Softwaretechnik

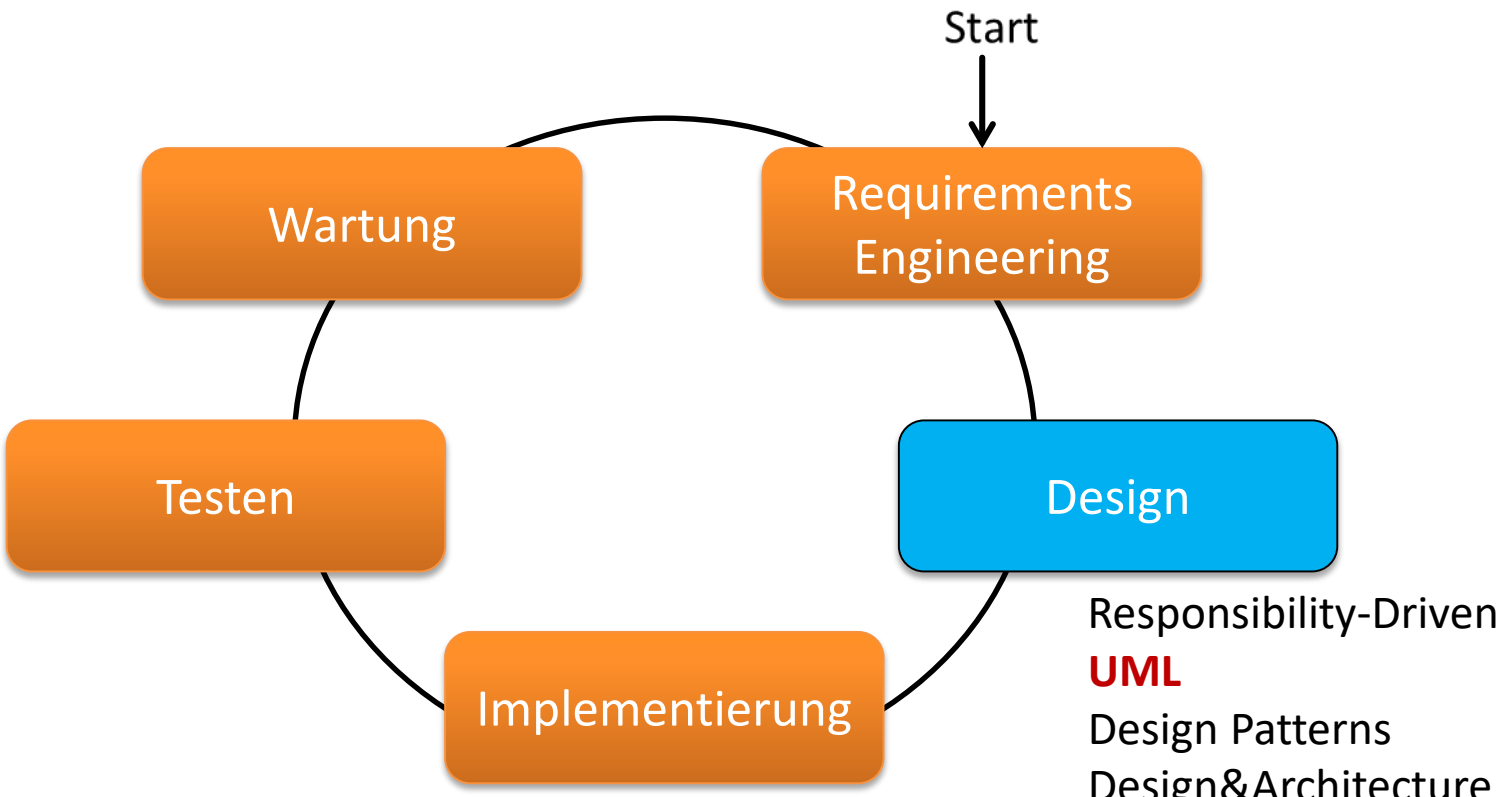
Modeling Behavior

SOFTWARE SYSTEME

UNIVERSITÄT LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund
Software Systems

Einordnung



Responsibility-Driven
UML
Design Patterns
Design&Architecture

Use-Case Diagramme

Use-Case Diagramme

Ein use case ist eine *generische Beschreibung einer gesamten Transaktion* welche eine oder mehrere Aktoren involviert.

Ein use-case Diagramm präsentiert eine *Menge von use cases* (Ellipsen) und deren externe Aktoren, die mit dem System interagieren.

Abhängigkeiten und *Assoziationen* zwischen use cases können dargestellt werden.

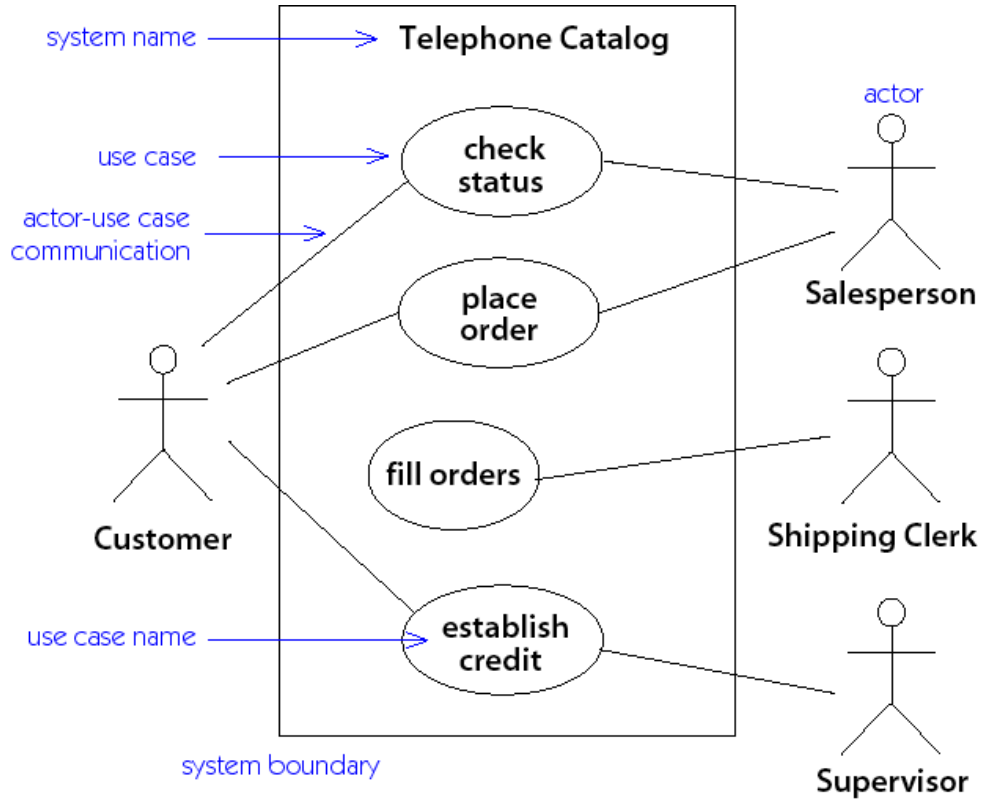


Figure 5-1. *Use case diagram*

Anwendung

- Aufzeigen der Ziele der User-System Interaktionen
- Definition und Organisation der funktionalen Anforderungen
- Spezifikation des Kontext des Systems
- Modellierung der grundlegenden Abläufe eines Use Cases

Identifikation von Aktoren

- Wer verwendet das System?
- Wer installiert das System?
- Wer wartet das System?
- Wer administriert das System?
- Wer beendet das System?
- Mit wem oder was kommuniziert das System?
- Wer erhält Informationen vom System?
- Welche anderen Systemen verwenden dieses System?

Was kann ein Aktor sein?
- Person
- Organisation
- Anderes System
- Externes Gerät

Identifikation von Use Cases

- Falls das System Informationen speichert: Welche Aktoren werden diese Informationen kreieren, updaten, verwalten, löschen, lesen?
- Welche Funktionen möchten die einzelnen Aktoren vom System verwenden?
- Gibt es externe Events, die das System betreffen und wie wird das System darüber informiert?
- Informiert das System Aktoren über einen geänderten Zustand innerhalb des Systems?

Was ist ein Use Case?
- Aktion, die eine bestimmte Aufgabe im System erfüllt

Beziehungen (Relationships)

- Association: Kommunikation und Interaktion zw. Aktor und Use Case

- <<Include>>: Repräsentiert eine Abhängigkeit von eine Use Case zu einem „included“ Use Case; keine Aktoren initiieren diese Interaktion, sondern sie wird von einem anderen Use Case gestartet;
  Beispiel: (Log In) ---<<include>>---> (Verify Password)

- <<Extend>>: Wenn der Basis Use Case ausgeführt wird, wird *unter Umständen* (Kriterium erforderlich) der Extended Use Case ebenfalls ausgeführt;
  Beispiel: (Log In) <---<<extend>>--- (Display failed log in)

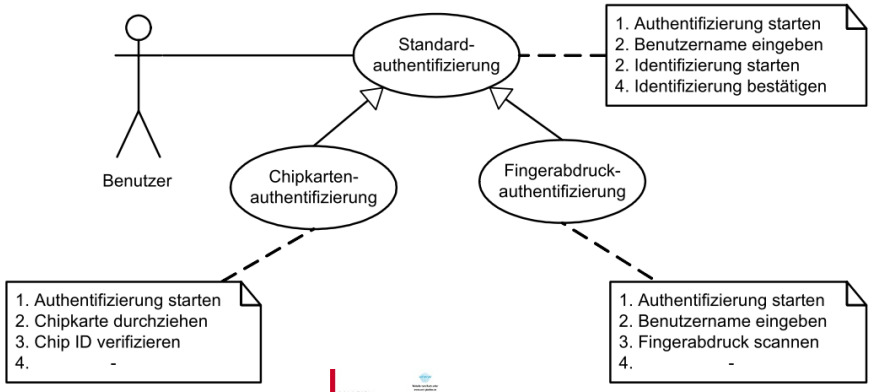
- Generalization: Ähnlich zu Vererbung: Erweiterung der Basisfunktionalität;
  Beispiel: (Make Payment) <|——— (Pay with Credit Card)

Verwendung: Use-Case Diagramm

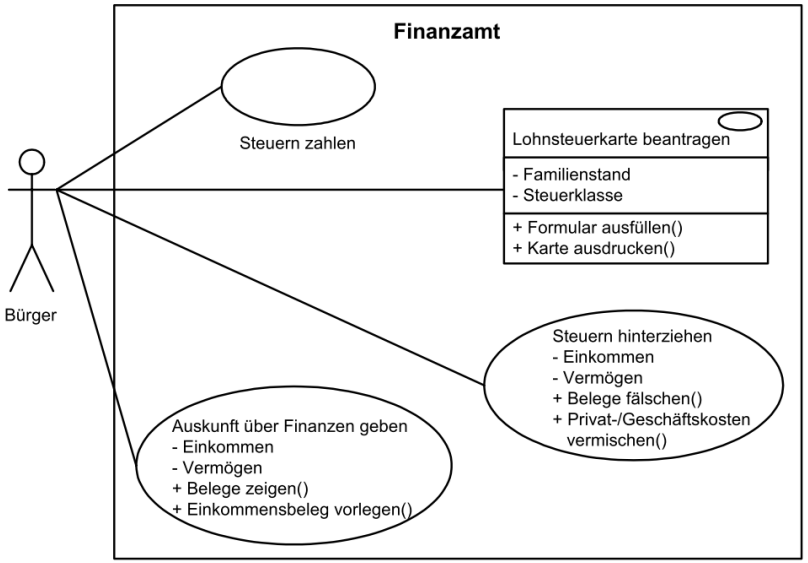
"A use case is a *snapshot of one aspect* of your system. The sum of all use cases is *the external picture* of your system …"

—UML Distilled

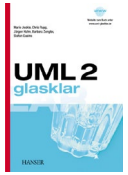
Generalisierung und Kommentare



Auch Attribute und Operationen möglich



Quelle:



Sequenz Diagramme

Szenarien

Ein Szenario ist eine *Instanz* von einem use case, dass ein *typisches Beispiel* einer Ausführung zeigt.

Szenarien können durch UML repräsentiert werden, entweder durch *sequence diagrams* oder *collaboration diagrams*.

*Beachtet: Ein Szenario beschreibt nur **ein** Beispiel eines use cases, so dass Besonderheiten oder Bedingungen nicht ausgedrückt werden können!*

Sequenzdiagramme

Ein sequence diagram beschreibt ein Szenario durch das Zeigen von Interaktionen zwischen einer Menge von Objekten in einer *zeitlichen Abfolge*.

Objekte (keine Klassen!) werden als *vertikale Balken* gezeichnet. *Events* oder Nachrichtensendungen werden als horizontale (oder schräge) *Pfeile* vom Sender zum Empfänger gezeichnet.

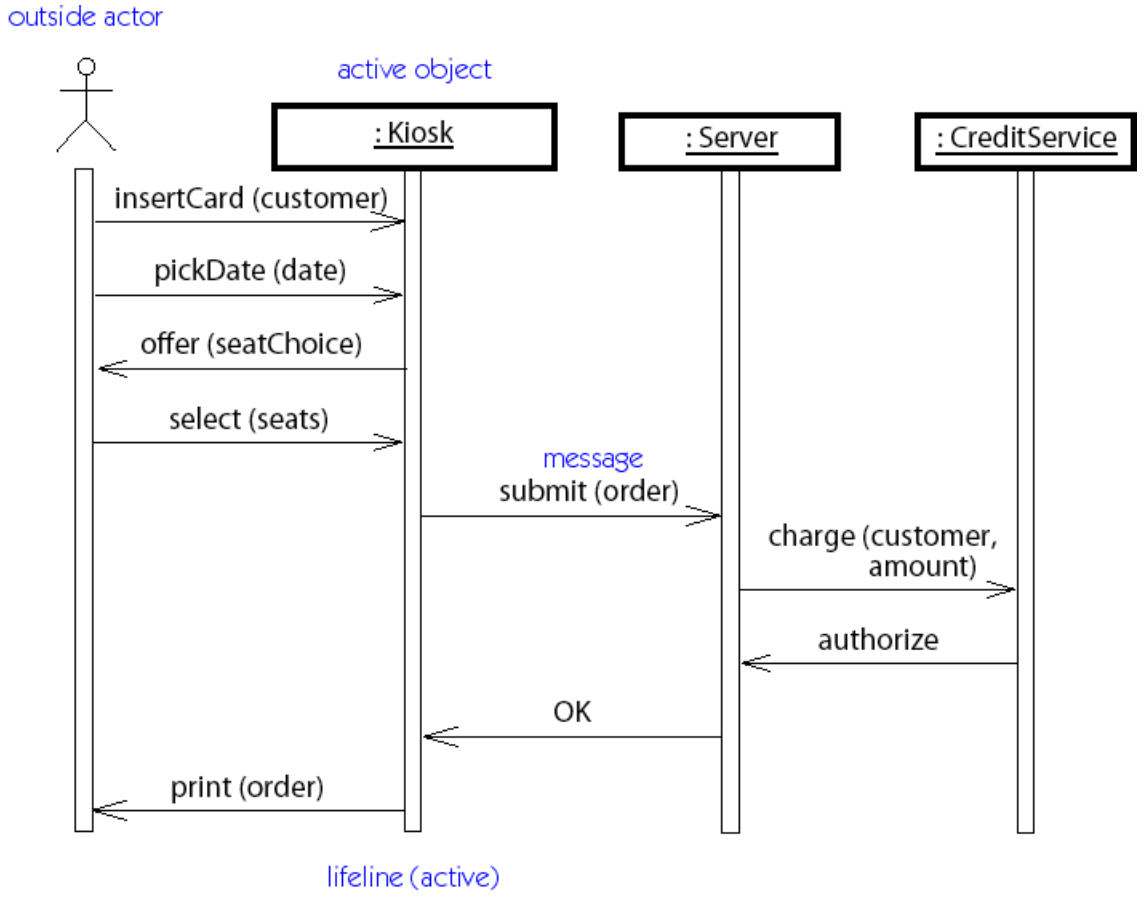
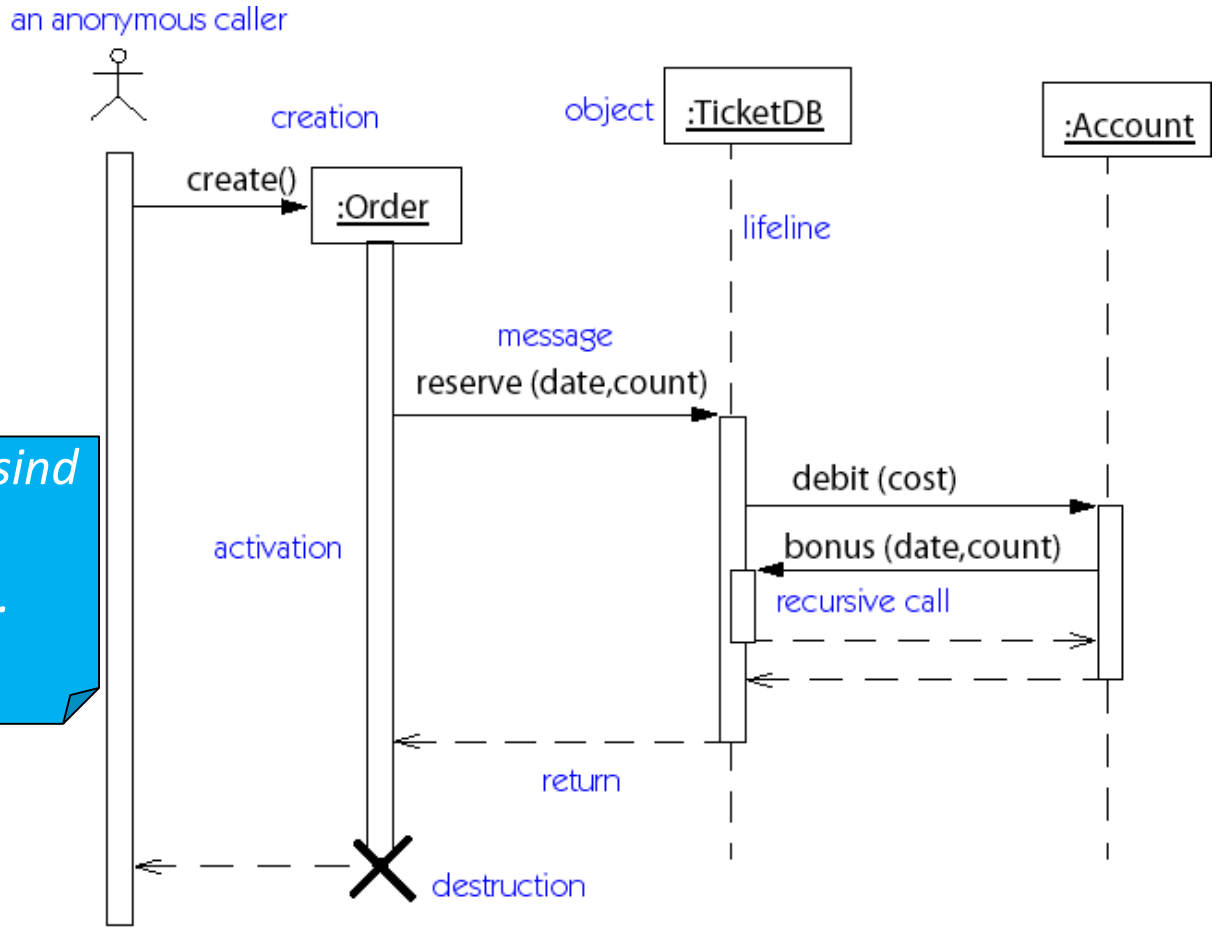


Figure 8-1. *Sequence diagram*

Szenario: Sitzplatz im Kino reservieren

Aktivierungen



Return-Statements sind optional. Abhängig vom Detailgrad evtl. wichtig.

Figure 8-2. *Sequence diagram with activations*

Asynchronität und Bedingungen

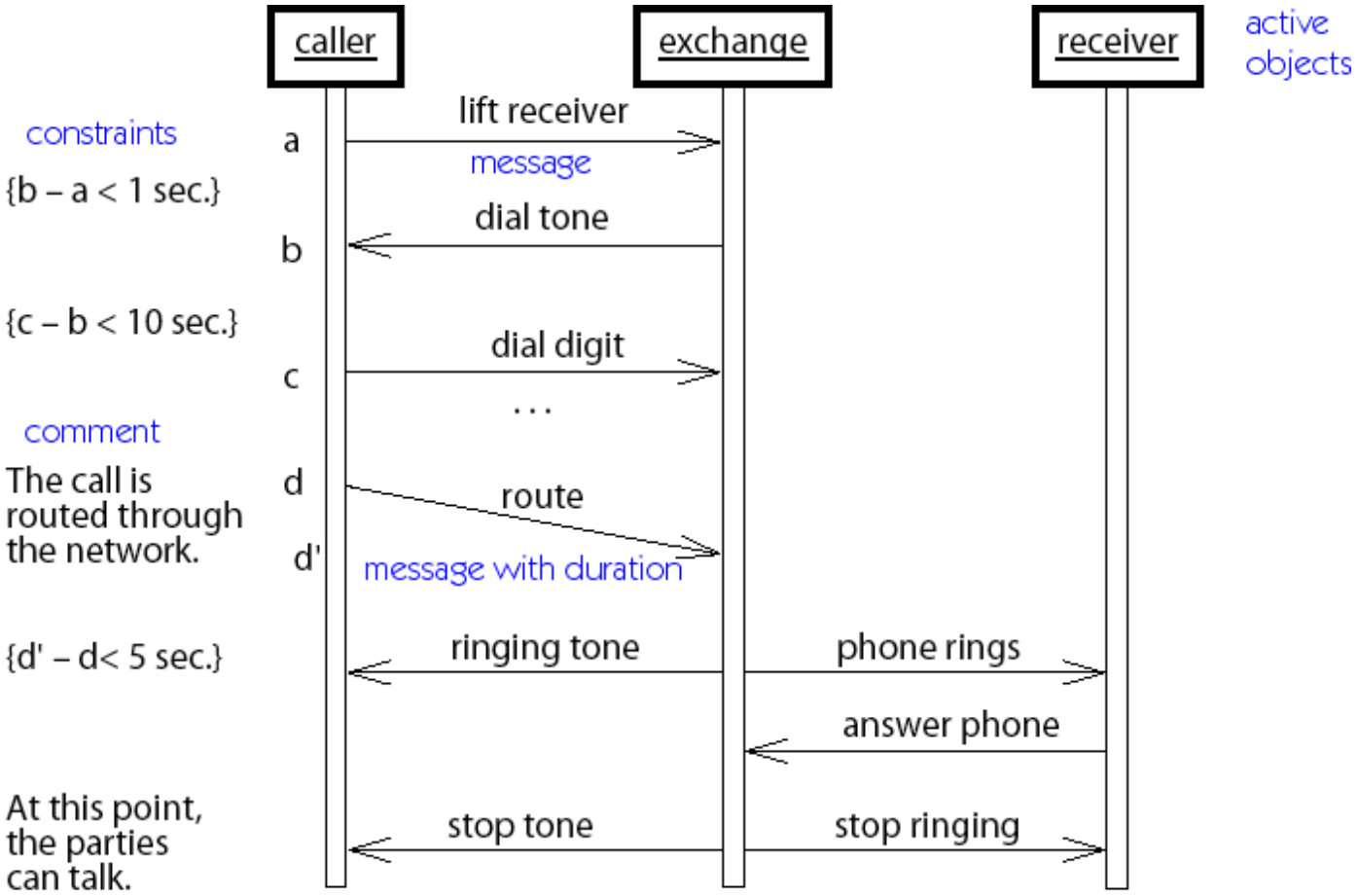
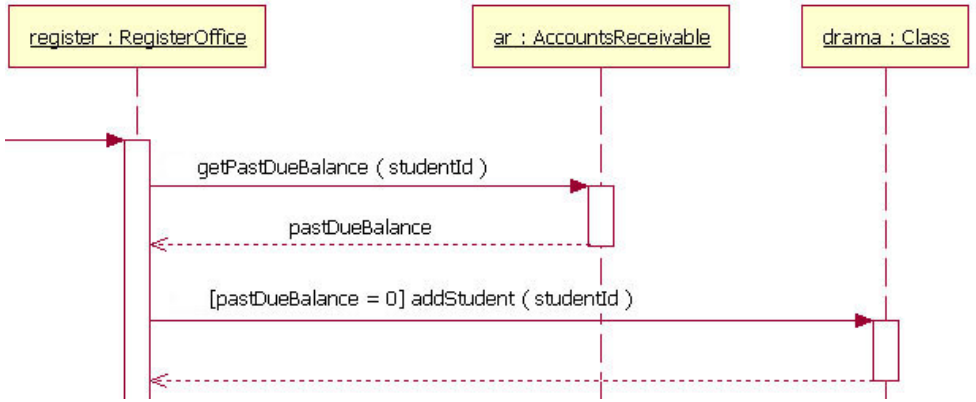


Figure 13-161. *Sequence diagram with asynchronous control*

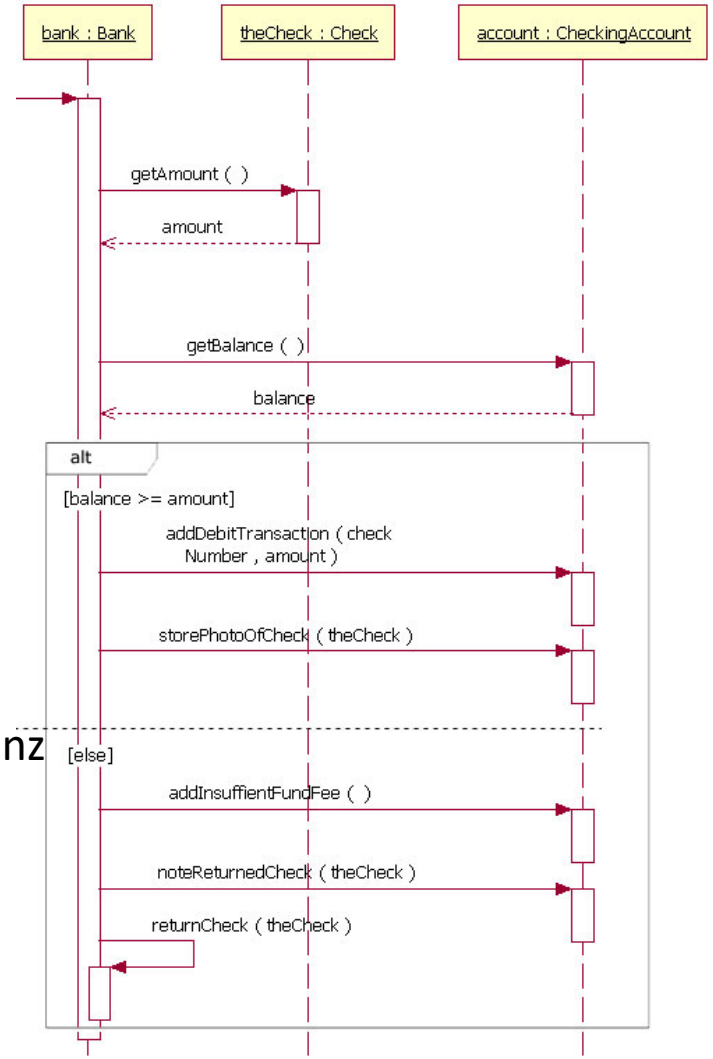
Alternativen und Guards

Guard: Bedingung muss erfüllt sein, bevor eine Nachricht verschickt wird.



Syntax: [Boolean Test]

Alternative Sequenz



Quelle: http://www.ibm.com/developerworks/rational/library/3101.html

Statechart (Zustands-)Diagramme

Definition

Ein Statechart Diagram beschreibt die *zeitliche Evolution* eines Objektes von einer gegebenen Klasse in Abhängigkeit von *Interaktionen* mit anderen Objekten innerhalb und außerhalb des Systems.

Ein Event ist eine one-way (asynchrone) Kommunikation von einem Objekt zu einem Anderen:

- *atomar* (nicht unterbrechbar)
- Beinhaltet *Hardware* und Realwelt-Objekte, z.B., Nachrichteneingang, input Ereignis, Zeitüberschreitung, ...
- Notation: ***eventName(parameter: type, ...)***
- Kann das Objekt zu einer *Transition* zwischen Zuständen veranlassen

Definition...

Ein Zustand ist eine Zeitperiode, bei der ein Objekt auf ein Ereignis *wartet*:

- Dargestellt als *abgerundete Box* mit (bis zu) drei Sektionen:
  - *name* — optional
  - *state* variables — name: type = value (valid only for that state)
  - *triggered operations* — internal transitions and ongoing operations
- Kann *geschachtelt* sein

Beispiel

- Zustand eines Objektes

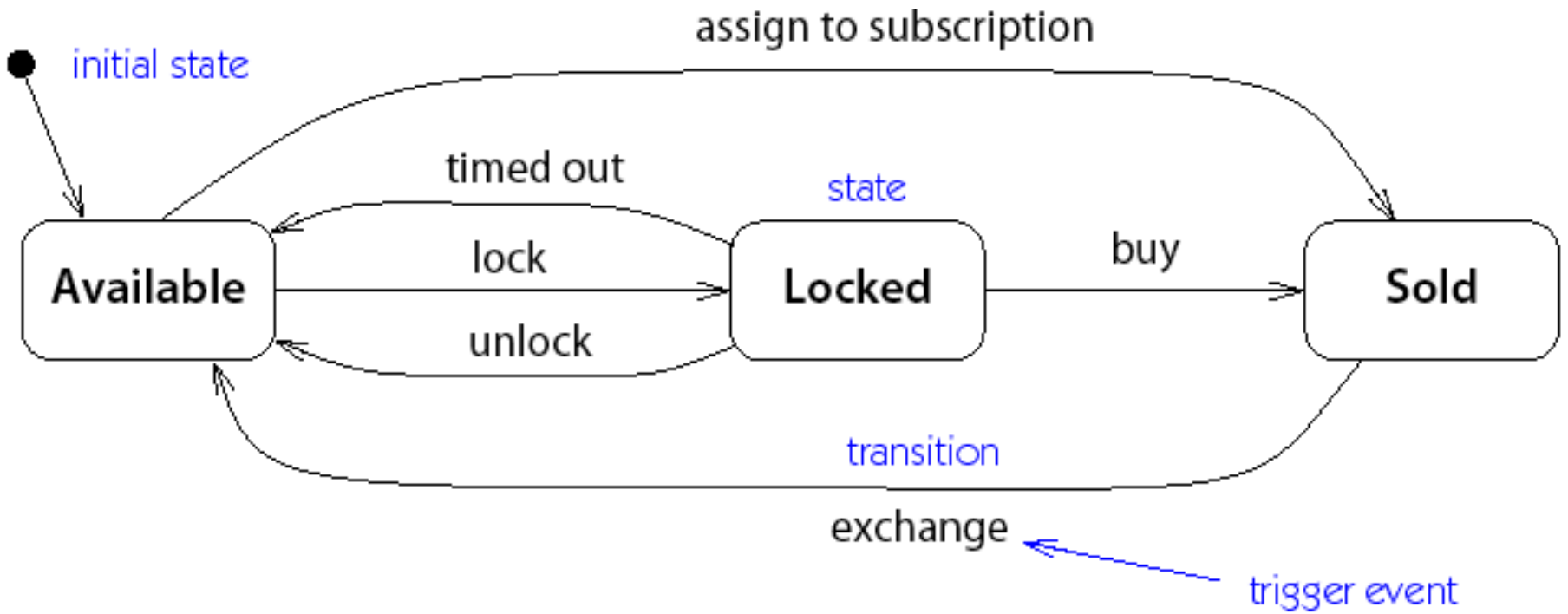


Figure 3-5. *Statechart diagram*

Statusbox mit Regionen

Das *Eingangs-Event* tritt auf, wann immer eine Transition zu diesem Zustand getätigt wird.

Das *Ausgangs-Event* tritt auf, wenn eine Transition aus diesem Zustand hinaus führt.

Die *Hilfs-* und *Zeichenereignisse* lösen interne Transitionen aus ohne den Zustand zu ändern, so dass keine Eingangs- oder Ausgangsoperation durchgeführt wird.

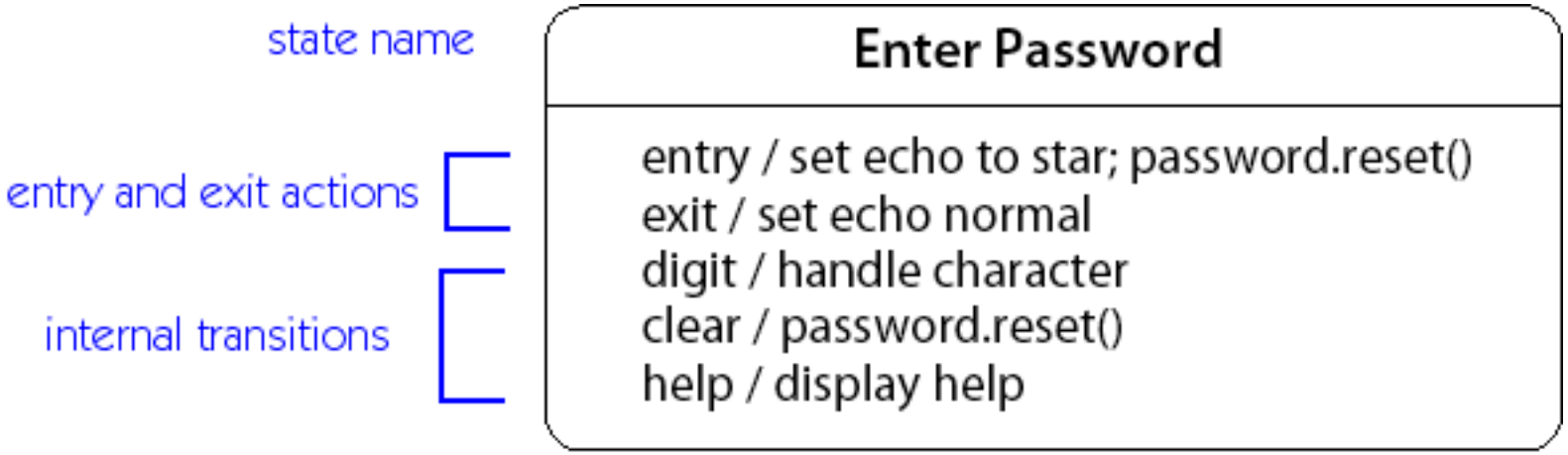


Figure 6-4. *Internal transitions, and entry and exit actions*

Transitionen

Eine Transition ist eine *Antwort auf ein externes Ereignis* welches das Objekt in einem *bestimmten Zustand* erhalten hat

- Kann zur *Ausführung* einer Operation und zum Wechsel des Zustands des Objekts führen
- Kann ein Ereignis zu einem anderen externen Objekten *senden*
- Transitionssyntax (jeder Teil ist optional):
  event(arguments) [condition]
  / target.sendEvent operation(arguments)
- *Externe Transitionen* markieren Kreisbögen zwischen Zuständen
- *Interne Transitionen* sind Teil der ausgelösten Operationen eines Zustandes

Operationen und Aktivitäten

Eine Operation ist eine *atomare Aktion* angestoßen von einer Transition

- *Eingangs- und Ausgangsoperationen* können mit Zuständen assoziiert werden

Eine Aktivität ist eine *laufende Operation* die dann läuft, während ein Objekt in einem bestimmten Zustand ist

- Modelliert als “interne Transitionen” markiert mit dem pseudo-event ***do***

Schachtelung: Nested Statecharts

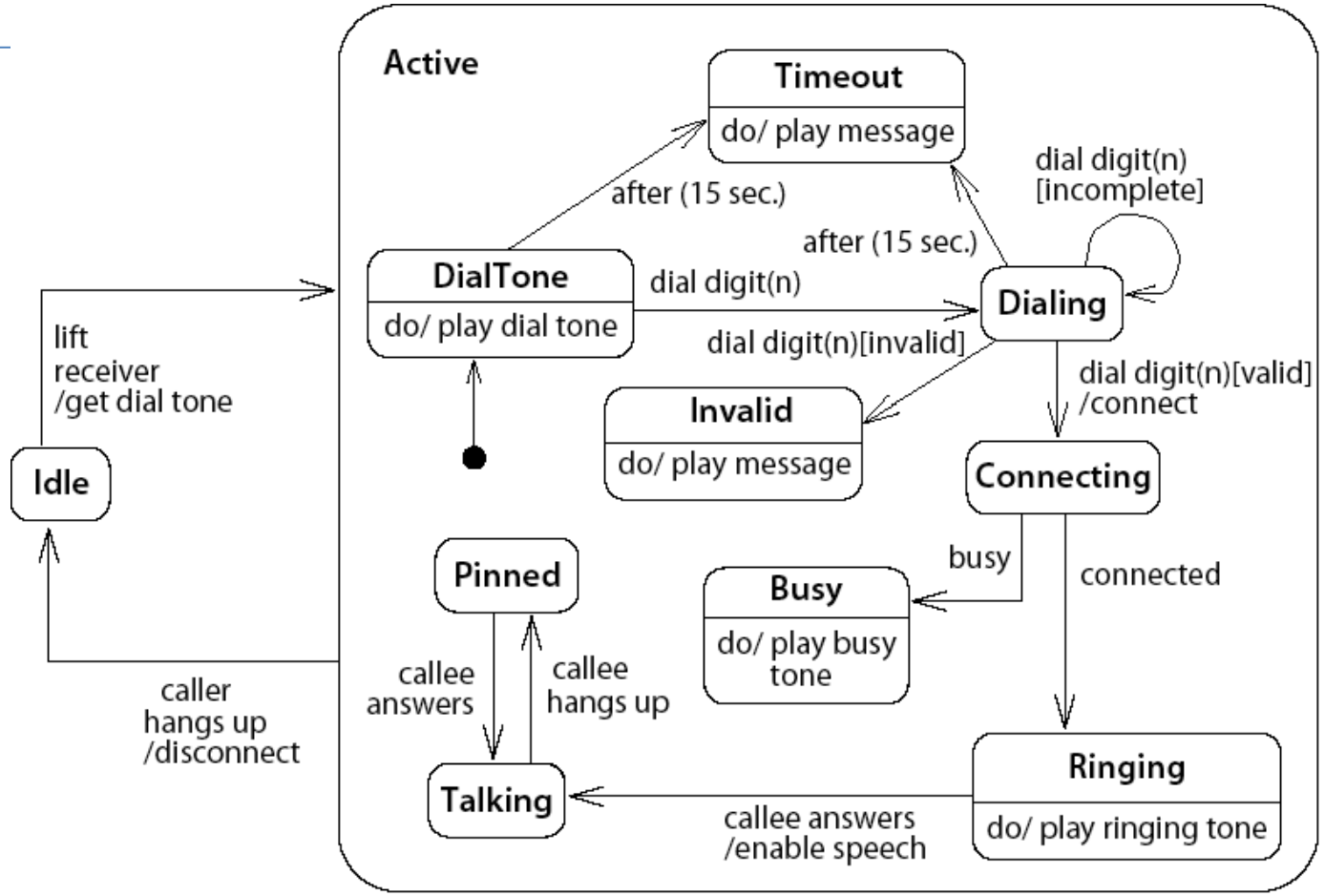


Figure 13-169. *State diagram*

Nested Statecharts

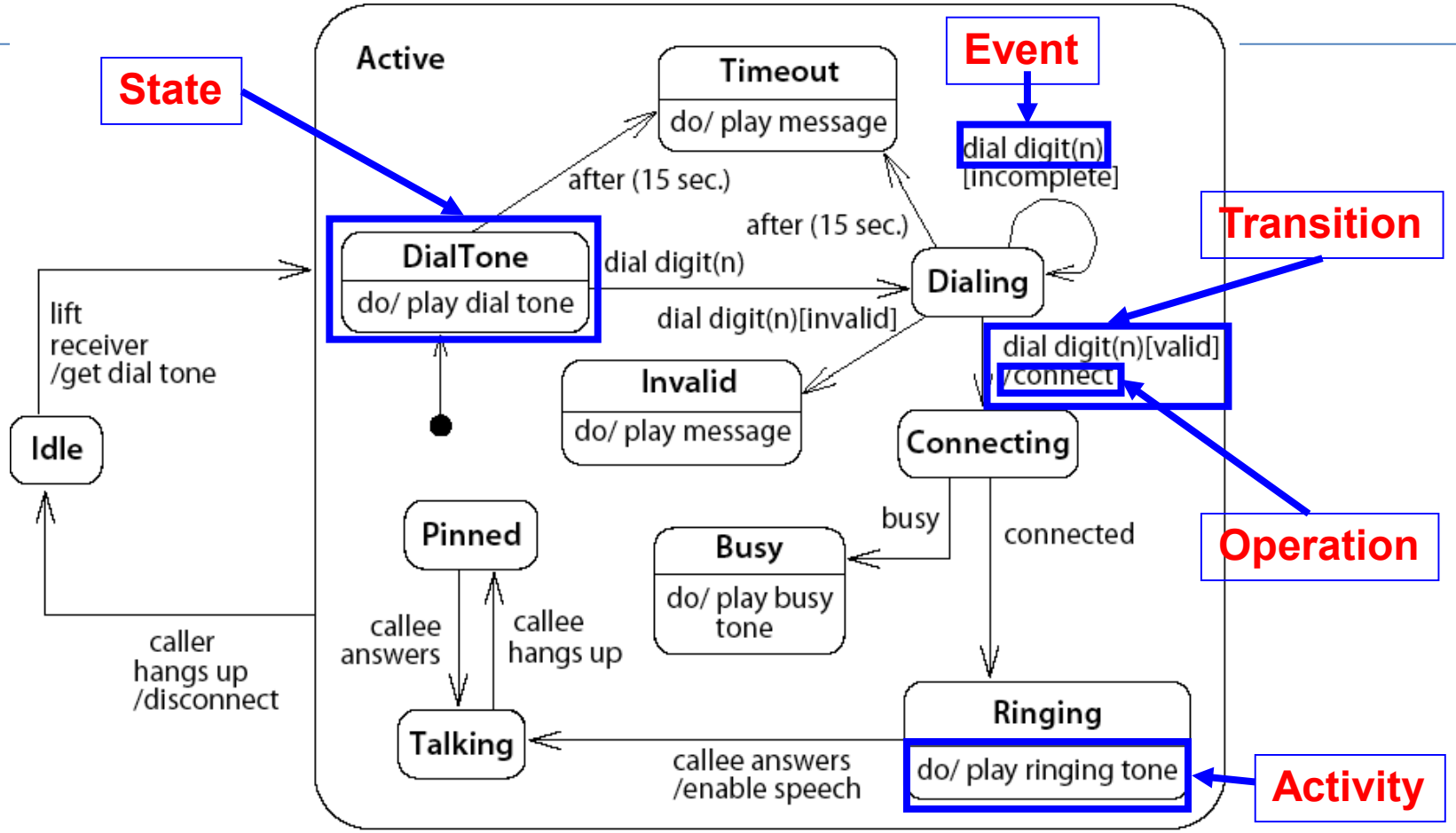


Figure 13-169. *State diagram*

Aufgabe

- Modellieren Sie ein Flugzeug-Objekt, welches den Zustand des Flugzeuges bzgl. der Platzreservierung wiedergibt. Definieren Sie geeignete Zustandsübergänge und evtl. Bedingungen dafür.

UML Benutzung: Perspektiven

Perspektiven

Drei Perspektiven beim Erstellen von UML Diagrammen:

1. ***Konzeptionell***
   - Repräsentieren Domänenkonzepte
     - Ignoriere Software Belange
2. ***Spezifikation***
   - Fokus auf sichtbare Interfaces und Verhalten
     - Ignoriere interne Implementierung
3. ***Implementierung***
   - Dokumentiere Implementierungsentscheidungen
     - Häufigste, aber am wenigsten nützlichste Perspektive (!)

—UML Distilled

Was Ihr mitgenommen haben solltet

- Was ist der Zweck von use case Diagrammen?
- Warum beschreiben Szenarien Objekte und nicht Klassen?
- Wie können zeitliche Bedingungen in Szenarien beschrieben werden?
- Wie spezifiziert und interpretiert man Nachrichten-Labels in einem Szenario?
- Wie benutzt man genestete Zustandsdiagramme, um Objektverhalten zu modellieren?
- Was ist der Unterschied zwischen “externen” und “internen” Transitionen?

Literatur

- *The Unified Modeling Language Reference Manual*, James Rumbaugh, Ivar Jacobson and Grady Booch, Addison Wesley, 1999.