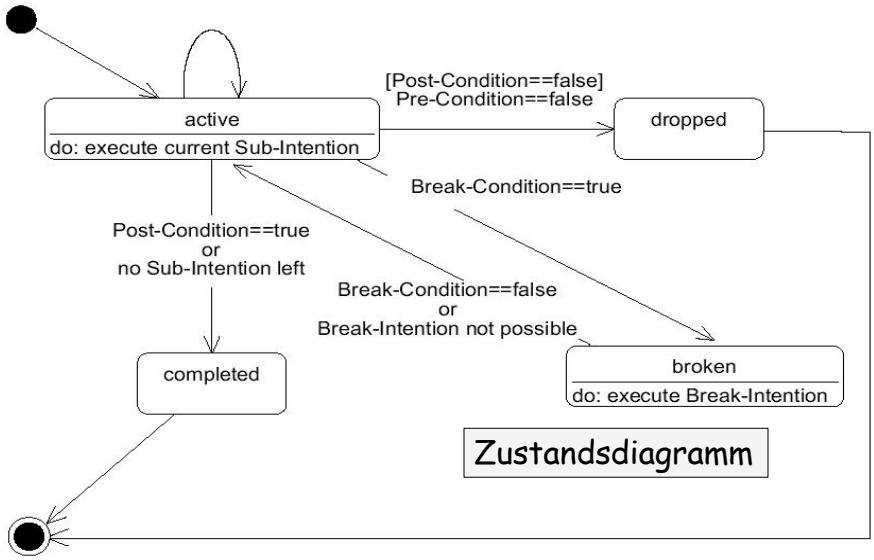
Beschreibung interner Abläufe: Zustandsdiagramm

© www.oose.de/uml

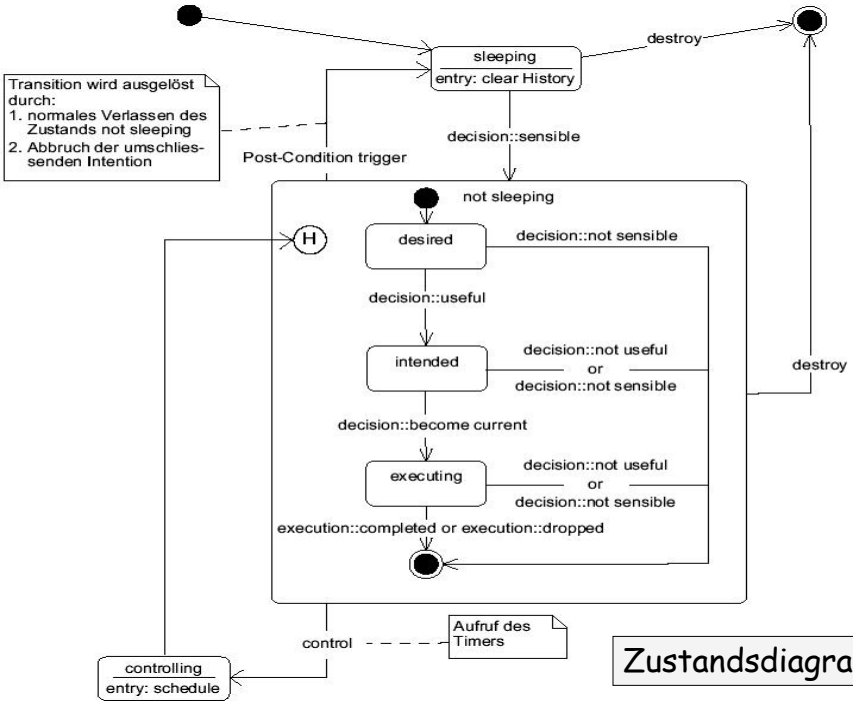


Zustandsdiagramm:
Abfolge von Ereignissen/Zuständen
während der Lebensdauer eines Objekts

Beschreibung interner Abläufe: Zustandsdiagramm

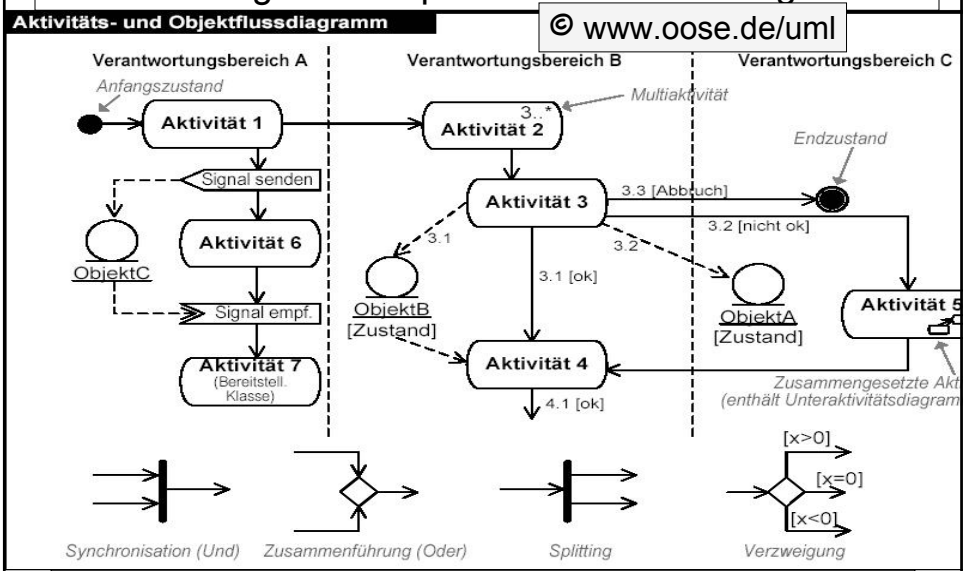


Zustandsdiagramm

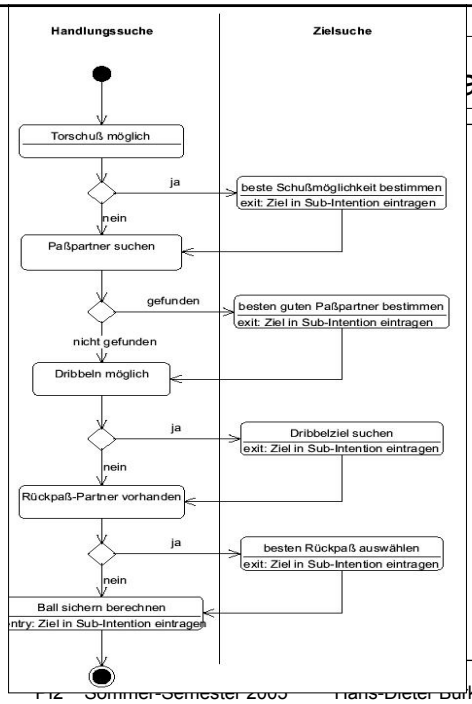


Zustandsdiagramm

Beschreibung von Kooperation: Aktivitätsdiagramme

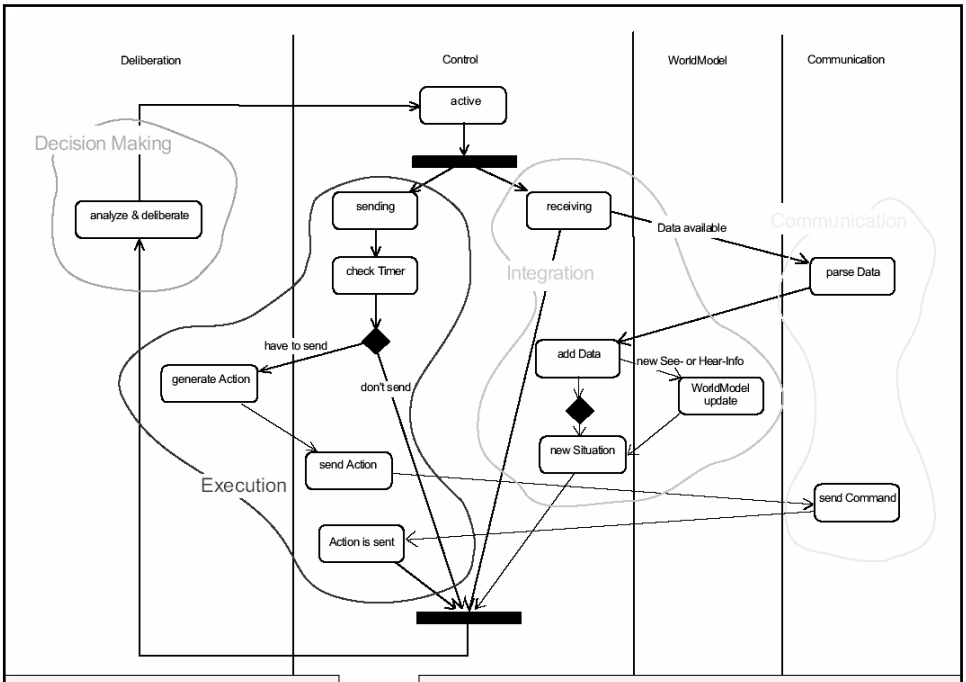


Zusammenarbeit von Objekten, Parallele Arbeit



ation: Aktivitätsdiagramme

Aktivitätsdiagramm



Aktivitätssdiagramm | Identifizierung von Komponenten