

## 34. Verhaltensanalyse für die Lebenszyklen von Objekten: Aktionsdiagramme (Aktivitäten-, Zustandsdiagramme) - wie modelliere ich das Leben von Objekten und andere Abläufe in Prozessen?

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 20-0.2, 15.06.20

- 1) Aktivitätendiagramme in UML
  - 2) Zustandsdiagramme in UML
  - 3) Verhaltens-, Steuerungs-, und Protokollmaschinen
  - 4) Implementierung von Steuerungsmaschinen
  - 5) Einsatz im Test
- A1) Andere Notationen  
A2) Impl. von Protokollmaschinen



- ▶ Zuser 7.5.3
- ▶ Störrle Kap. 10 (Zustandsdiagramme), Kap. 11 (Aktivitätsdiagramme)
- ▶ ST für Einsteiger: Kap. 10

**Grameen has given me an unshakeable faith in the creativity of human beings.** This has led me to believe that human beings are not born to suffer the misery of hunger and poverty.

**To me poor people are like bonsai trees.**

When you plant the best seed of the tallest tree in a flower-pot, you get a replica of the tallest tree, only inches tall. There is nothing wrong with the seed you planted, only the soil-base that is too inadequate. Poor people are bonsai people. There is nothing wrong in their seeds. Simply, society never gave them the base to grow on. All it needs to get the poor people out of poverty for us to create an enabling environment for them. Once the poor can unleash their energy and creativity, poverty will disappear very quickly.

**Let us join hands to give every human being a fair chance to unleash their energy and creativity.**

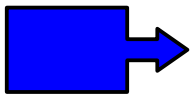
M. Yunus, Peace Nobel price lecture, Oslo, Dec 10, 2006

- ▶ Modellierung des Tamagotchi-Spiels mit Statemate, einer professionellen Statechart-Umgebung
  - <http://www4.in.tum.de/~philippj/tamasemi/slides/statemate.pdf>
- ▶ Wie man professionell Zustandsmaschinen aus einer domänenspezifischen Sprache auf C übersetzt, zeigt der RAGEL compiler
  - <http://www.colm.net/open-source/ragel/>

Notation: Wir schreiben:

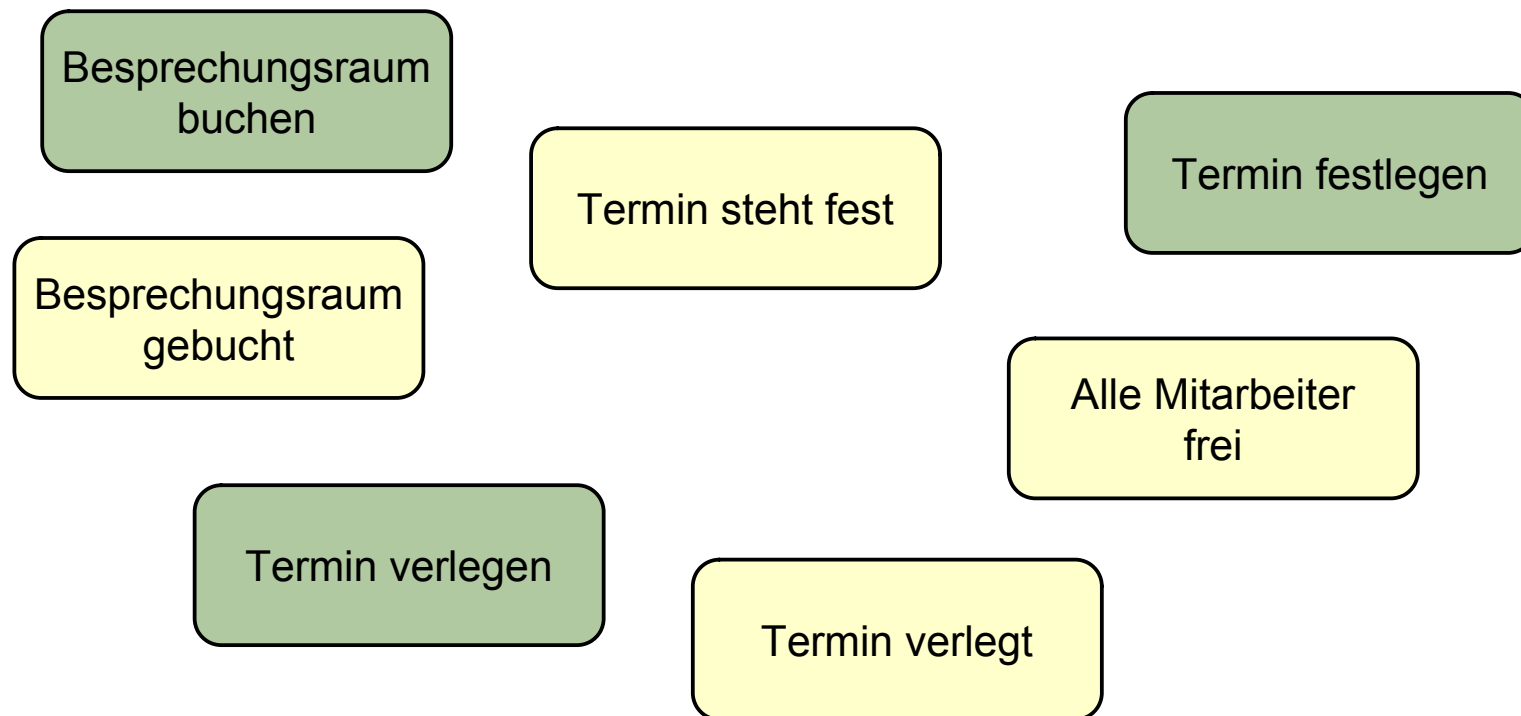
- ▶ Analysefragen in Font LeckerliOne: *Which tools does a text contain?*
- ▶ Zu analysierende Texte des Kunden in Font ComicJens: *Eine Teambesprechung ist ein Termin.*

# Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
  1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodelldgetriebene Modellierung mit UML
  1. Strukturelle metamodelldgetriebene Modellierung für das Domänenmodell
  2. Strukturelle Modellierung von komplexen Objekten
  3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
  1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
  2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
  3. (Funktionale querschneidende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent

# Problem: Wie analysiert man einen Text vom Kunden, der das Verhalten eines Objekts beschreibt?

- ▶ "Eine Teambesprechung benötigt einen Besprechungsraum. Bevor der gebucht werden kann, muss der Termin feststehen; das kann aber nur festgelegt werden, wenn alle Mitarbeiter, die an dem Termin teilnehmen sollen, frei sind. Andernfalls muss der Termin verlegt werden."



# Dynamische Analyse (Verhaltensanalyse)

**Dynamische Analyse** fragt nach den zeitlichen Aussagen des Kunden über Reihenfolgen, zeitliche Abfolgen, Ordnungen.

- ▶ Ergebnis:
  - **Verhaltensmodelle**
  - **Steuerungsmaschinen** (Lebenszyklen von komplexen Objekten)
  - **Technische Steuerungsmaschinen** (Lebenszyklen von eingebetteten Systemen)

*Was steckt genau in einem Objekt?  
Welche Phasen durchläuft ein Objekt oder ein System?  
In welche Zuständen befindet sich das Objekt?*

# Punktweise und querschneidende dynamische Verfeinerung in der OOA

**Punktweise funktionale Verfeinerung** ist eine funktionale Verfeinerung eines Modellfragmentes (meist Objekt oder Methode), die *punktweise* geschieht, d.h. pro Modellfragment separat durchgeführt wird.

Kap. 34

- ▶ Ergebnis der Verfeinerung einer Klasse im Strukturmodell:
  - **Lebenszyklus** des komplexen Objektes
  - **Implementierung** einer Methode

*Welchen Lebenszyklus durchläuft ein Objekt?  
Welche Zustände oder Aktivitäten hat eine Methode?*

**Querschneidende funktionale Verfeinerung** ist eine funktionale Verfeinerung *mehrerer* Modellfragmente gleichzeitig, die *querschneidend* geschieht.

Kap. 35

- ▶ Damit kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen, eine *Szenarienanalyse*, die quasi die Draufsicht auf ein Szenario ermittelt
  - Siehe Kapitel “Szenarienanalyse”

*Welches Interaktionsprotokoll durchläuft eine Gruppe von Objekten?*

## 34.1. Modellierung von Prozessen mit Aktionsdiagrammen

Aktionsdiagramme: Aktivitätsdiagramme (AD),  
Statecharts (SC)

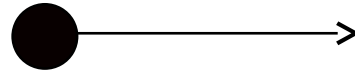




# Start- und Endzustand

- ▶ Jedes Aktionsdiagramm (Statechart, Aktivitätendiagramm) sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".

- ▶ **Notation:**



- ▶ Ein Aktionsdiagramm kann einen oder mehrere Endzustände haben.

- ▶ **Notation:** ("*bull's eye*")



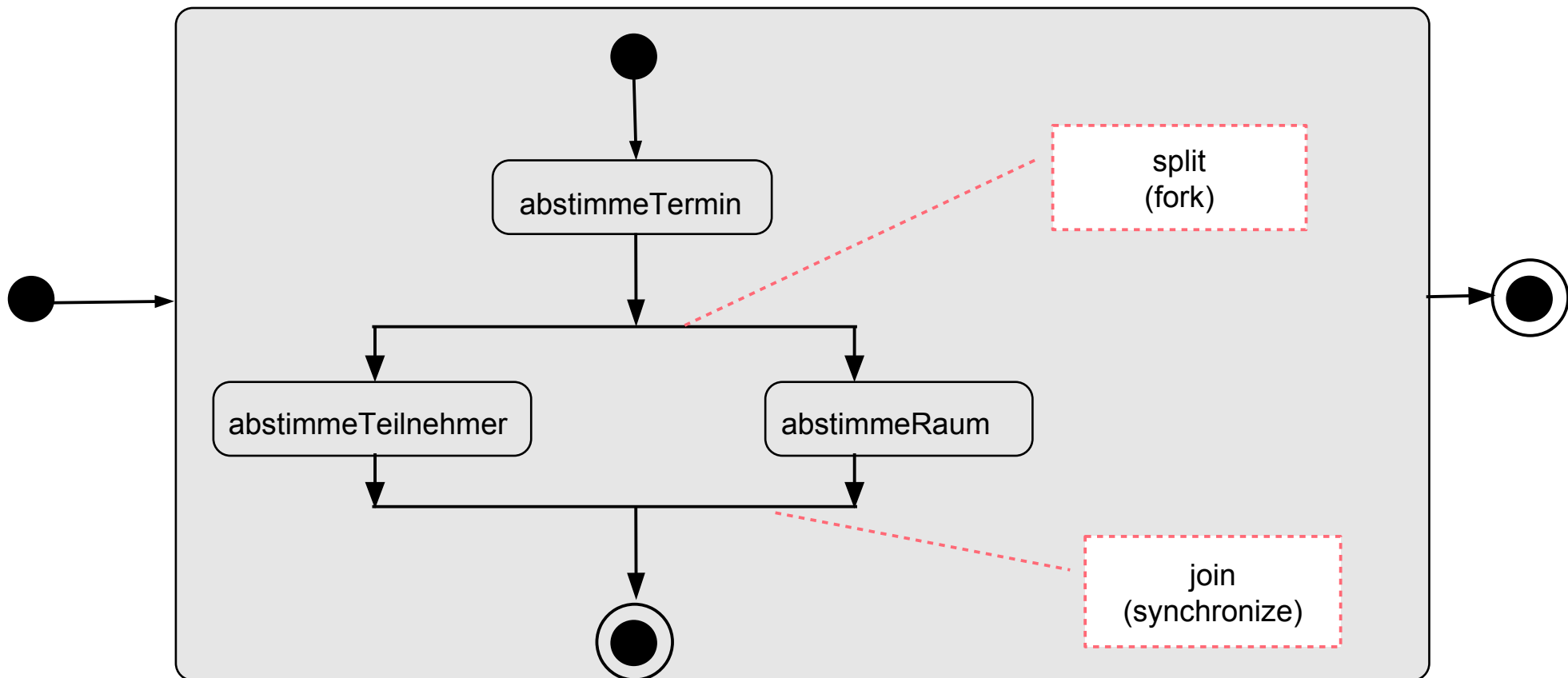
## 34.1.1. Aktivitätsdiagramme

- ▶ zur Spezifikation von Aktivitäten und ihren Abhängigkeiten



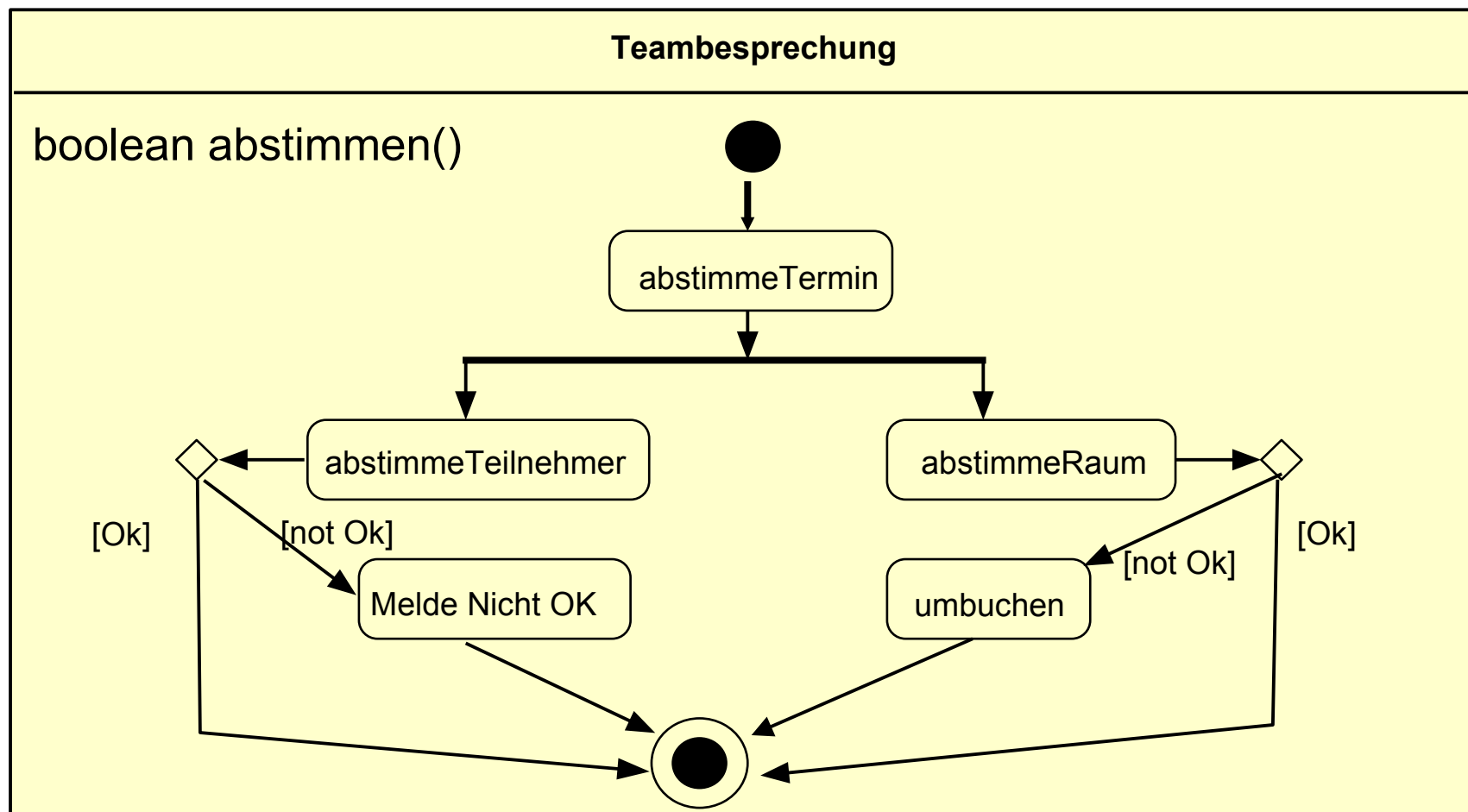
# Dynamische Modellierung (Verhaltensmodellierung)

- ▶ Eine Signatur eines Objektes oder einer Methode muss **funktional verfeinert** werden
  - Das Verhalten (dynamische Semantik) muss spezifiziert werden (partiell oder vollständig)
  - Daher spricht man von *Verhaltensmodellierung* oder *dynamischer Modellierung*
  - und von *punktweiser Verfeinerung* einer Klassen- oder Methoden-Signatur
- ▶ Einfachste Form: Angabe von Aktivitäten, verknüpft mit Steuer- und Datenfluss
  - Geschachtelt in eine Oberaktivität



# Aktivitätsdiagramm als Verhalten einer Methode (activity diagram)

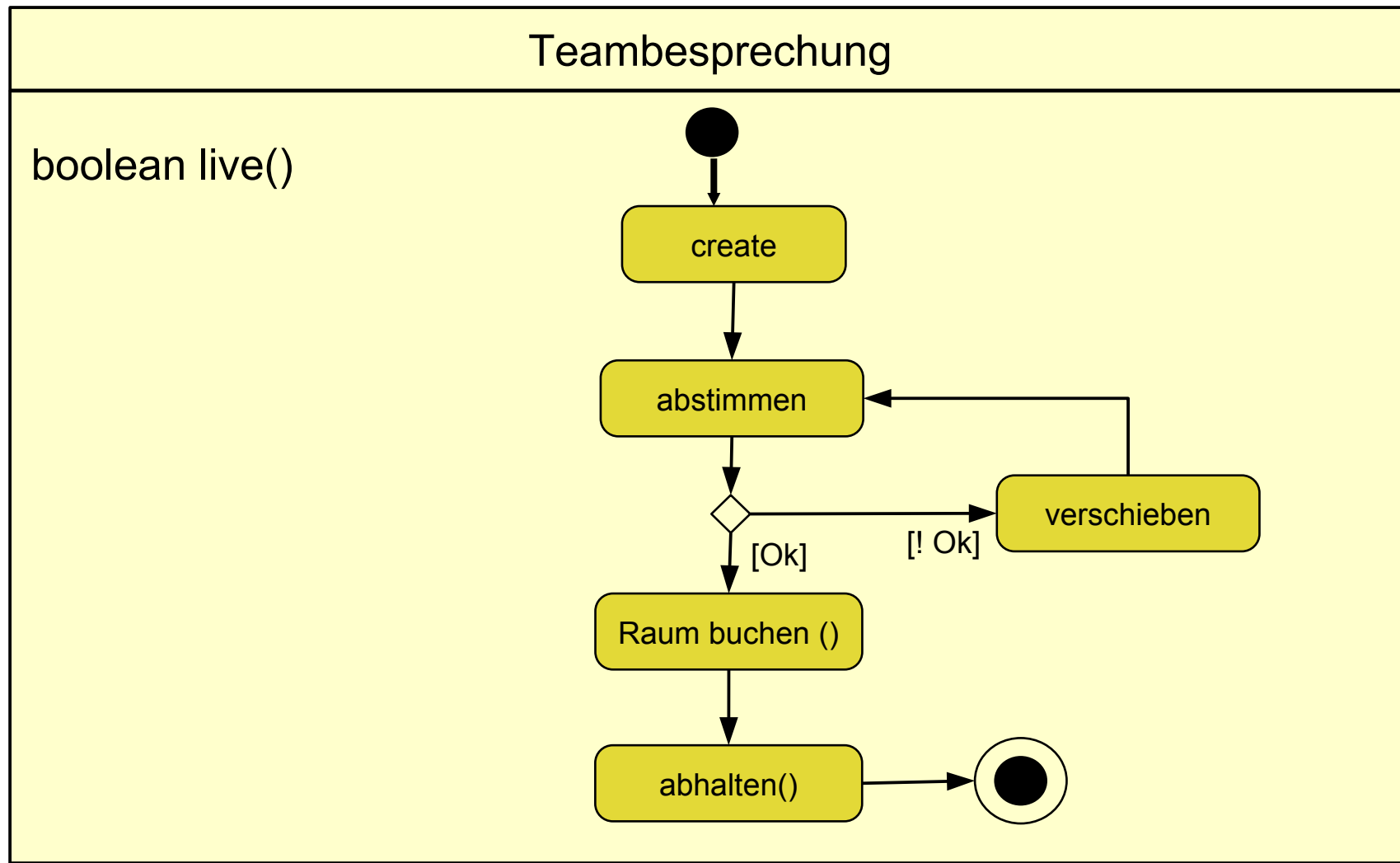
- ▶ Aktivitätsdiagramme können das Verhalten einer Methode beschreiben, dann werden sie in ein Abeitel der Klasse notiert
- ▶ Aktivitäten, verbunden durch Datenfluß (Datenflußdiagramm, data-flow diagram)
  - Parallele Aktivitäten in parallelen Zweigen
  - Bedingungen (guards) bestimmen, ob über eine Kante Daten fließen (*bedingter Datenfluß*)



[nach  
Pfleeger]

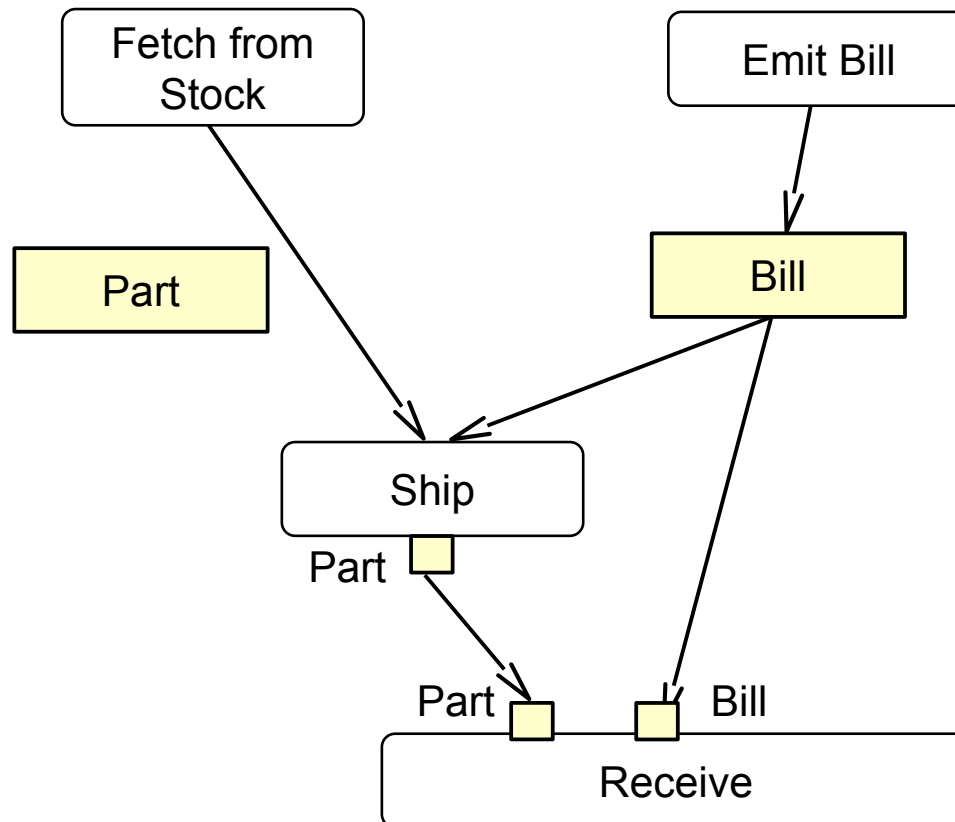
# Aktivitätsdiagramm für Lebenszyklus eines Objekts

- ▶ Viele Objekte müssen in einer bestimmten Art und Weise aufgerufen werden, von ihrer Geburt bis zum Tod
- ▶ AD beschreiben den *Arbeitsfluss (Workflow)* der Methoden (hier: abstimmen() wird aufgerufen)



# Verschiedene Notationen für Datenfluß

- ▶ Objekte, die zwischen Aktivitäten fließen, können verschieden notiert werden
- ▶ Pins sind benannte Parameter der Aktivitäten
  - Unterscheide Pins (von Aktivitäten) von Ports (von Komponenten)!



## 34.2 UML-Zustandsdiagramme (Zustandsmaschinen, Statecharts)

Zustandsmaschinen spezifizieren *Zustände* und ihr Aufeinanderfolgen

Zustandsmaschinen gehören zu jUML, wenn sie verlustfrei in Code überführt und zurücküberführt werden können (round-trip engineering)



# Zustandsbasierte dynamische Modellierung für sicherheitskritische Systeme

17

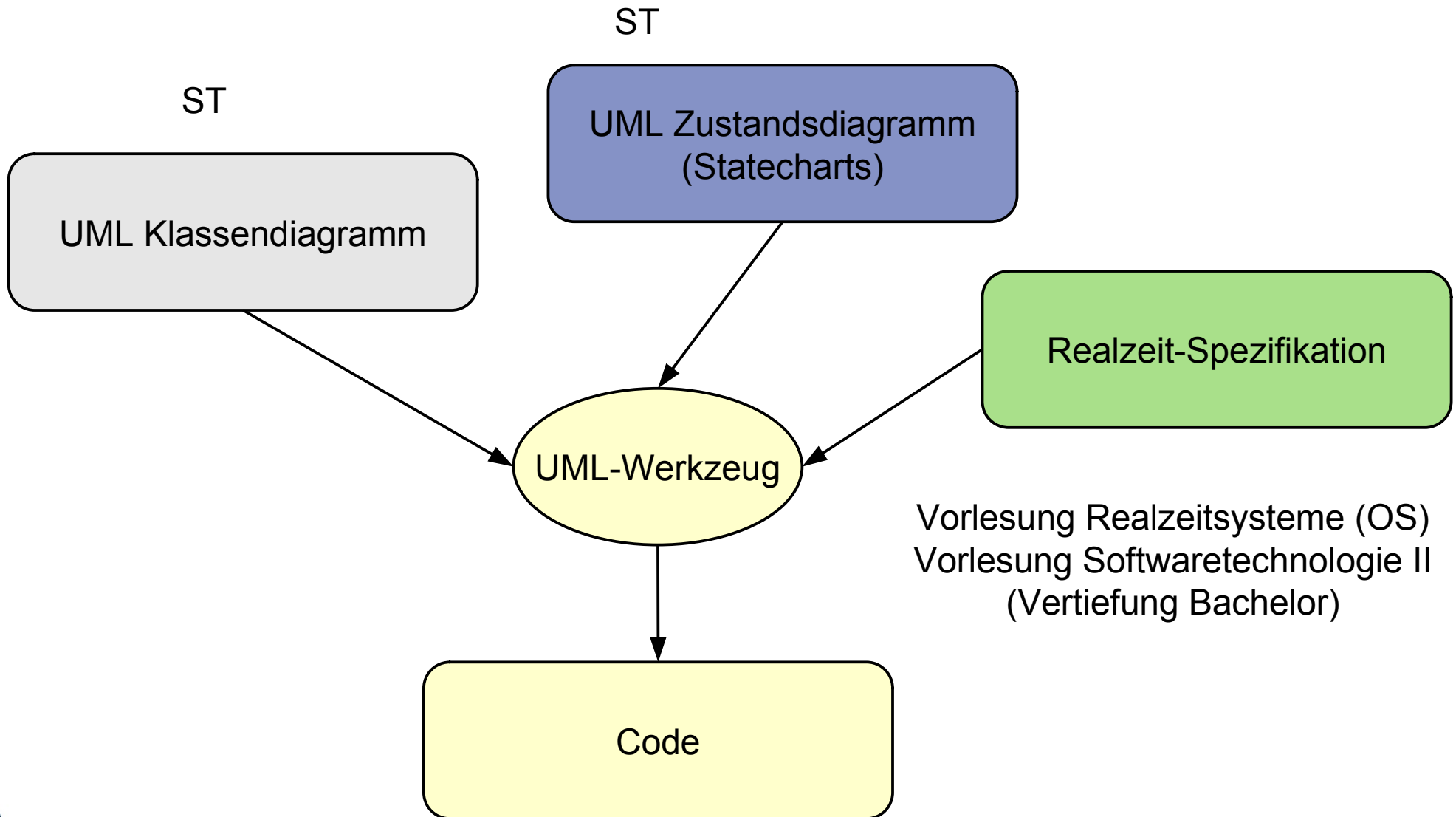
Softwaretechnologie (ST)

- ▶ Objekt-Verhalten und Szenarien können auch *zustandsbetont (phasenorientiert, modi-orientiert)* analysiert werden
- ▶ Beispiel: Spezifikationen von Zustandsautomaten für Fly-by-wire und Drive-by-wire
- ▶ Methodik: Analyse und Entwurf mit UML-Statecharts





- ▶ Die Paderborner Railcabs arbeiten mit UML-Statecharts, die mit Realzeit-attributen angereichert sind (real-time statecharts):



Vorlesung Realzeitsysteme (OS)  
Vorlesung Softwaretechnologie II  
(Vertiefung Bachelor)

- ▶ **Definition:** Ein *Zustand (Phase, Modus)* ist eine Eigenschaft eines Objektes oder Systems, die über einen begrenzten Zeitraum besteht.

- ▶ **Notation:**



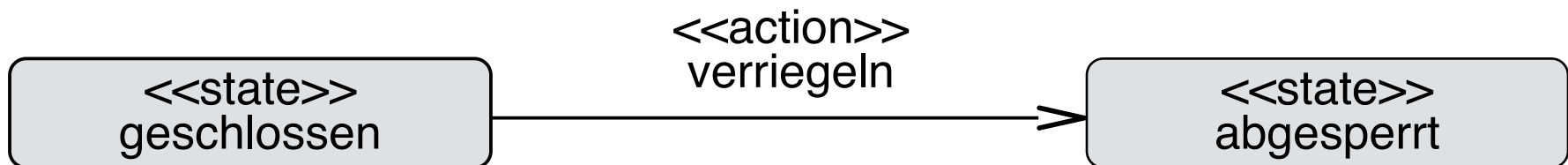
- ▶ Ein (technisches) **System** besteht aus einem (komplexen) Objekt oder einem Netz von Objekten, das Aufgaben ausführt
  - eines komplexes Softwaresystem
  - ein Arbeitsprozess
  - ein Produkt eines Arbeitsprozesses
  - ein einzelnes Objekt
- *Vorsicht! Zustände werden wie Aktivitäten mit ovalen Boxen notiert, gehören aber zu einem Zustandsdiagramm!*

## 34.2.1 Endliche Automaten ohne Aktionen (Akzeptoren)

- ▶ Theoretische Informatik, Automatentheorie:

Ein **endlicher Zustandsautomat (Akzeptor)** über einem Eingabealphabet  $A$  ist ein Tupel, bestehend aus:

- einer Menge  $S$  von Zuständen (Phasen, Modi)
  - einer (partiellen) Übergangsfunktion  $\text{trans} : S \times A \rightarrow S$
- einem Startzustand  $s_0 \in S$
- einer Menge von Endzuständen  $S_f \subseteq S$

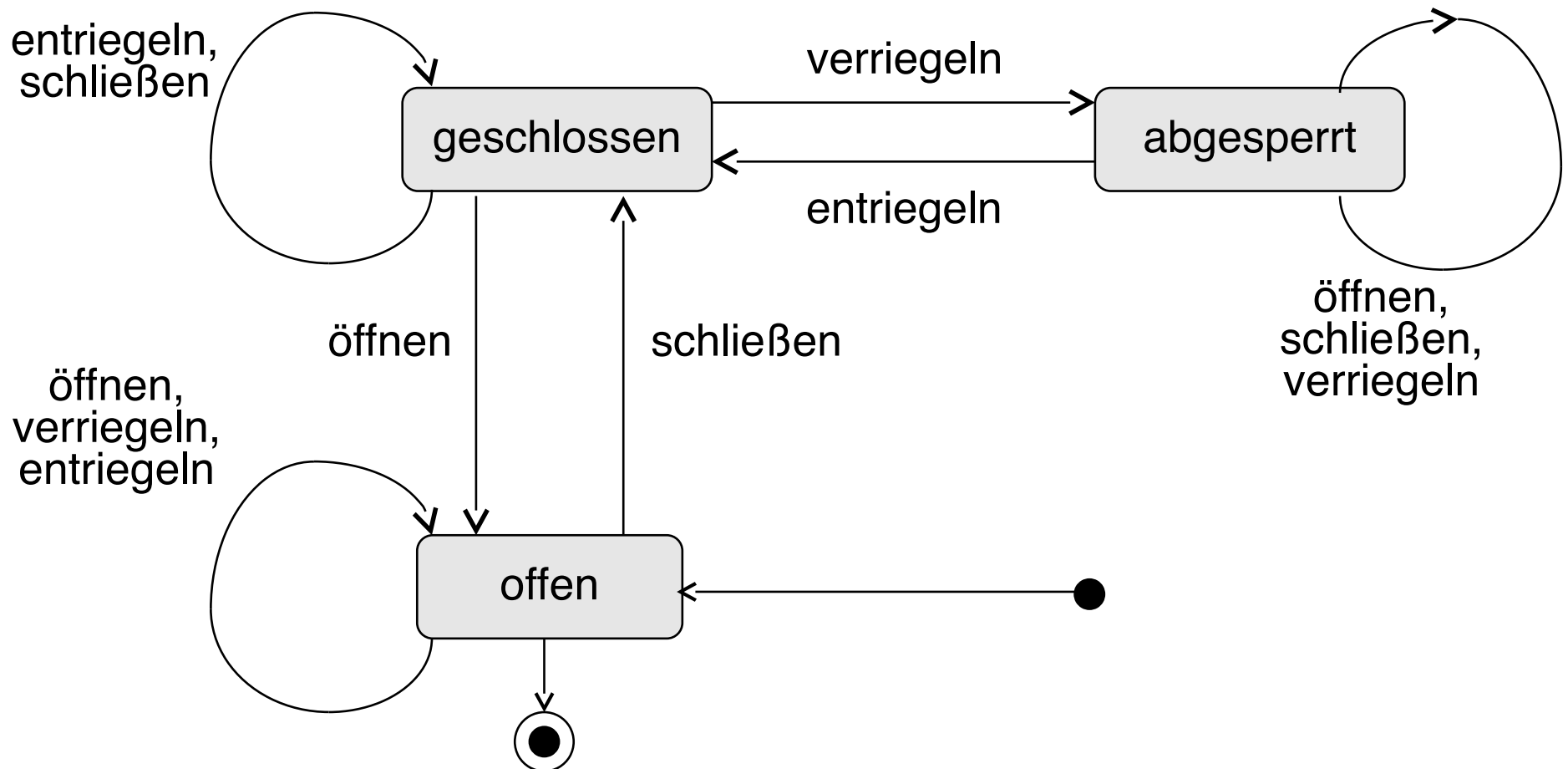


$\text{trans}(\text{geschlossen}, \text{verriegeln}) = \text{abgesperrt}$

**Achtung! Notation von Zuständen ähnlich zur Notation von Aktivitäten!**

# Beispiel: Zustandsmodell einer Tür

- ▶ Der Tür-Akzeptor stellt einen Prüfer für mögliche Aktionsfolgen für eine Tür dar
- ▶ In UML heisst der Akzeptor **Protokoll(zustands)maschine**, denn er akzeptiert eine Folge von Ereignissen (Protokoll)
- ▶ Der Zustandsautomat bildet die Phasen im Leben einer Tür ab



# Zustandsübergangstabellen von Protokollmaschinen

Tür Ausgangs-/ Endzustand	geschlossen	offen	abgesperrt
geschlossen	entriegeln, schließen	öffnen	verriegeln
offen	schließen	öffnen, verriegeln, entriegeln	-
abgesperrt	entriegeln	-	öffnen, schließen, verriegeln

- ▶ Tabellen bilden eine alternative Notation

## 34.2.2. Endliche Automaten mit Aktionen (Transduktoren)

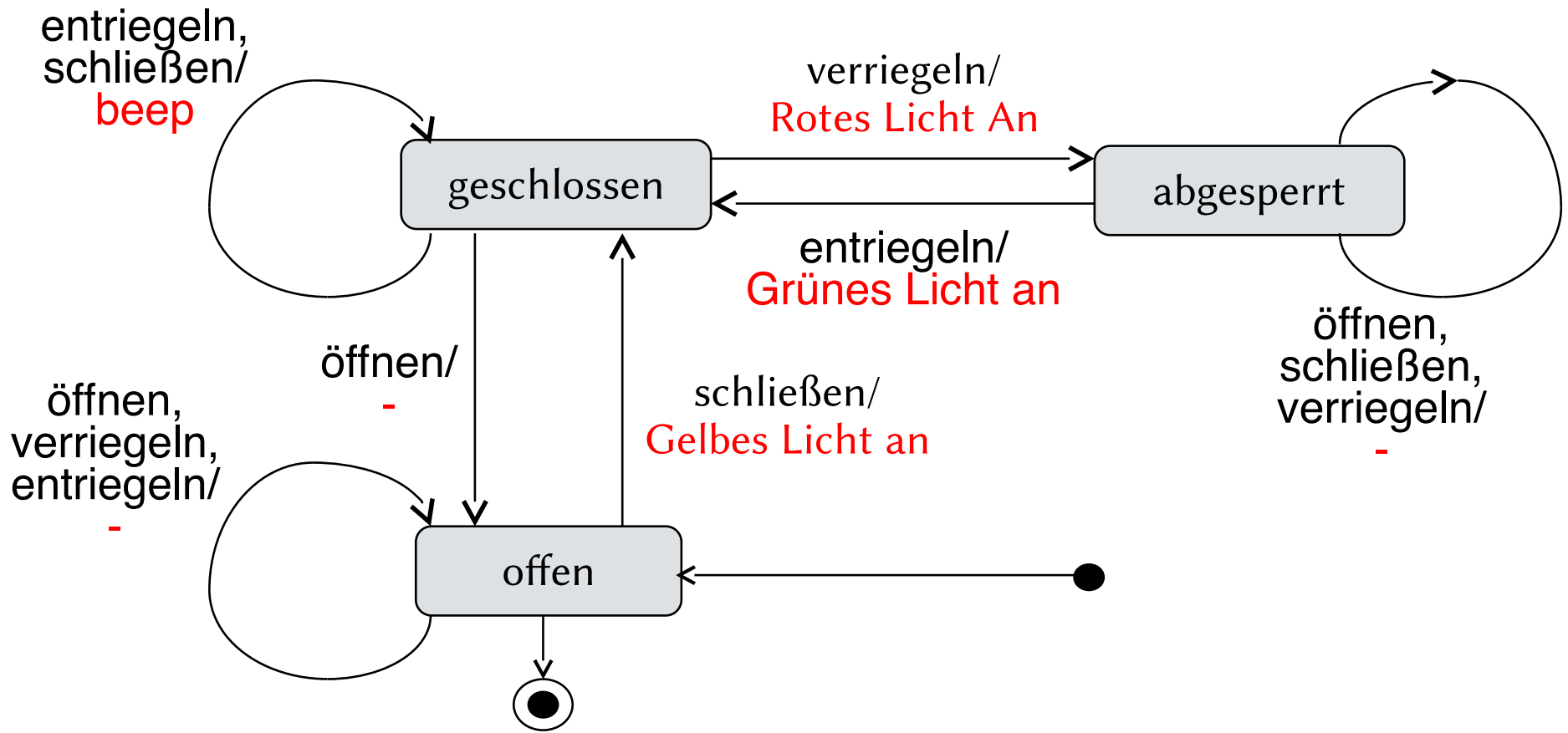
- ▶ Ein **endlicher Zustandsübersetzer (Transduktor, endlicher Übersetzer)** übersetzt eine Folge von Zeichen aus einem Eingabealphabet  $A$  in eine Folge von Zeichen aus **einem Ausgabealphabet  $B$** . Er ist ein Tupel, bestehend aus:
  - einer Menge  $S$  von Zuständen
  - einer (partiellen) Übergangsfunktion  $\text{trans} : S \times A \rightarrow S$ 
    - einem Startzustand  $s_0 \in S$
    - einer Menge von Endzuständen  $S_f \subseteq S$



$\text{trans}(\text{geschlossen}, \text{verriegeln}) = (\text{abgesperrt}) / \text{rotes Licht einschalten}$

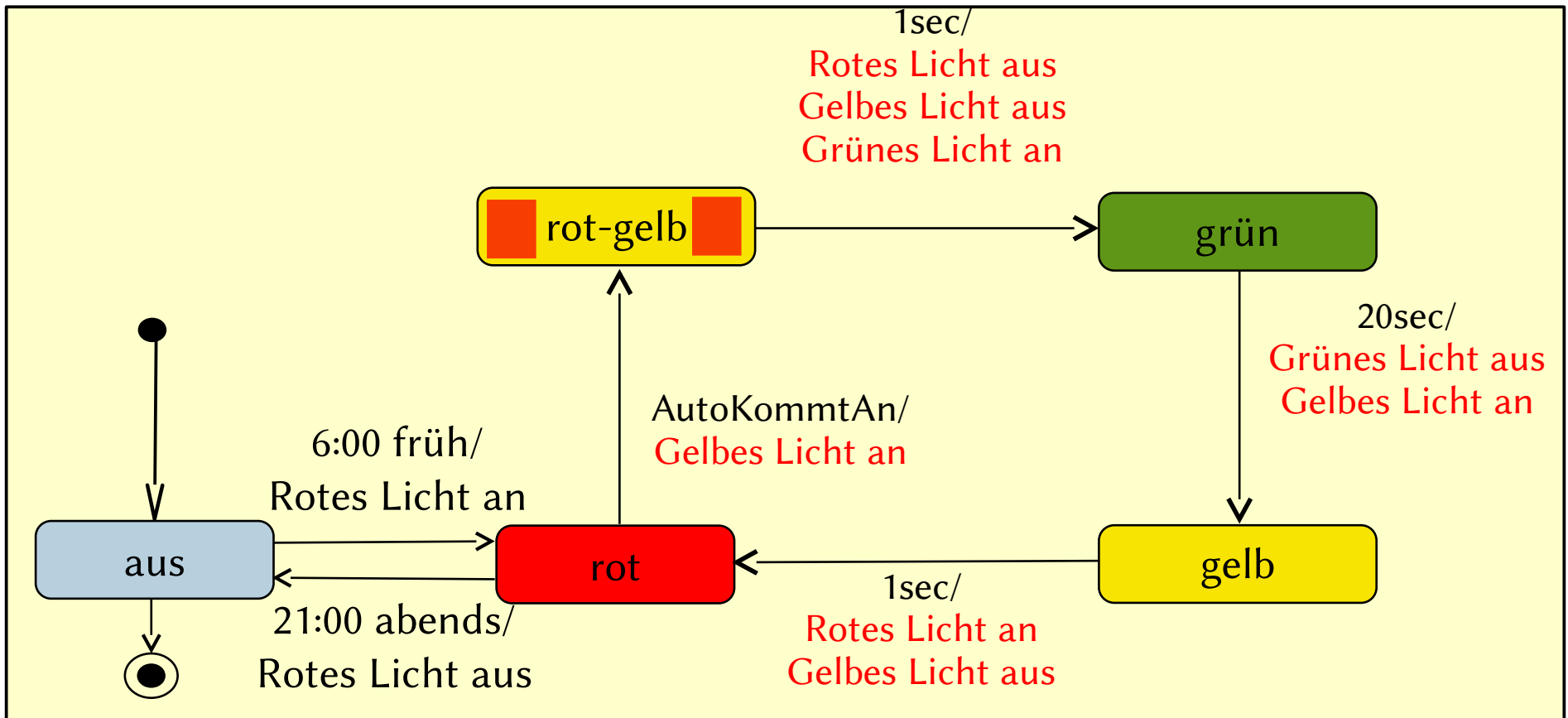
# Beispiel: Zustandsmaschine einer Tür

- ▶ Der Tür-Transduktor stellt zusätzlich zum Prüfer einen Steuerer (controller) für eine Tür-Zustandsmeldeampel dar
  - aus ihm kann ein Steuerungsalgorithmus für die Türampel abgeleitet werden
- ▶ In UML: **Zustandsmaschine (Verhaltensmaschine)**



# Beispiel: Zustandsmaschine einer bedarfsgesteuerten Ampel

- ▶ Welches Ereignis löst den Ampelzyklus aus?
- ▶ Welches Eingabealphabet hat der Transduktor (Ereignisse)? Welches Ausgabealphabet?
- ▶ Welche Sprachen (Mengen von Folgen von Ereignissen) übersetzt der Transduktor in einander?



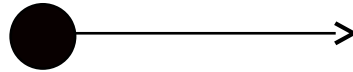


# Semantik eines Zustandsmodells

- ▶ Wir verstehen unter “Semantik eines Modells” seine Interpretation bzw. Seinen Ablauf bzgl. eines Interpretierers der Modellierungssprache.
- ▶ Ein Zustandsmodell ist endlich, definiert aber einen unendlichen Zustandsfolgenraum:
  - seine Semantik ist eine unendliche Menge von Pfaden über Zuständen und Aktionen (Aktionsfolgen):
  - Rollt man die Pfade des Zustandsmodells ab, entsteht ein unendlich tiefer Zustandsfolgen-Baum
- ▶ Die Semantik eines Zustandsmodells ist definiert als **Menge von Sequenzen (Aktionsfolgen)**:
  - in der Theoretischen Informatik:
    - Menge von "akzeptierten Wörtern" (Sprache über Grundalphabet von Ereignissen)
  - in der Softwaretechnik wird das interpretiert als:
    - Menge von zulässigen *Ereignisfolgen (Ereignissprache)*
    - Menge von zulässigen *Aufruffolgen oder Aktionen (Aufrufsprache)*
    - Menge von zulässigen *Pfaden in einem Graphen (Pfadsprache)*
- ▶ Wichtige Arten von Automaten:
  - *Akzeptor (Moore-Automat)*: "Automaten mit Ausgabe": Ausgabe bei Erreichen eines Zustands → Protokollmaschine
  - *Transduktor (Mealy-Automat)*: Ausgabe bei Übergang → Zustandsmaschine
    - ◆ Softwaretechnik: *Aktion* bei Übergang

# Start- und Endzustand (wie bei AD)

- ▶ Jedes Zustandsdiagramm sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".
- ▶ **Notation:**



- ▶ Ein Zustandsdiagramm kann einen oder mehrere Endzustände haben.
- ▶ **Notation:** ("*bull's eye*")



# Bedingte Zustandsübergänge in *Protokollmaschinen*

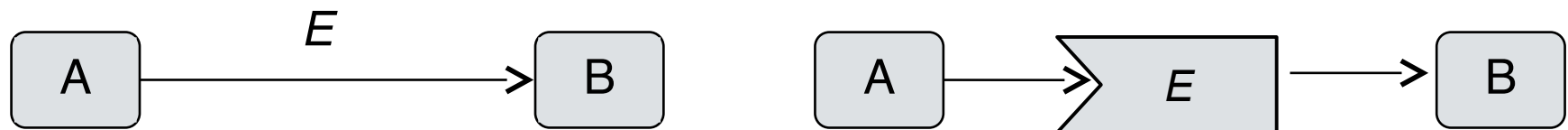


- ▶ **Definition** Eine **Bedingung** (*guard*) ist eine Boolesche Bedingung, die zusätzlich bei Auftreten des Ereignisses erfüllt sein muß, damit der beschriebene Übergang eintritt.
- ▶ **Notation:** Eine Bedingung kann folgende Informationen verwenden:
  - Parameterwerte des Ereignisses
  - Attributwerte und Assoziationsinstanzen (Links) der Objekte
  - ggf. Navigation über Links zu anderen Objekten
- ▶ **Beispiel:**

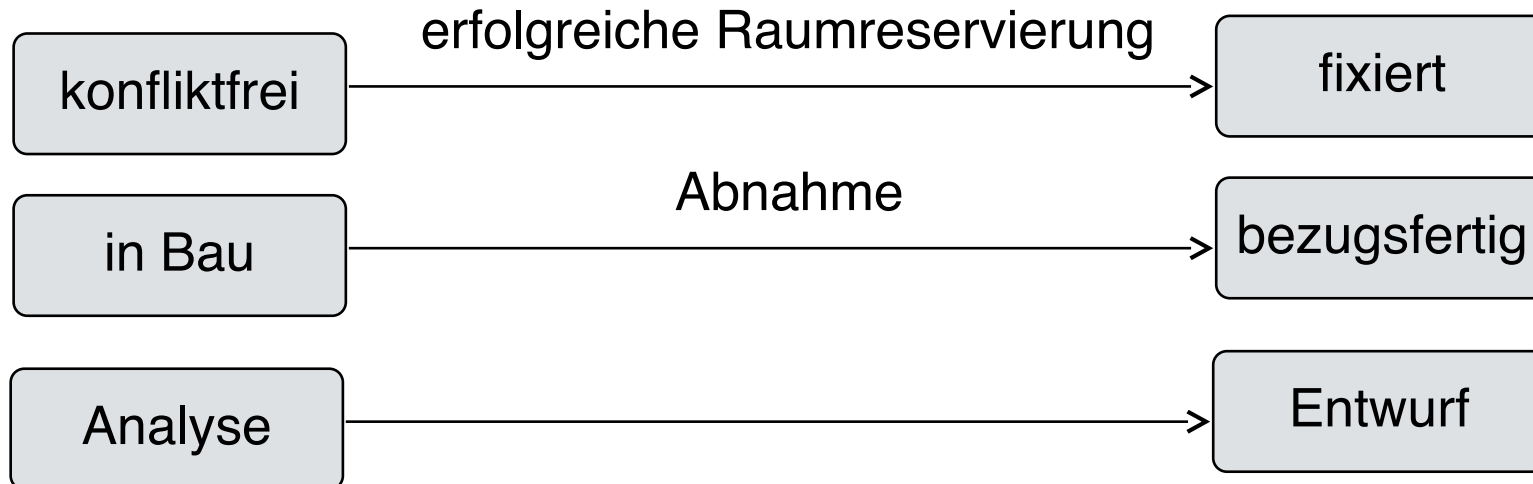


# Zustandsübergänge nach Ereignissen in *Protokollmaschinen*

- ▶ **Definition:** Ein *Zustandsübergang* von Zustand *A* nach Zustand *B* mit Ereignisnamen *E* besagt, daß im Zustand *A* bei Auftreten eines *E*-Ereignisses der neue Zustand *B* angenommen wird.
- ▶ In UML: *Protokollzustandsübergang (protocol transition)*, da Teil eines Akzeptors
- ▶ **Notation:**

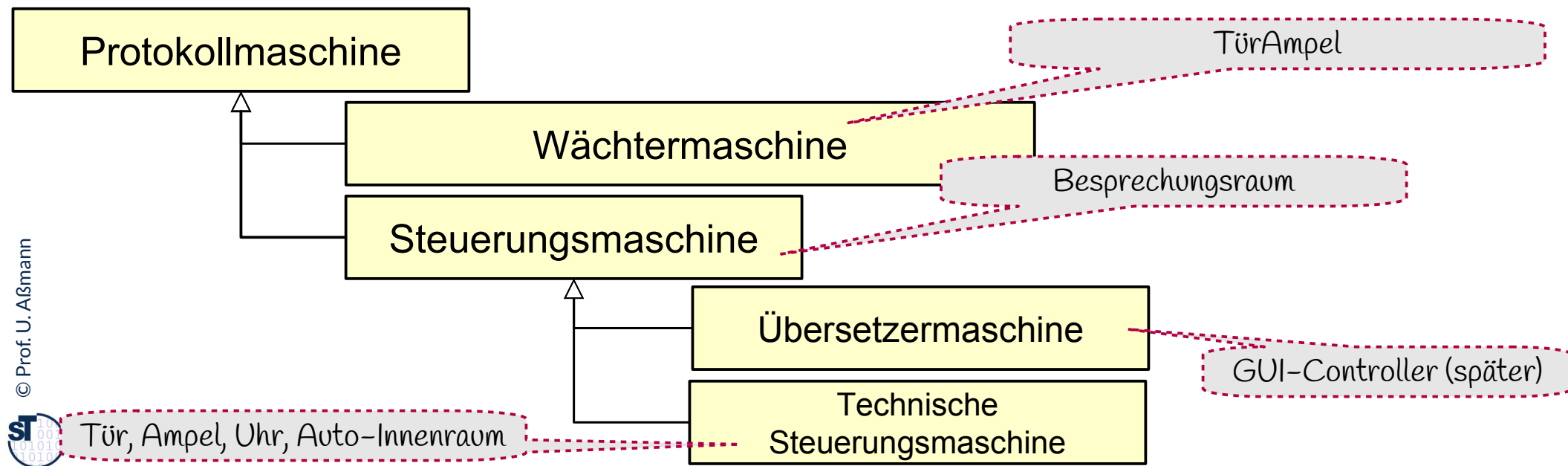


- ▶ **Beispiele:**

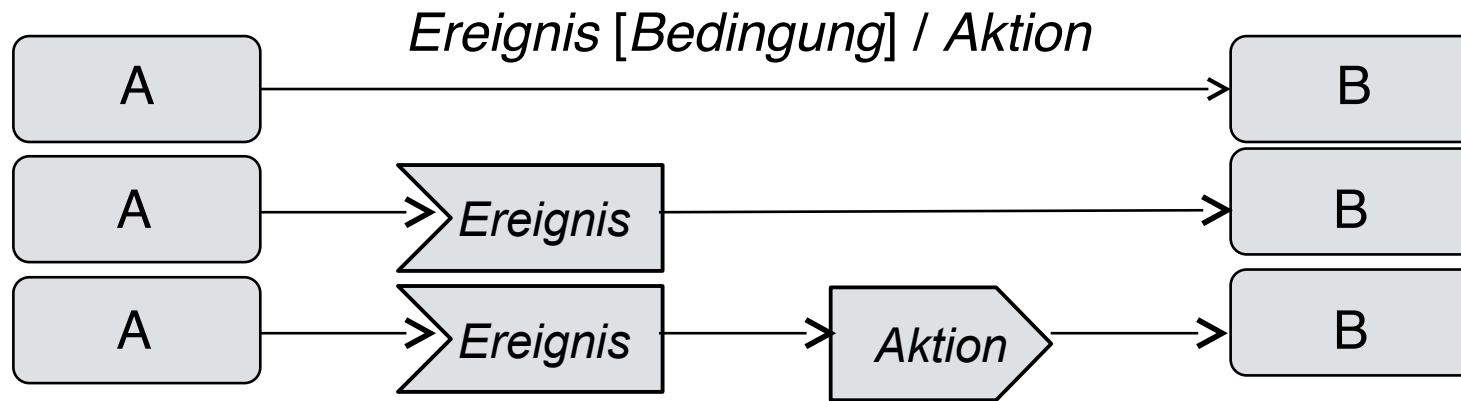


# Spezielle Protokollmaschinen (Akzeptoren)

- ▶ Ein **Zustandsmodell** (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model) ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- ▶ Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die einem Objekt zugeordnet ist und das Verhalten eines Objekts beschreibt
  - Sie beschreibt dann einen vollständigen *Objektlebenszyklus* (white-box object life cycle)
- ▶ Eine **technische Steuerungsmaschine** beschreibt das Verhalten eines technischen Gerätes
  - Aus Steuerungsmaschinen kann die Implementierung der Steuerungssoftware des Objekts bzw. des Geräts abgeleitet werden (wichtig für eingebettete Systeme)
- ▶ Eine reine **Übersetzermaschine** bildet ein Übersetzer-Objekt



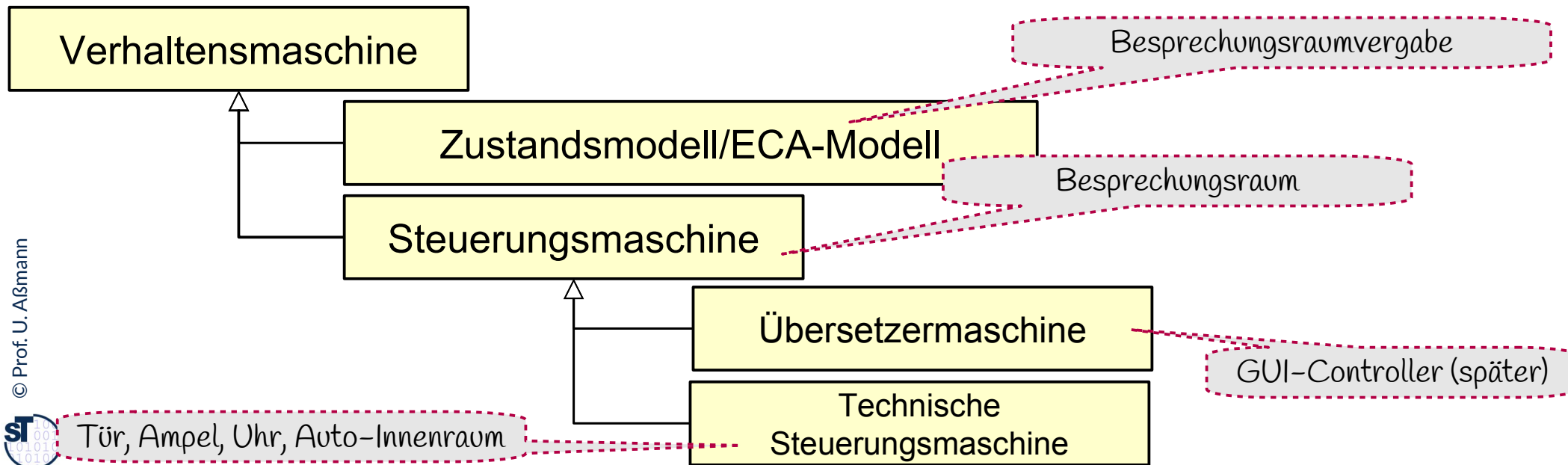
# Aktionen bei Zustandsübergängen in Verhaltensmaschinen (Zustandsmaschinen)



- ▶ **Definition** Eine *Aktion* ist die Beschreibung einer ausführbaren Anweisung. Dauer der Ausführung vernachlässigbar. Nicht unterbrechbar. Eine Aktion kann auch eine Folge von Einzelaktionen sein.
- ▶ In UML heissen Zustandsübergänge mit Aktionen **volle Zustandsübergänge**
- ▶ Typische Arten von Aktionen:
  - Lokale Änderung eines Attributwerts
  - Versenden einer Nachricht an ein anderes Objekt (bzw. eine Klasse)
  - Erzeugen oder Löschen eines Objekts
  - Rückgabe eines Ergebniswertes für eine früher empfangene Nachricht

# Spezielle Verhaltensmaschinen (Transduktoren)

- ▶ Ein **Zustandsmodell** (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model) ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- ▶ Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die einem Objekt zugeordnet ist und das Verhalten eines Objekts beschreibt
  - Sie beschreibt dann einen vollständigen *Objektlebenszyklus* (white-box object life cycle)
- ▶ Eine **technische Steuerungsmaschine** beschreibt das Verhalten eines technischen Gerätes
  - Aus Steuerungsmaschinen kann die Implementierung der Steuerungssoftware des Objekts bzw. des Geräts abgeleitet werden (wichtig für eingebettete Systeme)
- ▶ Eine reine **Übersetzermaschine** bildet ein Übersetzer-Objekt



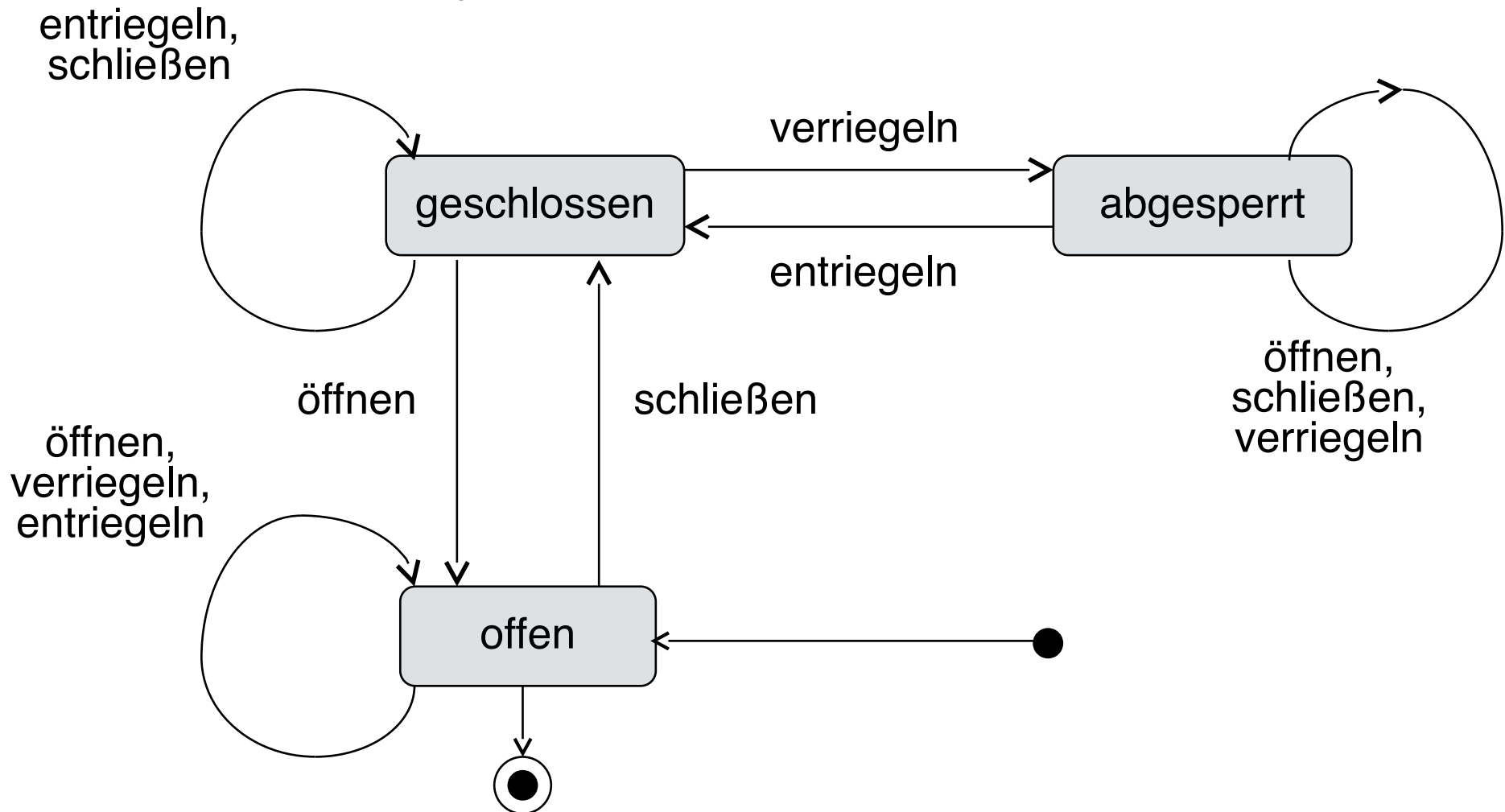
## 34.3 Unterschied von Verhaltens-, Steuer und Protokollmaschinen





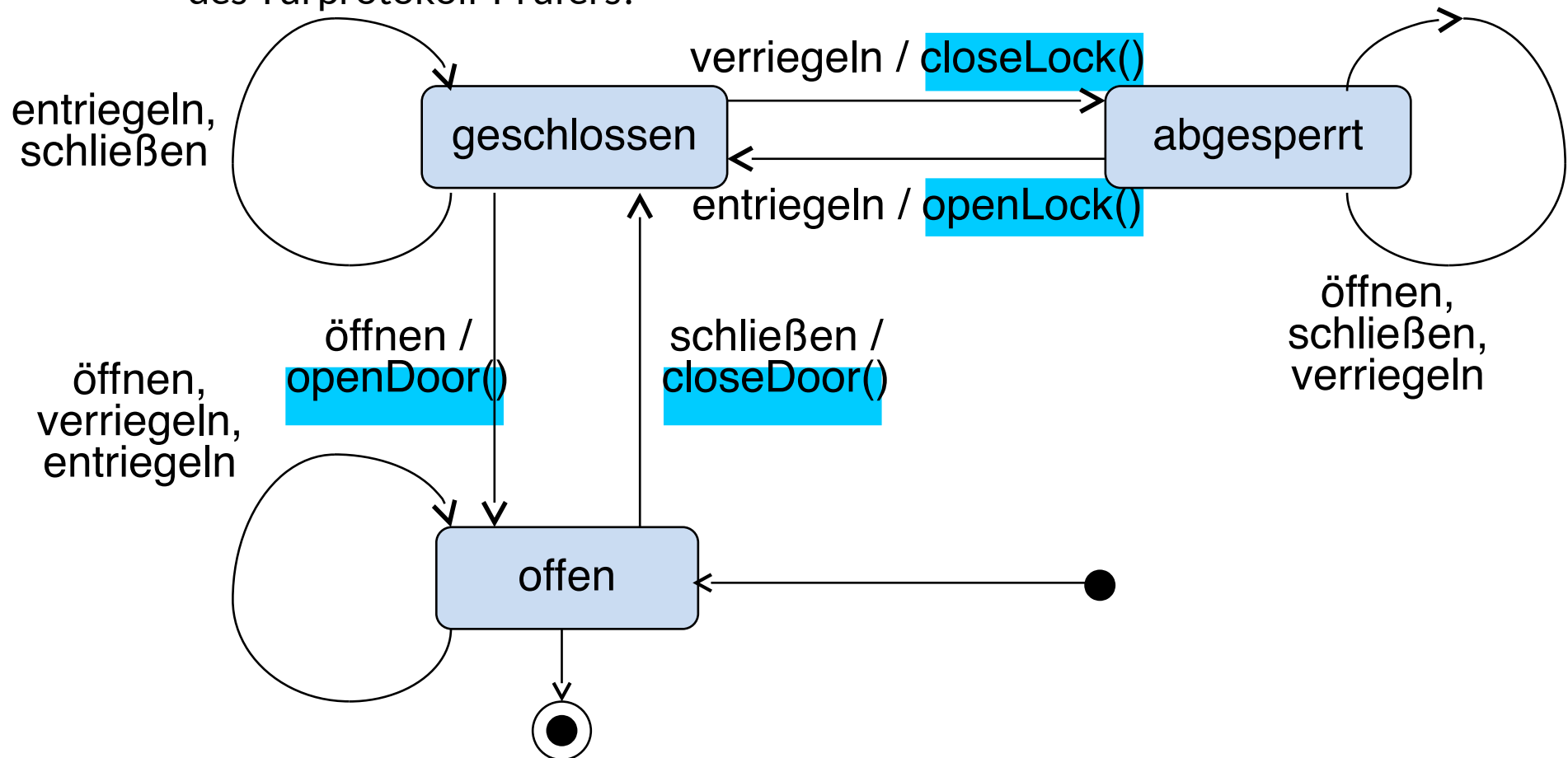
# Beispiel: Protokollmaschine kontrolliert die Benutzung einer Tür

- ▶ Eine **Protokollmaschine kontrolliert**, ob ein Benutzer eine Zustandsmaschine richtig bedient,
  - d.h. ob die Benutzungsreihenfolge einer Zustandsmaschine folgt (akzeptierend, beobachtend, prüfend).



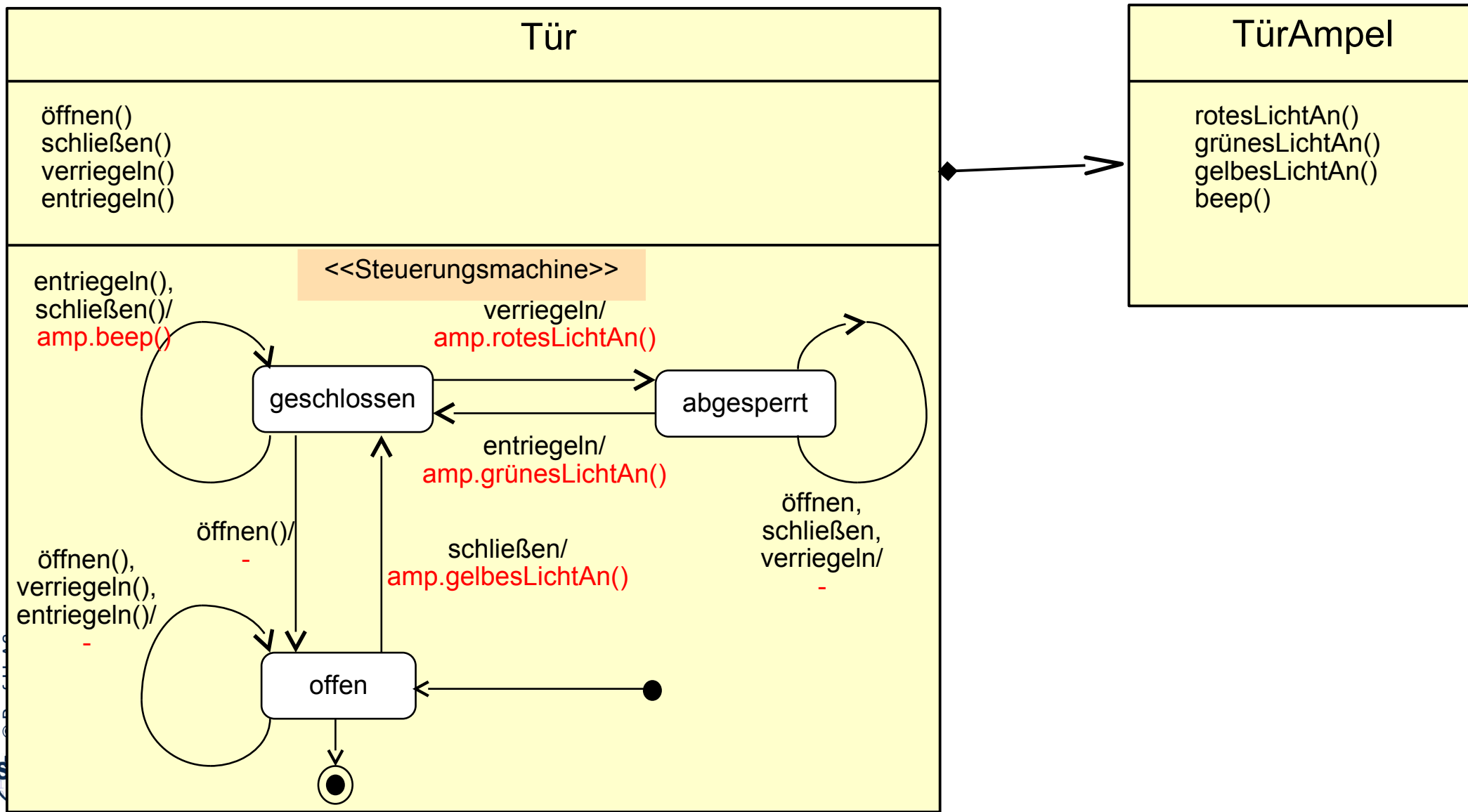
# Beispiel: Steuerungsmaschine für eine Tür einer Behindertentoilette

- ▶ Eine **Steuerungsmaschine** steuert zusätzlich weitere Objekte/Klassen an
- ▶ Hier: die **Türsteuerung** empfängt die Signale des Türbenutzers und **steuert Servo-Motoren an**
  - Achtung: das ist bereits die zweite Steuerungsmaschine zur Protokollmaschine des Türprotokoll-Prüfers!



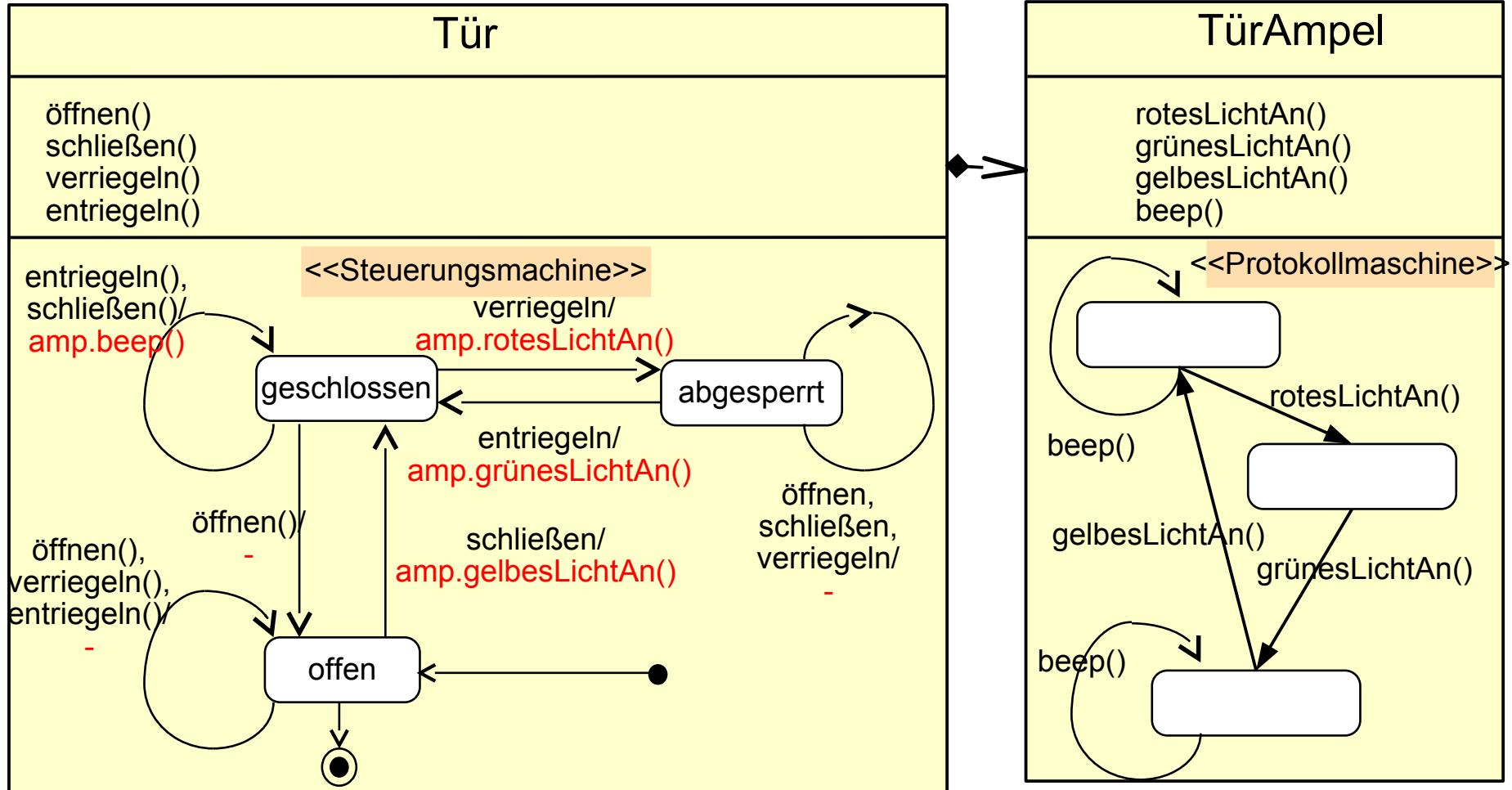
# Übersetzermaschinen: Objektlebenszyklus von innen und aussen

- ▶ Eine **Steuerungsmaschine** kann im Compartment einer Klasse erscheinen; sie beschreibt einen *whitebox*-Objektlebenszyklus (**Übersetzermaschine**)



# Wächtermaschinen: Objektlebenszyklus von nur von aussen

- ▶ Eine Protokollmaschine, die die legale Reihenfolge der Aufrufe an eine Klasse prüft, heißt **Schnittstellenautomat (interface automaton) oder Wächtermaschine des Protokolls** der Klasse.
  - Die Wächtermaschine kann in einem separaten Compartment einer Klasse erscheinen.
  - Sie beschreibt einen *blackbox*-Objektlebenszyklus, d.h. die beobachtbare Sicht von aussen, das *Protokoll* der Schnittstelle oder Klasse



## Verhaltens-(Steuerungs-)maschinen

- ▶ steuern
- ▶ müssen das Wissen über das gesteuerte System *vollständig* repräsentieren, ansonsten gerät das System ausser Kontrolle
- ▶ geben mit ihren Aktionen eine Implementierung der Steuerungssoftware des technischen Systems an
- ▶ können verschiedene Dinge steuern:
  - sich selbst (reine Steuerungsmaschine)
  - andere Klassen
  - ein Subsystem von Klassen
- ▶ **Übersetzermaschinen (transducer objects)** übersetzen das Aufrufprotokoll in ein Steuerprotokoll

## Protokollmaschinen

- ▶ kontrollieren, prüfen
- ▶ können ein *partielles* Wissen über das geprüfte System kontrollieren (der Rest des Verhaltens wird *nicht* abgeprüft)
- ▶ Beschreiben eine Sicht von aussen auf das System
- ▶ Beschreiben das Aufruf- oder Ereignisprotokoll des Systems
- ▶ **Wächtermaschinen (interface automata)** überwachen das Aufrufprotokoll an ein Objekt

## 34.4 Implementierung von Steuerungsmaschinen



# Implementierung von Steuerungsmaschinen mit Implementierungsmuster *IntegerState*

- ▶ **Zustand wird als Integer-Variable repräsentiert**, Bereich [1..n]
  - Alle Ereignisse werden zu “Reaktions”-Methoden, die von aussen aufgerufen werden
  - Externe Ereignisse werden mit “Reaktions-Methoden” modelliert
  - Interne Ereignisse werden den Implementierungen der Methoden zugeordnet
- ▶ **Reaktionsmethoden** schalten den Zustand fort, indem sie Fallanalyse betreiben
  - In jeder Methode wird eine Fallunterscheidung über den Zustand durchgeführt
  - Jeder Fall beschreibt also ein Paar (Ereignis, Zustand)
  - Der Rumpf des Falles beschreibt
    - ◆ den Zustandsübergang (Wechsel des Zustands)
    - ◆ die auszulösende Aktion
- ▶ Einsatz von *IntegerState*:
  - Einsatz in der Phase eines Compilers, der die Zeichen von Datei liest
  - Einsatz bei Protokollprüfern in der Telekommunikation
  - Einsatz bei allen Anwendungen von Zustandsmaschinen, die Performanz benötigen

# IntegerState

## Beispiel: Code zur Steuerung einer Tür (1)

41

Softwaretechnologie (ST)

```
class Tuer {
```

Modifier "final" bei Attributen: unveränderlich

```
    // Konstante zur Zustandskodierung
    private static final int Z_offen = 0;
    private static final int Z_geschlossen = 1;
    private static final int Z_abgesperrt = 2;
```

```
    // Zustandsvariable
    private int zustand = Z_offen;
```

Reaktionsmethode

```
    // Reaktionsmethode oeffnen (Reaktion auf Ereignis)
    public void oeffnen() {
        // Fallanalyse über Zustand
        switch (zustand) {
            case Z_offen:
                break;
            case Z_geschlossen:
                zustand = Z_offen;
                System.out.println("Klack");
                break;
            case Z_abgesperrt:
                break;
        }
    }
}
```



# IntegerState

## Beispiel: Code zur Steuerung einer Tür (2)

```
public void schliessen() {
    // Fallanalyse
    switch (zustand) {
        case Z_offen:
            zustand = Z_geschlossen;
            System.out.println("Klick");
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            break;
    }
}

public void verriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            zustand = Z_abgesperrt;
            System.out.println("Knirsch");
            break;
        case Z_abgesperrt:
            break;
    }
}
```

# IntegerState

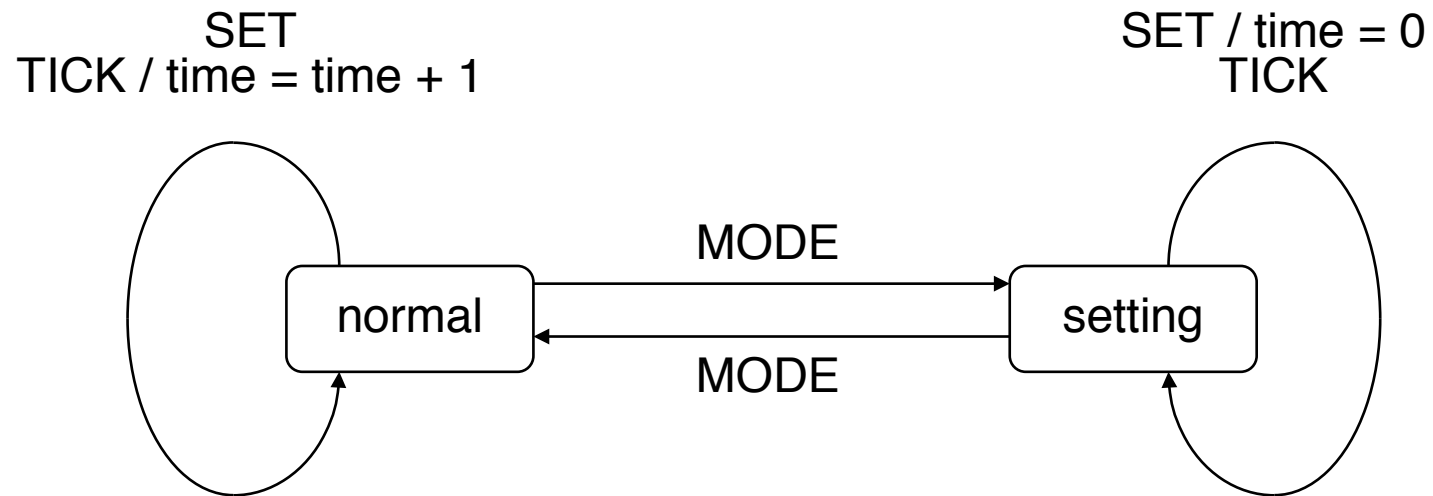
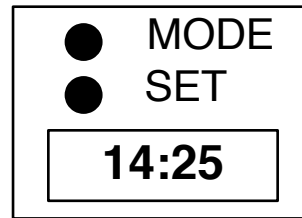
## Beispiel: Code zur Steuerung einer Tür (3)

```
public void entriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            zustand = Z_geschlossen;
            System.out.println("Knirsch");
            break;
    }
}

// Client-Klasse
class TuerBediener {
    public static void main(String[] args) {
        Tuer t1 = new Tuer();
        t1.oeffnen();
        t1.schliessen();
        t1.verriegeln();
        t1.entriegeln();
        t1.oeffnen();
        t1.schliessen();
    }
}
```

# Aufgabe: Steuerungsmaschine realisieren

- ▶ Beispiel: Betriebsmodi einer Taschenuhr (stark vereinfacht)



# Implementierung mit IntegerState

```
class Clock {  
  
    private int time = 0;  
    private static final int NORMAL = 0;  
    private static final int SETTING = 1;  
  
    private int state = NORMAL;  
  
    // Reaktionsmethode  
    public void set () {  
        switch (state) {  
            case NORMAL: {  
                time = time+1;  
                break;  
            };  
            case SETTING: {  
                time = 0;  
                setChanged();  
                break;  
            };  
        };  
    }  
  
    ...// analog tick(), mode()  
}
```

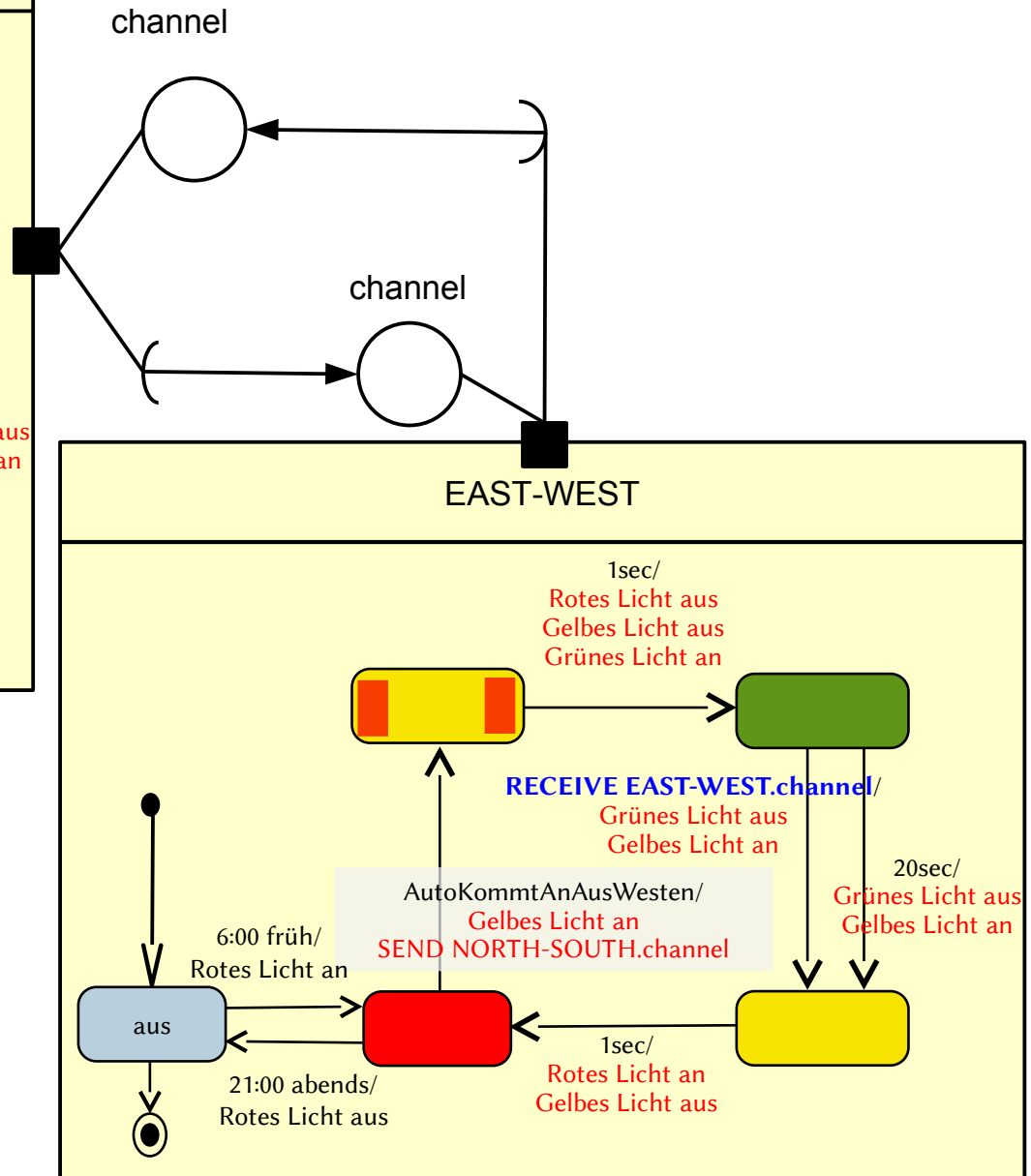
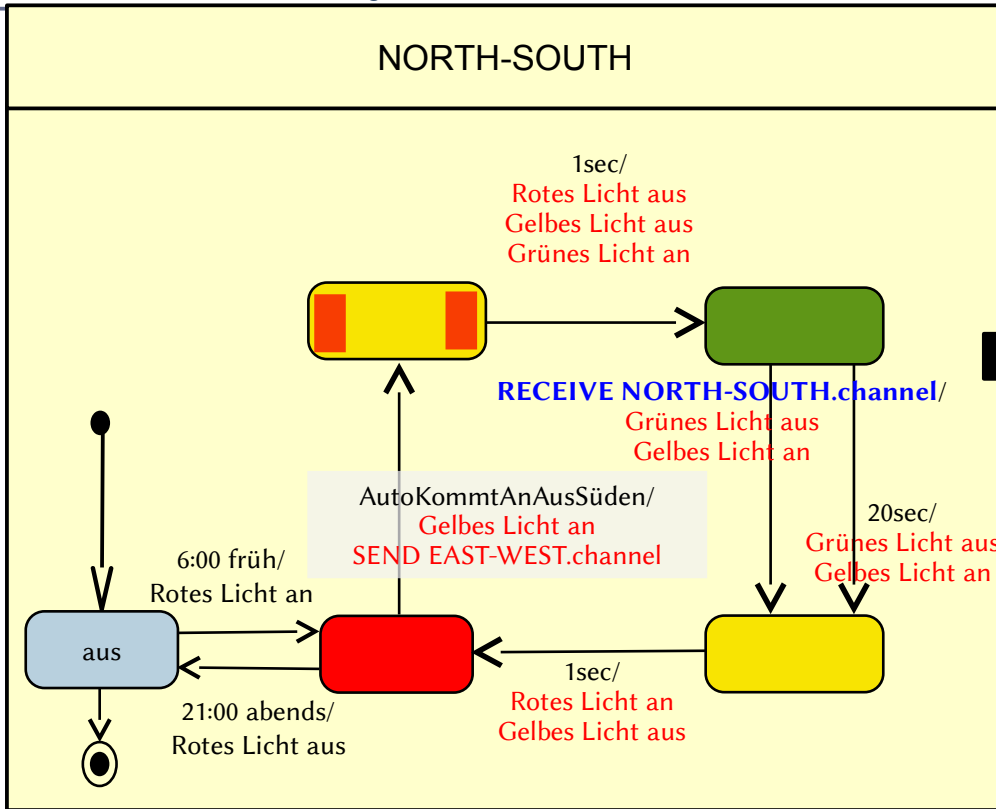
- ▶ Implementieren Sie den Ampelautomaten mit dem Implementierungsmuster IntegerState.
- ▶ Wieviele Reaktionsmethoden brauchen Sie?

## 34.5 Kooperierende Zustandsmaschinen

- ▶ Eine besonders einfache Art von Objektnetz bilden solche, in denen alle Objekte Zustandsmaschinen bilden, die kooperieren
  - sich Nachrichten senden (Ereignisse)
  - auf Ereignisse in Nachbarobjekten mit eigenen Reaktionsmethoden reagieren



# Bsp.: Kopplung zweier Ampeln an einer Kreuzung durch Ereignis-Kanäle



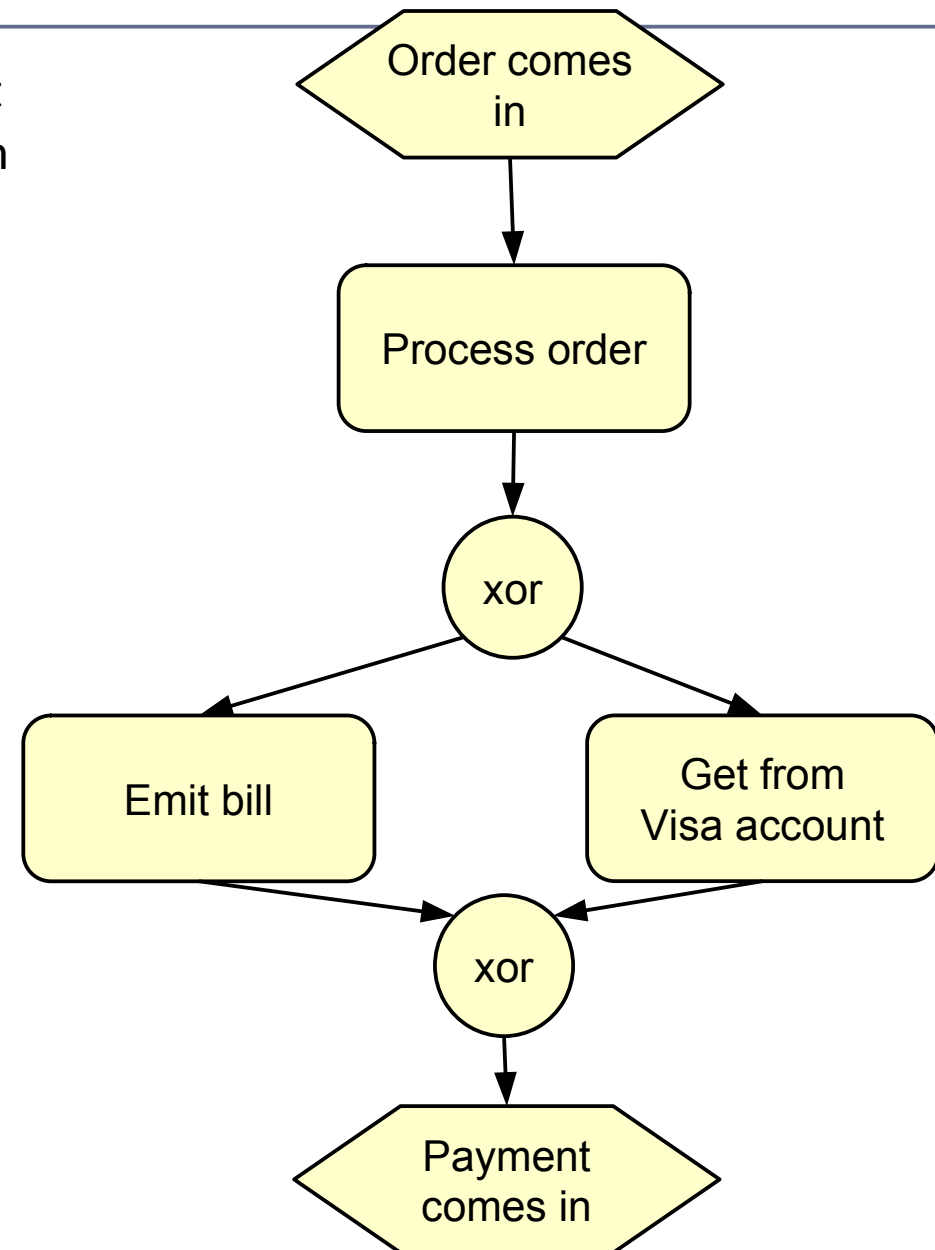
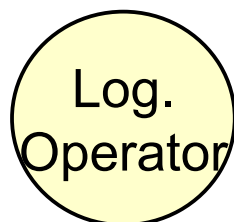
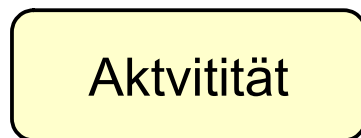
- ▶ Synchronisation der Automaten durch Senden von Token auf Channels
- ▶ “Kommunizierende Automaten”

- ▶ Many slides courtesy to © Prof. Dr. Heinrich Hussmann. Used by permission.
- ▶ Warum betreibt man Verhaltensanalyse mit Aktivitätsdiagrammen und Zustandsdiagrammen?
- ▶ Erklären Sie typische Steuerungsmaschinen:
  - Erweitern Sie das Stellverhalten der Uhr zu einer Funkarmbanduhr
  - Beschreiben Sie den Lebenszyklus von Autotüren und -heckklappen
  - Warum ist ein Geldautomat nur mit endlich vielen Zuständen ausgestattet?
  - Ist ein Bahnkarten-Verkaufsautomat ebenfalls noch ein endlicher Automat?
  - Fahrstühle können sehr viele Zustände haben. [Jazayeri] Wann wäre ein Fahrstuhl ein Automat mit unendlich vielen Zuständen?
- ▶ Erklären Sie typische Zustandsmaschinen für Abläufe, und beschreiben Sie sie jeweils alternativ mit einem Aktionsdiagramm. Was ist jeweils einfacher?
  - Hausbau
  - Projekte
  - Immatrikulation eines Studenten



# 34.A.1 Andere Notationen für Aktivitätsdiagramme

- ▶ Ähnlich, wird in der industriellen Praxis oft benutzt: Ereignisgesteuerte Prozessketten (EPK), Sprache des ARIS-Toolkits für Prozessmodellierung von SAP-Systemen

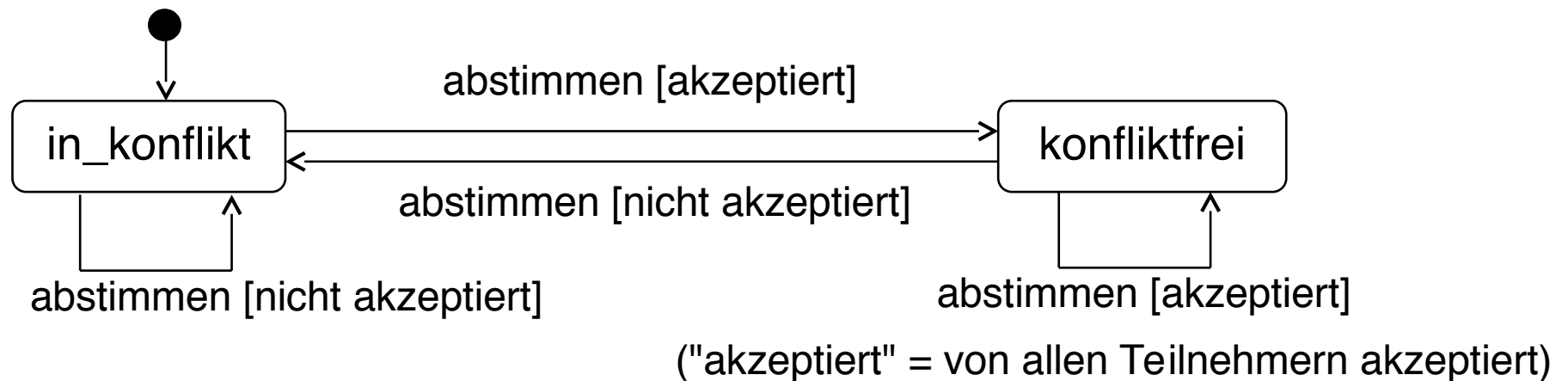


## 34.A.2 Implementierung von Protokollmaschinen



# Beispiel: Protokollmaschine

- ▶ Folgende Protokollmaschine definiert die zulässigen Aufrufreihenfolgen der Klasse Terminverschiebung:



- Begriff "Protokoll":
  - Kommunikationstechnologie
  - Regelwerk für Nachrichtenaustausch
- Protokollmaschinen in der Softwarespezifikation:
  - **zusätzliche** abstrakte Sicht auf komplexen Code (partielles Wissen)
  - Vertragsprüfer zur Einhaltung von Aufrufreihenfolgen

# Implementierungsmuster Protokollmaschine

## Explicit Tracing State

```
public Teambesprechung
    (String titel, Hour beginn, int dauer,
     Teammitglied[] teilnehmer) {
    int zustand = Z_nicht_abgestimmt;
    super(titel, beginn, dauer);
    this.teilnahme = teilnehmer;
    if (! abstimmen(beginn, dauer)) {
        System.out.println("Termin bitte verschieben!");
        zustand = Z_in_konflikt;
    }
    else {
        for (int i=0; i<teilnahme.length; i++)
            teilnahme[i].teilnahmeSetzen(this);
        zustand = Z_konfliktfrei;
    }
}
```

Explizites  
Zustandsattribut

- Analog zu IntegerState, aber keine Aktionen
- Ablauflogik kann den Zustandswert benutzen (muß aber nicht!)

# Implementierungsmuster Protokollmaschine

## Implicit Tracing State

- ▶ Information über Zustand jederzeit berechenbar - hier aus den Werten der Assoziationen und den Datumsangaben
- ▶ Zustandsinformation gibt zusätzliches Modell, nicht direkt im Code wiederzufinden

```
public Teambesprechung
    (String titel, Hour beginn, int dauer,
     Teammitglied[] teilnehmer) {
    super(titel, beginn, dauer);
    this.teilnahme = teilnehmer;
    if (! abstimmen(beginn, dauer)) {
        System.out.println("Termin bitte verschieben!");
    }
    else {
        for (int i=0; i<teilnahme.length; i++)
            teilnahme[i].teilnahmeSetzen(this);
    }
}
```

Zustandsw  
echsel

Zustand  
unklar

Zustand  
in\_konflikt

Zustand  
konfliktfrei

- ▶ Anwendungsgebiet: Prüfen von Aufrufreihenfolgen
- ▶ Codegenerierung von Implementierungen aus Zustandsmodell:
  - Implementierungsmuster ImplicitTracingState, ExplicitTracingState, State (aber ohne Aktionen)
  - Nur zur Ableitung von Prüfcode! Zustandsmodell liefert Information für Teilaspekte des Codes (zulässige Reihenfolgen), keine vollständige Implementierung
- ▶ Praktische Aspekte:
  - In der Analyse zur Darstellung von Geschäftsprozessen und -regeln
  - komplexen Lebenszyklen für Geschäftsobjekte (Modellierung mit Sichten, die jeweils durch eine Protokollmaschine beschrieben werden)
  - Nützlich für den Darstellung von Klassen mit komplexen Regeln für die Aufrufreihenfolge
  - Hilfreich zur Ableitung von Status-Informationen für Benutzungs-Schnittstellen
  - Hilfreich zum Definieren sinnvoller Testfälle für Klassen

## 34. Verhaltensanalyse für die Lebenszyklen von Objekten: Aktionsdiagramme (Aktivitäten-, Zustandsdiagramme) - wie modelliere ich das Leben von Objekten und andere Abläufe in Prozessen?

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 20-0.2, 15.06.20

- 1) Aktivitätendiagramme in UML
  - 2) Zustandsdiagramme in UML
  - 3) Verhaltens-, Steuerungs-, und  
Protokollmaschinen
  - 4) Implementierung von  
Steuerungsmaschinen
  - 5) Einsatz im Test
- A1) Andere Notationen  
A2) Impl. von Protokollmaschinen



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

- ▶ Zuser 7.5.3
- ▶ Störrle Kap. 10 (Zustandsdiagramme), Kap. 11 (Aktivitätsdiagramme)
- ▶ ST für Einsteiger: Kap. 10

**Grameen has given me an unshakeable faith in the creativity of human beings.** This has led me to believe that human beings are not born to suffer the misery of hunger and poverty.

**To me poor people are like bonsai trees.**

When you plant the best seed of the tallest tree in a flower-pot, you get a replica of the tallest tree, only inches tall. There is nothing wrong with the seed you planted, only the soil-base that is too inadequate. Poor people are bonsai people. There is nothing wrong in their seeds. Simply, society never gave them the base to grow on. All it needs to get the poor people out of poverty for us to create an enabling environment for them. Once the poor can unleash their energy and creativity, poverty will disappear very quickly.

**Let us join hands to give every human being a fair chance to unleash their energy and creativity.**

M. Yunus, Peace Nobel price lecture, Oslo, Dec 10, 2006





- ▶ Modellierung des Tamagotchi-Spiels mit StateMate, einer professionellen Statechart-Umgebung
  - <http://www4.in.tum.de/~philippj/tamasemi/slides/statemate.pdf>
- ▶ Wie man professionell Zustandsmaschinen aus einer domänenspezifischen Sprache auf C übersetzt, zeigt der RAGEL compiler
  - <http://www.colm.net/open-source/ragel/>

Notation: Wir schreiben:

- ▶ Analysefragen in Font LeckerliOne: *Which tools does a text contain?*
- ▶ Zu analysierende Texte des Kunden in Font ComicJens: *Eine Teambesprechung ist ein Termin.*



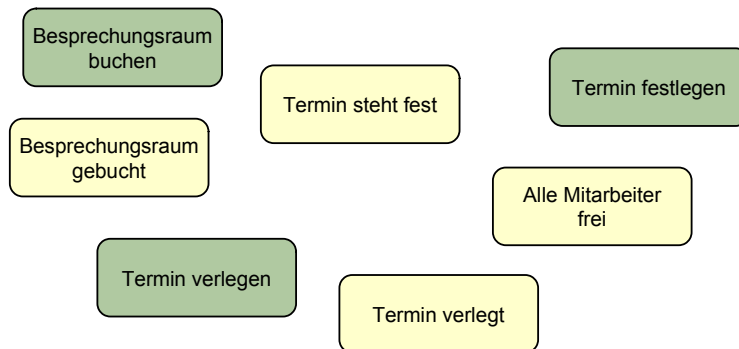
# Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
  1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML
  1. Strukturelle metamodellgetriebene Modellierung für das Domänenmodell
  2. Strukturelle Modellierung von komplexen Objekten
  3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
  1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
  2. Funktionale querschnittende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
  3. (Funktionale querschnittende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent



## Problem: Wie analysiert man einen Text vom Kunden, der das Verhalten eines Objekts beschreibt?

- ▶ "Eine Teambesprechung benötigt einen Besprechungsraum. Bevor der gebucht werden kann, muss der Termin feststehen; das kann aber nur festgelegt werden, wenn alle Mitarbeiter, die an dem Termin teilnehmen sollen, frei sind. Andernfalls muss der Termin verlegt werden."



**Dynamische Analyse** fragt nach den zeitlichen Aussagen des Kunden über Reihenfolgen, zeitliche Abfolgen, Ordnungen.

- ▶ Ergebnis:
  - Verhaltensmodelle
  - Steuerungsmaschinen (Lebenszyklen von komplexen Objekten)
  - Technische Steuerungsmaschinen (Lebenszyklen von eingebetteten Systemen)

*Was steckt genau in einem Objekt?  
Welche Phasen durchläuft ein Objekt oder ein System?  
In welche Zustände befindet sich das Objekt?*

Ein Beispiel:

Schreibt man an einem Dokument, bedeutet punktweise Verfeinerung, dass man an einer Stelle neuen Text hinzufügt.

Querschneidende Verfeinerung dagegen bedeutet, dass man an vielen Stellen eine ähnliche Ersetzung oder Erweiterung vornimmt, z.B. die Ersetzung eines Namens oder einer Definition.

## Punktweise und querschneidende dynamische Verfeinerung in der OOA

7 Softwaretechnologie (ST)

**Punktweise funktionale Verfeinerung** ist eine funktionale Verfeinerung eines Modellfragmentes (meist Objekt oder Methode), die *punktweise* geschieht, d.h. pro Modellfragment separat durchgeführt wird.

Kap. 34

- ▶ Ergebnis der Verfeinerung einer Klasse im Strukturmodell:

- **Lebenszyklus** des komplexen Objektes
- **Implementierung** einer Methode

*Welchen Lebenszyklus durchläuft ein Objekt?  
Welche Zustände oder Aktivitäten hat eine Methode?*

**Querschneidende funktionale Verfeinerung** ist eine funktionale Verfeinerung *mehrerer* Modellfragmente gleichzeitig, die *querschneidend* geschieht.

Kap. 35

- ▶ Damit kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen, eine *Szenarienanalyse*, die quasi die Draufsicht auf ein Szenario ermittelt
  - Siehe Kapitel "Szenarienanalyse"

*Welches Interaktionsprotokoll durchläuft eine Gruppe von Objekten?*



### Ein Beispiel:

Schreibt man an einem Dokument, bedeutet punktweise Verfeinerung, dass man an einer Stelle neuen Text hinzufügt.

Querschneidende Verfeinerung dagegen bedeutet, dass man an vielen Stellen eine ähnliche Ersetzung oder Erweiterung vornimmt, z.B. die Ersetzung eines Namens oder einer Definition.

## 34.1. Modellierung von Prozessen mit Aktionsdiagrammen

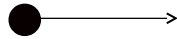
Aktionsdiagramme: Aktivitätsdiagramme (AD),  
Statecharts (SC)



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

## Start- und Endzustand

- ▶ Jedes Aktionsdiagramm (Statechart, Aktivitätendiagramm) sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".
- ▶ **Notation:**



- ▶ Ein Aktionsdiagramm kann einen oder mehrere Endzustände haben.
- ▶ **Notation:** ("bull's eye")



## 34.1.1. Aktivitätsdiagramme

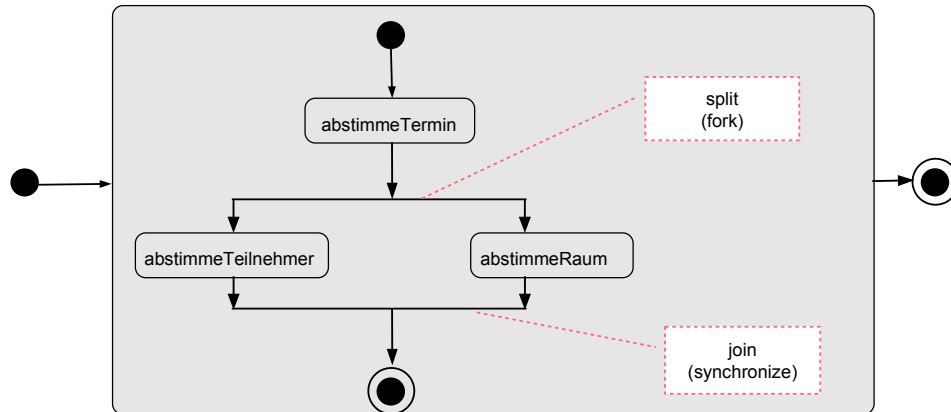
- ▶ zur Spezifikation von Aktivitäten und ihren Abhängigkeiten





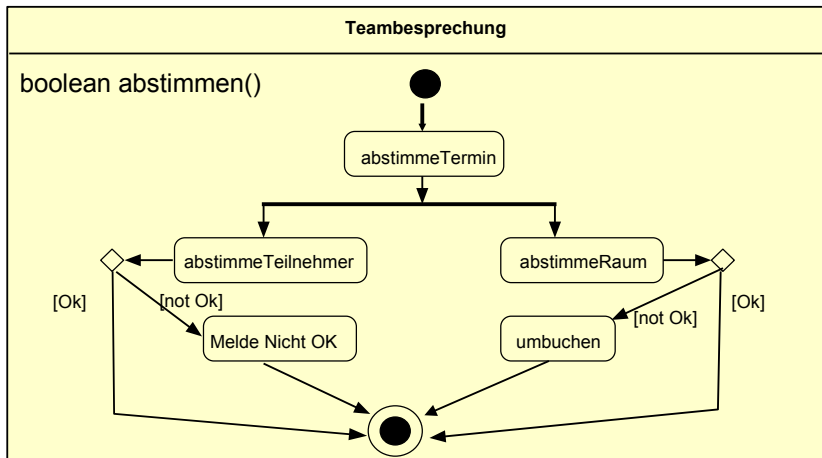
# Dynamische Modellierung (Verhaltensmodellierung)

- ▶ Eine Signatur eines Objektes oder einer Methode muss **funktional verfeinert** werden
  - Das Verhalten (dynamische Semantik) muss spezifiziert werden (partiell oder vollständig)
  - Daher spricht man von *Verhaltensmodellierung* oder *dynamischer Modellierung*
  - und von *punktweiser Verfeinerung* einer Klassen- oder Methoden-Signatur
- ▶ Einfachste Form: Angabe von Aktivitäten, verknüpft mit Steuer- und Datenfluss
  - Geschachtelt in eine Oberaktivität



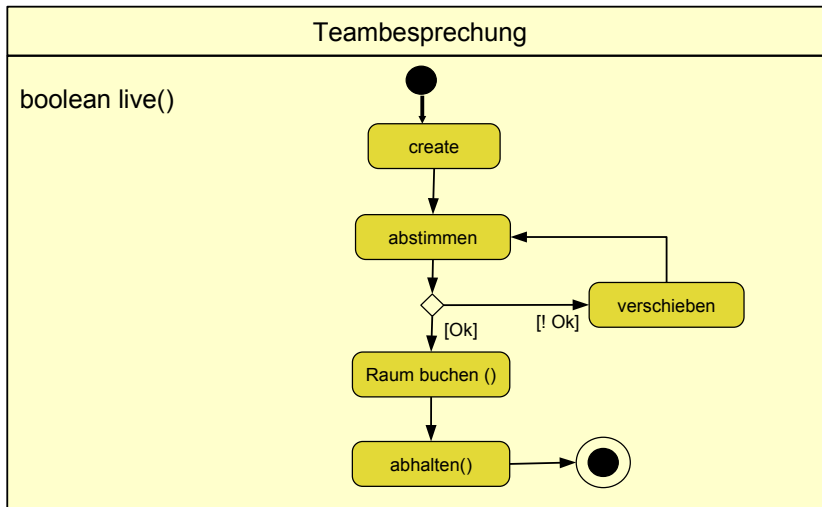
# Aktivitätsdiagramm als Verhalten einer Methode (activity diagram)

- ▶ Aktivitätsdiagramme können das Verhalten einer Methode beschreiben, dann werden sie in ein Abteil der Klasse notiert
- ▶ Aktivitäten, verbunden durch Datenfluß (Datenflußdiagramm, data-flow diagram)
  - Parallele Aktivitäten in parallelen Zweigen
  - Bedingungen (guards) bestimmen, ob über eine Kante Daten fließen (*bedingter Datenfluß*)



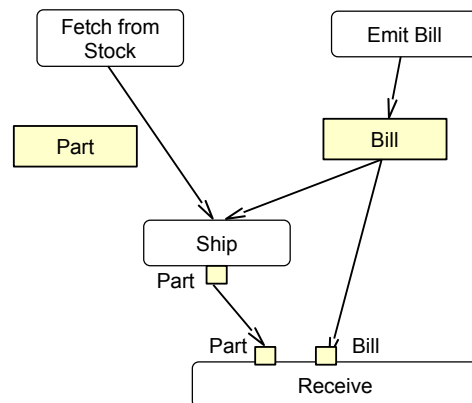
# Aktivitätsdiagramm für Lebenszyklus eines Objekts

- ▶ Viele Objekte müssen in einer bestimmten Art und Weise aufgerufen werden, von ihrer Geburt bis zum Tod
- ▶ AD beschreiben den *Arbeitsfluss (Workflow)* der Methoden (hier: abstimmen() wird aufgerufen)



## Verschiedene Notationen für Datenfluß

- ▶ Objekte, die zwischen Aktivitäten fließen, können verschieden notiert werden
- ▶ Pins sind benannte Parameter der Aktivitäten
  - Unterscheide Pins (von Aktivitäten) von Ports (von Komponenten)!



UML nutzt oft die gleiche grafische Syntax für verschiedene Konzepte:

- Pins entsprechen Parameter bzw. Streams und Senken.
- Ports entsprechen Port-Schnittstellen mit Methoden, Streams und Senken.

## 34.2 UML-Zustandsdiagramme (Zustandsmaschinen, Statecharts)

Zustandsmaschinen spezifizieren *Zustände* und ihr Aufeinanderfolgen

Zustandsmaschinen gehören zu jUML, wenn sie verlustfrei in Code überführt und zurücküberführt werden können (round-trip engineering)



DRESDEN  
CONCEPT  
Exzellenz aus  
Wissenschaft  
und Kultur

- ▶ Objekt-Verhalten und Szenarien können auch *zustandsbetont* (*phasenorientiert, modi-orientiert*) analysiert werden
- ▶ Beispiel: Spezifikationen von Zustandsautomaten für Fly-by-wire und Drive-by-wire
- ▶ Methodik: Analyse und Entwurf mit UML-Statecharts

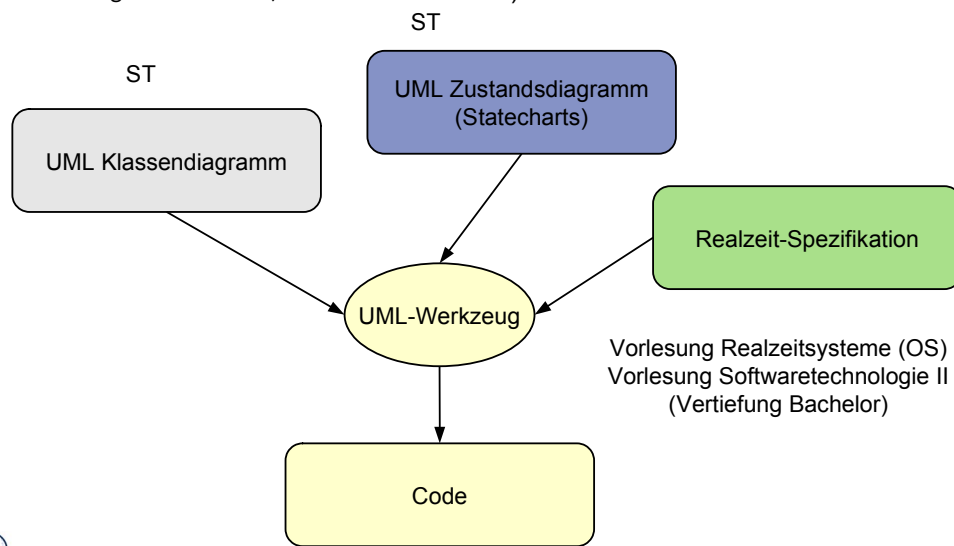


© Prof. U. Altmann

[http://en.wikipedia.org/wiki/File:Hands-free\\_Driving.jpg](http://en.wikipedia.org/wiki/File:Hands-free_Driving.jpg)

- Objekt-Verhalten und Szenarien können auch *zustandsbetont* analysiert werden
  - Man frage: *Wie ändern sich die Zustände des Systems, wenn bestimmte Ereignisse auftreten?*
  - Es entsteht ein ECA-Architekturstil (event-condition-action)
- Besonders wichtig bei:
  - Sicherheitskritischen Systemen: *Kann dieser illegale Zustand vermieden werden?*
  - Benutzerschnittstellen: *Ist diese Aktion in diesem Zustand des GUI erlaubt?*
  - Komponentenorientierten Systemen: *Darf diese Komponente mit dieser anderen kommunizieren? (Protokollprüfung)*
- Methodik: Analyse und Entwurf mit UML-Statecharts

- Die Paderborner Railcabs arbeiten mit UML-Statecharts, die mit Realzeit-attributen angereichert sind (real-time statecharts):



Die Paderborner autonomen Zugsysteme wurden auf einer Teststrecke in der Uni Paderborn getestet. Sie können autonom fahren, Waggons (Shuttles) koppeln, Züge teilen. Sie bilden den Prototyp der Bahn der Zukunft.

Wann werden sie in Deutschland zugelassen?

- ▶ **Definition:** Ein **Zustand (Phase, Modus)** ist eine Eigenschaft eines Objektes oder Systems, die über einen begrenzten Zeitraum besteht.

- ▶ **Notation:**



- ▶ Ein (technisches) **System** besteht aus einem (komplexen) Objekt oder einem Netz von Objekten, das Aufgaben ausführt
  - eines komplexes Softwaresystem
  - ein Arbeitsprozess
  - ein Produkt eines Arbeitsprozesses
  - ein einzelnes Objekt
- *Vorsicht! Zustände werden wie Aktivitäten mit ovalen Boxen notiert, gehören aber zu einem Zustandsdiagramm!*

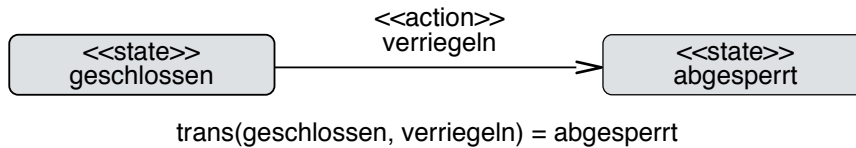


### 34.2.1 Endliche Automaten ohne Aktionen (Akzeptoren)

- ▶ Theoretische Informatik, Automatentheorie:

Ein **endlicher Zustandsautomat (Akzeptor)** über einem Eingabealphabet  $A$  ist ein Tupel, bestehend aus:

- einer Menge  $S$  von Zuständen (Phasen, Modi)
  - einer (partiellen) Übergangsfunktion  $\text{trans} : S \times A \rightarrow S$
- einem Startzustand  $s_0 \in S$
- einer Menge von Endzuständen  $S_f \subseteq S$



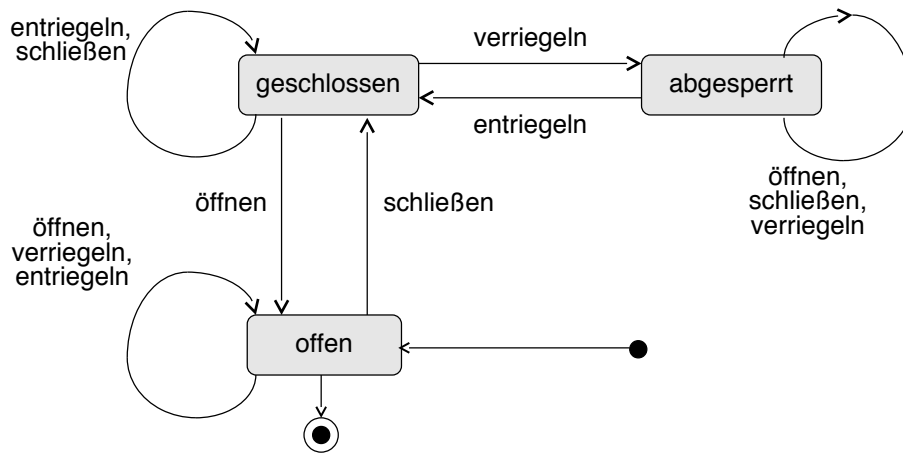
**Achtung! Notation von Zuständen ähnlich zur Notation von Aktivitäten!**



Vielen Objekte verhalten sich als Akzeptoren, d.h. einfache Protokollautomaten. Sie haben einen “Schnittstellenautomaten” (interface automaton), der die legale Reihenfolge der Aufrufe festhält.

## Beispiel: Zustandsmodell einer Tür

- ▶ Der Tür-Akzeptor stellt einen Prüfer für mögliche Aktionsfolgen für eine Tür dar
- ▶ In UML heisst der Akzeptor **Protokoll(zustands)maschine**, denn er akzeptiert eine Folge von Ereignissen (Protokoll)
- ▶ Der Zustandsautomat bildet die Phasen im Leben einer Tür ab



**Aufgabe:** Stellen Sie 5 legale Aktionsfolgen des Tür-Akzeptors dar.

**Hinweis:** Bilden Sie Pfade durch den Graphen des Automaten: Rollen Sie den Automaten "ab".

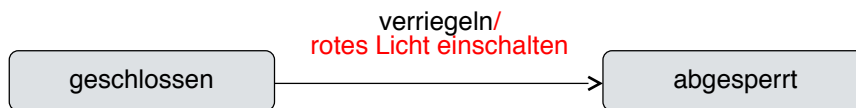
Tür Ausgangs-/ Endzustand	geschlossen	offen	abgesperrt
geschlossen	entriegeln, schließen	öffnen	verriegeln
offen	schließen	öffnen, verriegeln, entriegeln	-
abgesperrt	entriegeln	-	öffnen, schließen, verriegeln

- ▶ Tabellen bilden eine alternative Notation

- Ein gutes Werkzeug kann Zustandsautomaten in mehreren Visualisierungen darstellen:
- als Automat-Diagramm
  - als Zustandsübergangstabelle
  - als Entscheidungsbaum (siehe ST-2)

### 34.2.2. Endliche Automaten mit Aktionen (Transduktoren)

- ▶ Ein **endlicher Zustandsübersetzer (Transduktor, endlicher Übersetzer)** übersetzt eine Folge von Zeichen aus einem Eingabealphabet  $A$  in eine Folge von Zeichen aus **einem Ausgabealphabet  $B$** . Er ist ein Tupel, bestehend aus:
  - einer Menge  $S$  von Zuständen
  - einer (partiellen) Übergangsfunktion  $\text{trans} : S \times A \rightarrow S$ 
    - einem Startzustand  $s_0 \in S$
    - einer Menge von Endzuständen  $S_f \subseteq S$



$\text{trans}(\text{geschlossen}, \text{verriegeln}) = (\text{abgesperrt}) / \text{rotes Licht einschalten}$



Transduktoren transformieren einen Ereignisstrom der Sprache  $A$  in einen Ereignisstrom (Übersetzer einer Sprache mit endlichem Alphabet)

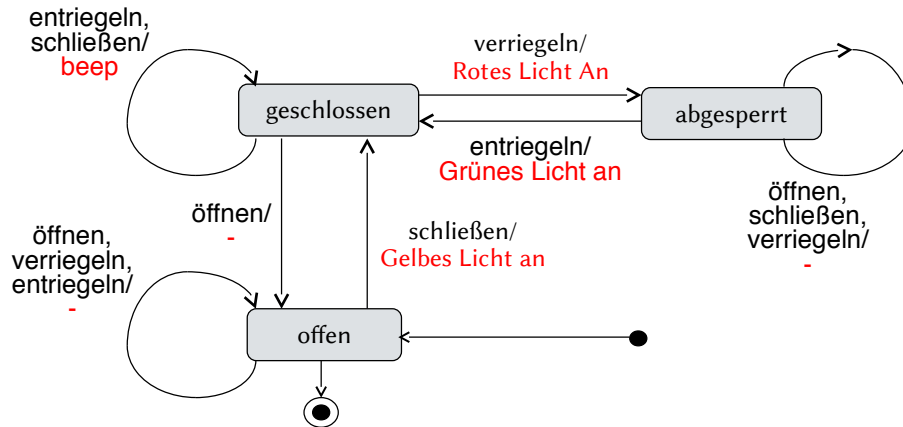
Transduktoren abstrahieren Ereignisse im Eingabebereich in Ereignisse im Ausgabebereich.

Viele Programme sind Transduktoren, d.h. einfache Übersetzer von einer regulären Sprache  $A$  in eine zweite reguläre Sprache  $B$ .

Vielen Objekte verhalten sich als Transduktoren, d.h. einfache Übersetzer.

## Beispiel: Zustandsmaschine einer Tür

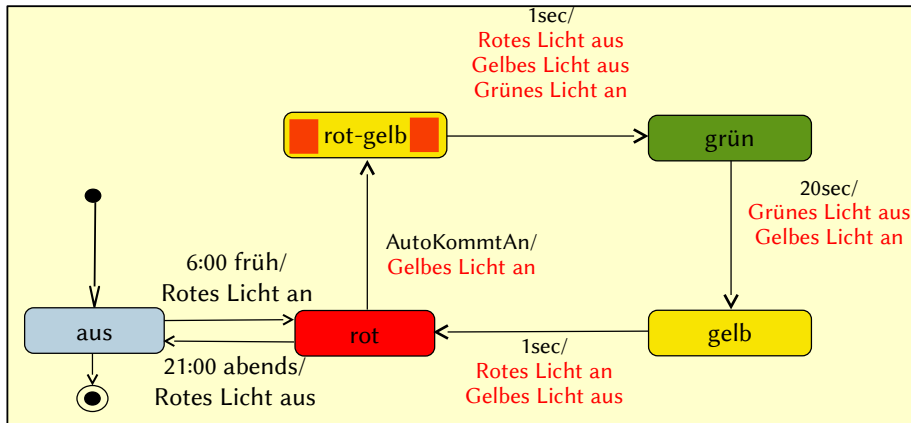
- ▶ Der Tür-Transduktor stellt zusätzlich zum Prüfer einen Steuerer (controller) für eine Tür-Zustandsmeldeampel dar
  - aus ihm kann ein Steuerungsalgorithmus für die Türampel abgeleitet werden
- ▶ In UML: **Zustandsmaschine (Verhaltensmaschine)**



## Beispiel: Zustandsmaschine einer bedarfsgesteuerten Ampel

25 Softwaretechnologie (ST)

- ▶ Welches Ereignis löst den Ampelzyklus aus?
- ▶ Welches Eingabealphabet hat der Transduktor (Ereignisse)? Welches Ausgabealphabet?
- ▶ Welche Sprachen (Mengen von Folgen von Ereignissen) übersetzt der Transduktor in einander?



Übung: Schreiben Sie drei Pfade durch den Automaten auf. Unterscheiden Sie sorgfältig eingehende Ereignisse von ausgelösten Aktionen.

- Wie lange kann ein Pfad werden?
- Wie viele Pfade gibt es insgesamt?
- Wie viele Schaltfolgen hat eine Verkehrsampel?

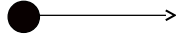
- ▶ Wir verstehen unter "Semantik eines Modells" seine Interpretation bzw. seinen Ablauf bzgl. eines Interpretierers der Modellierungssprache.
- ▶ Ein Zustandsmodell ist endlich, definiert aber einen unendlichen Zustandsfolgenraum:
  - seine Semantik ist eine unendliche Menge von Pfaden über Zuständen und Aktionen (Aktionsfolgen):
  - Rollt man die Pfade des Zustandsmodells ab, entsteht ein unendlich tiefer Zustandsfolgen-Baum
- ▶ Die Semantik eines Zustandsmodells ist definiert als **Menge von Sequenzen (Aktionsfolgen)**:
  - in der Theoretischen Informatik:
    - Menge von "akzeptierten Wörtern" (Sprache über Grundalphabet von Ereignissen)
  - in der Softwaretechnik wird das interpretiert als:
    - Menge von zulässigen *Ereignisfolgen* (*Ereignissprache*)
    - Menge von zulässigen *Aufruffolgen* oder *Aktionen* (*Aufrufsprache*)
    - Menge von zulässigen *Pfaden in einem Graphen* (*Pfadsprache*)
- ▶ Wichtige Arten von Automaten:
  - *Akzeptor (Moore-Automat)*: "Automaten mit Ausgabe": Ausgabe bei Erreichen eines Zustands → Protokollmaschine
  - *Transduktor (Mealy-Automat)*: Ausgabe bei Übergang → Zustandsmaschine
    - ♦ Softwaretechnik: *Aktion* bei Übergang



Übung 2: Rollen Sie den Ampelautomaten ab in einen Zustandsübergangsbaum. Wann wiederholen sich die Ereignisfolgen?

## Start- und Endzustand (wie bei AD)

- ▶ Jedes Zustandsdiagramm sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".
- ▶ **Notation:**



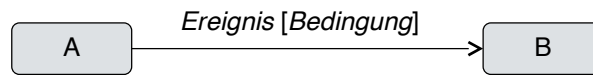
- ▶ Ein Zustandsdiagramm kann einen oder mehrere Endzustände haben.
- ▶ **Notation:** ("bull's eye")



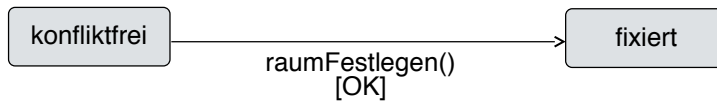
Start- und Endzustände werden in UML für mehrere Diagrammartent benutzt.



## Bedingte Zustandsübergänge in Protokollmaschinen



- ▶ **Definition** Eine **Bedingung** (*guard*) ist eine Boolesche Bedingung, die zusätzlich bei Auftreten des Ereignisses erfüllt sein muß, damit der beschriebene Übergang eintritt.
- ▶ **Notation:** Eine Bedingung kann folgende Informationen verwenden:
  - Parameterwerte des Ereignisses
  - Attributwerte und Assoziationsinstanzen (Links) der Objekte
  - ggf. Navigation über Links zu anderen Objekten
- ▶ **Beispiel:**

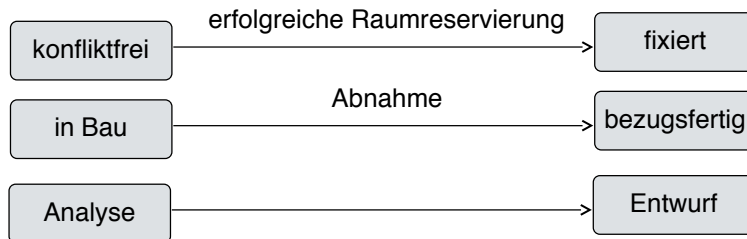


## Zustandsübergänge nach Ereignissen in *Protokollmaschinen*

- ▶ **Definition:** Ein **Zustandsübergang** von Zustand *A* nach Zustand *B* mit Ereignisnamen *E* besagt, daß im Zustand *A* bei Auftreten eines *E*-Ereignisses der neue Zustand *B* angenommen wird.
- ▶ In UML: **Protokollzustandsübergang (protocol transition)**, da Teil eines Akzeptors
- ▶ **Notation:**



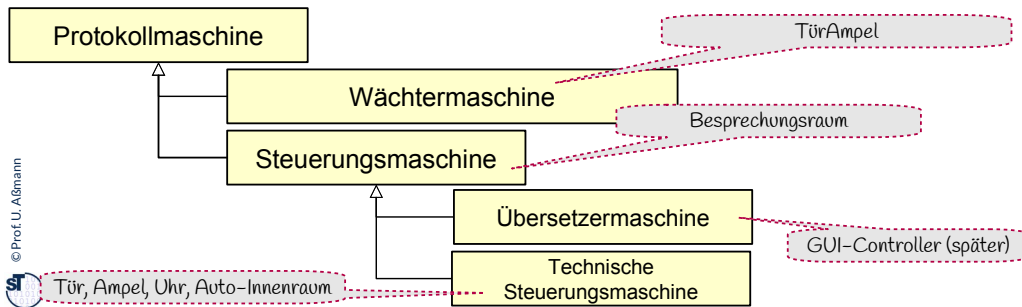
- ▶ **Beispiele:**



## Spezielle Protokollmaschinen (Akzeptoren)

30 Softwaretechnologie (ST)

- ▶ Ein **Zustandsmodell** (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model) ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- ▶ Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die einem Objekt zugeordnet ist und das Verhalten eines Objekts beschreibt
  - Sie beschreibt dann einen vollständigen *Objektlebenszyklus* (white-box object life cycle)
- ▶ Eine **technische Steuerungsmaschine** beschreibt das Verhalten eines technischen Gerätes
  - Aus Steuerungsmaschinen kann die Implementierung der Steuerungssoftware des Objekts bzw. des Geräts abgeleitet werden (wichtig für eingebettete Systeme)
- ▶ Eine reine **Übersetzermaschine** bildet ein Übersetzer-Objekt

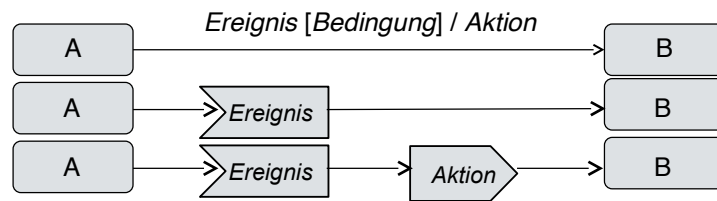


Fast alle technischen Geräte können mit Steuerungsmaschinen beschrieben werden. Daher sind Steuerungsmaschinen für die Modellierung von Software für eingebettete oder cyber-physische Systeme sehr wichtig.

Fragen:

- Kann man ein Auto mit einer Steuerungsmaschine beschreiben?
- Kann man ein autonomes Auto mit einer Steuerungsmaschine beschreiben?

## Aktionen bei Zustandsübergängen in Verhaltensmaschinen (Zustandsmaschinen)



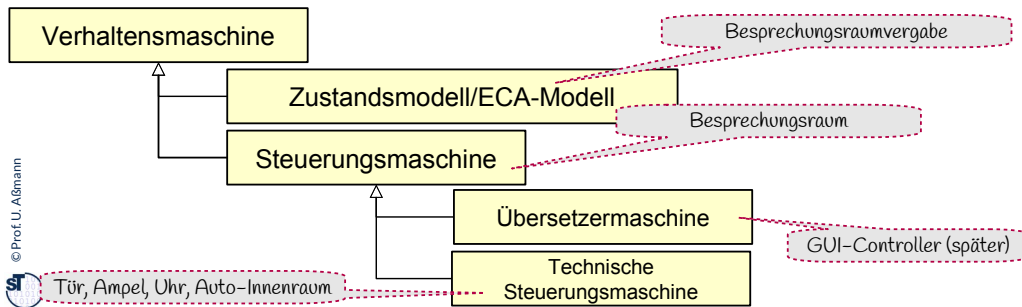
- ▶ **Definition** Eine *Aktion* ist die Beschreibung einer ausführbaren Anweisung. Dauer der Ausführung vernachlässigbar. Nicht unterbrechbar. Eine Aktion kann auch eine Folge von Einzelaktionen sein.
- ▶ In UML heißen Zustandsübergänge mit Aktionen **volle Zustandsübergänge**
- ▶ Typische Arten von Aktionen:
  - Lokale Änderung eines Attributwerts
  - Versenden einer Nachricht an ein anderes Objekt (bzw. eine Klasse)
  - Erzeugen oder Löschen eines Objekts
  - Rückgabe eines Ergebniswertes für eine früher empfangene Nachricht



## Spezielle Verhaltensmaschinen (Transduktoren)

32 Softwaretechnologie (ST)

- ▶ Ein **Zustandsmodell (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model)** ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- ▶ Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die einem Objekt zugeordnet ist und das Verhalten eines Objekts beschreibt
  - Sie beschreibt dann einen vollständigen *Objektlebenszyklus* (white-box object life cycle)
- ▶ Eine **technische Steuerungsmaschine** beschreibt das Verhalten eines technischen Gerätes
  - Aus Steuerungsmaschinen kann die Implementierung der Steuerungssoftware des Objekts bzw. des Geräts abgeleitet werden (wichtig für eingebettete Systeme)
- ▶ Eine reine **Übersetzermaschine** bildet ein Übersetzer-Objekt



Fast alle technischen Geräte können mit Steuerungsmaschinen beschrieben werden. Daher sind Steuerungsmaschinen für die Modellierung von Software für eingebettete oder cyber-physische Systeme sehr wichtig.

Fragen:

- Kann man ein Auto mit einer Steuerungsmaschine beschreiben?
- Kann man ein autonomes Auto mit einer Steuerungsmaschine beschreiben?



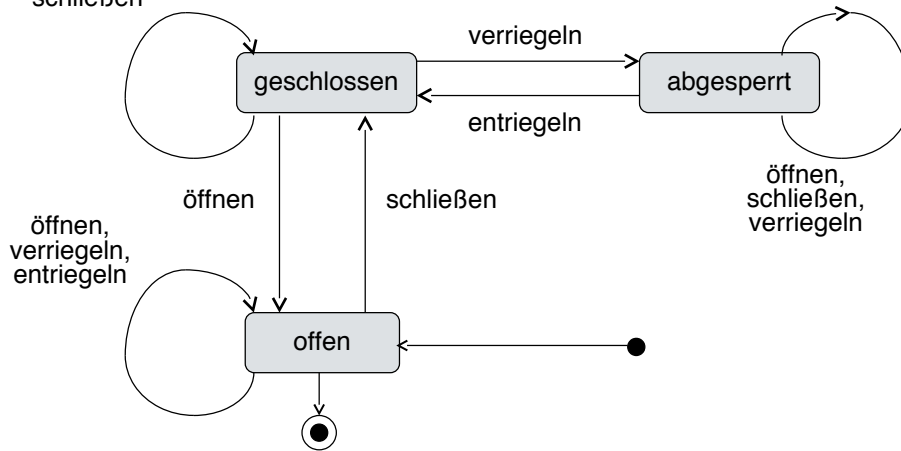
## 34.3 Unterschied von Verhaltens-, Steuer und Protokollmaschinen



## Beispiel: Protokollmaschine kontrolliert die Benutzung einer Tür

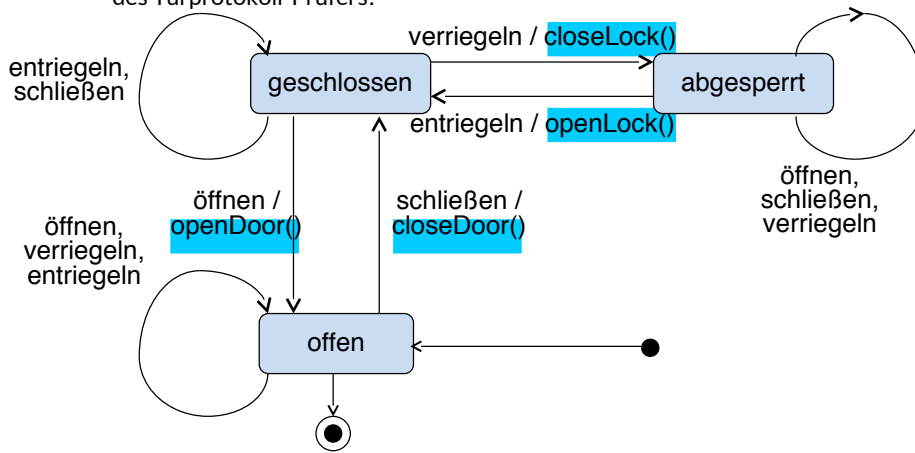
- ▶ Eine **Protokollmaschine kontrolliert**, ob ein Benutzer eine Zustandsmaschine richtig bedient,
  - d.h. ob die Benutzungsreihenfolge einer Zustandsmaschine folgt (akzeptierend, beobachtend, prüfend).

entriegeln,  
schließen



## Beispiel: Steuerungsmaschine für eine Tür einer Behindertentoilette

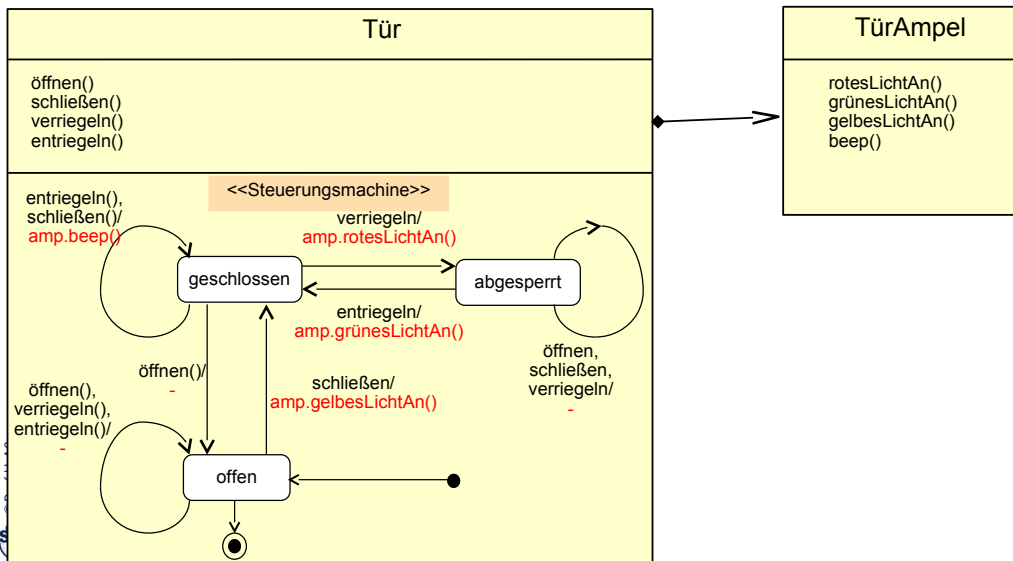
- ▶ Eine **Steuerungsmaschine** steuert zusätzlich weitere Objekte/Klassen an
- ▶ Hier: die Türsteuerung empfängt die Signale des Türbenutzers und steuert Servo-Motoren an
  - Achtung: das ist bereits die zweite Steuerungsmaschine zur Protokollmaschine des Türprotokoll-Prüfers!





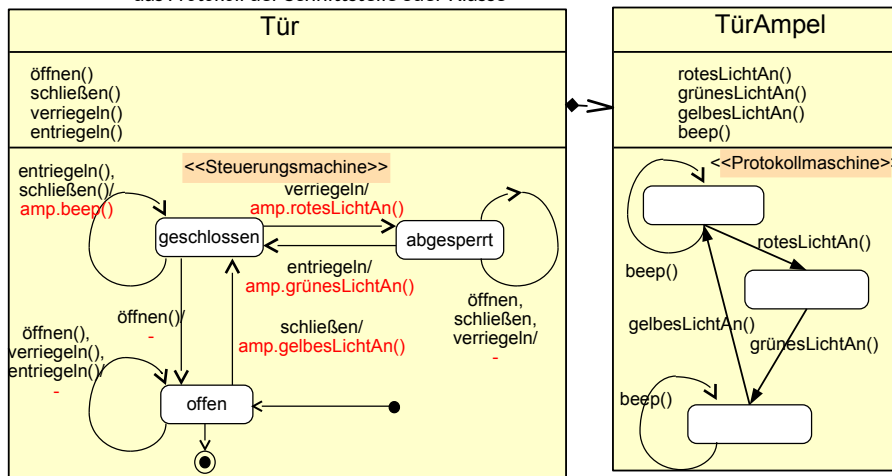
# Übersetzermaschinen: Objektlebenszyklus von innen und aussen

- ▶ Eine **Steuerungsmaschine** kann im Compartment einer Klasse erscheinen; sie beschreibt einen *whitebox*-Objektlebenszyklus (**Übersetzermaschine**)



# Wächtermaschinen: Objektlebenszyklus von nur von aussen

- ▶ Eine Protokollmaschine, die die legale Reihenfolge der Aufrufe an eine Klasse prüft, heißt **Schnittstellenautomat (interface automaton) oder Wächtermaschine des Protokolls** der Klasse.
  - Die Wächtermaschine kann in einem separaten Compartment einer Klasse erscheinen.
  - Sie beschreibt einen *blackbox*-Objektlebenszyklus, d.h. die beobachtbare Sicht von aussen, das *Protokoll* der Schnittstelle oder Klasse



## Verhaltens-(Steuerungs-)maschinen

- ▶ steuern
- ▶ müssen das Wissen über das gesteuerte System *vollständig* repräsentieren, ansonsten gerät das System ausser Kontrolle
- ▶ geben mit ihren Aktionen eine Implementierung der Steuerungssoftware des technischen Systems an
- ▶ können verschiedene Dinge steuern:
  - sich selbst (reine Steuerungsmaschine)
  - andere Klassen
  - ein Subsystem von Klassen
- ▶ **Übersetzermaschinen (transducer objects)** übersetzen das Aufrufprotokoll in ein Steuerprotokoll

## Protokollmaschinen

- ▶ kontrollieren, prüfen
- ▶ können ein *partielles* Wissen über das geprüfte System kontrollieren (der Rest des Verhaltens wird *nicht* abgeprüft)
- ▶ Beschreiben eine Sicht von aussen auf das System
- ▶ Beschreiben das Aufruf- oder Ereignisprotokoll des Systems
- ▶ **Wächtermaschinen (interface automata)** überwachen das Aufrufprotokoll an ein Objekt





## 34.4 Implementierung von Steuerungsmaschinen



# Implementierung von Steuerungsmaschinen mit Implementierungsmuster *IntegerState*

- ▶ **Zustand wird als Integer-Variable repräsentiert**, Bereich [1..n]
  - Alle Ereignisse werden zu "Reaktions"-Methoden, die von aussen aufgerufen werden
    - Externe Ereignisse werden mit "Reaktions-Methoden" modelliert
    - Interne Ereignisse werden den Implementierungen der Methoden zugeordnet
- ▶ **Reaktionsmethoden** schalten den Zustand fort, indem sie Fallanalyse betreiben
  - In jeder Methode wird eine Fallunterscheidung über den Zustand durchgeführt
  - Jeder Fall beschreibt also ein Paar (Ereignis, Zustand)
  - Der Rumpf des Falles beschreibt
    - ♦ den Zustandsübergang (Wechsel des Zustands)
    - ♦ die auszulösende Aktion
- ▶ Einsatz von *IntegerState*:
  - Einsatz in der Phase eines Compilers, der die Zeichen von Datei liest
  - Einsatz bei Protokollprüfern in der Telekommunikation
  - Einsatz bei allen Anwendungen von Zustandsmaschinen, die Performanz benötigen



# IntegerState

## Beispiel: Code zur Steuerung einer Tür (1)

41 Softwaretechnologie (ST)

```
class Tuer {  
    // Konstante zur Zustandskodierung  
    private static final int Z_offen = 0;  
    private static final int Z_geschlossen = 1;  
    private static final int Z_abgesperrt = 2;  
  
    // Zustandsvariable  
    private int zustand = Z_offen;  
  
    // Reaktionsmethode oeffnen (Reaktion auf Ereignis)  
    public void oeffnen() {  
        // Fallanalyse über Zustand  
        switch (zustand) {  
            case Z_offen:  
                break;  
            case Z_geschlossen:  
                zustand = Z_offen;  
                System.out.println("Klack");  
                break;  
            case Z_abgesperrt:  
                break;  
        }  
    }  
}
```

Modifier "final" bei Attributen: unveränderlich

Reaktionsmethode



```
public void schliessen() {
    // Fallanalyse
    switch (zustand) {
        case Z_offen:
            zustand = Z_geschlossen;
            System.out.println("Klick");
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            break;
    }
}

public void verriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            zustand = Z_abgesperrt;
            System.out.println("Knirsch");
            break;
        case Z_abgesperrt:
            break;
    }
}
```



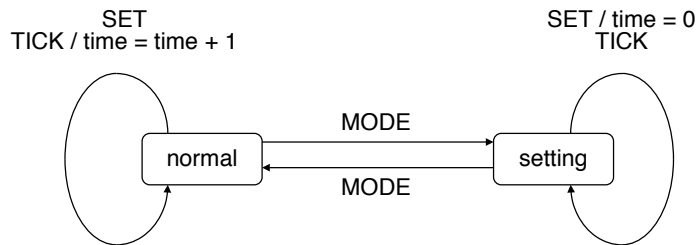
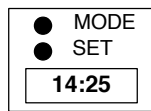
```
public void entriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            zustand = Z_geschlossen;
            System.out.println("Knirsch");
            break;
    }
}
// Client-Klasse
class TuerBediener {
    public static void main(String[] args) {
        Tuer t1 = new Tuer();
        t1.oeffnen();
        t1.schliessen();
        t1.verriegeln();
        t1.entriegeln();
        t1.oeffnen();
        t1.schliessen();
    }
}
```





# Aufgabe: Steuerungsmaschine realisieren

- ▶ Beispiel: Betriebsmodi einer Taschenuhr (stark vereinfacht)



## Implementierung mit IntegerState

```
class Clock {  
    private int time = 0;  
    private static final int NORMAL = 0;  
    private static final int SETTING = 1;  
  
    private int state = NORMAL;  
  
    // Reaktionsmethode  
    public void set () {  
        switch (state) {  
            case NORMAL: {  
                time = time+1;  
                break;  
            };  
            case SETTING: {  
                time = 0;  
                setChanged();  
                break;  
            };  
        };  
    }  
    ...// analog tick(), mode()  
}
```



# Übung

- ▶ Implementieren Sie den Ampelautomaten mit dem Implementierungsmuster IntegerState.
- ▶ Wieviele Reaktionsmethoden brauchen Sie?

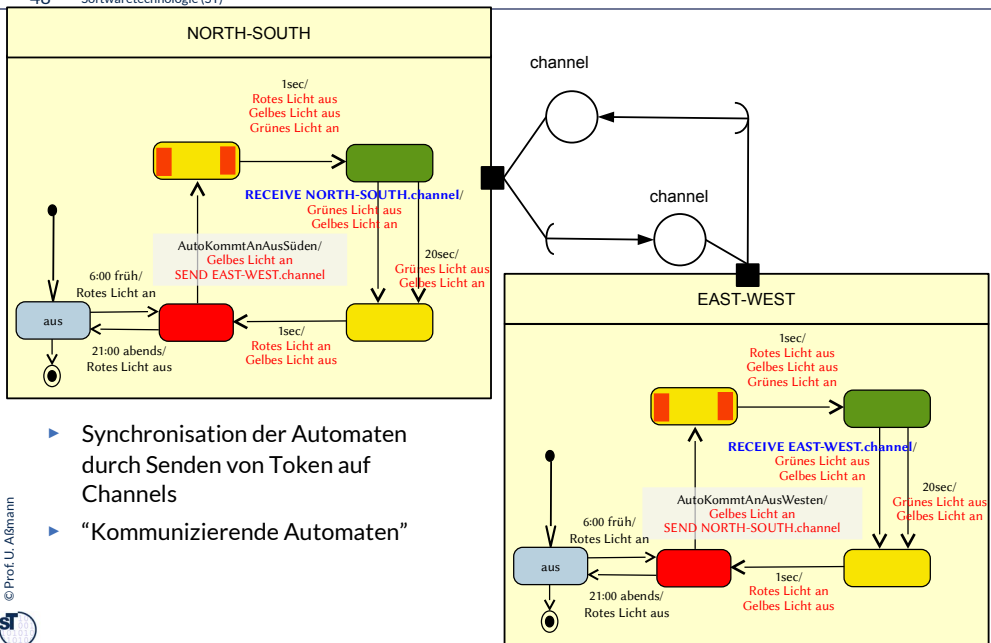


## 34.5 Kooperierende Zustandsmaschinen

- ▶ Eine besonders einfache Art von Objektnetz bilden solche, in denen alle Objekte Zustandsmaschinen bilden, die kooperieren
  - sich Nachrichten senden (Ereignisse)
  - auf Ereignisse in Nachbarobjekten mit eigenen Reaktionsmethoden reagieren



# Bsp.: Kopplung zweier Ampeln an einer Kreuzung durch Ereignis-Kanäle



- ▶ Synchronisation der Automaten durch Senden von Token auf Channels
- ▶ “Kommunizierende Automaten”

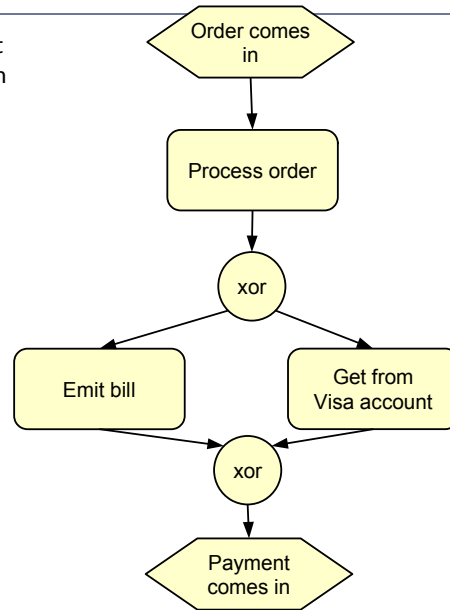
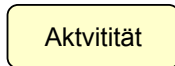


- ▶ Many slides courtesy to © Prof. Dr. Heinrich Hussmann. Used by permission.
- ▶ Warum betreibt man Verhaltensanalyse mit Aktivitätsdiagrammen und Zustandsdiagrammen?
- ▶ Erklären Sie typische Steuerungsmaschinen:
  - Erweitern Sie das Stellverhalten der Uhr zu einer Funkarmbanduhr
  - Beschreiben Sie den Lebenszyklus von Autotüren und -heckklappen
  - Warum ist ein Geldautomat nur mit endlich vielen Zuständen ausgestattet?
  - Ist ein Bahnkarten-Verkaufsautomat ebenfalls noch ein endlicher Automat?
  - Fahrstühle können sehr viele Zustände haben. [Jazayeri] Wann wäre ein Fahrstuhl ein Automat mit unendlich vielen Zuständen?
- ▶ Erklären Sie typische Zustandsmaschinen für Abläufe, und beschreiben Sie sie jeweils alternativ mit einem Aktionsdiagramm. Was ist jeweils einfacher?
  - Hausbau
  - Projekte
  - Immatrikulation eines Studenten



## 34.A.1 Andere Notationen für Aktivitätsdiagramme

- ▶ Ähnlich, wird in der industriellen Praxis oft benutzt: Ereignisgesteuerte Prozessketten (EPK), Sprache des ARIS-Toolkits für Prozessmodellierung von SAP-Systemen



## 34.A.2 Implementierung von Protokollmaschinen

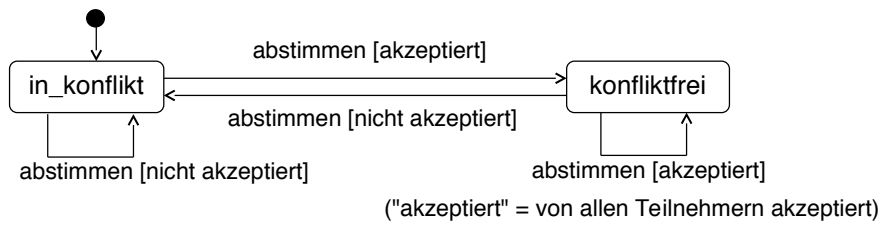


DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur



## Beispiel: Protokollmaschine

- ▶ Folgende Protokollmaschine definiert die zulässigen Aufrufreihenfolgen der Klasse Terminverschiebung:



- Begriff "Protokoll":
  - Kommunikationstechnologie
  - Regelwerk für Nachrichtenaustausch
- Protokollmaschinen in der Softwarespezifikation:
  - **zusätzliche** abstrakte Sicht auf komplexen Code (partiell Wissen)
  - Vertragsprüfer zur Einhaltung von Aufrufreihenfolgen



# Implementierungsmuster Protokollmaschine

## Explicit Tracing State

```
public Teambesprechung
    (String titel, Hour beginn, int dauer,
     Teammitglied[] teilnehmer) {
    int zustand = Z_nicht_abgestimmt;
    super(titel, beginn, dauer);
    this.teilnahme = teilnehmer;
    if (! abstimmen(beginn, dauer)){
        System.out.println("Termin bitte verschieben!");
        zustand = Z_in_konflikt;
    }
    else {
        for (int i=0; i<teilnahme.length; i++)
            teilnahme[i].teilnahmeSetzen(this);
        zustand = Z_konfliktfrei;
    }
}
```

Explizites  
Zustandsattribut

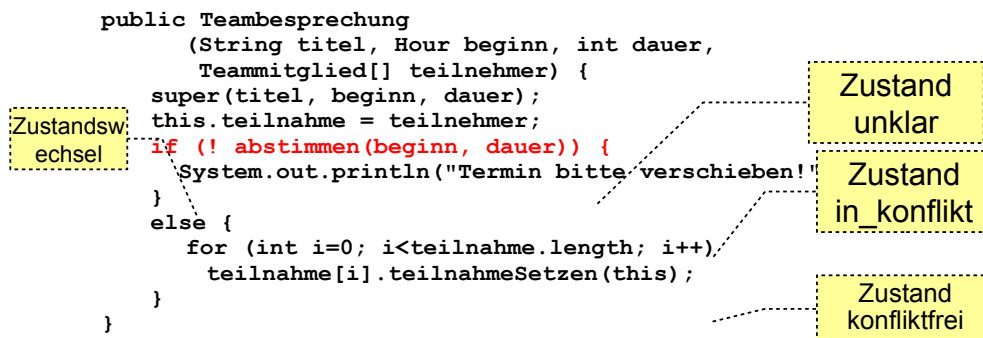
- Analog zu IntegerState, aber keine Aktionen
- Ablauflogik kann den Zustandswert benutzen (muß aber nicht!)



# Implementierungsmuster Protokollmaschine

## Implicit Tracing State

- ▶ Information über Zustand jederzeit berechenbar - hier aus den Werten der Assoziationen und den Datumsangaben
- ▶ Zustandsinformation gibt zusätzliches Modell, nicht direkt im Code wiederzufinden



- ▶ Anwendungsgebiet: Prüfen von Aufrufreihenfolgen
- ▶ Codegenerierung von Implementierungen aus Zustandsmodell:
  - Implementierungsmuster ImplicitTracingState, ExplicitTracingState, State (aber ohne Aktionen)
  - Nur zur Ableitung von Prüfcode! Zustandsmodell liefert Information für Teilaspekte des Codes (zulässige Reihenfolgen), keine vollständige Implementierung
- ▶ Praktische Aspekte:
  - In der Analyse zur Darstellung von Geschäftsprozessen und -regeln
  - komplexen Lebenszyklen für Geschäftsobjekte (Modellierung mit Sichten, die jeweils durch eine Protokollmaschine beschrieben werden)
  - Nützlich für den Darstellung von Klassen mit komplexen Regeln für die Aufrufreihenfolge
  - Hilfreich zur Ableitung von Status-Informationen für Benutzungs-Schnittstellen
  - Hilfreich zum Definieren sinnvoller Testfälle für Klassen

