Objektorientierte Analyse
34 Dynamische Modellierung und Szenarioanalyse mit Aktionsdiagrammen

Prof. Dr. rer. nat. habil. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 11-0.2, 20.06.11

1) Aktivitätendiagramme in UML
2) Zustandsdiagramme in UML
3) Verhaltens-, Steuerungs-, und Protokollmaschinen
4) Einsatz
5) Implementierung von Steuerungsmaschinen
6) Impl. von Protokollmaschinen
7) Strukturierte Zustände

Obligatorische Literatur

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

Überblick Teil III: Objektorientierte Analyse (OOA)

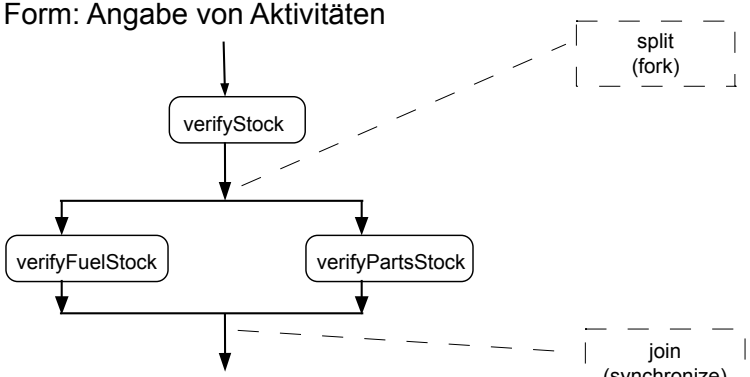
1. Überblick Objektorientierte Analyse
   1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML für das Domänenmodell
   1. Strukturelle metamodellgetriebene Modellierung
   2. Modellierung von komplexen Objekten
      1. Modellierung von Hierarchien
      2. (Modellierung von komplexen Objekten und ihren Unterobjekten)
      3. Modellierung von Komponenten (Groß-Objekte)
   3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen
   1. Funktionale Verfeinerung: Dynamische Modellierung von Lebenszyklen mit Aktionsdiagrammen (34)
   2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
   3. (Funktionale querschneidende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent

34.1. Dynamische Modellierung und Szenarienanalyse mit Aktivitätsdiagrammen

Aktionsdiagramme: Aktivitätsdiagramme, Statecharts

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*
- Einfachste Form: Angabe von Aktivitäten



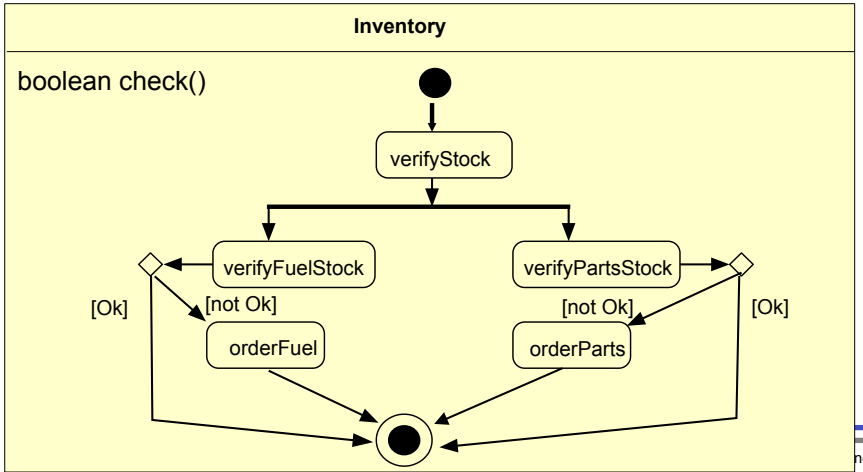
Start- und Endzustand

- 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 (activity diagram)

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

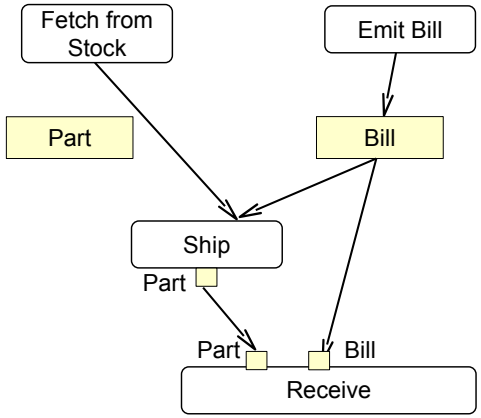


Aktivitätsdiagramm für Lebenszyklus

AD für Ports von Komponenten

Verschiedene Notationen für Datenfluß

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



Punktweise und querschneidende dynamische Verfeinerung

- Die funktionalen Verfeinerung per Objekt oder Methode geschieht *punktweise*, d.h. pro Objekt oder Methode (**punktweise funktionale Verfeinerung**).
- Ergebnis:
  - **Lebenszyklus** des Objekts
  - **Implementierung** einer Methode
- Daneben kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen (*querschneidende dynamische Modellierung,* **querschneidende funktionale Verfeinerung**).
  - Dazu führt man eine *Szenarienanalyse* durch, die quasi die Draufsicht auf ein Szenario ermittelt
  - Siehe nächstes Kapitel

34.2 UML-Zustandsdiagramme (Statecharts)

Zur punktweisen funktionalen Verfeinerung

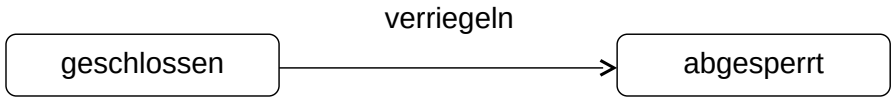
Zustandsbasierte dynamische Modellierung

- 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 mit UML-Statecharts

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 S

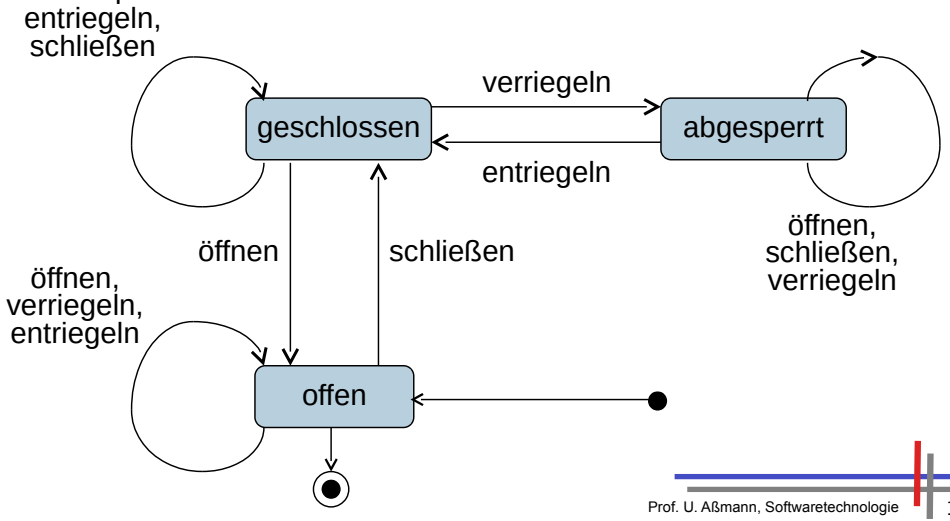


trans(geschlossen, verriegeln) = abgesperrt

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

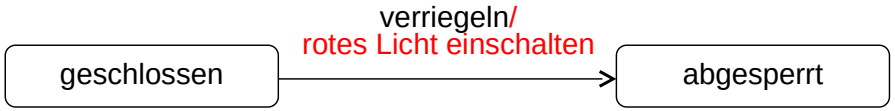


Zustandstabellen von Protokollmaschinen

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

Endliche Automaten 2 (Transduktoren)

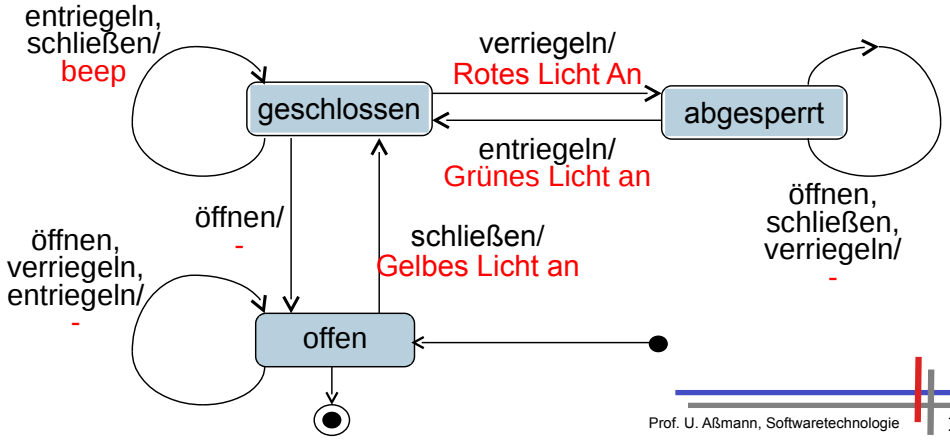
- Ein **endlicher Zustandsübersetzer (Transduktor)** über einem Eingabealphabet A und einem Ausgabealphabet B ist ein Tupel, bestehend aus:
  - einer Menge S von Zuständen
  - einer (partiellen) Übergangsfunktion trans : S x A → S
  - einem Startzustand s0 S
  - einer Menge von Endzuständen S_f S



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
- Heisst in UML ***Verhaltens(zustands)maschine***

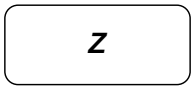


Semantik eines Zustandsmodells

- Die Semantik eines Zustandsmodells ist definiert als Menge von Sequenzen (Folgen):
  - in der Theoretischen Informatik:
    - Menge von "akzeptierten Wörtern" (über Grundalphabet von Ereignissen)
  - in der Softwaretechnik:
    - Menge von zulässigen *Ereignisfolgen*
    - Menge von zulässigen *Aufruffolgen*
- Wichtige Verallgemeinerung: "Automaten mit Ausgabe"
  - *Transduktor (Mealy-Automat)*: Ausgabe bei Übergang
    - Softwaretechnik: *Aktion* bei Übergang
  - *Akzeptor (Moore-Automat)*: Ausgabe bei Erreichen eines Zustands

UML-Zustandsmodelle

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

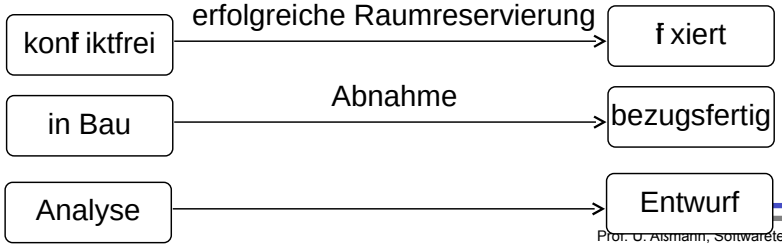


- Was ist ein "System"?
  - Technisch: Ein Objekt oder eine Gruppe von Objekten, ein hierarchisches Objekt, auch ein komplexes Objekt
  - Praktisch:
    - Eigenschaft eines komplexen Softwaresystems
    - Eigenschaft eines Arbeitsprozesses
    - Eigenschaft eines Produkts eines Arbeitsprozesses
    -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

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:**

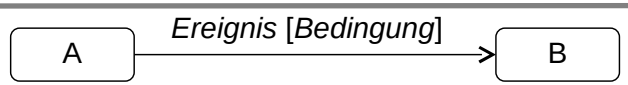


Start- und Endzustand (Wdh.)

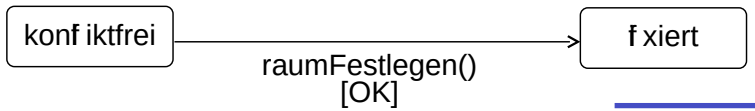
- Jedes Zustandsdiagramm sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".
- **Notation:**

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

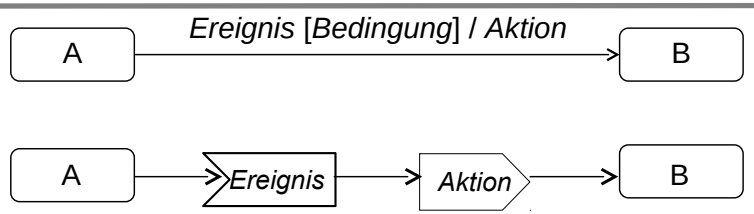
Bedingte Zustandsübergänge in Protokollmaschinen



- **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:**



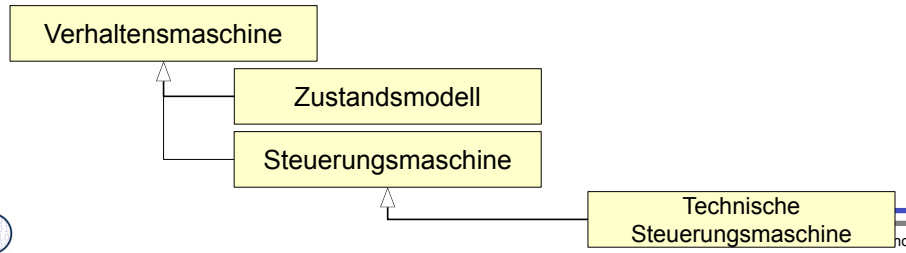
Aktionen bei Zustandsübergängen in Verhaltensmaschinen



- **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):

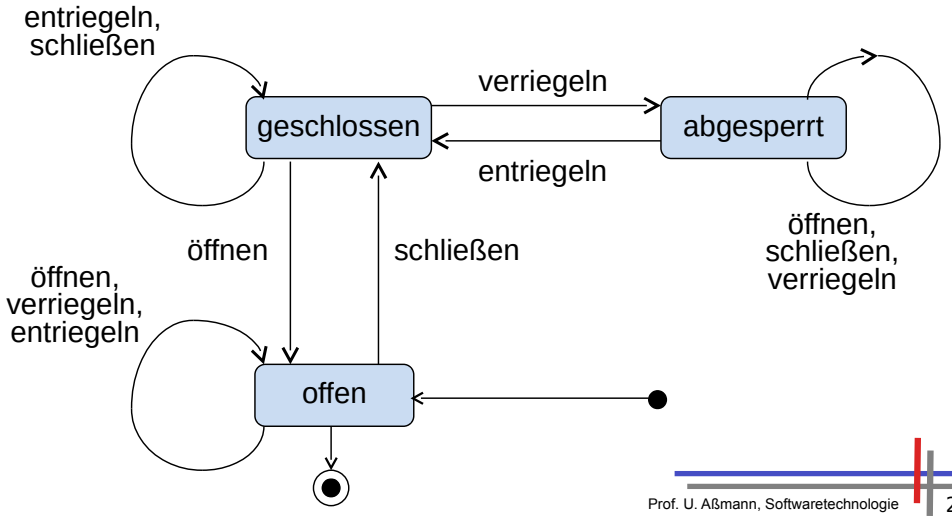
- Ein **Zustandsmodell (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model)** ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die das Verhalten eines Objekts beschreibt
  - Sie beschreibt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

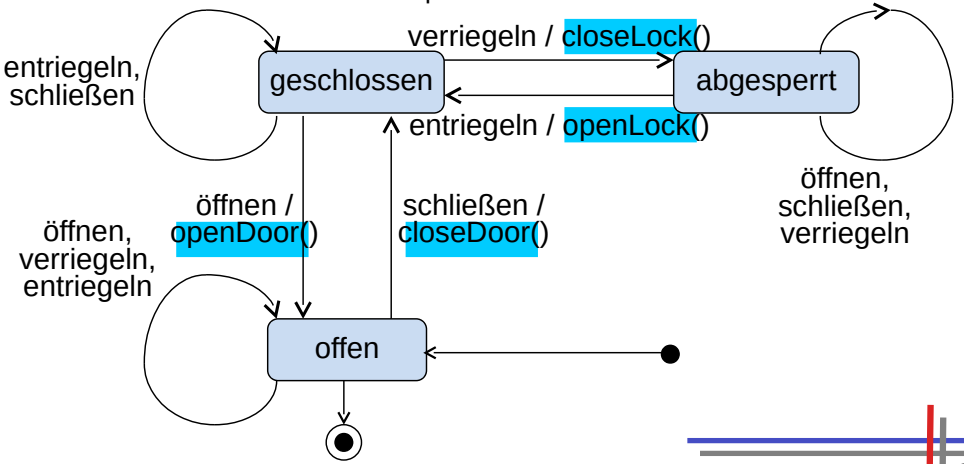
Beispiel: Protokollmaschine für eine Tür

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



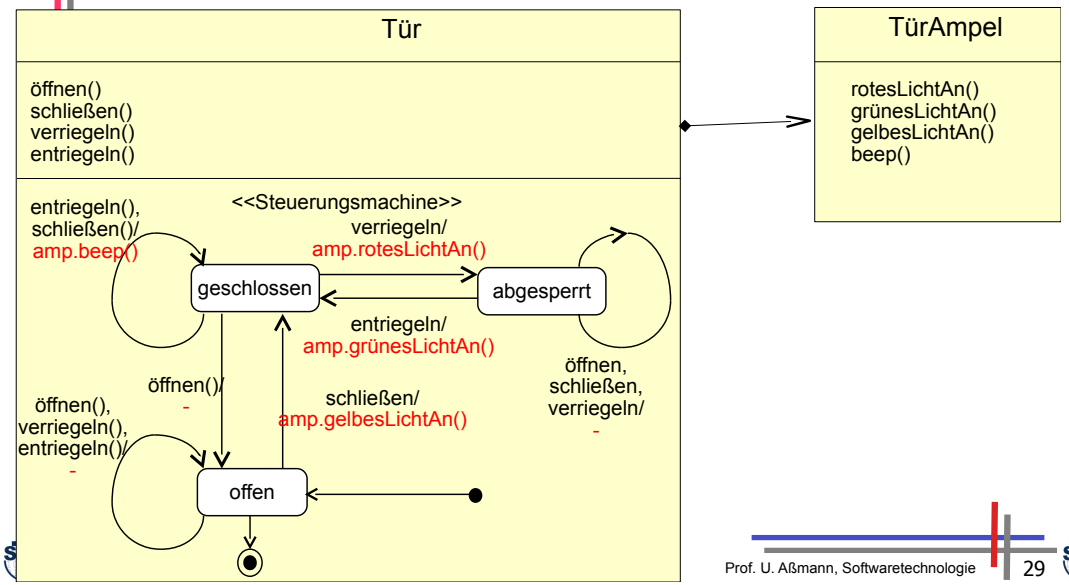
Beispiel: Steuerungsmaschine für eine Tür einer Behindertentoilette

- 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!



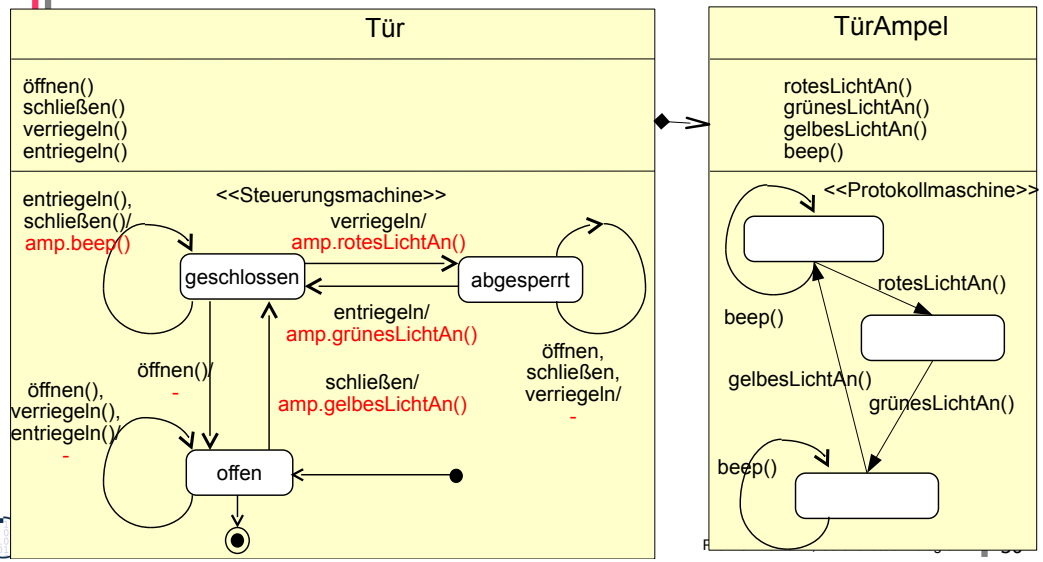
Objektlebenszyklus von innen und aussen

- 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 Einsatzzwecke von Zustandsdiagrammen

Zustandsdiagramme in der Analyse

- Einsatz von Zustandsmodellen:
  - Anwendungsfälle können mit Zustandsmodellen verfeinert werden (Szenarienanalyse)
    - die Aktion aus dem Anwendungsfall kann als Aktion einer Verhaltensmaschine aufgefasst werden
    - Achtung: dann ist die Verhaltensmaschine noch nicht einer Klasse zugeordnet
- Einsatz von Steuerungsmaschinen (Objektlebenszyklen)
  - Für komplexe Objekte kann eine Steuerungsmaschine angegeben werden
  - Auch für alle Unterobjekte des komplexen Objektes
- Einsatz von Protokollmaschinen
  - Zur Modellierung von Geschäftsprozessen in Geschäftssoftwaresystemen
  - Modellierung von Protokollen für Anschlüssen (ports) im Kontextmodell
  - Ihr Einsatz in der Szenarienanalyse ist nicht möglich:
    - Es ist nicht möglich, eine Aktion aus einem Anwendungsfall als Ereignis einer Protokollmaschine aufzufassen und damit Szenarienanalyse zu betreiben
    - da die Protokollmaschine nur Aufrufreihenfolgen beschreibt, aber keine Verfeinerungen von Aktionen zulässt

Zustandsdiagramme im Entwurf

- Zustandsmodelle (ohne Objektzuordnung) sind im Entwurf Objekten/Klassen zuzuordnen, da alle Zustandsdiagramme zu Objektlebenszyklen werden müssen
  - Aus den Zustandsmodellen entstehen also Steuerungsmaschinen
- Steuerungsmaschinen
  - können Verhalten, d.h., Implementierungen von beliebigen Klassen spezifizieren (*Objektlebenszyklen, white-box object life cycle)*
  - können als technische Steuerungsmaschinen Implementierungen von technischen Geräten beschreiben (*Gerätelebenszyklus*)
- Protokollmaschinen
  - können gültige Aufrufreihenfolgen an Objekte beschreiben und zur Ableitung von Vertragsprüfern eingesetzt werden (*black-box object life cycle)*

Einsatzzwecke für Zustandsmodelle

	Anwendungsfall-Lebenszyklus	Objektlebenszyklus (OLC)	Steuerung (technischer Geräte)
Verhaltensmaschine (behavioral state machine)	Zustandsmodell: Szenario-Analyse: Verhalten von Objekten im Kontextmodell und Top-Level-Architektur	Steuerungsmaschine: Verhaltens-beschreibung in Analyse Entwurf Implementierung (white-box OLC)	Technische Steuerungsmaschine: Verhaltens-beschreibung in Analyse Entwurf Implementierung (white-box OLC)
Protokollmaschine (protocol state machine)	Zur Darstellung von Geschäftsprozessen	Vertragsprüfung in Analyse Entwurf Implementierung (black-box OLC)	<< nicht möglich >>

Verwendung von UML-Zustandsmodellen

Verhaltens-Maschinen (Transduktor):

- Beschreiben das *Verhalten (Implementierung) eines Systems*
  - z. B. die *Steuerung* eines Systems der realen Welt, zum Steuern von Systemen, eingebettete Systeme etc.
  - Ereignisse sind Signale der Umgebung oder anderer Systemteile
- Reaktion in gegebenem Zustand auf ein bestimmtes Signal:
  - neuer Zustand
  - **ausgelöste Aktion** (wie im Zustandsmodell spezifiziert)
- Zustandsmodelle definieren die *Reaktion* des gesteuerten Systems auf mögliche Ereignisse, d.h. geben eine Implementierung an

Protokoll-Maschinen (Akzeptoren, Prüfmaschinen):

- Zum Überprüfen der *korrekten Aufrufsreihenfolgen*, die ein Benutzer an ein System absetzt
  - Ereignisse sind eingeschränkt auf Operationsaufrufe, d.h. es werden nur Aufruf-Ereignisse berücksichtigt
- Reaktion in gegebenem Zustand auf bestimmten Aufruf:
  - neuer Zustand
  - **Keine Aktionen** im Zustandsmodell!
- Zustandsmodelle definieren zulässige *Reihenfolgen* von Aufrufen (Schnittstelle)
- Protokollmaschinen sind Vertragsprüfer (“checker”), d.h. *Prüfer*, ob das System bzw. das Objekt einem Zustandsmodell folgt

34.5 Implementierung von Steuerungsmaschinen

... gehört eigentlich zum Übergang vom Implementierungsmodell zur Implementierung, hier aber zur Verständlichkeit aufgenommen

Implementierung von Steuerungsmaschinen mit Implementierungsmuster *IntegerState*

- Entwurfsmuster *IntegerState*
  - Zustand wird als Integer-Variable repräsentiert, Bereich [1..n]
  - Alle Ereignisse werden zu Methoden (die von aussen aufgerufen werden)
    - Externe Ereignisse werden mit “Reaktions-Methoden” modelliert
    - Interne Ereignisse Methoden zugeordnet
- Methoden schalten 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)

"f nal" 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;


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

"f nal" bei Methoden : nicht überschreibbar

IntegerState
Beispiel: Code zur Steuerung einer Tür (2)

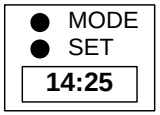
```
    public void schliessen() {
      // Fallanalyse
      switch (zustand) {
            case Z_offen:
                  zustand = Z_geschlossen;
                  System.out.println("Klick");
                  break;
            case Z_geschlossen:
                  break;
            case Z_abgesperrt: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

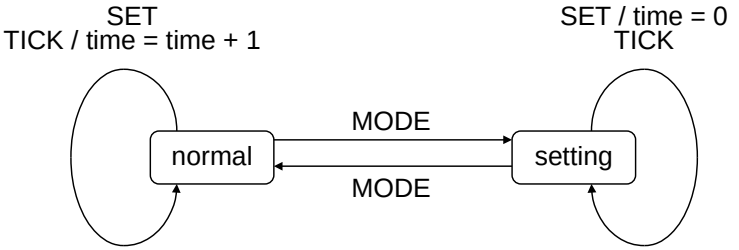
IntegerState
Beispiel: Code zur Steuerung einer Tür (3)

```
    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)



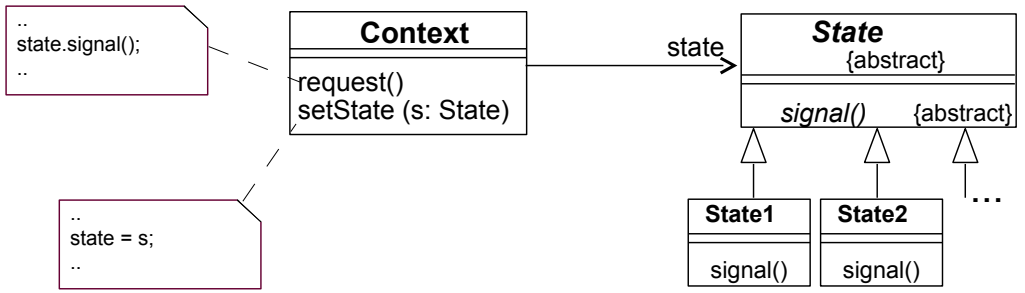


Implementierung mit IntegerState

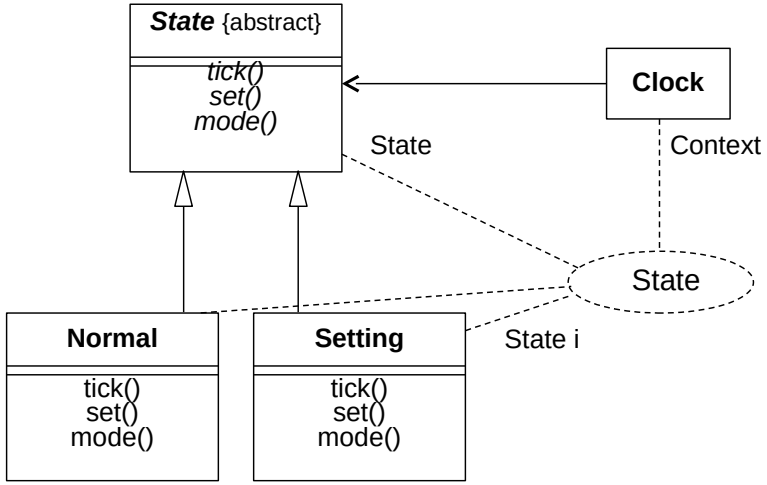
```
class Clock {

  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()
}
```


Implementierungsmuster State

- Problem: Was, wenn der Zustand Informationen (Attribute) enthält?
- Lösung: Darstellung des Zustands durch *Zustandsobjekt*
  - Weiterschalten von Zuständen durch Auswechseln des Zustandsobjekts (Polymorphie)



State-Beispiel für Uhr (1)



State-Beispiel für Uhr in Java (2)

```
abstract class State {
  abstract void tick();
  abstract void set();
  abstract void mode();
}
```

```
class Normal extends State {
  void tick() {
    clock.time++;
    setChanged();
  }
  void set() {}
  void mode() {
    clock.setState
      (new Setting());
  }
}
```

```
class Setting extends State {
  void tick() {}
  void set() {
    clock.time = 0;
    setChanged();
  }
  void mode() {
   clock.setState
      (new Normal());
  }
}
```


State

Variante mit geschachtelten Klassen *(inner classes)*

```
class Clock {
  private int time = 0;
  private State normal = new Normal();
  private State setting = new Setting();
  private State s = normal;
  abstract class State {...}
  class Normal extends State {
      void tick() {...}
      void set() {}
      void mode() { s = setting; }
  }
  class Setting extends State {
      ... analog
  }
  public void tick () {
      s.tick();
  }
  ... set(), mode() analog
}
```

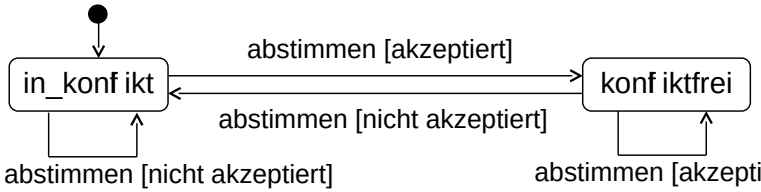

Steuerungsmaschinen: Zusammenfassung

- Anwendungsgebiet:
  - white-box-Objektlebenszyklen
  - Gerätesteuerungen
    - Mikrowelle, Stoppuhr, Thermostat, ...
    - Große Bedeutung z.B. in Automobil- und Luftfahrtindustrie
    - Problem: Verhalten des gesteuerten Geräts muss *regulär* sein, d.h. die Zustandsmenge muss einer reguläre Sprache entsprechen
- Codegenerierung möglich mit State und IntegerState
  - bei genau definierter Aktionssprache (Aus den Aktionen muss Code generiert werden). Werkzeuge existieren
- Praktische Aspekte:
  - Kommunikation: Nachrichten empfangen/versenden
  - Nebenläufigkeit
  - Reaktivität (Akzeptieren von Nachrichten zu beliebigem Zeitpunkt)
  - Realzeitaspekte

34.6 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

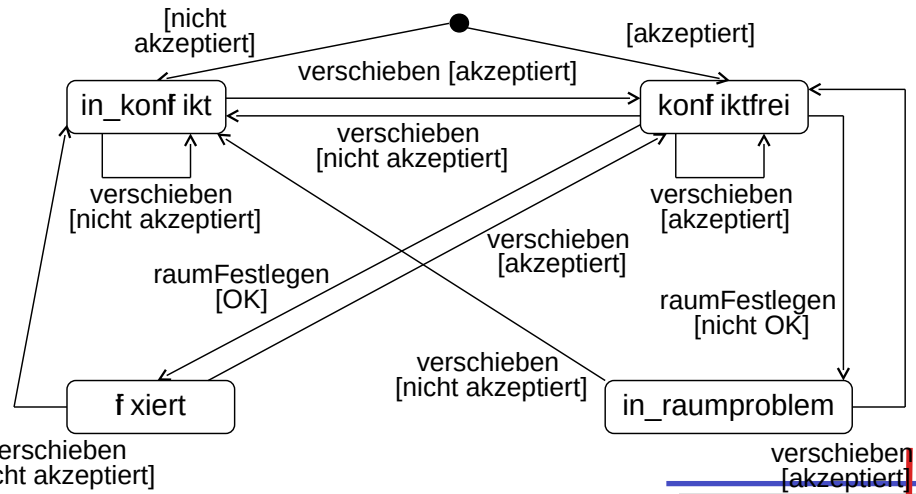
34.7 Vereinfachung von Zustandsdiagrammen durch Strukturierung

Unterspezifikation und Vervollständigung von Übergängen

- Was passiert, wenn ***kein*** Übergang im aktuellen Zustand für das aktuelle Ereignis angegeben ist?
- Möglichkeiten:
  - Unzulässig
    - Fehlermeldung (Fehlerzustand)
    - Ausnahmebehandlung
  - Zustand unverändert (impliziter "Schleifen"-Übergang)
  - Warteschlange für Ereignisse
  - Unterspezifikation ("wird später festgelegt")
- Achtung: Ein vollständiges Zustandsmodell (totale Übergangsfunktion) ist meist sehr umfangreich und unübersichtlich!

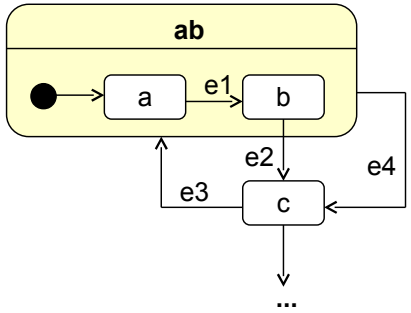
Black-Box Objektlebenszyklus (Protokollmaschine)

- Zulässige Zustände von Objekten der Klasse "Teambesprechung":
  - Merke: nur zur Generierung eines Vertragsprüfers einsetzbar, nicht zu einer vollständigen Implementierung

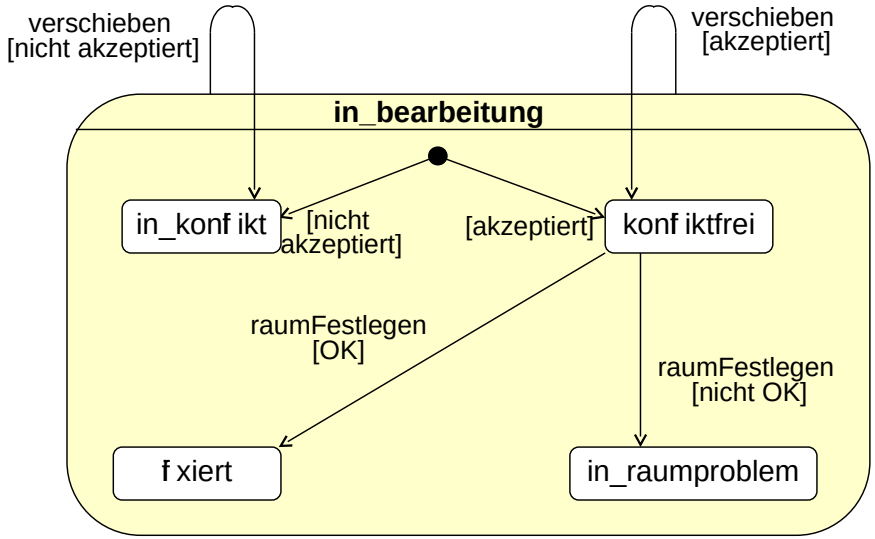


Ober- und Unterzustände

- Zur Vereinfachung, insbesondere, um eine ganze Gruppe von Zuständen einheitlich zu behandeln, können ***Oberzustände*** eingeführt werden.
  - Ein Zustand in den Oberzustand ist ein Übergang in den Startzustand des enthaltenen Zustandsdiagramms.
  - Ein Zustand aus dem Oberzustand gilt für alle Zustände des enthaltenen Zustandsdiagramms (*Vererbung* von Übergangsverhalten).

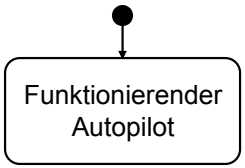


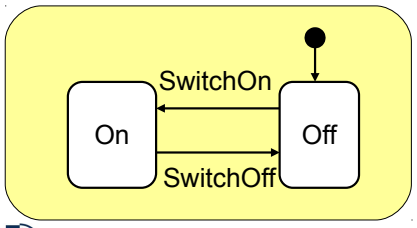
Zustandshierarchie Teambesprechung (jetzt einfacher notiert)

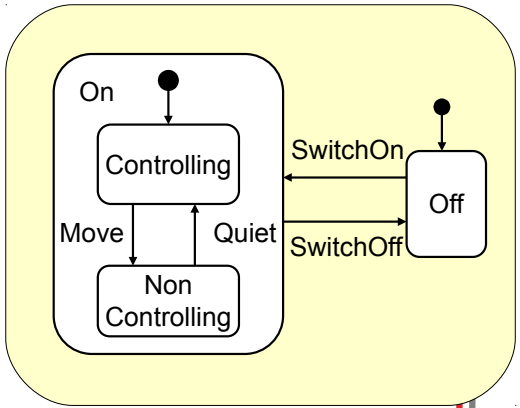


Warum kann man ein hierarchisches Zustandsdiagramm einfach verstehen?

- Es ist kein flacher Automat, sondern ein *hierarchisch gegliedeter,* der in einen einzigen Oberzustand gefaltet werden kann (*reduzibel)*

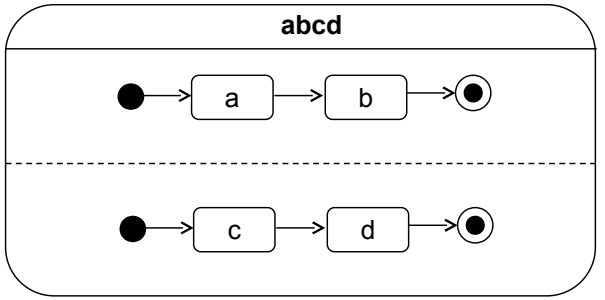






Nebenläufige Teilzustände

- Um voneinander zeitlich unabhängige Vorgänge einfach darzustellen, kann ein Zustand in nebenläufige Teilbereiche zerlegt werden (getrennte "Schwimmbahnen").
  - Ein Zustand des Oberzustands ist ein Tupel von Zuständen der Teilbereiche (Schwimmbahnen).



Interne Übergänge

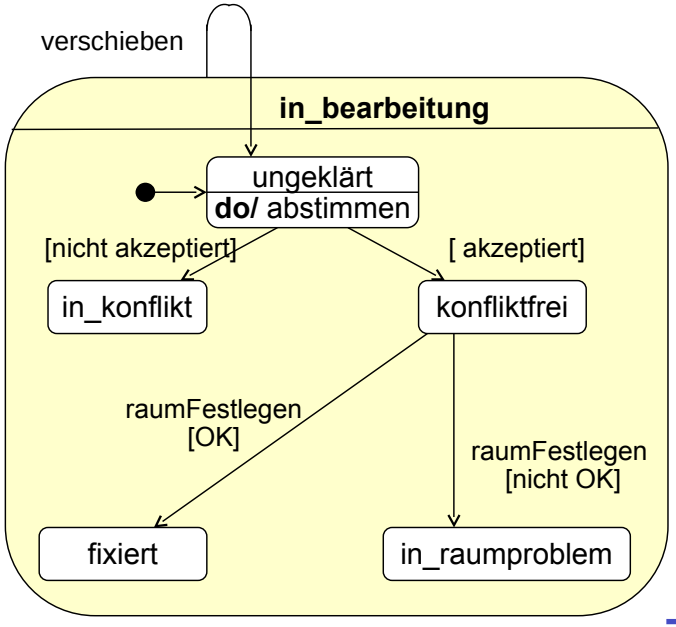
- **Definition:** Ein ***interner Übergang*** eines Zustands *S* beschreibt einen Übergang, der stattfindet, während das Objekt im Zustand *S* ist.
- Es gibt folgende Fälle von internen Übergängen:
  - Eintrittsübergang *(entry transition)*
  - Austrittsübergang *(exit transition)*
  - Fortlaufende Aktivität *(do transition)*
  - Unterdiagrammaufruf *(include transition)*
  - Reaktion auf benanntes Ereignis
- **Notation:**

Zustand
label */ interner Übergang*

label = **entry**, **exit**, **do**, **include** oder *Ereignisname*

Achtung: UML 1.3 !

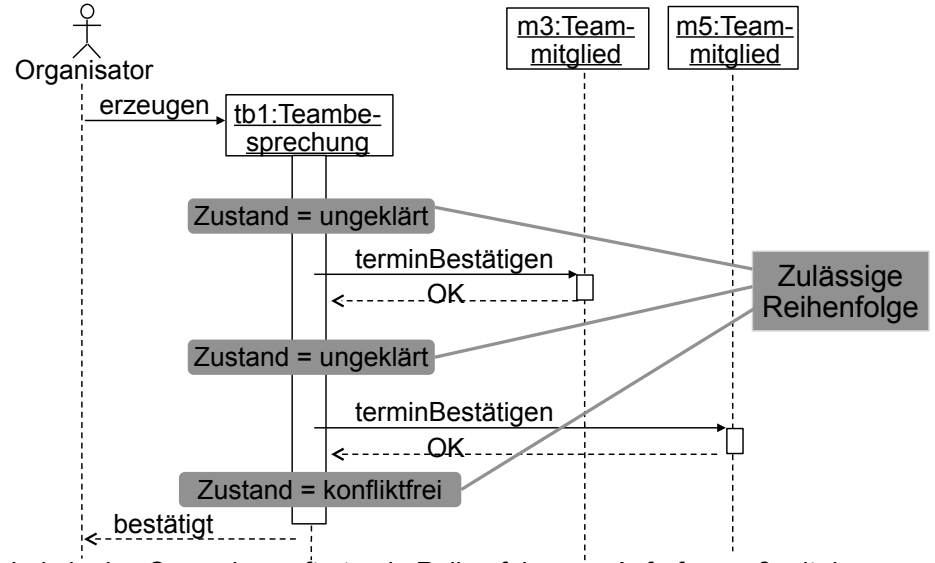
Verschiedene Aktivitäten



Aktivität "abstimmen" = Abstimmung mit den Teammitgliedern per Operation "terminBestätigen"

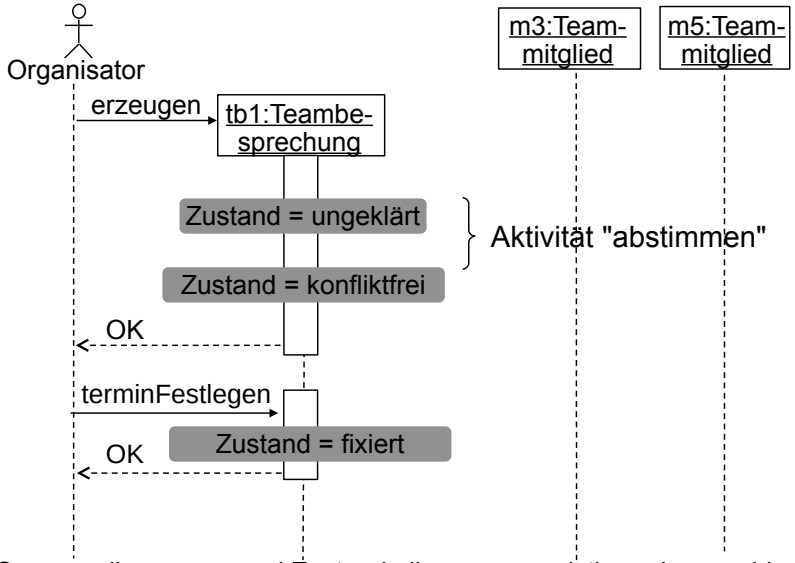
(Modellierbar als Unterdiagramm *UD*, d.h. **include** *UD* anstelle von **do**)

Zusammenhang: Zustandsdiagramm - Sequenzdiagramm (1)



Jede in den Szenarien auftretende Reihenfolge von Aufrufen muß mit dem Zustandsmodell verträglich sein.

Zusammenhang: Zustandsdiagramm – Sequenzdiagramm (2)



Sequenzdiagramme und Zustandsdiagramme existieren in verschiedenen Abstraktionsstufen.

Zustandsmodellierung: Zusammenfassung

Typische Anwendung:	Analysephase	Entwurfsphase
Zeitbezogene Anwendungen (Echtzeit, Embedded, safety-critical)	**Verhaltens- und Steuerungsmaschinen**: Skizzen der Steuerung für Teilsysteme und Gesamtsystem	Detaillierte Angaben zur Implem., automatische Codegenerierung, Verifikation mit Model checking
Datenbezogene Anwendungen (Informations-systeme, DB-Anwendungen)	**Protokollmaschinen**: Lebenszyklen für zentrale Geschäftsobjekte, Geschäftsprozesse	Genaue Spezifikation von Aufrufreihenfolge (Vertragsprüfung)

The End

- Many slides courtesy to © Prof. Dr. Heinrich Hussmann, 2003. Used by permission.