34. Lebenszyklen einfacher Objekte: Aktionsdiagramme

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden

Version 13-0.5, 14.06.13

1) Aktivitätendiagramme in UML
2) Zustandsdiagramme in UML
3) Verhaltens-, Steuerungs-, und Protokollmaschinen
4) Implementierung von Steuerungsmaschinen
5) Einsatz im Test

A1) Impl. von Protokollmaschinen

Obligatorische Literatur

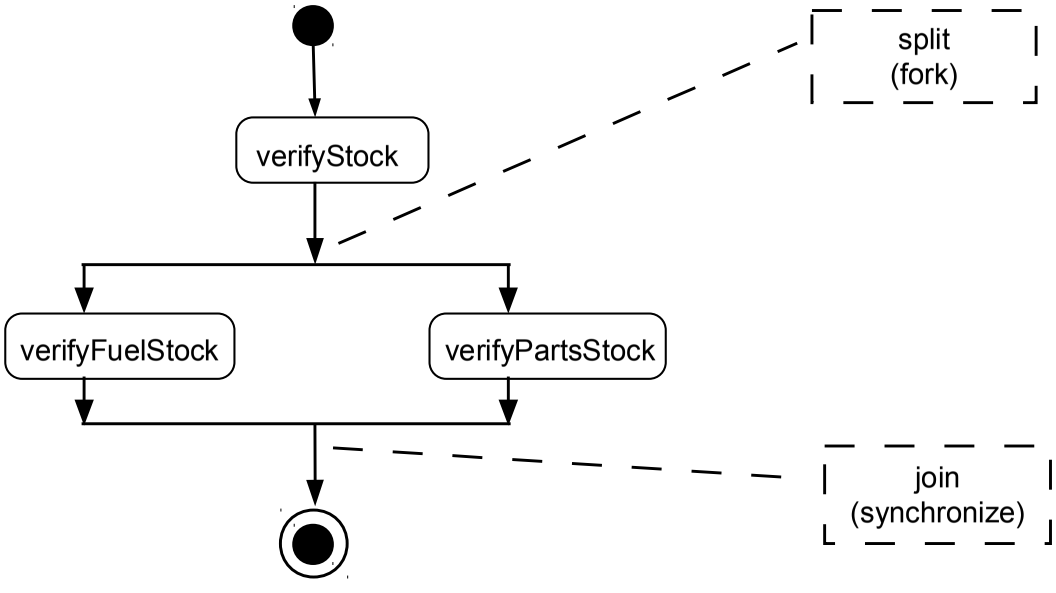
- Zuser 7.5.3
- Störrle Kap. 10 (Zustandsdiagramme), Kap. 11 (Aktivitätsdiagramme)
- ST für Einsteiger: Kap. 10

34.1. Aktivitätsdiagramme

Aktionsdiagramme: Aktivitätsdiagramme (AD), Statecharts (SC)

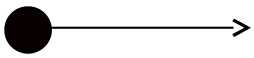
Dynamische Modellierung (Verhaltensmodellierung)

- 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



Start- und Endzustand

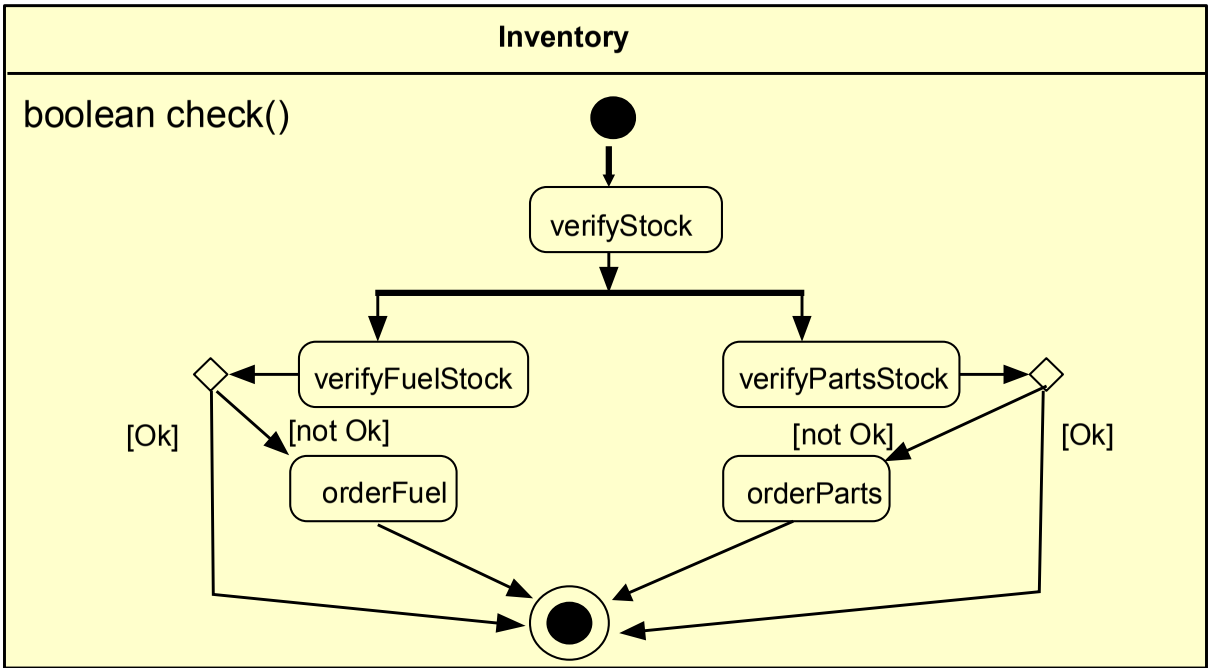
- Jedes Aktionsdiagramm sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".
- **Notation:**



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

Aktivitätsdiagramm als Verhalten einer Methode (activity diagram)

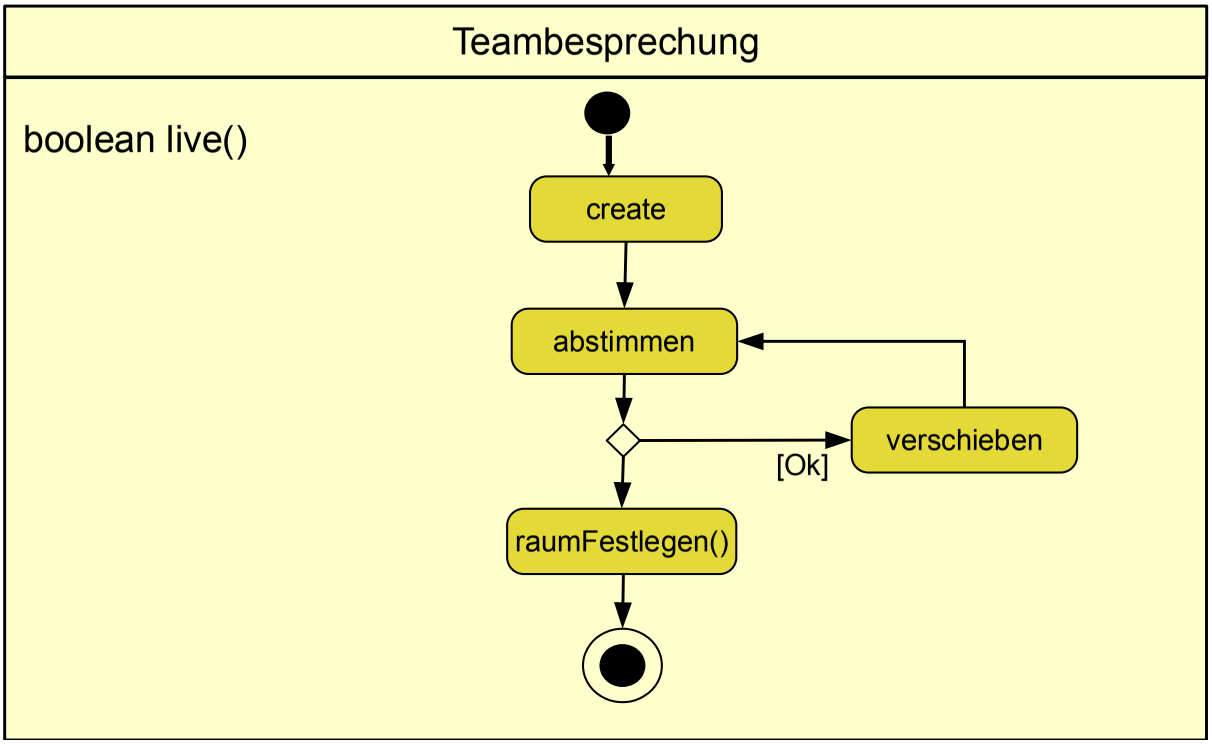
- Aktivitätsdiagramme können das Verhalten einer Methode beschreiben, dann werden sie in ein Abteil der Klasse notiert
- Aktivitäten, verbunden durch Datenfluß (Datenflußdiagramm, data-flow diagram)
  - Parallele Aktivitäten in parallelen Zweigen
  - Bedingungen (guards) bestimmen, ob über eine Kante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



[Pfleeger]

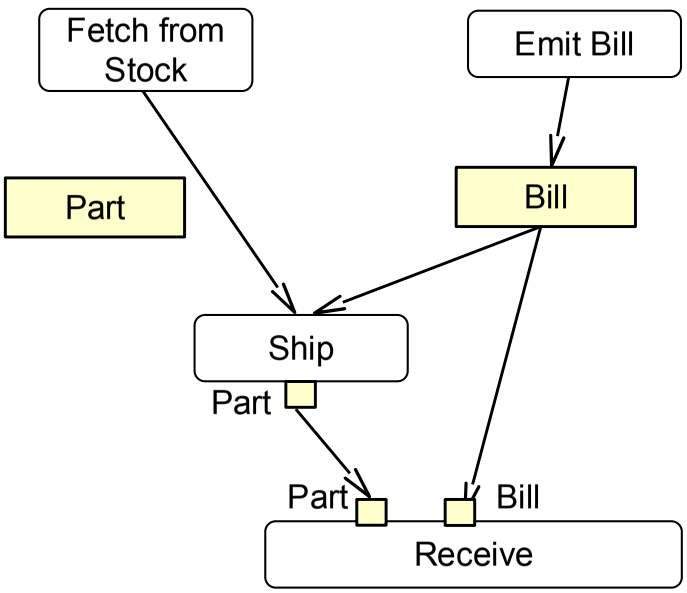
Aktivitätsdiagramm für Lebenszyklus eines Objekts

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



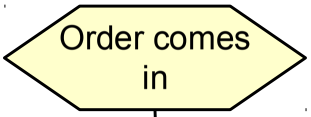
Verschiedene Notationen für Datenfluß

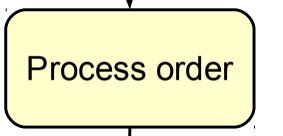
- Objekte, die zwischen Aktivitäten fließen, können verschieden notiert werden
- *Pins* sind benamte Parameter der Aktivitäten

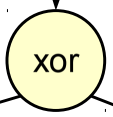


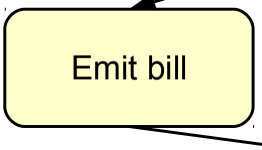
Andere Notationen für Aktivitätsdiagramme

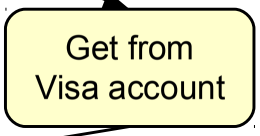
- Ähnlich, wird in der industriellen Praxis oft benutzt: Ereignisgesteuerte Prozessketten (EPK), Sprache des ARIS-Toolkits für Prozessmodellierung von SAP-Systemen

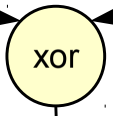


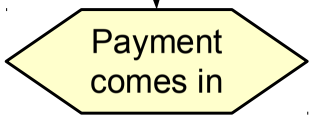












http://de.wikipedia.org/wiki/Ereignisgesteuerte_Prozesskette

34.2 UML-Zustandsdiagramme (Zustandsmaschinen, Statecharts)

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

Zustandsbasierte dynamische Modellierung

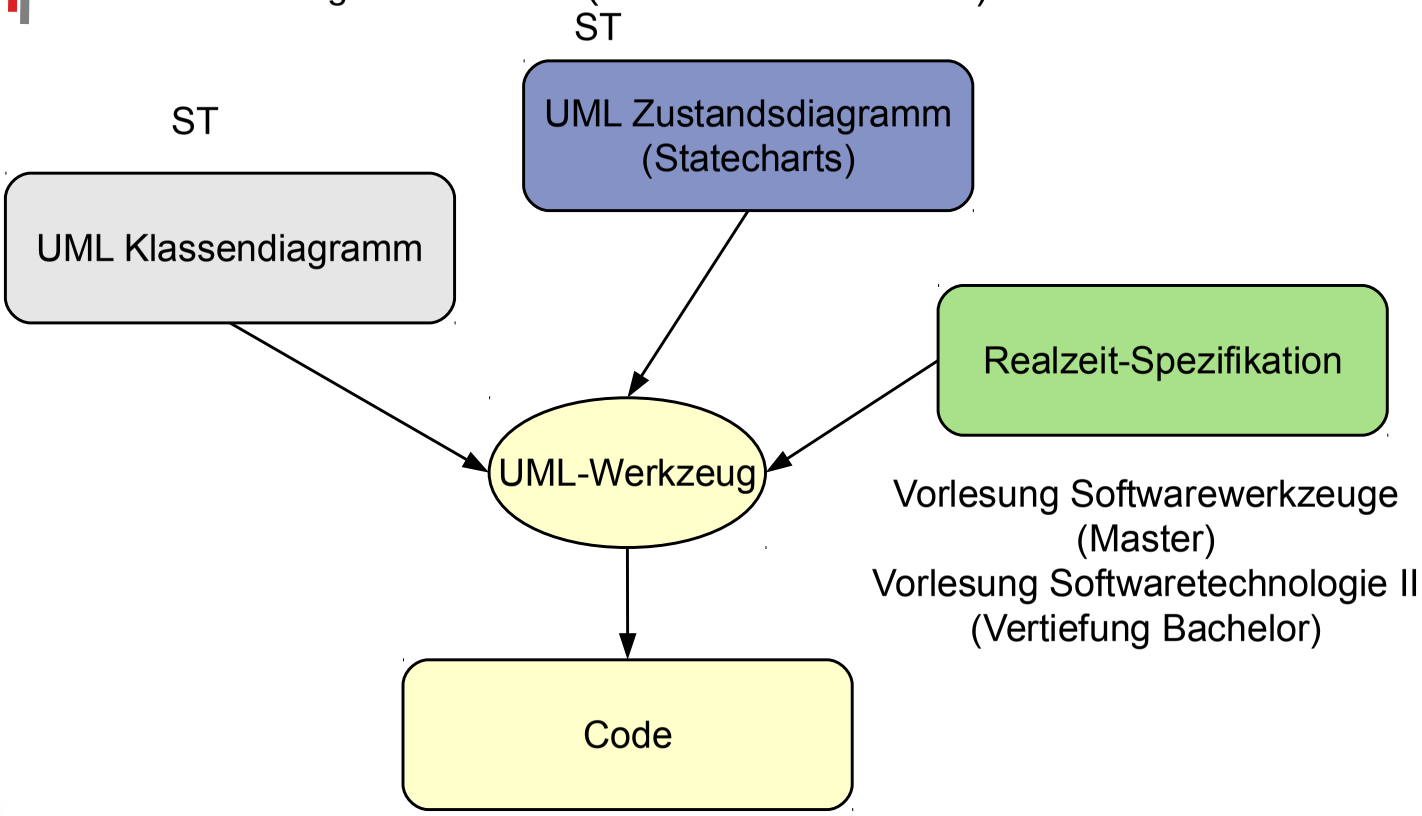
- Objekt-Verhalten und Szenarien können auch *zustandsbetont* analysiert werden
  - Man frage: *Wie ändern sich die Zustände des Systems, wenn bestimmte Ereignisse auftreten?*
  - Es entsteht ein ECA-Architekturstil (event-condition-action)
- Besonders wichtig bei:
  - Sicherheitskritischen Sytemen: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

Beispiel: Spezifikation von Software für sicherheitskritische Systeme

- Spezifikationen von Zustandsautomaten für Fly-by-wire und Drive-by-wire

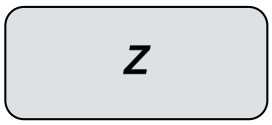
http://en.wikipedia.org/wiki/File:Hands-free_Driving.jpg

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



UML-Zustandsmodelle

- **Definition:** Ein ***Zustand*** ist eine Eigenschaft eines Objektes oder Systems, die über einen begrenzten Zeitraum besteht.
- **Notation:**



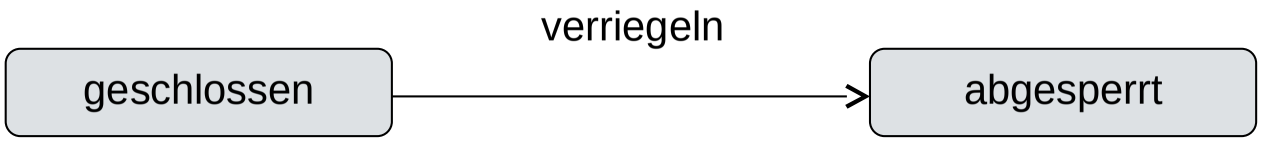
- Was ist ein "System"?
  - Technisch: Ein Objekt oder eine Netz von Objekten, ein hierarchisches Objekt, auch ein komplexes Objekt
  - Praktisch:
    - Eigenschaft eines komplexen Softwaresystems
    - Eigenschaft eines Arbeitsprozesses
    - Eigenschaft eines Produkts eines Arbeitsprozesses
    - Eigenschaft eines einzelnen Objekts

Endliche Automaten 1 (Akzeptoren)

- Theoretische Informatik, Automatentheorie:

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

  - einer Menge S von Zuständen
  - einer (partiellen) Übergangsfunktion trans : S x A → S
  - einem Startzustand s0 ∈ S
  - einer Menge von Endzuständen $S_f \subseteq S$

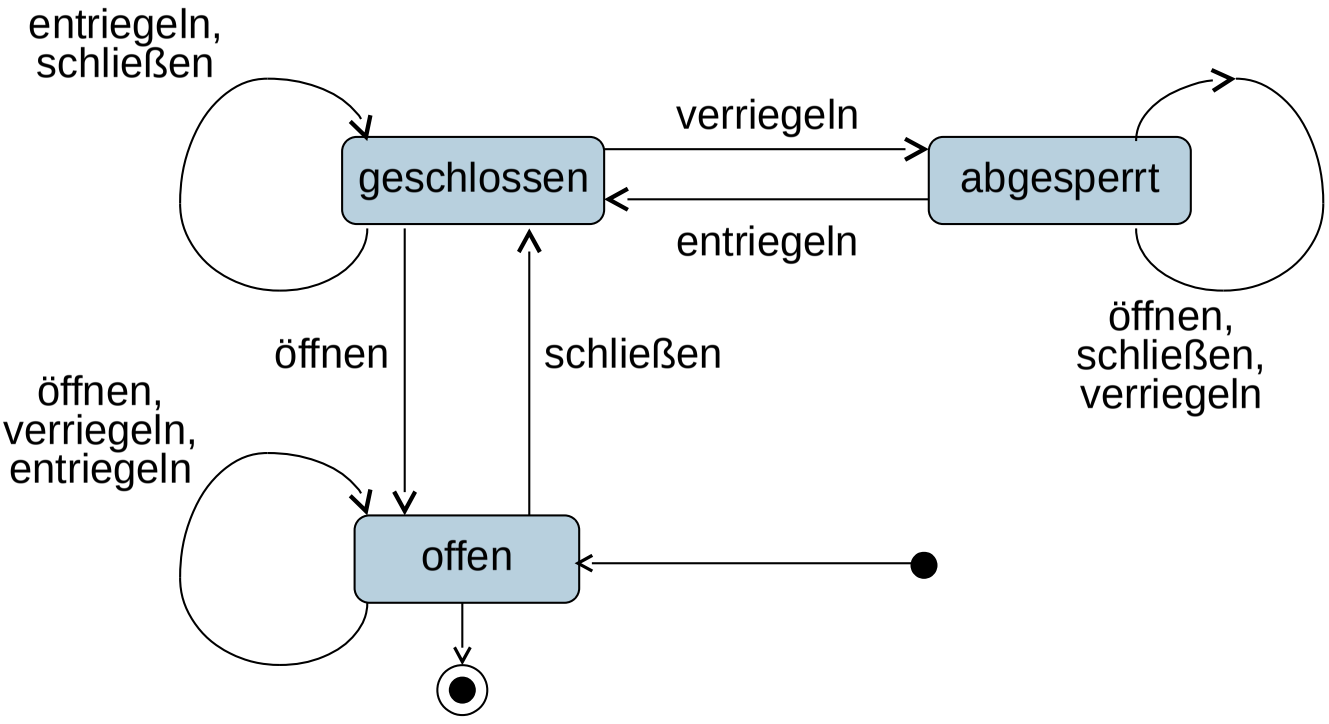


trans(geschlossen, verriegeln) = abgesperrt

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 ein Protokoll



Zustandstabellen von Protokollmaschinen

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

- Alternative Notation

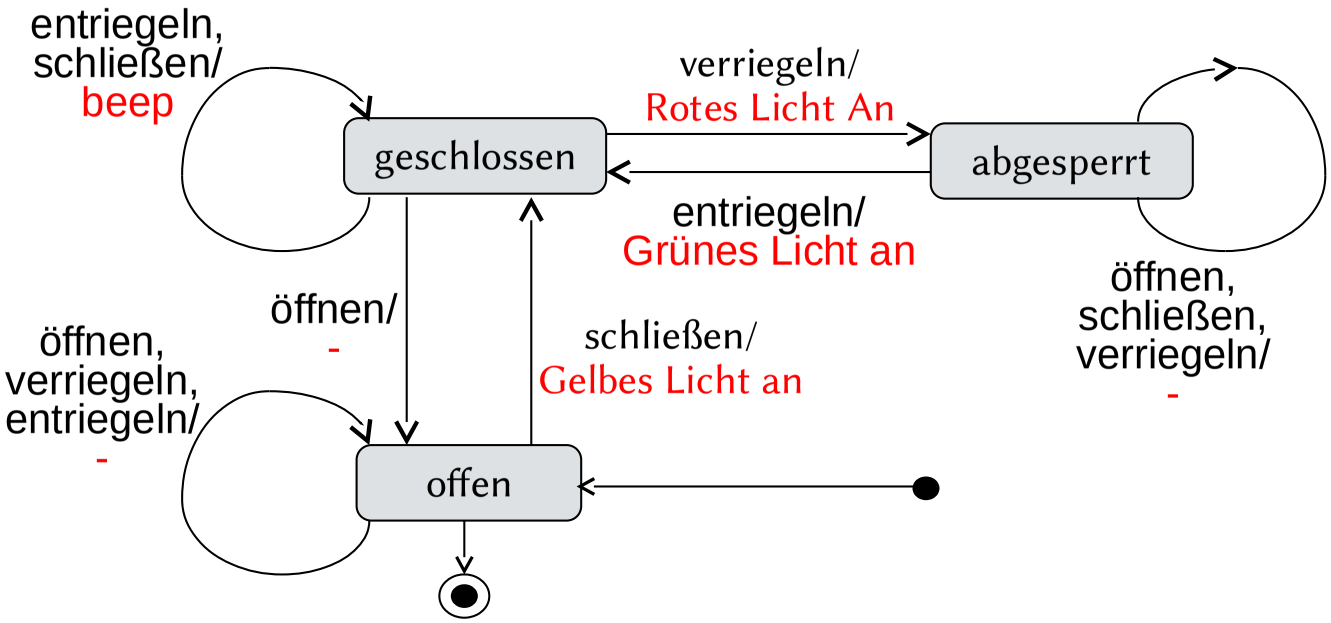
Endliche Automaten 2 (Transduktoren)

- Ein **endlicher Zustandsübersetzer (Transduktor)** über einem Eingabealphabet A und einem Ausgabealphabet B ist ein Tupel, bestehend aus:
  - einer Menge S von Zuständen
  - einer (partiellen) Übergangsfunktion trans : S x A → S
  - einem Startzustand s0 ∈ S
  - einer Menge von Endzuständen $S_f \subseteq S$

trans(geschlossen, verriegeln) = (abgesperrt) / rotes Licht einschalten

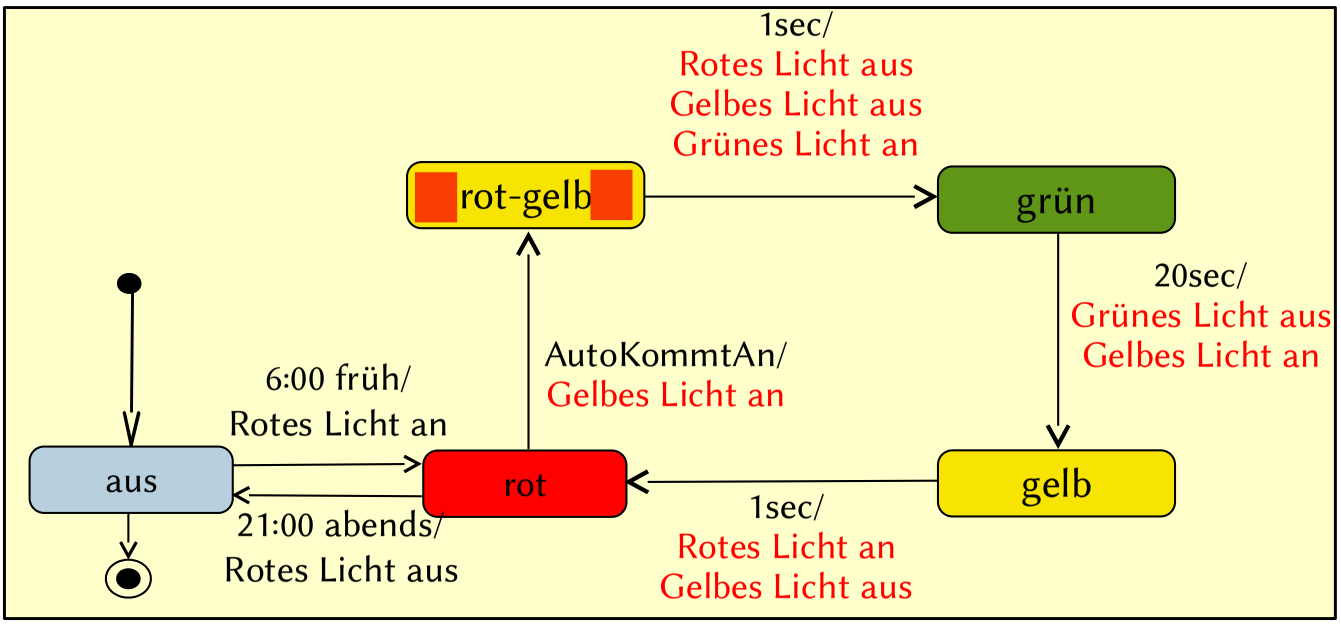
Beispiel: Zustandsmodell einer Tür

- 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: Zustandsmodell einer bedarfsgesteuerten Ampel

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



Semantik eines Zustandsmodells

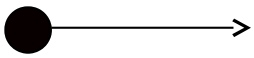
- Ein Zustandsmodell ist endlich, definiert aber einen unendlichen Zustandsraum (Semantik)
- Die Semantik eines Zustandsmodells ist definiert als Menge von Sequenzen (Folgen):
  - in der Theoretischen Informatik:
    - Menge von "akzeptierten Wörtern" (Sprache über Grundalphabet von Ereignissen)
  - 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 Verallgemeinerung: "Automaten mit Ausgabe"
  - *Transduktor (Mealy-Automat)*: Ausgabe bei Übergang
    - Softwaretechnik: *Aktion* bei Übergang
  - *Akzeptor (Moore-Automat)*: Ausgabe bei Erreichen eines Zustands

Übung

- Schreiben Sie 4 zulässige Schaltfolgen der bedarfsgesteuerten Ampel aus
- Was ähnelt sich?

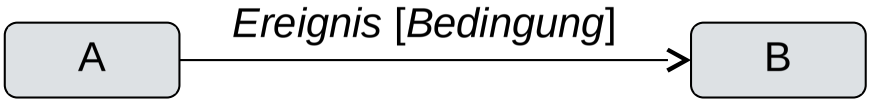
Start- und Endzustand (wie bei AD)

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

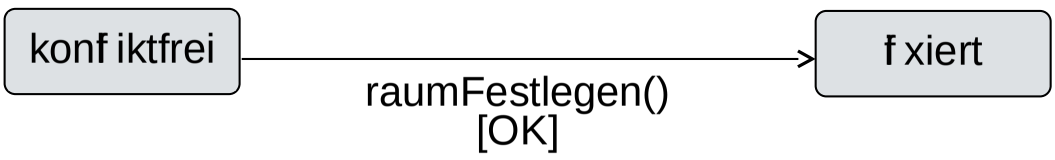


- Ein Zustandsdiagramm kann einen oder mehrere Endzustände haben.
- **Notation:** *("bull's eye")*

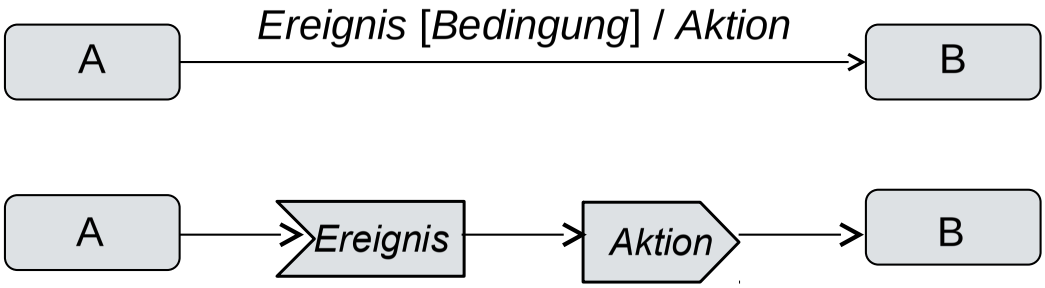
Bedingte Zustandsübergänge in Protokollmaschinen



- **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:**



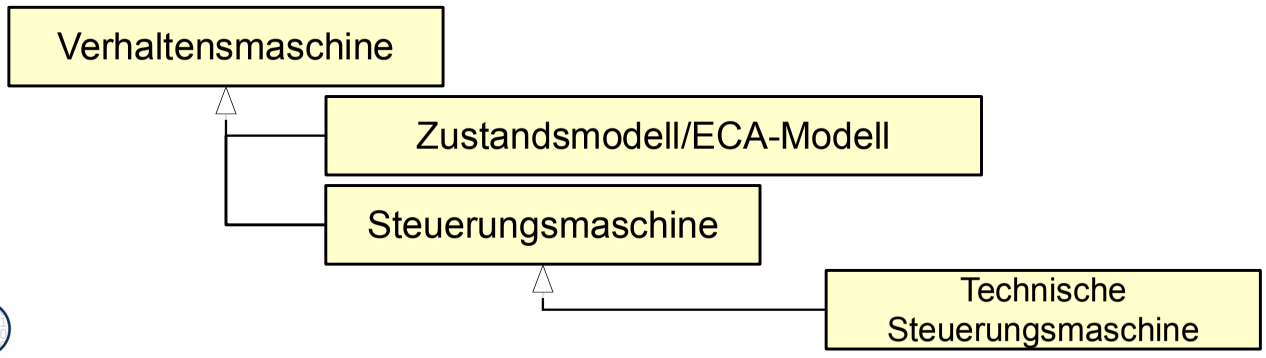
Aktionen bei Zustandsübergängen in Verhaltensmaschinen



- **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):

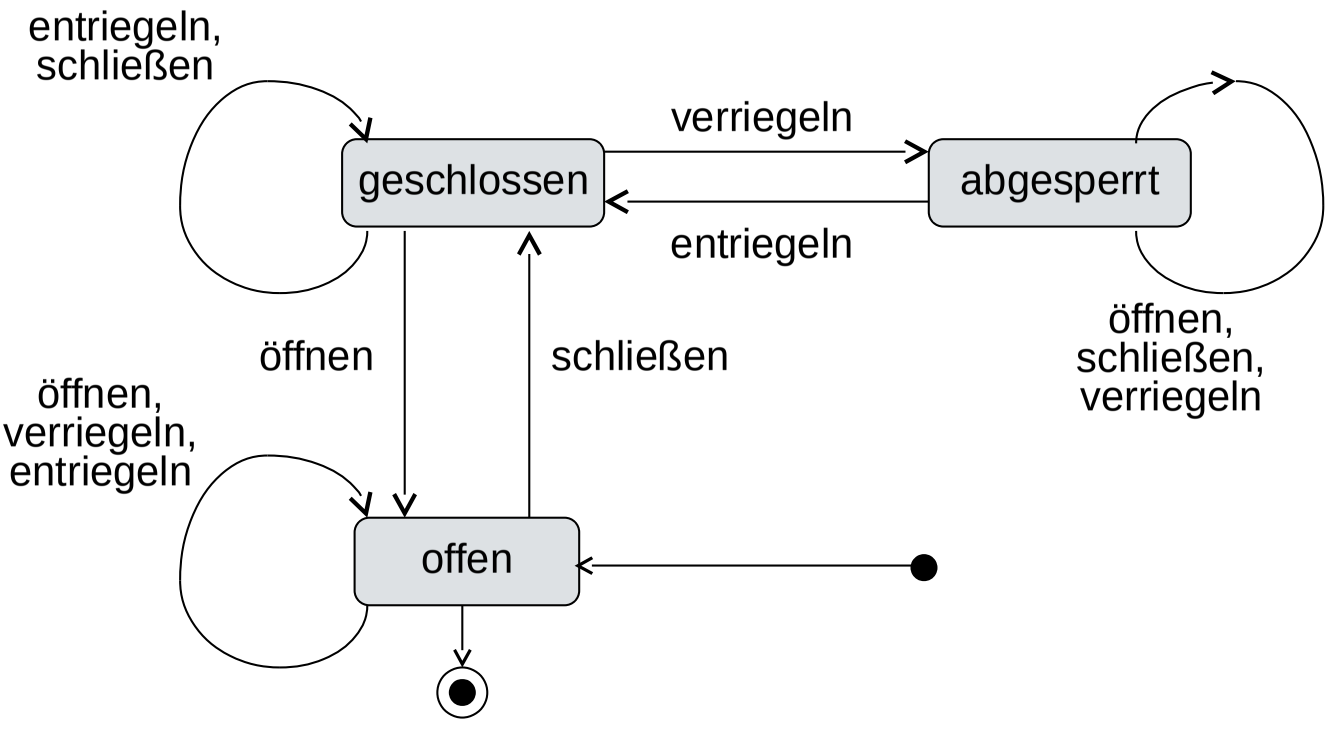
- Ein **Zustandsmodell (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model)** ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die das Verhalten eines Objekts beschreibt
  - Sie beschreibt 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)



34.3 Unterschied von Verhaltens-, Steuer und Protokollmaschinen

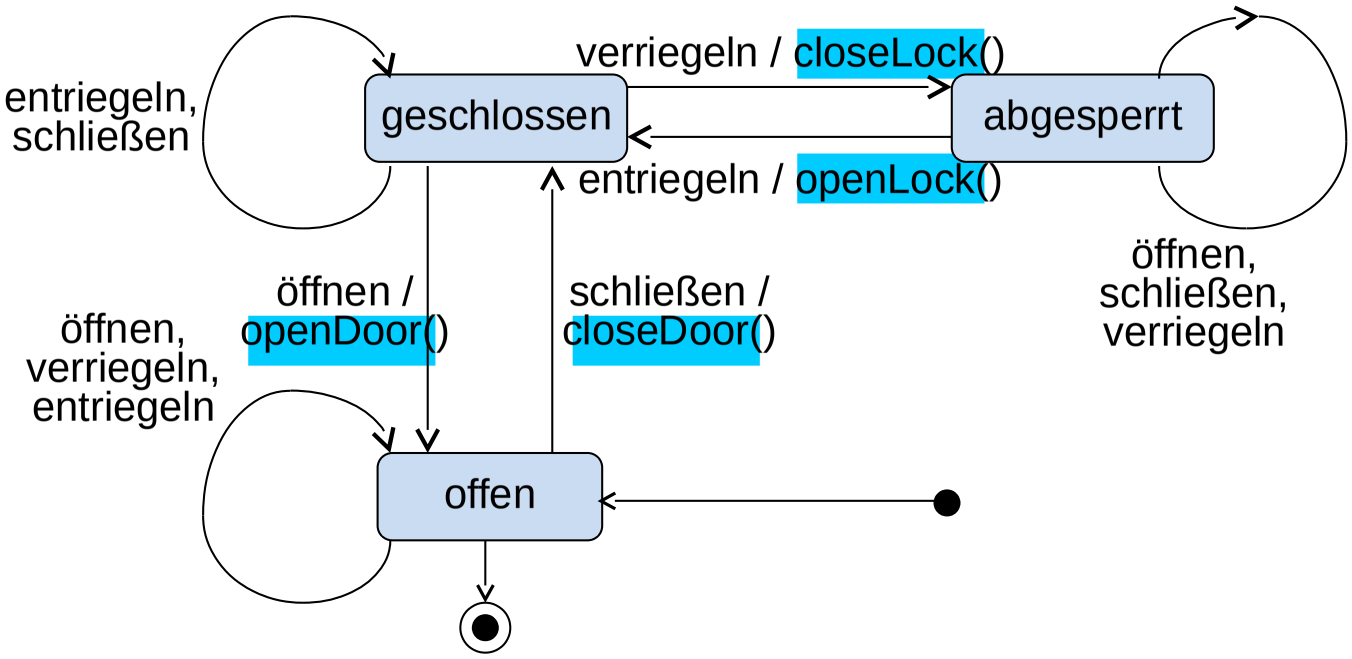
Beispiel: Protokollmaschine für eine Tür

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



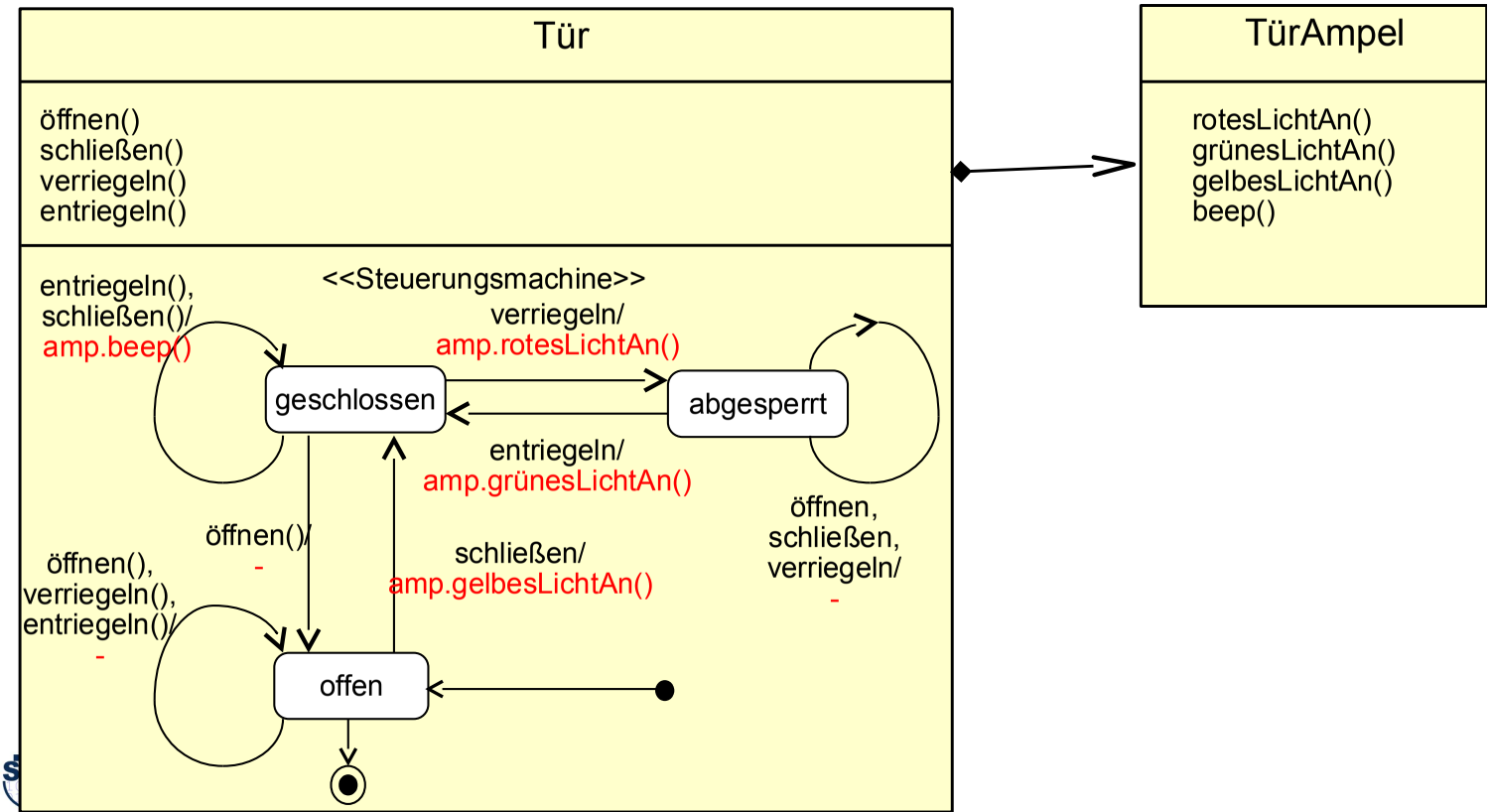
Beispiel: Steuerungsmaschine für eine Tür einer Behindertentoilette

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



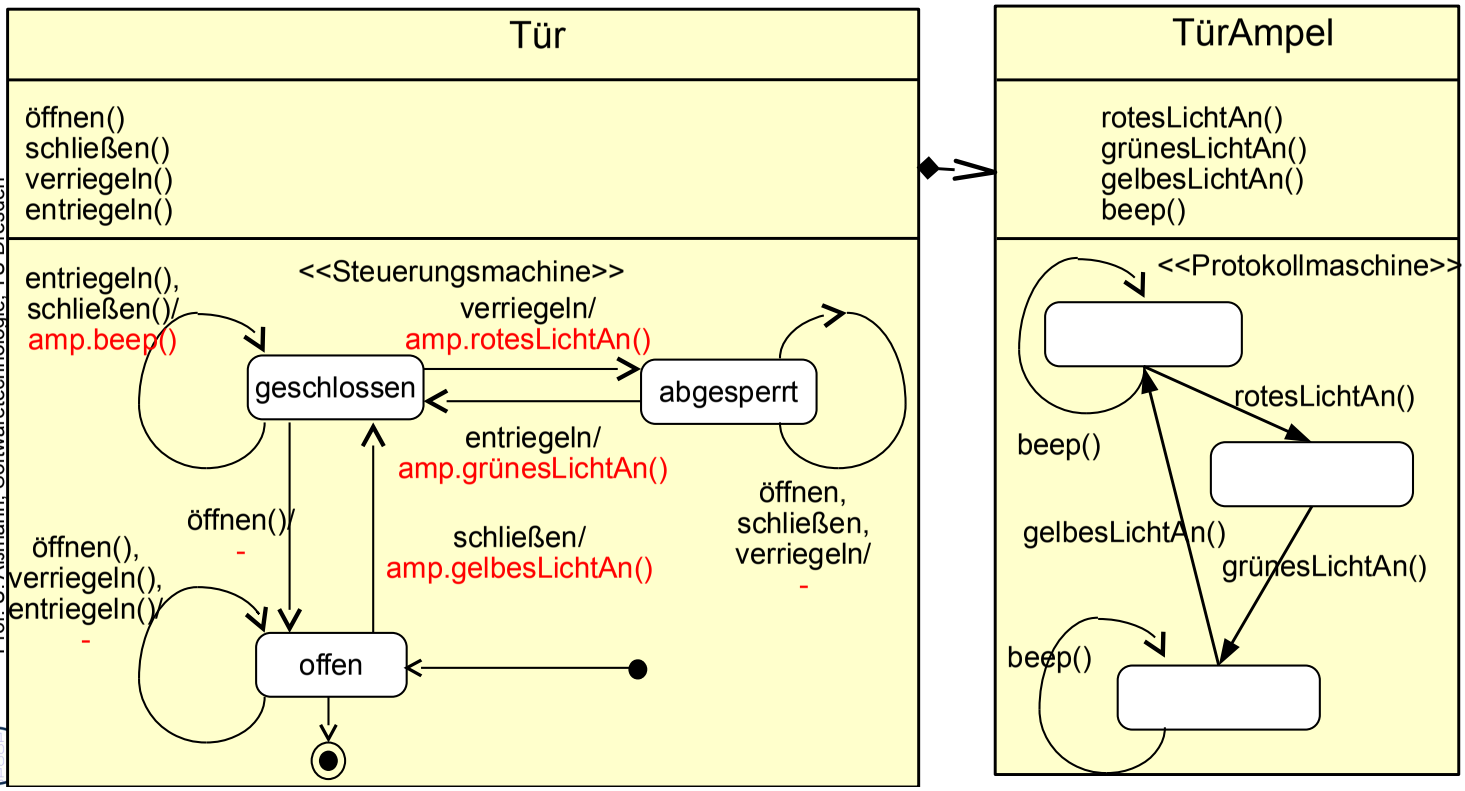
Objektlebenszyklus von innen und aussen

- Eine Steuerungsmaschine kann in einer Klasse erscheinen; sie beschreibt einen *whitebox*-Objektlebenszyklus



Objektlebenszyklus von nur von aussen

- Auch eine Protokollmaschine kann in einer Klasse erscheinen, sie beschreibt einen *blackbox*-Objektlebenszyklus, d.h. die beobachtbare Sicht von aussen, das *Protokoll* der Schnittstelle oder Klasse



Unterschied

- 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
- Protokollmaschinen
  - kontrollieren
  - 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

34.4 Implementierung von Steuerungsmaschinen

Implementierung von Steuerungsmaschinen mit Implementierungsmuster *IntegerState*

- Entwurfsmuster *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

IntegerState
Beispiel: Code zur Steuerung einer Tür (1)

Modifier "final" bei Attributen: unveränderlich

```
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
    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)

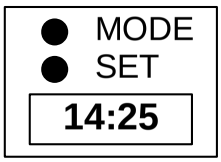

```
    public void schliessen() {
      // Fallanalyse
      switch (zustand) {
            case Z_offen:
                  zustand = Z_geschlossen;
                  System.out.println("Klick");
                  break;
            case Z_geschlossen:
                  break;
            case Z_abgesperrt: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

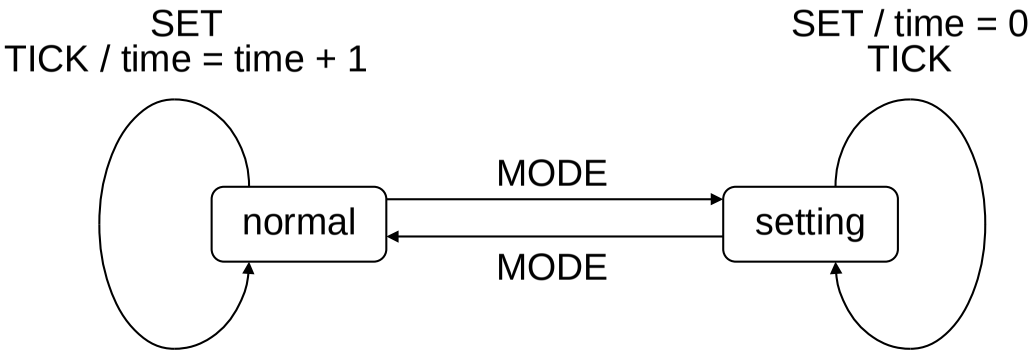

IntegerState
Beispiel: Code zur Steuerung einer Tür (3)


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





Implementierung mit IntegerState

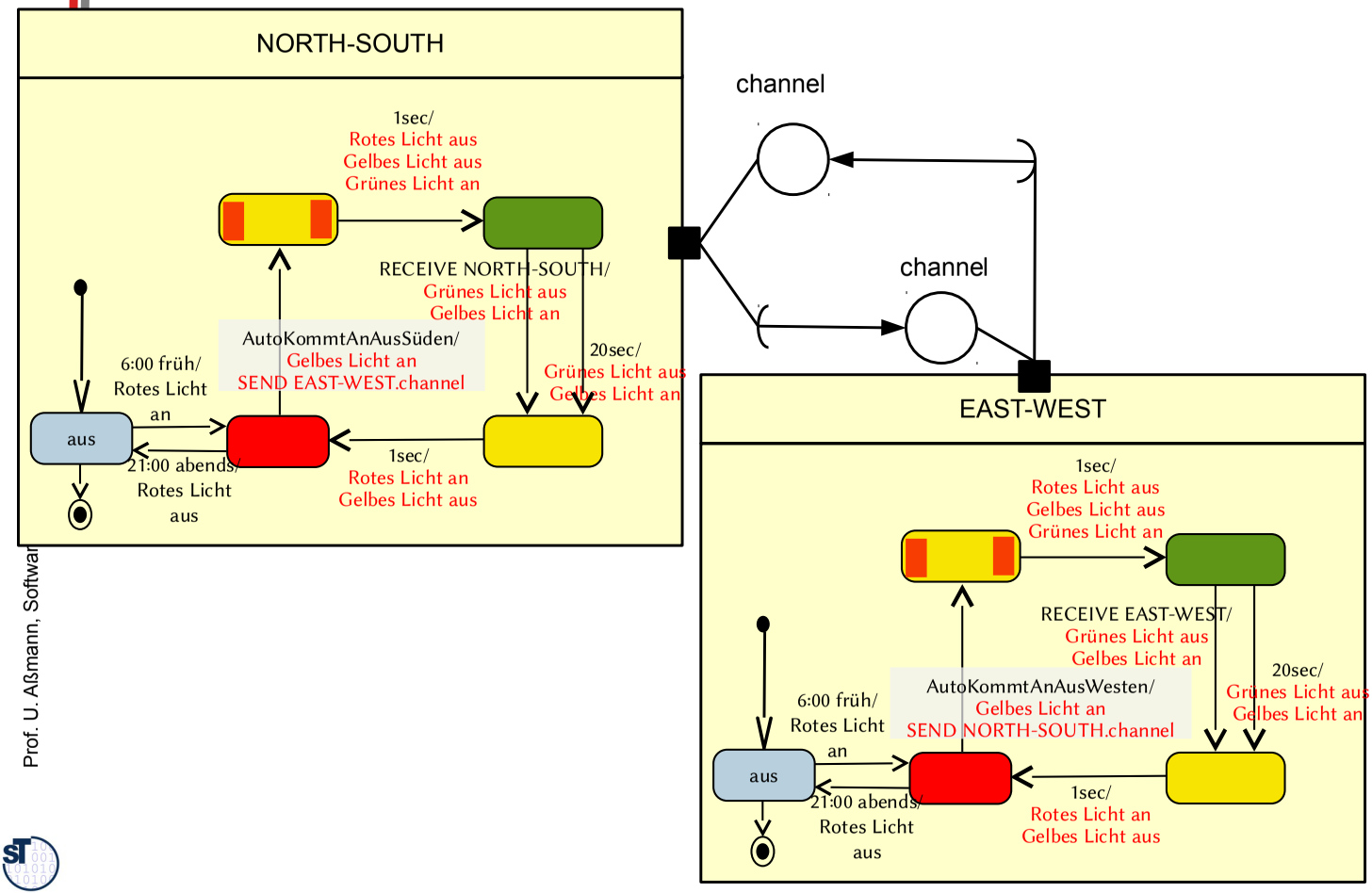

```
class Clock {

  private int time = 0;
  private static final int NORMAL = 0;
  private static final int SETTING = 1;

  private int state = NORMAL;

  public void set () {
      switch (mode) {
            case NORMAL: {
                  time = time+1;
                  break;
            };
            case SETTING: {
                  time = 0;
                  setChanged();
                  break;
            };
      };
  }

  ...// analog tick(), mode()
}
```


34.5 Kooperierende Zustandsmaschinen

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



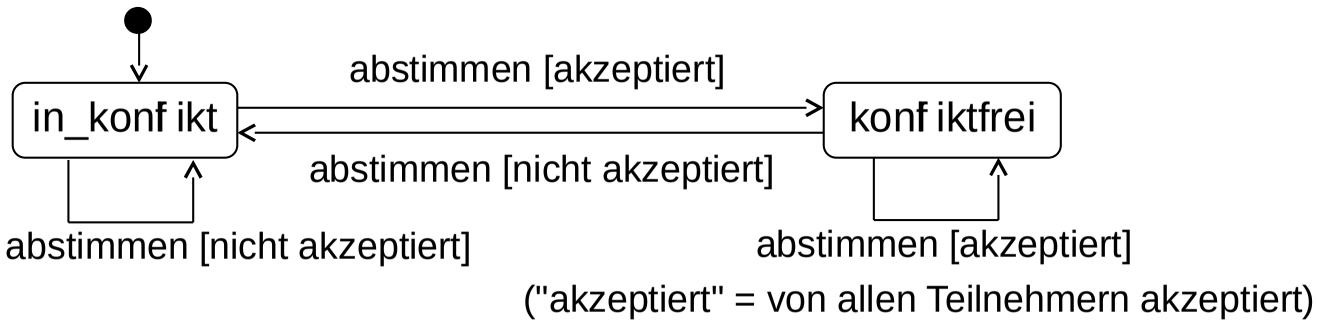
Appendix

- Many slides courtesy to © Prof. Dr. Heinrich Hussmann, 2003. Used by permission.
- Typische Steuerungsmaschinen
  - Stellverhalten von Uhren
  - Autotüren und -heckklappen
  - Geldautomaten
  - Bahnkarten-Verkaufsautomat
  - Fahrstühle [Jazayeri]
- Typische Zustandsmaschinen für Abläufe:
  - Hausbau
  - Projekte
  - Immatrikulation eines Studenten

34.A.1 Implementierung von Protokollmaschinen

Beispiel: Protokollmaschine

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



("akzeptiert" = von allen Teilnehmern akzeptiert)

- 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);
   }
}
```

Zustands wechsel

Zustand unklar

Zustand in_konflikt

Zustand konfliktfrei

Protokoll-Maschinen: Zusammenfassung

- 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