Zustandsdiagramme:
Modellierung und Implementation
OOPM, Ralf Lämmel

UML: Unified Modeling Language

- ★ *UML ist eine Modellierungssprache.*
- ★ *UML dient der Anforderungsbeschreibung*
  - für Organisationssysteme und
  - Softwaresysteme.
- ★ *UML ermöglicht die Beschreibung*
  - von Struktur
  - und Verhalten (Abläufe).

UML: Unified Modeling Language

- ★ *Benutzer*
  - Softwareentwickler
  - **Systemanalytiker**
  - **Systemarchitekten**
- ★ *Anwendungsbereiche*
  - Informatik
  - Prozessmanagement
  - Informationsmanagement

UML: Unified Modeling Language

★ *Wichtige Ziele des UML-Sprachentwurfs:*

- Höhere Abstraktionsform
- Verständlichkeit
  - durch die Verwendung der Diagrammform
  - Lesbarkeit für diese Zielgruppen:
    - **Manager**
    - **Benutzer**
- Ausführbarkeit (z.B. Code-Generierung)

UML: Unified Modeling Language

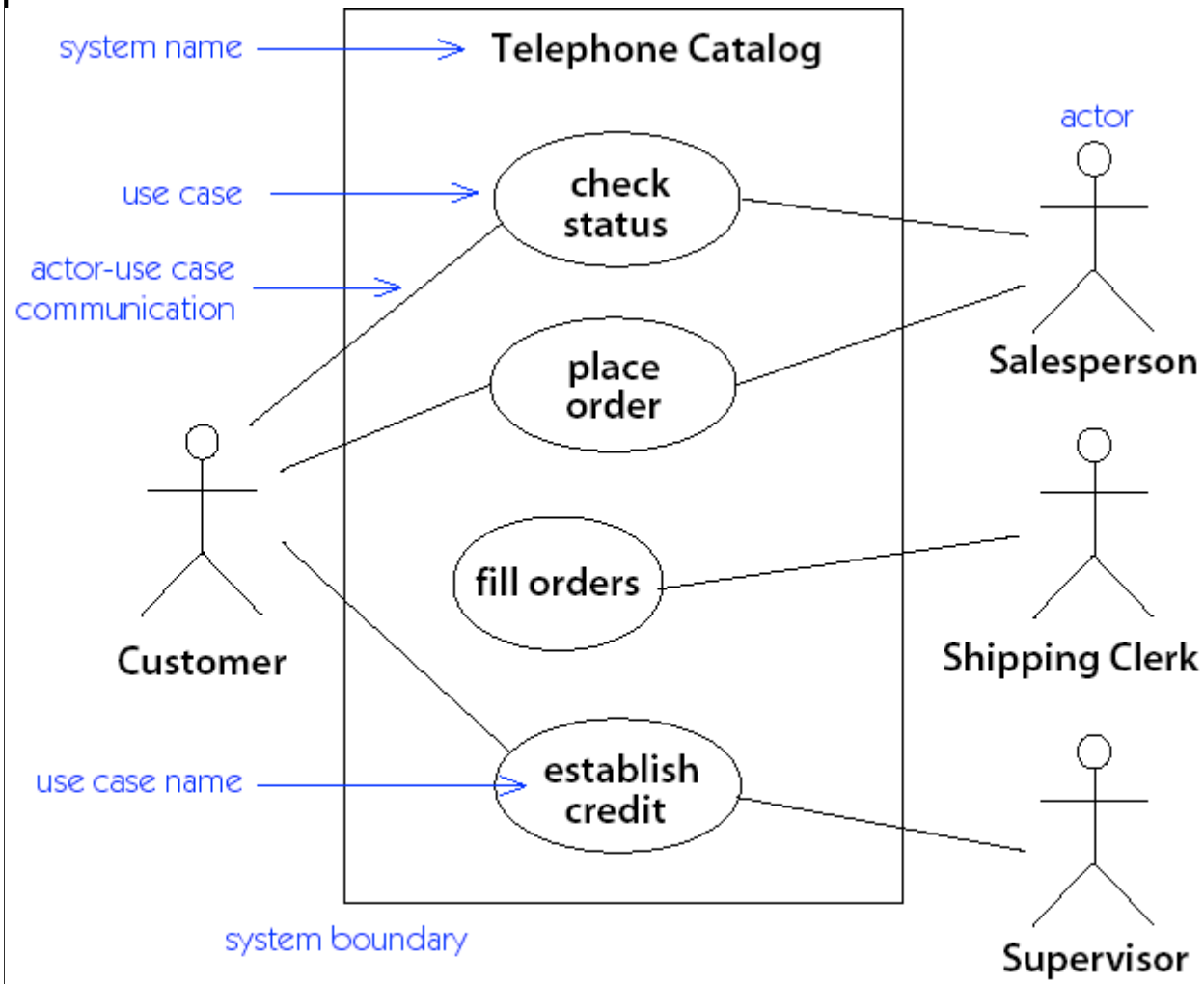
- **Strukturdiagramme**
  - ***Klassendiagramme***
  - Kompositionsstrukturdiagramme
  - Komponentendiagramme
  - **Objektdiagramme**
  - Paketdiagramme
  - Verteilungsdiagramme
- **Verhaltensdiagramme**
  - **Aktivitätsdiagramme**
  - **Anwendungsfalldiagramme**
  - Interaktionsübersichtsdiagramme
  - Kommunikationsdiagramme
  - **Sequenzdiagramme**
  - Zeitverlaufsdiagramme
  - ***Zustandsdiagramme***

UML: Unified Modeling Language

- **Strukturdiagramme**
  - ***Klassendiagramme***
  - Kompositionsstrukturdiagramme
  - Komponentendiagramme
  - **Objektdiagramme**
  - Paketdiagramme
  - Verteilungsdiagramme
- **Verhaltensdiagramme**
  - **Aktivitätsdiagramme**
  - **Anwendungsfalldiagramme**
  - Interaktionsübersichtsdiagramme
  - Kommunikationsdiagramme
  - **Sequenzdiagramme**
  - Zeitverlaufsdiagramme
  - ***Zustandsdiagramme***

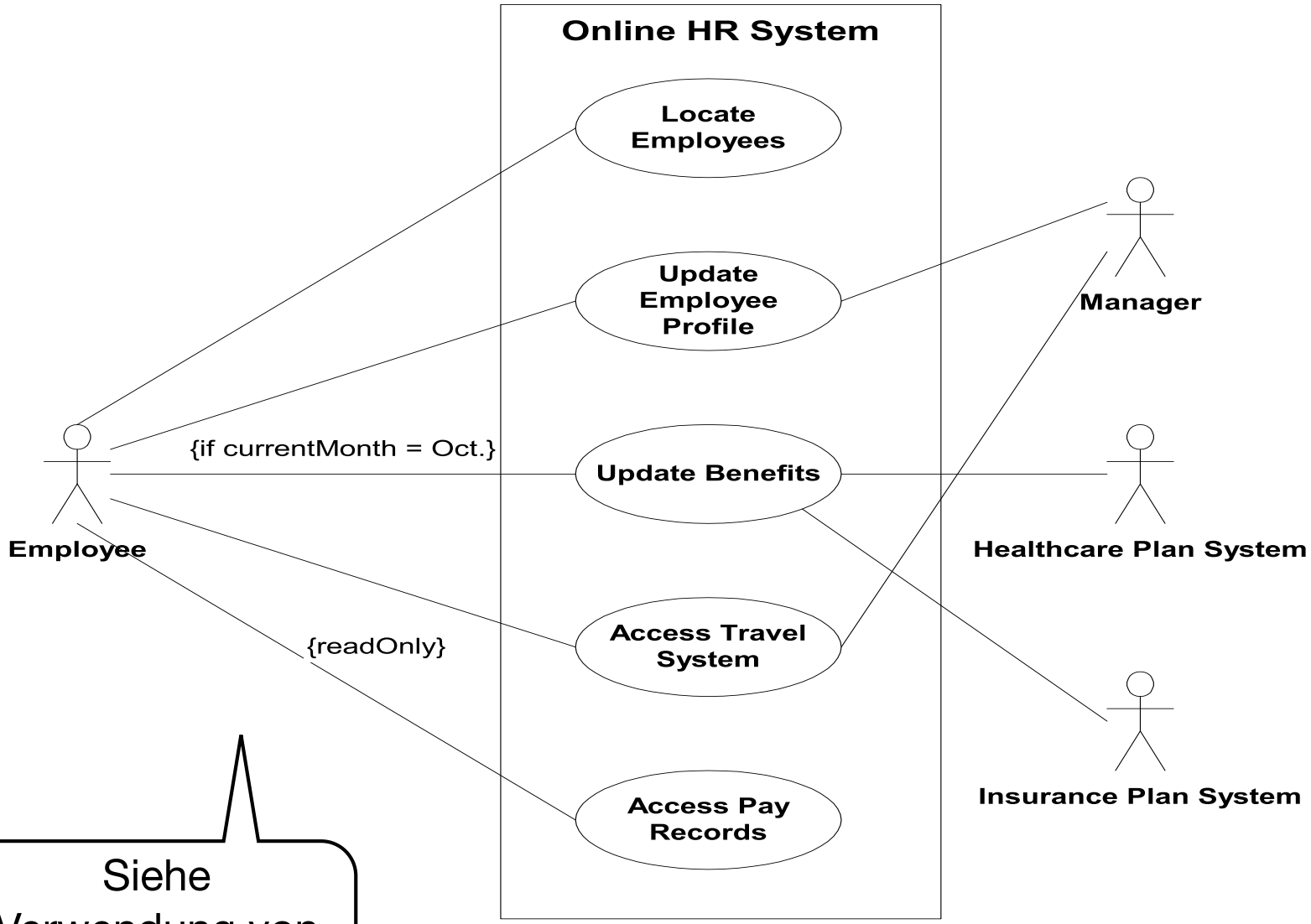
Basis der Modellierung im Allgemeinen.
Basis der Verhaltensmodellierung im Speziellen.

Beispiel eines Anwendungsfalldiagramms



Ein Modell von Anwendungsfällen ist eine Sicht auf das System welches das Systemverhalten aus externer Sicht (oder aus Nutzersicht) betont.

Weiteres Beispiel



Siehe Verwendung von Constraints

(C) OMG TUTORIAL SERIES

Modellierungsphilosophien

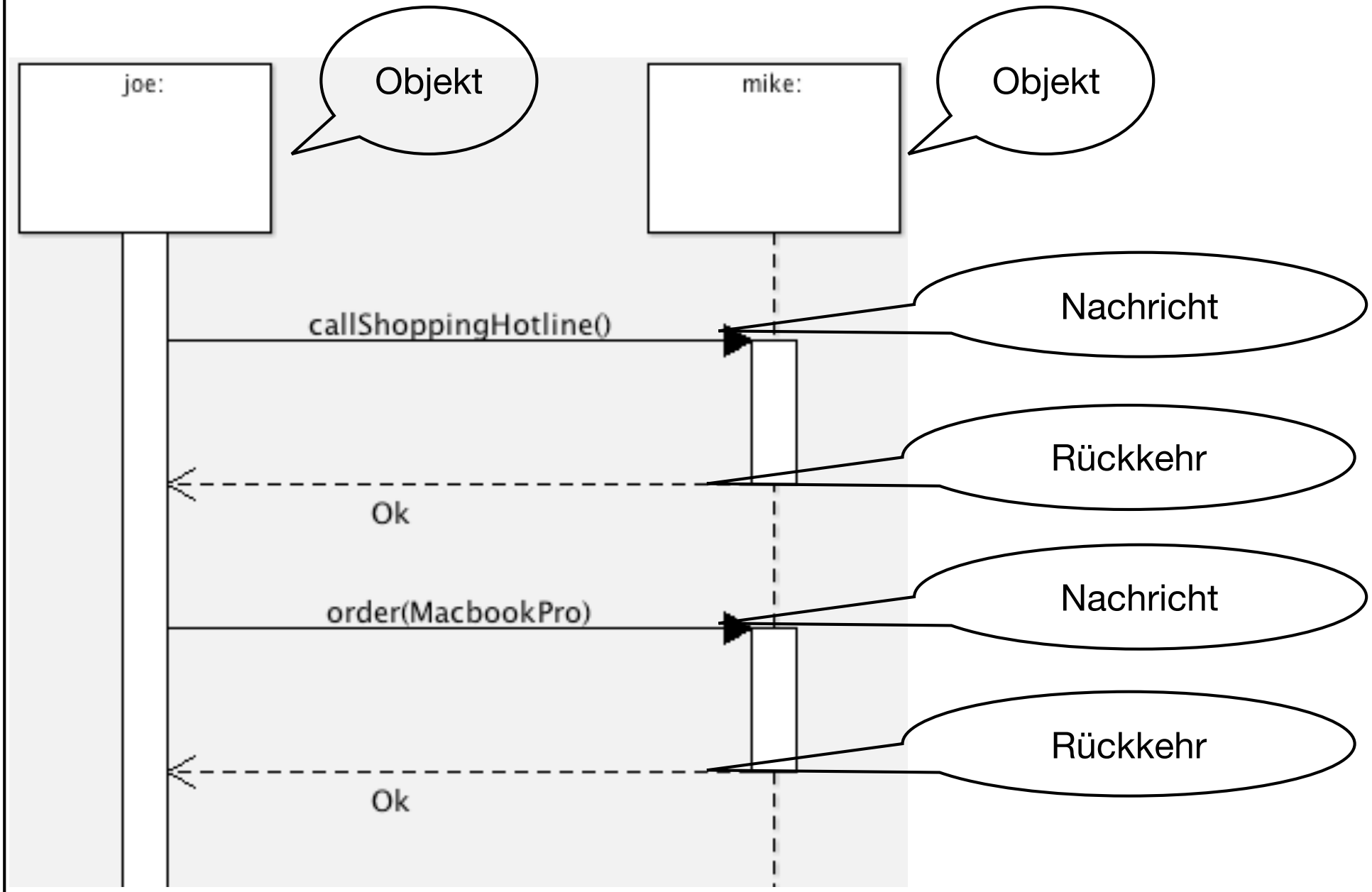
- **Anwendungsfälle** zuerst
  - Beginne mit Anwendungsfällen
  - Leite Struktur- und Verhaltensmodelle ab
- Nicht Anwendungsfall-getrieben
  - Regelmäßige Konsistenzkontrolle zwischen
    - Anwendungsfallmodellen und
    - allen anderen Modellen

UML: Unified Modeling Language

- **Strukturdiagramme**
  - ***Klassendiagramme***
  - Kompositionsstrukturdiagramme
  - Komponentendiagramme
  - **Objektdiagramme**
  - Paketdiagramme
  - Verteilungsdiagramme
- **Verhaltensdiagramme**
  - **Aktivitätsdiagramme**
  - **Anwendungsfalldiagramme**
  - Interaktionsübersichtsdiagramme
  - Kommunikationsdiagramme
  - **Sequenzdiagramme**
  - Zeitverlaufsdiagramme
  - ***Zustandsdiagramme***

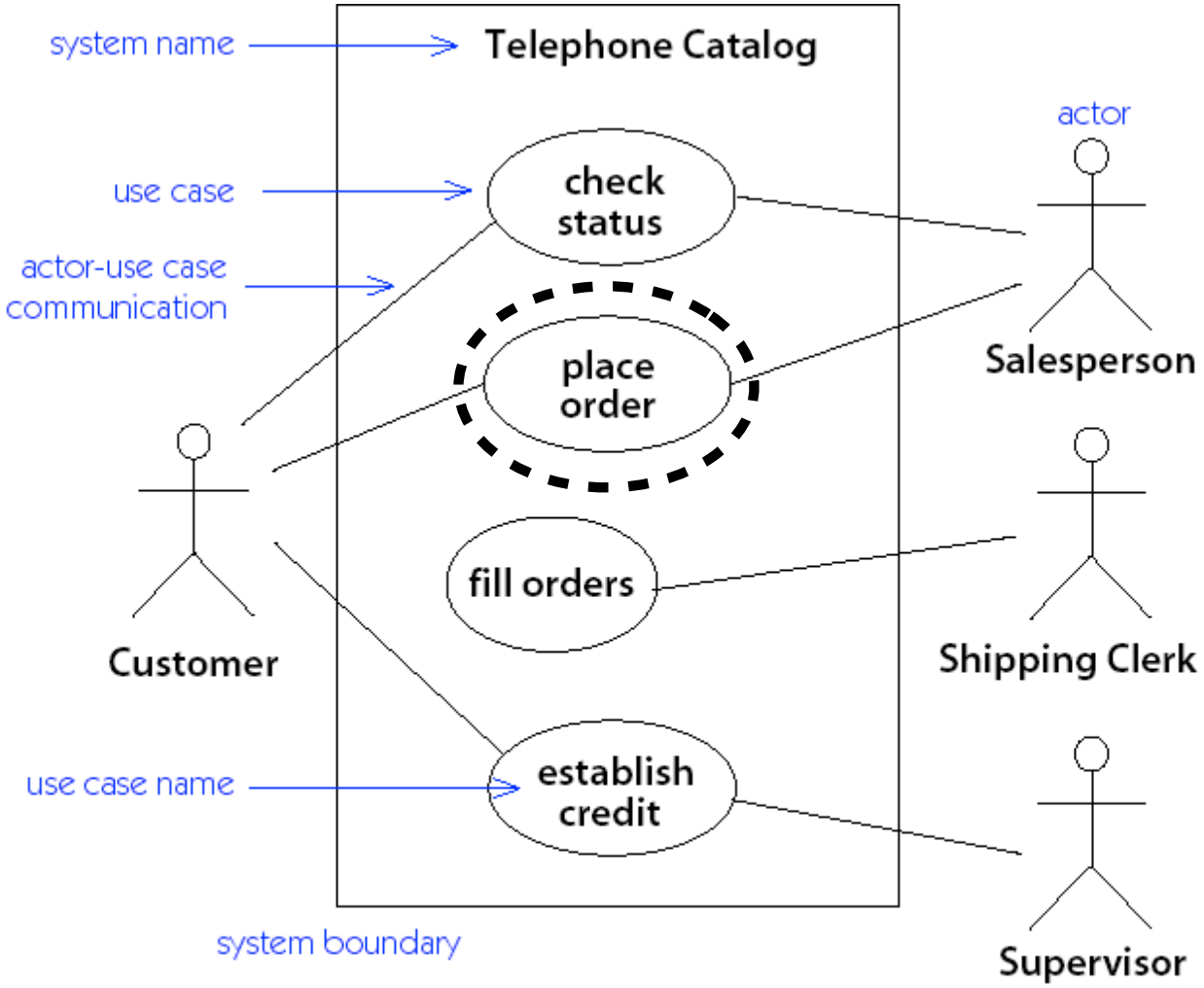
Zur Darstellung von Szenarien = Instanzen eines Anwendungsfalles im Sinne eines typischen Beispiels seiner Ausführung

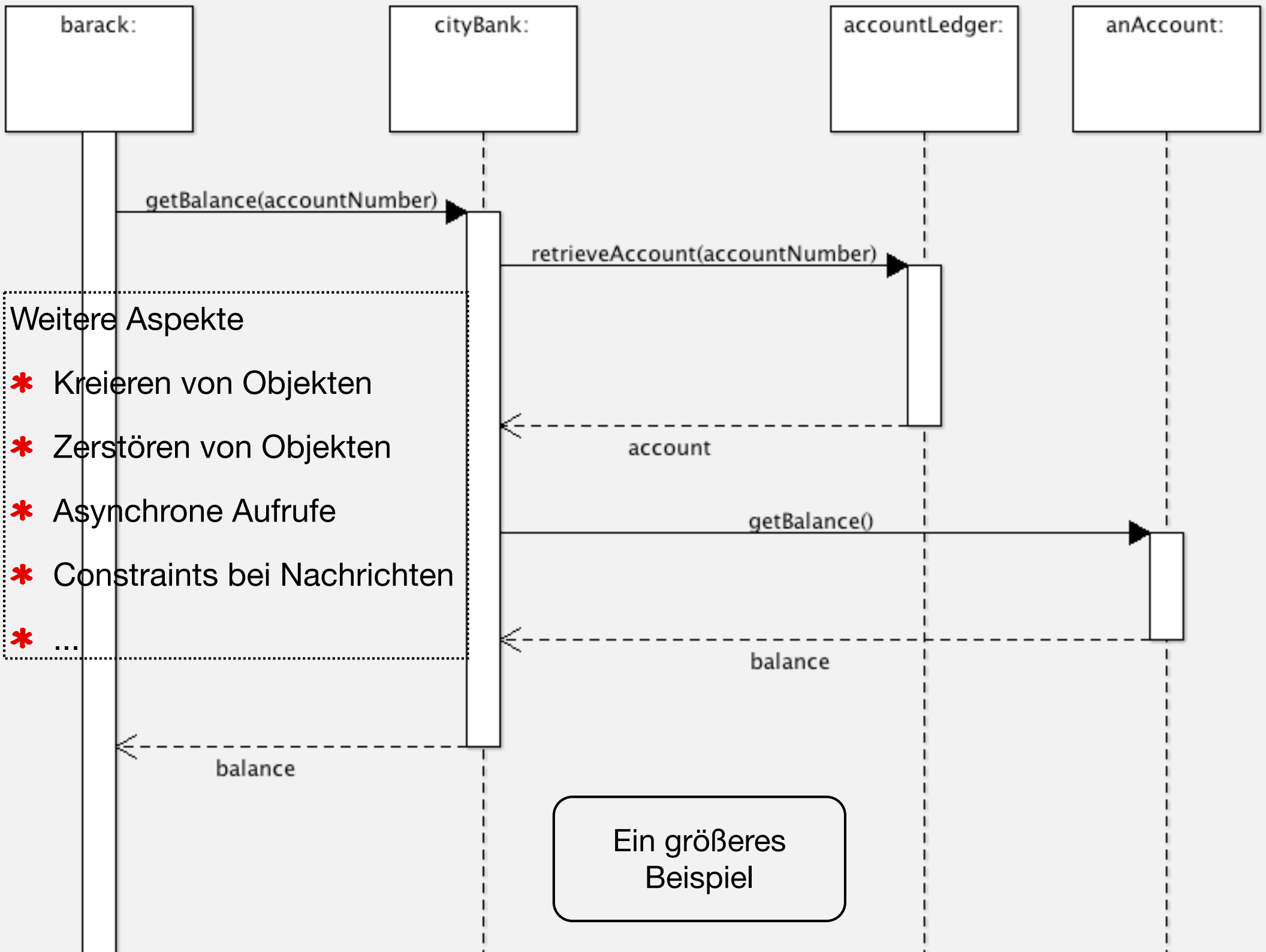
Ein Sequenzdiagramm für den Kauf am Telefon



Merke: Instanzen (Objekte) müssen nicht zwangsläufig zu implementierten Objekten korrespondieren. Es können z.B. auch Systemnutzer sein.

Zum Vergleich: Ein Anwendungsfalldiagramm



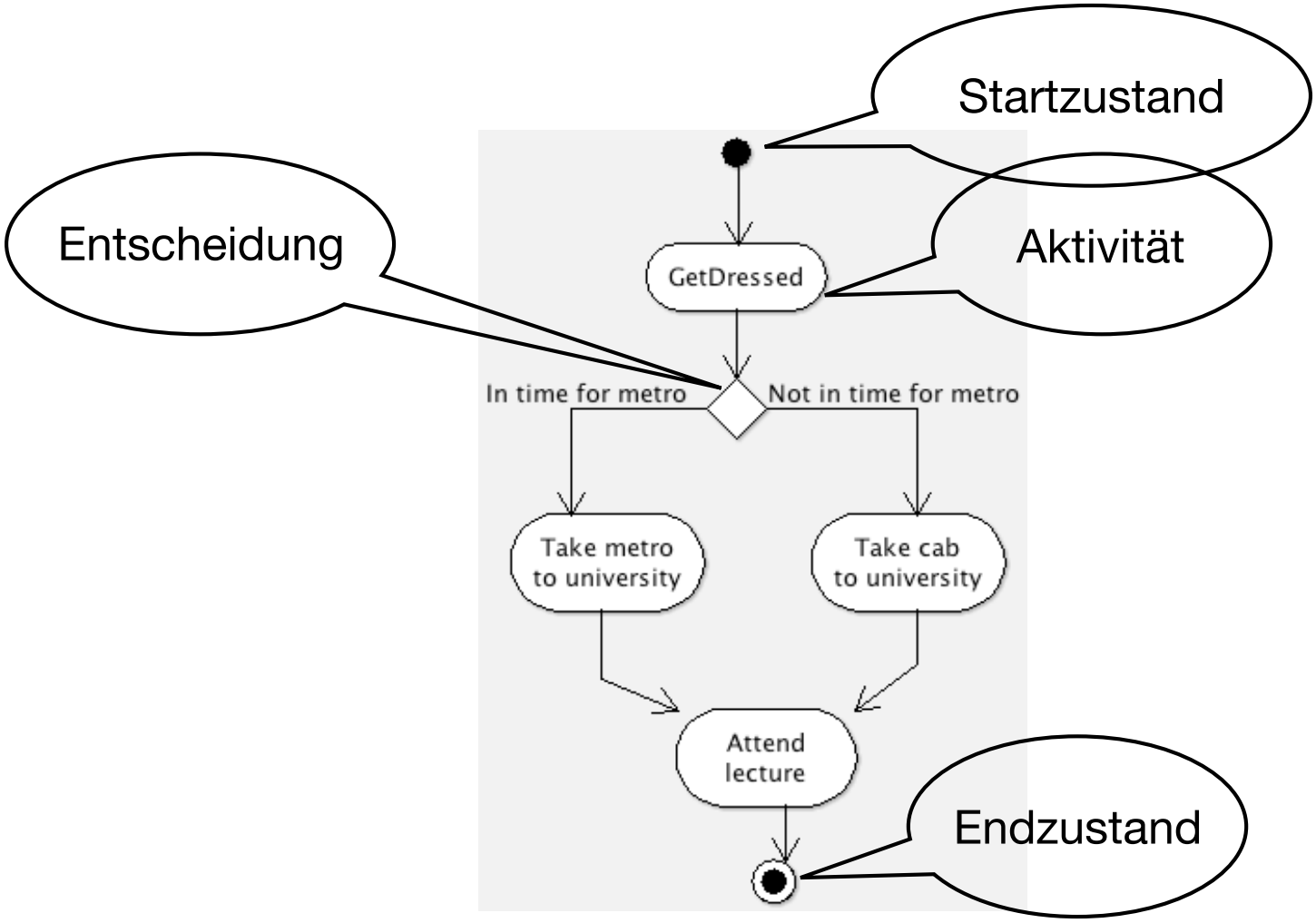

barack:
cityBank:
accountLedger:
anAccount:
getBalance(accountNumber)
retrieveAccount(accountNumber)
account
getBalance()
balance
balance
Weitere Aspekte
* Kreieren von Objekten
* Zerstören von Objekten
* Asynchrone Aufrufe
* Constraints bei Nachrichten
* ...
Ein größeres Beispiel

UML: Unified Modeling Language

- ★ **Strukturdiagramme**
  - ❋ ***Klassendiagramme***
  - ❋ Kompositionsstrukturdiagramme
  - ❋ Komponentendiagramme
  - ❋ **Objektdiagramme**
  - ❋ Paketdiagramme
  - ❋ Verteilungsdiagramme
- ★ **Verhaltensdiagramme**
  - ❋ **Aktivitätsdiagramme**
  - ❋ **Anwendungsfalldiagramme**
  - ❋ Interaktionsübersichtsdiagramme
  - ❋ Kommunikationsdiagramme
  - ❋ **Sequenzdiagramme**
  - ❋ Zeitverlaufsdiagramme
  - ❋ ***Zustandsdiagramme***

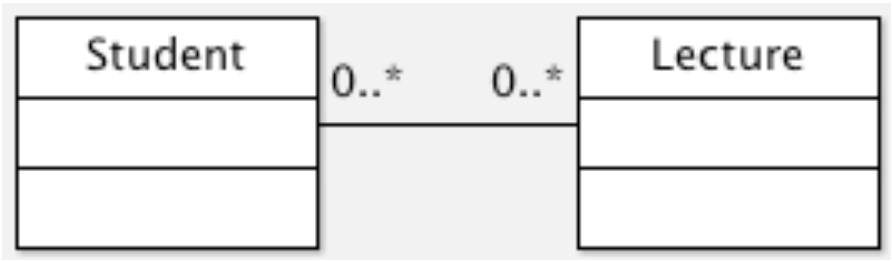
Verallgemeinerte Programmflusspläne

Ein Aktivitätsdiagramm zum Besuch einer Vorlesung

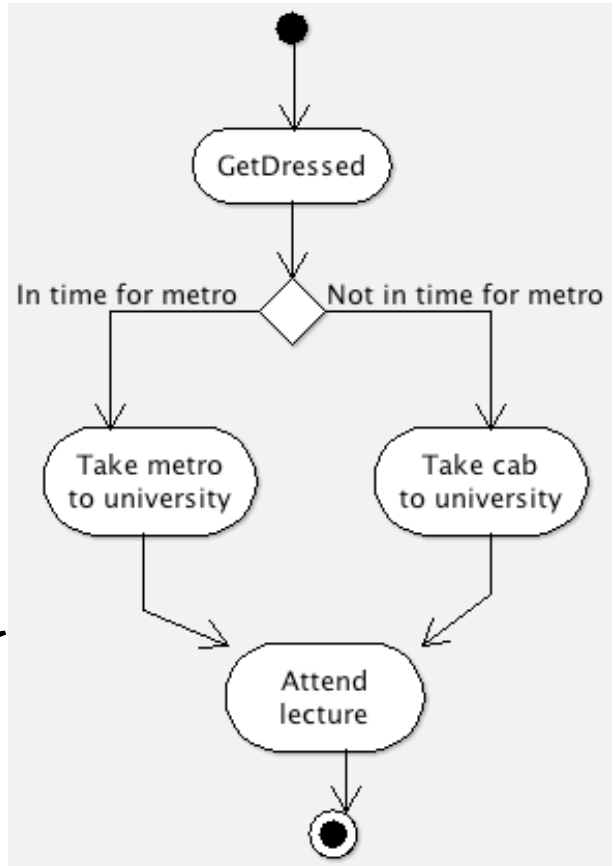


Strukturelle Modellierung vs. ***Verhaltens***modellierung

Klassendiagramm für die Modellierung einer Universität

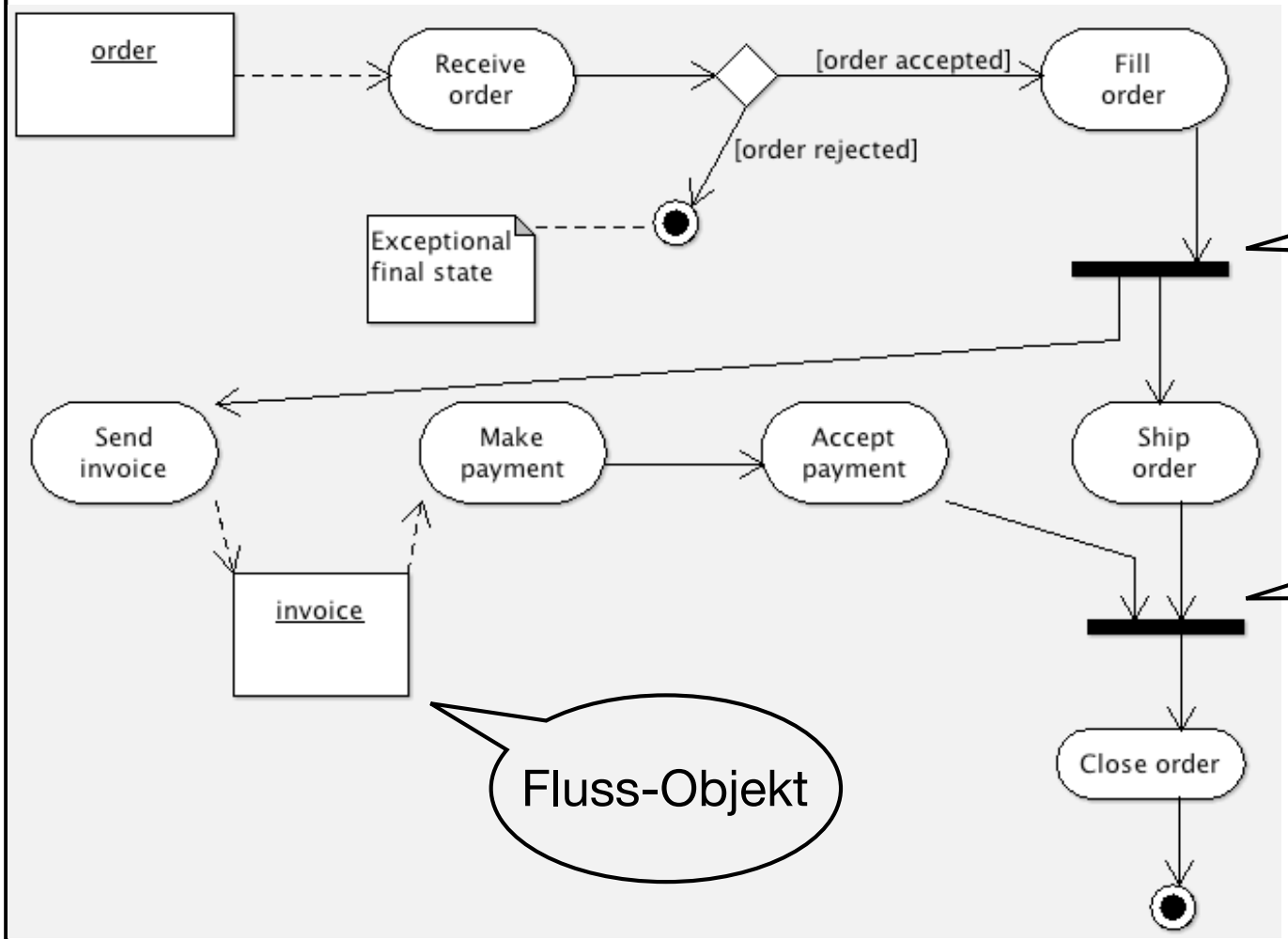


Aktivitätsdiagramm für die Modellierung eines Vorlesungsbesuches



Parallelität und Datenfluss am Beispiel

- Weitere (nicht illustrierte) Ausdrucksmittel
  - Unterbrechungen
  - Signale (Eingabe/Ausgabe)
  - ...

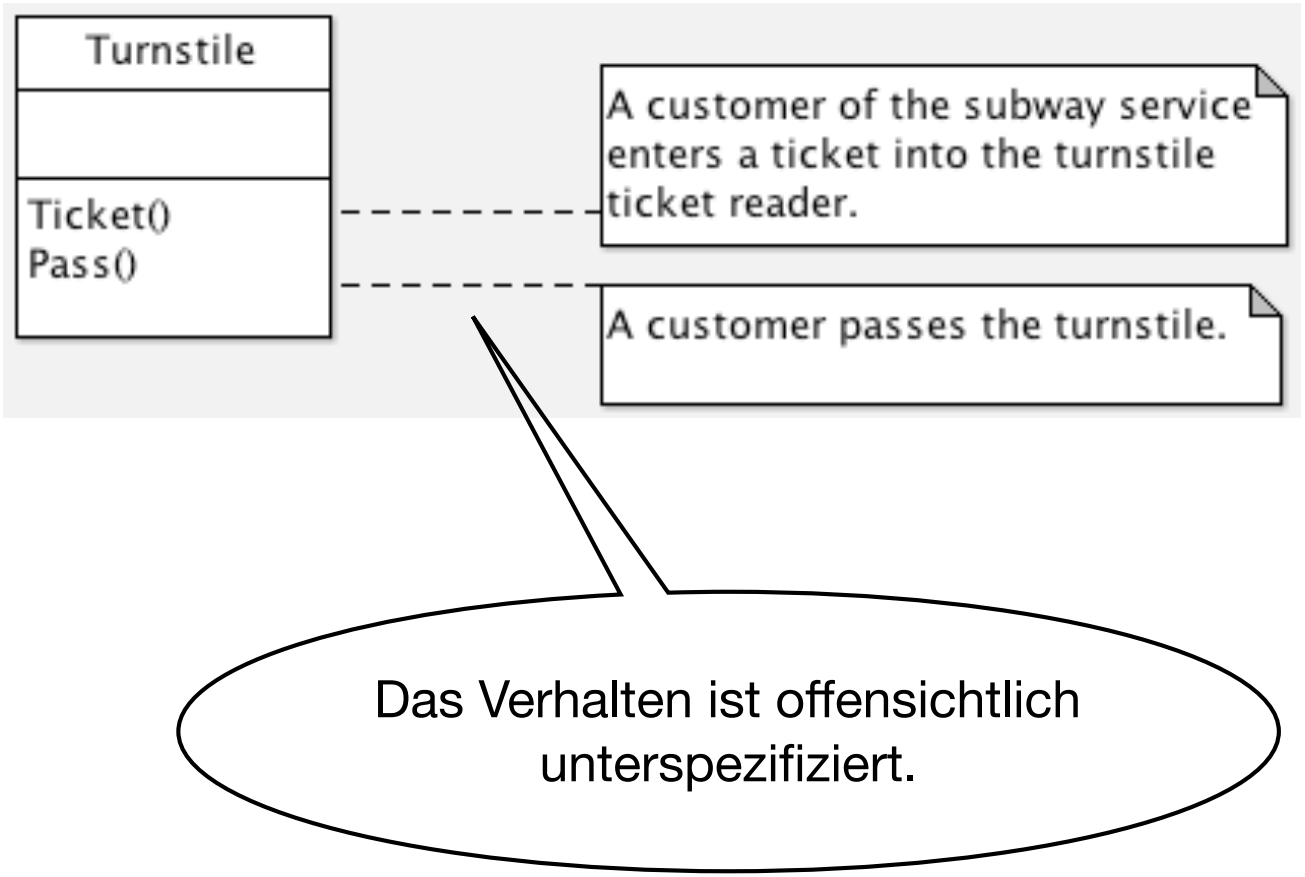


UML: Unified Modeling Language

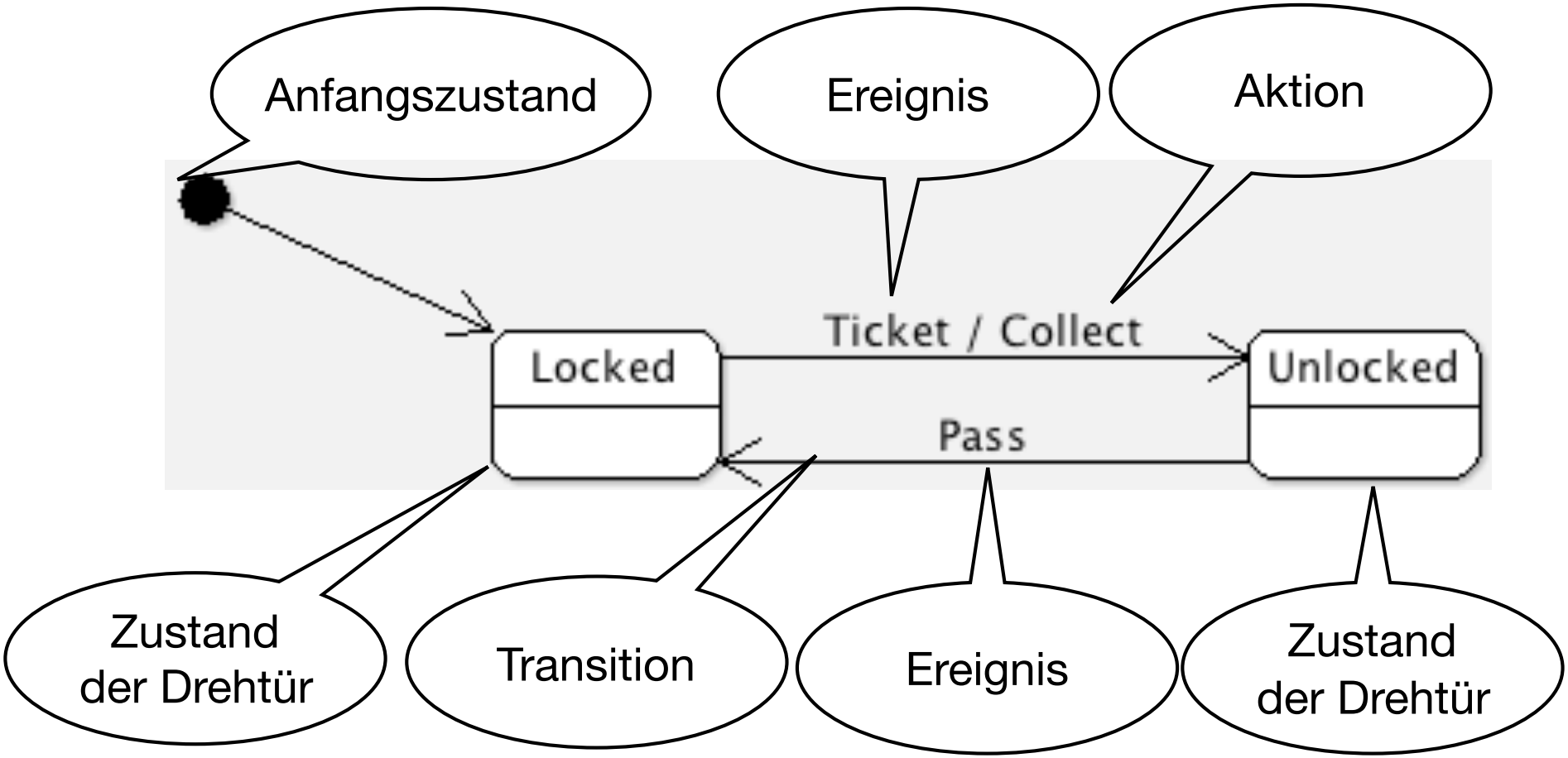
- ★ **Strukturdiagramme**
  - ***Klassendiagramme***
  - Kompositionsstrukturdiagramme
  - Komponentendiagramme
  - **Objektdiagramme**
  - Paketdiagramme
  - Verteilungsdiagramme
- ★ **Verhaltensdiagramme**
  - **Aktivitätsdiagramme**
  - **Anwendungsfalldiagramme**
  - Interaktionsübersichtsdiagramme
  - Kommunikationsdiagramme
  - **Sequenzdiagramme**
  - Zeitverlaufsdiagramme
  - ***Zustandsdiagramme***

Zustandsänderung von Objekten bei Ereignissen

Modellierung der Struktur eines U-Bahn-Systems



Ein Zustandsdiagramm für die Drehtür



Eigenschaften von Zustandsdiagrammen

* ***Vollständigkeit***
  - Ist das Verhalten für jedes mögliche Ereignis in jedem möglichen Zustand definiert?

Im Beispiel: Nein!

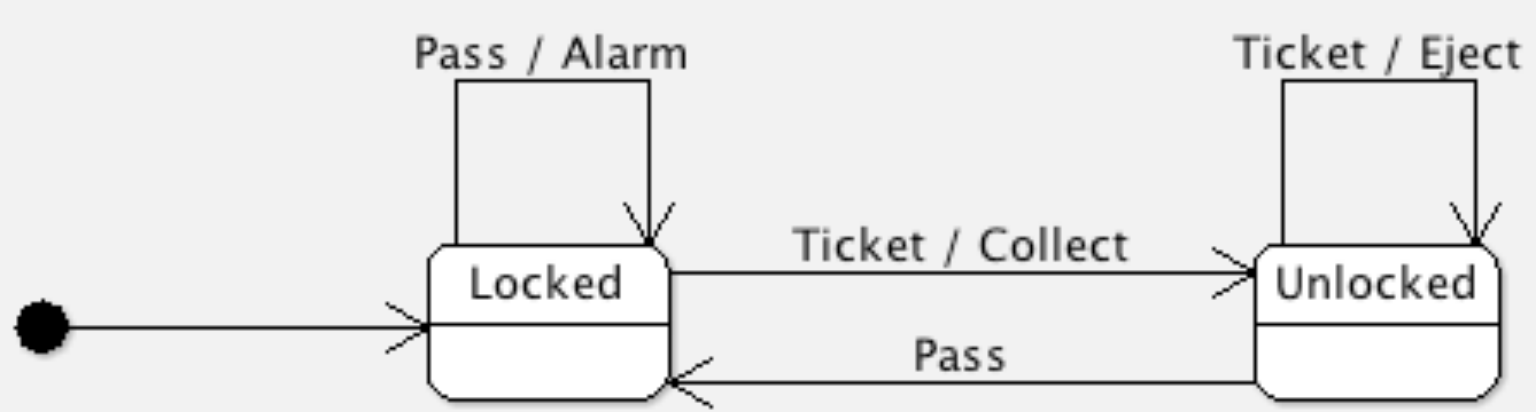
* ***Terminierung***
  - Gibt es Folgen von Zustandsübergängen die irgendwann zu einem Endzustand führen?

Im Beispiel: Nein!

* ***Determinismus***
  - Ist der Folgestand in jedem Zustand durch das eintreffende Ereignis determiniert?

Im Beispiel: Ja!

Ergänztes Zustandsdiagramm



- Der Versuch des Passierens der Drehtür im blockierten Zustand führt zu einem Alarm.
- Das Einfügen eines Tickets im unblockierten Zustand impliziert das Auswerfen des Tickets.

Talking to a finite state machine?

Locked > *Pass?*

Alarm!

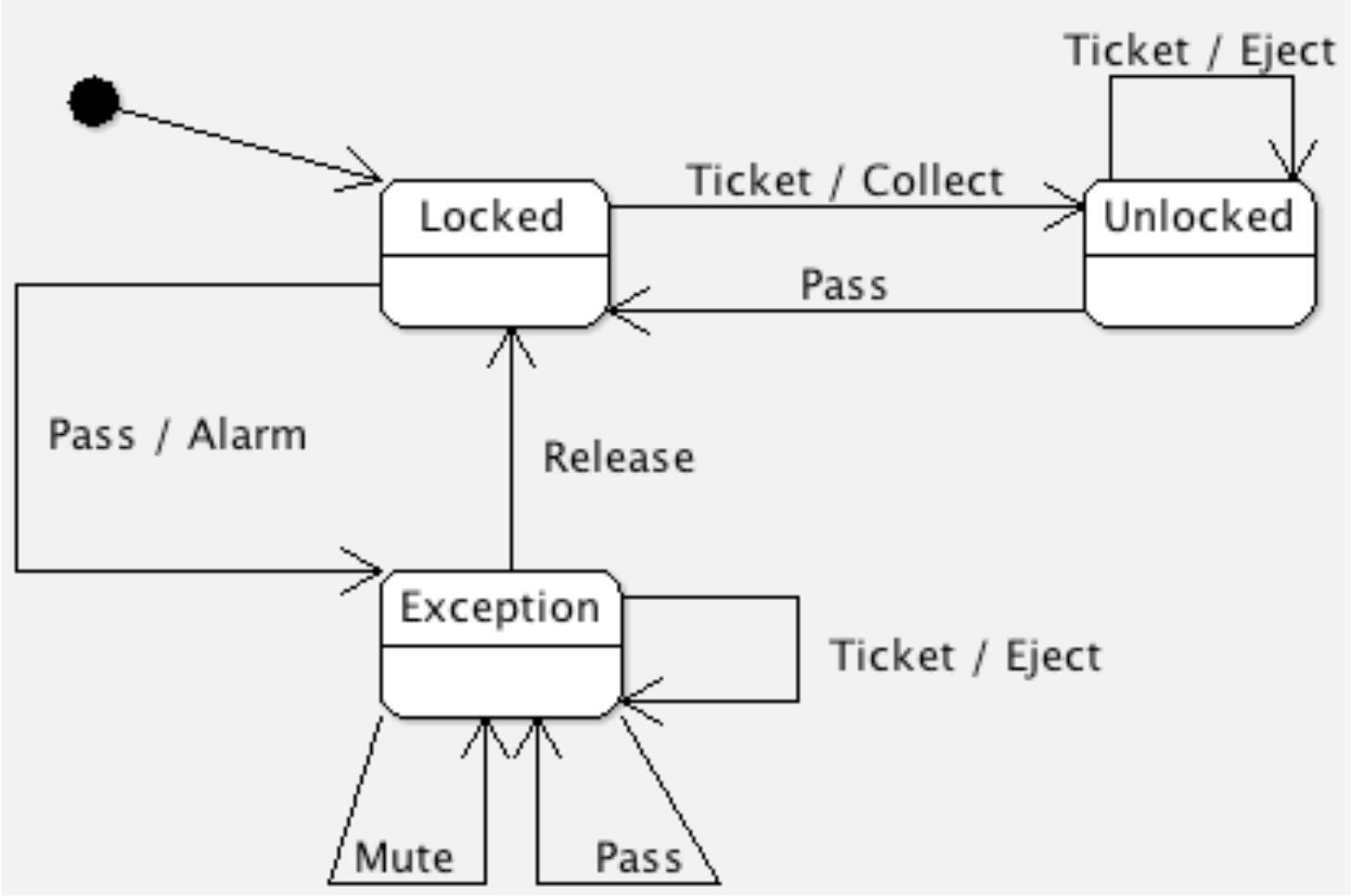
Locked > *Ticket?*

Collect!

Unlocked > *Pass?*

Locked > …

Weiter ergänztes Zustandsdiagramm



Nebenbemerkung/Übung: Der Lebenszyklus eines Objektes kann auch als ein Zustandsdiagramm verstanden werden.

1. Konstruktion
   a) Stempeln
   b) Initialisieren
   c) Assoziieren
2. Mitarbeit
   a) Nachrichten verarbeiten
   b) Nachrichten versenden
   c) Zustand anpassen
3. Persistieren (optional)
4. Dekonstruktion
   a) Aufgeben
   b) Abschließen
   c) Vergehen
5. Auferstehen (optional)

Implementation von Zustandsdiagrammen

- Zweck der Übung?
  - Demonstration der Ausführbarkeit von Objekten
  - Illustration des Abstandes zwischen
    - Modell und
    - Implementation.
  - Illustration stark unterschiedlicher Optionen.
  - Illustration von Programmiertechniken.

Optionen der Implementation von Zustandsdiagrammen

- SwitchCase -- Überführung mittels switch-case-Anweisung
- MethodDispatch -- Überführung mittels dynamischer Bindung
- TableDriven -- Überführung mittels Datenstruktur

https://svn.code.sf.net/p/developers/code/repository/oopm/eclipse/chrestomathy/uml/state/

Optionen der Implementation von Zustandsdiagrammen

- SwitchCase -- Überführung mittels switch-case-Anweisung
- MethodDispatch -- Überführung mittels dynamischer Bindung
- TableDriven -- Überführung mittels Datenstruktur

SwitchCase: Aufzählungstyp für Zustände

```
public enum State {
  Locked,
  Unlocked
}
```

Kurzform für eine abstrakte Klasse State mit zwei Unterklassen Locked und Unlocked.

SwitchCase: Ereignis “(insert a) ticket”

```
public void ticket() {
   System.out.println("Someone trying to \"Ticket\".");
   switch (state) {
     case Locked :
        Collect();
        state = State.Unlocked;
        break;
     case Unlocked :
        Eject();
        break;
   }
   print();
}
```

Abfrage des aktuellen Zustandes (Attribut)

Aufruf einer Methode für die Aktion

Zuweisung des Zielzustandes

Aufruf einer Methode für die Aktion

SwitchCase: Ereignis “pass (the turnstile)”

```
public void pass() {
   System.out.println("Someone trying to \"Pass\".");
   switch (state) {
    case Locked :
       Alarm();
       break;
    case Unlocked :
       state = State.Locked;
       break;
   }
   print();
}
```

SwitchCase: Aktionen

```
protected void alarm() {
   System.out.println("Alarm!");
}

protected void collect() {
   System.out.println("Ticket collected.");
}

protected void eject() {
   System.out.println("Ticket ejected.");
}
```

Kompakte Variante in C

```
enum State { EXCEPTION, LOCKED, UNDEFINED, UNLOCKED };
enum Event { RELEASE, TICKET, MUTE, PASS };

void collect() { }
void alarm() { }
void nop() { }
void eject() { }

enum State next(enum State s, enum Event e) {
  switch(s) {
  case EXCEPTION:
    switch(e) {
    case RELEASE: nop(); return LOCKED;
    case TICKET: eject(); return EXCEPTION;
    case PASS: nop(); return EXCEPTION;
    case MUTE: nop(); return EXCEPTION;
    default: return UNDEFINED;
    }
  case LOCKED:
    switch(e) {
    case TICKET: collect(); return UNLOCKED;
    case PASS: alarm(); return EXCEPTION;
    default: return UNDEFINED;
    }
  case UNLOCKED:
    switch(e) {
    case TICKET: eject(); return UNLOCKED;
    case PASS: nop(); return LOCKED;
    default: return UNDEFINED;
    }
  default: return UNDEFINED;
  }
}
```

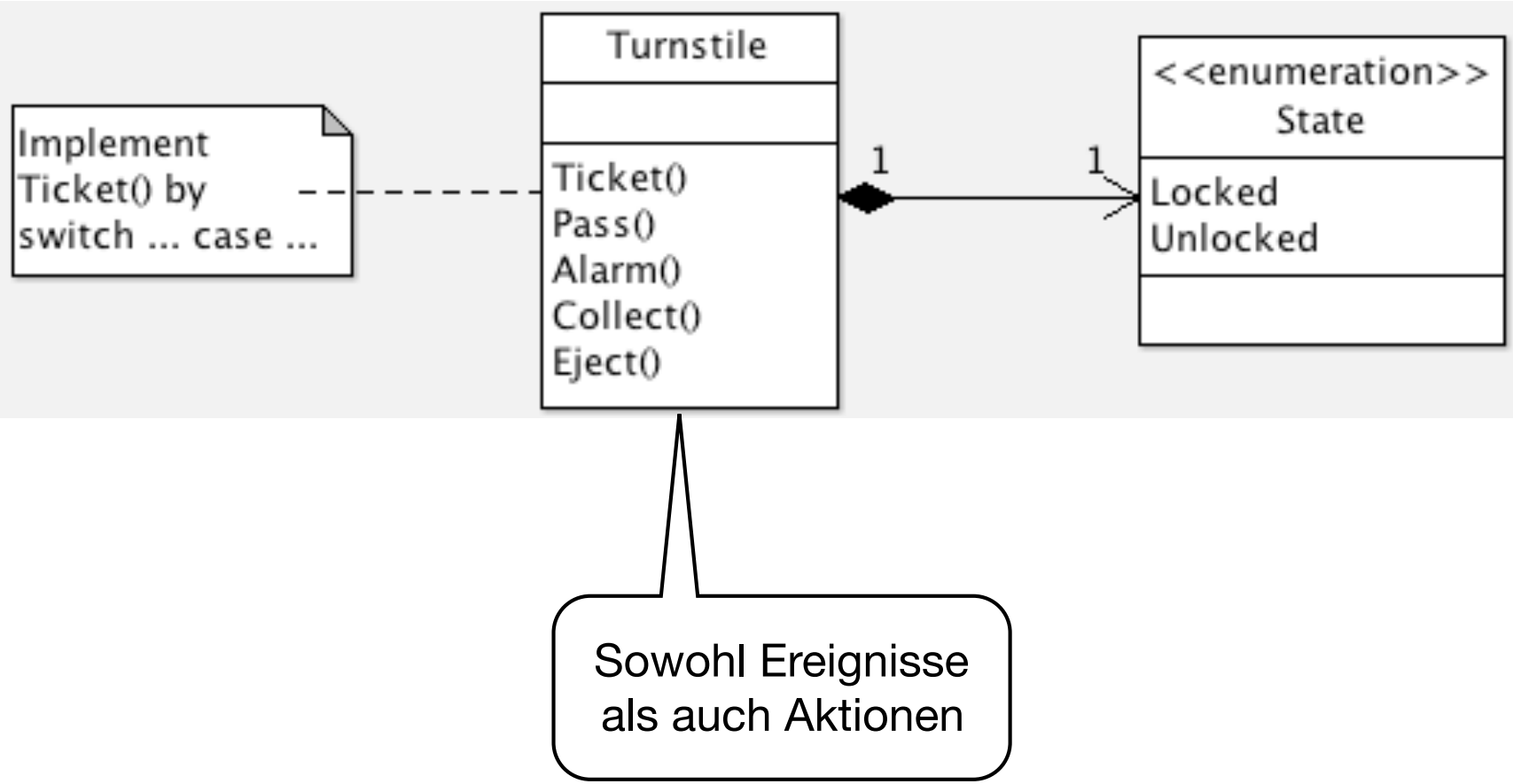
Online-Lokation mit vollständigem Java-Code:

https://svn.code.sf.net/p/developers/code/repository/oopm/eclipse/chrestomathy/uml/state/

Rezept der ***SwitchCase-***Option

- Ein Aufzählungstyp zur Modellierung der Zustände.
- Ein zentrales Objekt mit Zustand und Ereignisbehandlung.
  - Eine Methode per Aktion.
  - Eine Methode per Ereignis.
    - Eine Switch-Case-Anweisung zur Überführung.

Die ***SwitchCase***-Option in UML



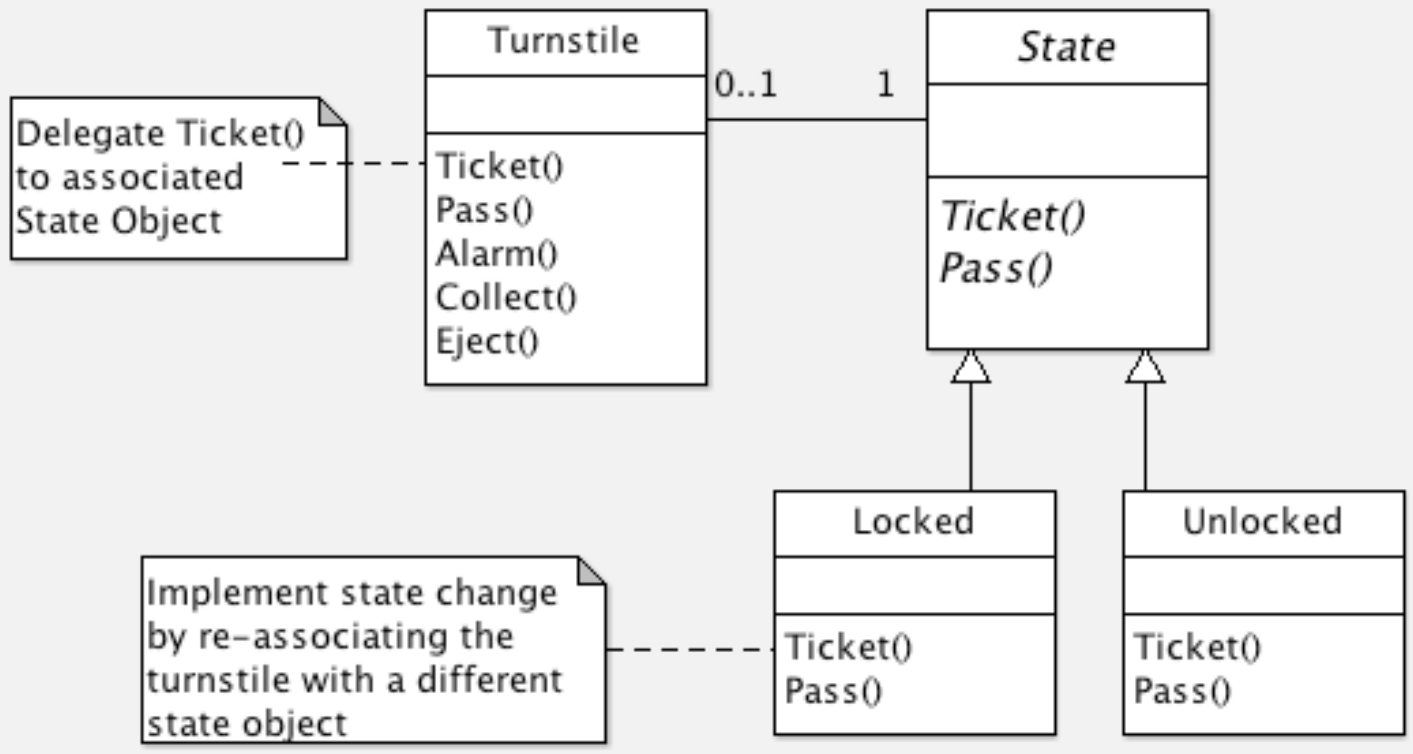
Diskussion der *SwitchCase*-Option

- Pros
  - Einfachheit des Modells.
  - Wiederkennbarkeit von Ereignissen und Aktionen.
- Cons
  - Fallunterscheidung basieren auf Low-Level-Code.
  - Monolithisch im Sinne des zentralen Objektes.
  - Zustände sind nur Werte (also keine Objekte).
  - Keine Laufzeitanpassung vorgesehen.

Optionen der Implementation von Zustandsdiagrammen

- SwitchCase -- Überführung mittels switch-case-Anweisung
- MethodDispatch -- Überführung mittels dynamischer Bindung
- TableDriven -- Überführung mittels Datenstruktur

Die ***MethodDispatch***-Option in UML



MethodDispatch: Die Basisklasse für Zustände

```
public abstract class State {
  public abstract void ticket(Turnstile me);

  public abstract void pass(Turnstile me);
}
```

Zur zustandsspezifischen Ereignisbehandlung

MethodDispatch: *Wiederverwendbare* Zustände

Zur Wiederverwendung von Zuständen

```
public abstract class State {
  static protected Locked locked = new Locked();

  static protected Unlocked unlocked = new Unlocked();
   public abstract void ticket(Turnstile me);

   public abstract void pass(Turnstile me);

}
```

```
/**
 * Behavior of turnstile in the locked state
 */
public class Locked extends State {

   public String printState() { return "Locked"; }

   /** Event: (Insert a) ticket */
   public void ticket(Turnstile me) {
      super.ticket(me);
      me.collect();
      me.state = unlocked;
   }

   /** Event: Pass (the turnstile) */
   public void pass(Turnstile me) {
      super.pass(me);
      me.alarm();
   }
}
```

Analog für den Zustand *Unlocked*.

MethodDispatch:
Der Kontext für die Überführungsfunktion

```
public class Turnstile {
  State state = State.locked;
  public void alarm() { ... }
  public void collect() { ... }
  public void eject() { ... }
  public void ticket() {
     state.ticket(this);
  }
  public void pass() {
     state.pass(this);
  }
}
```

Zustandslokus des Automaten mit Initialisierung

Aktionen des Automaten

Delegation der Ereignisbehandlung an aktuellen Zustand

Rezept der ***MethodDispatch***-Option

- Eine Klassenhierarchie zur Modellierung der Zustände.
  - Eine abstrakte Klasse

    mit einer abstrakten Methode pro Ereignis.
  - Eine konkrete Klasse pro Zustand

    mit einer konkreten Methode pro Ereignis.
- Ein Kontext-Objekt

  zur Verwaltung des aktuellen Zustands.

Diskussion der *MethodDispatch*-Option

- Pros
  - Typüberprüfung impliziert Vollständigkeit.
  - Zustände sind Objekte (Attribute+Verhalten).
- Cons
  - Keine Laufzeitanpassung vorgesehen.
  - Erweiterbarkeit durch neue Klassen praktisch nicht nutzbar.

Optionen der Implementation von Zustandsdiagrammen

- SwitchCase -- Überführung mittels switch-case-Anweisung
- MethodDispatch -- Überführung mittels dynamischer Bindung
- TableDriven -- Überführung mittels Datenstruktur

TableDriven

(Zustand, Ereignis) → (Aktion, Zustand)

- **Eine Datenstruktur (“Map”)**
  - *Schlüssel*: Zustand und Ereignis
  - *Wert*: Aktion und Zustand
- Ein Wert (eines Aufzählungstyps) per Zustand.
- Ein Wert (eines Aufzählungstyps) per Ereignis.
- Ein Objekt (mit Methode) per Aktion.

TableDriven: Ereignisse und Zustände

```
public enum Event {
   Ticket,
   Pass
}
```

```
public enum State {
   Locked,
   Unlocked
}
```

TableDriven: Aktionen

```
public abstract class Action {
   public abstract void execute();
}
```

Zur Wiederverwendung

```
public class Alarm extends Action {
  public static final Alarm singleton = new Alarm();
   public void execute() {
      System.out.println("Alarm!");
   }
}
```

TableDriven: Die Überführungsfunktion als Tabelle

(Zustand, Ereignis) → (Aktion, Zustand)

```
private HashMap<State, HashMap<Event, ActionState>> table =
    new HashMap<State, HashMap<Event,ActionState>>();
```

```
public class ActionState {
   public Action action;
   public State state;
   public ActionState(Action a, State s) {
      action = a;
      state = s;
   }
}
```

TableDriven: Füllen der Überführungsfunktion

```
HashMap<Event, ActionState> atLocked =
     new HashMap<Event, ActionState>();
atLocked.put(Event.Ticket,
     new ActionState(
          Collect.singleton,
          State.Unlocked));
atLocked.put(Event.Pass,
     new ActionState(
          Alarm.singleton,
          State.Locked));
table.put(State.Locked, atLocked);
```

Diskussion der *TableDriven*-Option

- Pros
  - Überführungsfunktion ist anpassbar zur Laufzeit.
- Cons
  - Vollständigkeit ist nicht garantiert.
  - Zustände/Ereignisse sind nur Werte (keine Objekte).
    - Beschränkte Erweiterbarkeit.
    - Keine Attribute.
    - Kein Verhalten.

All dies ist leicht zu adressieren.

- **Zusammenfassung**
  - Verhaltensmodellierung
    - ***Anwendungsfalldiagramm***

      Nutzersicht auf System
    - ***Sequenzdiagramm***

      Darstellung der Ausführung konkreter Szenarien
    - ***Aktivitätsdiagramm***

      Sequentielle, parallele, kommunizierende Prozesse
    - ***Zustandsdiagramm***

      Zustandsänderung von Objekten bei Ereignissen
- **Ausblick**
  - Generics
  - …