Software Reuse

Unified Modeling Language (UML)

Peter Sturm
Universität Trier

Bereiche der Modellierung

- Strukturmodellierung
  - Logische Struktur der Anwendung
  - Aufteilung in Klassen, Beziehungen zwischen Klassen, ...
- Verhaltensmodellierung
  - Dynamische Abläufe innerhalb einer Anwendung
  - Rollen, Nachrichten, Aktionen, Kontrollflüsse, ...
  - Use Case Diagrams
- Architekturmodellierung
  - Physische Struktur der Anwendung
  - Komponenten, Schnittstellen, Dateien, Bibliotheken, ...

Sichten

- UML unterstützt 5 verschiedene Sichten für die Visualisierung, Spezifikation, Konstruktion und Dokumentation eines Softwaresystems
  - Design View
  - Use Case View
  - Process View
  - Implementation View
  - Deployment View
- In allen Sichten gibt es
  - Strukturelle Modelle (statische Sichten)
  - Verhaltensmodelle (dynamische Sichten)

Software Reuse

UML

Elemente

Die Klasse

- Zentraler Baustein objektorientierter Systeme
  - Beschreibt Objektmenge mit gleichen Attributen, Operationen, Beziehungen und gleicher Semantik
  - Implementiert ein oder mehrere Schnittstellen
- Klassen haben unterscheidbare Namen
  - Einfache Namen
  - Pfadnamen X::Y (Klasse Y enthalten in Package X)

Sensor

java::awt::Rectangle

UML: Klasse

- Graphische Klassendarstellung
  - Name
  - Daten
  - Funktionen
- Members können für Übersichten entfallen
- Stereotypen <<X>>: “kind of X”

<<Singleton>>
MemoryHandler

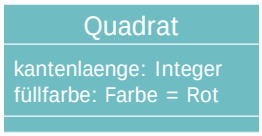
Einstein: Person
name: “Albert Einstein”
address: “Princeton, NJ”

:Person

Anonymes Objekt

Attribute

- Eigenschaft der Klasse
- Attribute haben
  - einen Namen
  - ggf. einen Typ
  - ggf. einen initialen Defaultwert



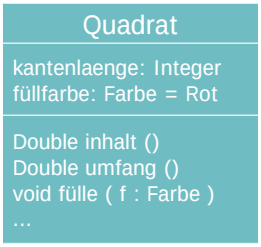
Attribute - revisited

[visibility] name [multiplicity] [: type] [= initial] [{properties}]

- Property
  - changeable: Keine Einschränkungen
  - addOnly: Sinnvoll bei Multiplicity > 1
  - frozen: Keine Änderung nach Initialisierung

Operationen

- “Implementation of a service that can be requested from any object of the class to affect behavior”
- Operationen bestehen aus
  - Namen
  - Name, Typ und initialer Defaultwert aller Parameter
  - Typ des Rückgabewerts



Operationen - revisited

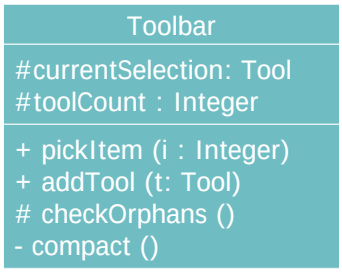
[visibility] name [(parameter-list)] [:return-type] [{property}]

Parameter: [direction] name [:type] [=default]

- Direction
  - in, out, inout
- Properties
  - isQuery
  - sequential: Nur ein Kontrollfluß im Objekt pro Zeitpunkt
  - guarded: MT-safe: Sequentialisierung aller guarded-Aufrufe
  - concurrent: MT-safe: Interne Konsistenzsicherung

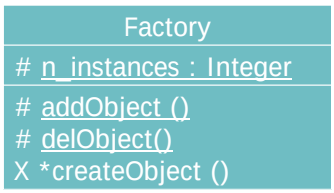
Sichtbarkeit

- Analog zu C++ 3 Sichtbarkeitsebenen
  - public: Dargestellt durch vorangestelltes +
  - protected: #
  - private: -



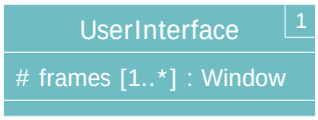
Instance Member vs. Class Member

- Instance Member
- Class Member
  - Unterstrichen



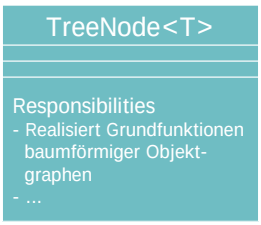
Objektanzahl

- Klassenebene
  - Objektanzahl einer Klasse im Einzelfall beschränken bzw. festlegen
- Attributebene
  - Attributeanzahl (impliziert Array bzw. dynamische Datenstruktur)



Verpflichtungen (Responsibilities)

- Wesentliche Eigenschaften aller Objekte einer Klasse
  - Prosa, z.B. Gründe für diese Klasse, etc.
- Ergebnis der Analyse

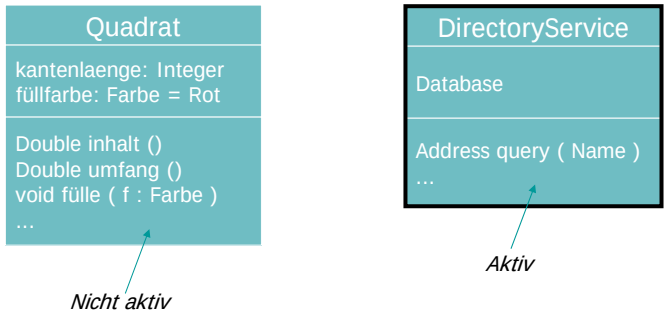


Hints and Tips

- A well-structured class
  - provides a crisp abstraction of something drawn from the vocabulary if the problem domain or the solution domain.
  - embodies a small, well-defined set of responsibilities and carries them all out very well.
  - provides a clear separation of the abstraction's specification and its implementation.
  - Is understandable and simple yet extensible and adaptable.
- When drawing a class in UML
  - show only those properties of the class that are important to understanding the abstraction in its context.
  - organize long lists of attributes and operations by grouping them according to their category.
  - shoe related classes in the same class diagrams.

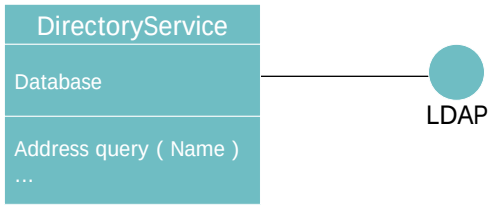
Aktive Klasse

- Objekte dieser Klasse beinhalten ein oder mehrere Kontrollflüsse
  - Können Aktivitäten auslösen
  - Verhalten konkurrent zu anderen Teilen des Systems



Schnittstellen

- Sammlung von Operationensdeklarationen
  - Keine Implementierung
  - Vgl. abstrakte Klasse in C++
- Spezifizieren Service einer Klasse oder Komponente



Beziehungen (Relationships)

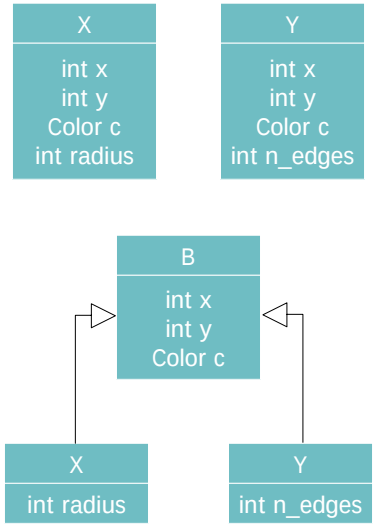
- Beziehungsarten
  - Abhängigkeiten (Dependencies)
    - Beziehungen zwischen Klassen
  - Generalisierungen
    - Beziehungen zwischen generalisierten Klassen und deren Spezialisierungen
  - Assoziationen
    - Strukturelle Beziehungen zwischen Objekten

Generalisierung - Spezialisierung

- Wesentliche Modellierungsschritte
- Generalisieren

  Semantische Gemeinsamkeiten identifizieren und separat beschreiben
- Spezialisieren

  Ergänzen / Erweitern / Einschränken einer vorhandenen Klasse
  - Untertyp (Subtyping)
  - Code Reuse
- Abstraktionshierarchie

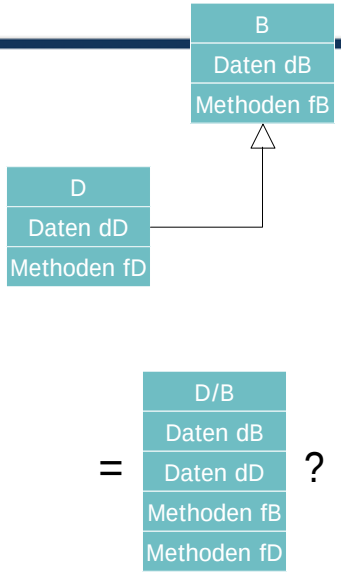


Die Ableitung

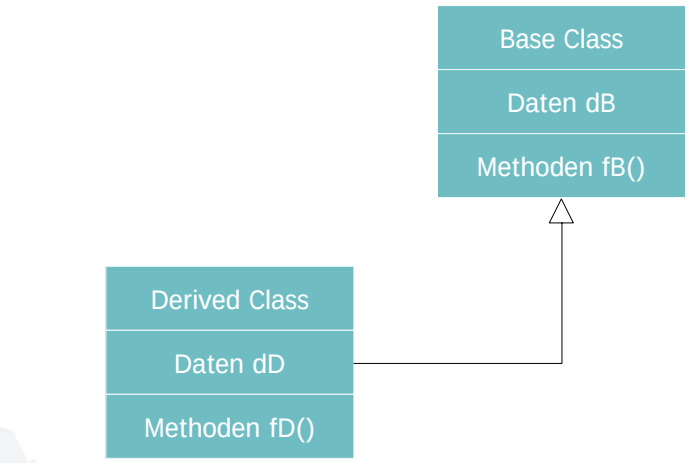
- "Klasse D wird von B abgeleitet"

```
class D: public B {
...
}
```

- Sind D-Objekte auch B-Objekte?
- Wann?
- Mehrfachimplementierungen?
  - `D::f()` und `B::f()`
- Einfachvererbung (Single Inheritance)
- Mehrfachvererbung (Multiple Inheritance)



UML: Ableitung

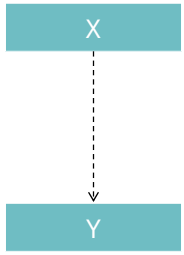


Details bei Klassenableitung

- Ausgezeichnete Positionen in der Ableitungshierarchie
  - {root}: Klasse soll keine Ableitung sein
  - {leaf}: Nicht von Klasse ableiten
- Abstrakt
  - Abstrakte Klasse: kursiv
  - Abstrakte Methode: kursiv
- Polymorphie
  - {leaf} Methode: Kein Overriding für diese Methode

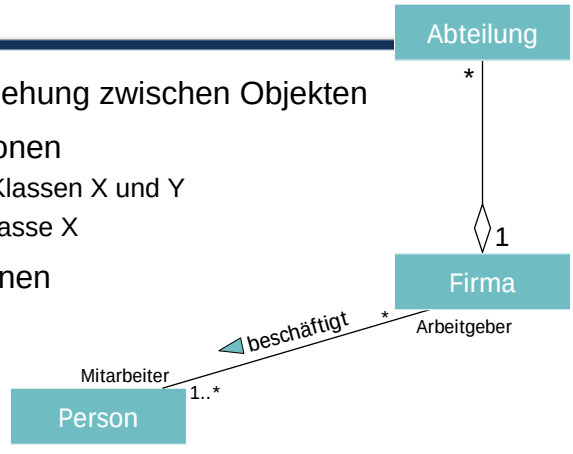
Dependency

- "X is using Y"-Beziehung
  - Änderungen an der Spezifikation Y beeinflußt u.U. X
  - Gerichtete Beziehung
- Beispiele
  - Argumente in einer Operation
  - Rückgabewert
  - ...



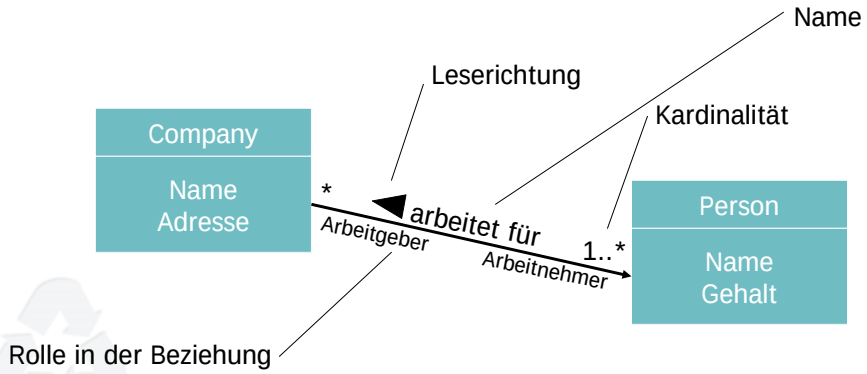
Association

- Strukturelle Beziehung zwischen Objekten
- Binäre Assozationen
  - zwischen zwei Klassen X und Y
  - innerhalb der Klasse X
- n-äre Assoziationen
- Bestandteile
  - Name
  - Rolle
  - Kardinalität
- Sonderfall Aggregation (Has a)



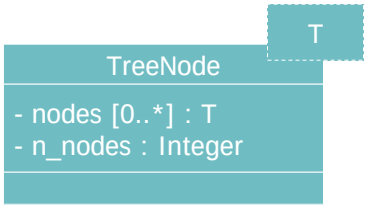
UML: HAS_A-Beziehungen

- Aggregation, Komposition, Containment
- Aufbau über Data Members
- Erweiterte Entity-Relationship-Diagramme



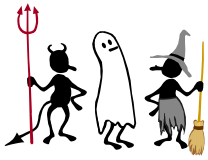
Templates

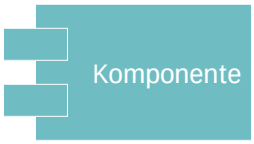
- Darstellung wie normale Klasse
- Hervorhebung der Typparameter



Gruppierungen

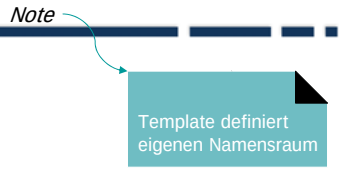
- Zusammenfassen von Klassen, Schnittstellen, Beziehungen
- Mehrstufige Zerlegung komplexer Systeme
- Komponenten
  - Eigenständiges Element; existiert zur Laufzeit
- Package
  - Konzeptionelle Gruppierung

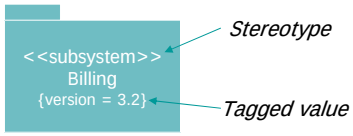


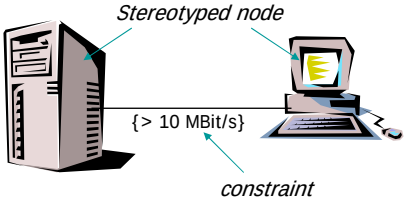


Ergänzungen

- Note
  - Kommentar, Einschränkung, Bemerkung
- Stereotyp
  - Definition eigener Bausteine; UML-Erweiterung; Metatyp
- Tagged Value
  - Erweiterte Eigenschaft eines UML-Elements; Metawert (manifestiert sich nicht als Datenmember einer Klasse)
  - Welche Programmiersprache?
  - Author
- Constraint
  - Einschränkung bei der Semantik eines UML-Elements







Beziehungsstereotypen

- Zwischen Klassen und Objekten
  - bind: Quelle instanziiert Zieltemplate mit aktuellen Parametern
  - derive: Quelle berechnet sich aus Ziel (konzeptionell nach konkret)
  - friend: analog C++
  - instanceOf
  - instantiate: Quelle erzeugt Instanzen des Ziels
  - powertype: Ziel ist Powertyp für Quelle (?)
  - refine: Quelle ist Verfeinerung des Ziels
  - use

Software Reuse

UML

Diagramme

Diagramme

- Diagramme für statische Eigenschaften
  - Klassendiagramm
  - Objektdiagramm
  - Komponentendiagramm
  - Deployment Diagram
- Mehrstufige Darstellung für Subsysteme
- Diagramme für dynamische Eigenschaften
  - Use Case Diagram
  - Sequence Diagram
  - Collaboration Diagram
  - Statechart Diagram
  - Activity Diagram

Klassendiagramm

- Ziel:
  - Statische, logische Sicht auf ein System
  - Statische Prozeßsicht (bei aktiven Klassen)
- Elemente: Klassen, Interfaces, Collaborations, Beziehungen
- Gängigster Diagrammtyp

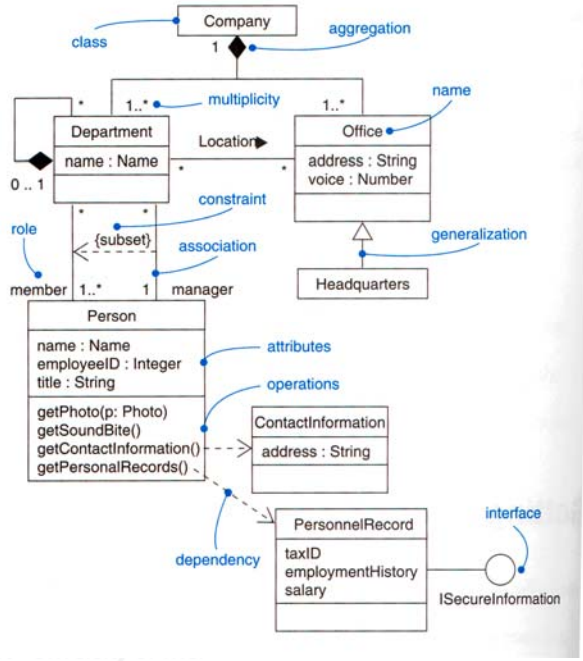
Hints and Tips

- A well-structured class diagram
  - is focused on communicating one aspect of a system‘s static design view.
  - contains only elements that are essential to understanding that aspect.
  - provides detail consistent with its level of abstraction, with only those adornments that are essential to understanding.
  - is not so minimalist that it misinforms the reader about important semantics.
- When a class diagram is drawn
  - give it a name that communicates its purpose :-)
  - lay out its elements to minimize lines that crosee :-))

Übung

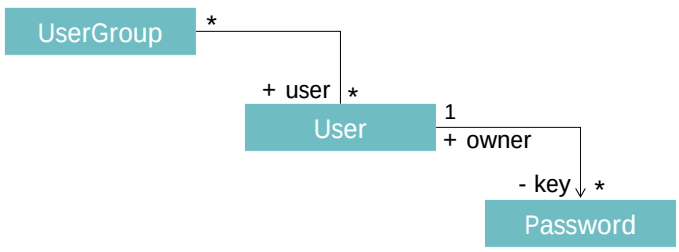
- Modellierung einer Universität
- Elemente
  - Universität
  - Fachbereich
  - Institut
  - Vorlesung
  - Seminar
  - Praktikum
  - Student
  - Promotionsstudent
  - Hiwi
  - Mitarbeiter
  - Professor

Beispiel Klassen-diagramm



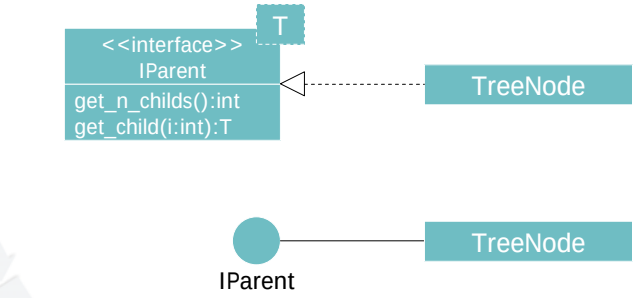
Zusätzliche Beschreibungsmittel

- Gerichtete Beziehungen
  - Allgemein sind Beziehungen bidirektional
- Sichtbarkeit von Rollen
  - +, #, - vor Rollenname definiert Sichtbarkeit



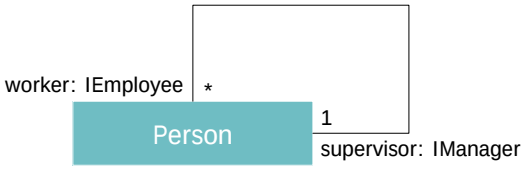
Zusätzliche Beschreibungsmittel (2)

- Realisierungsbeziehung
  - Realisierung von Interfaces
  - Collaborations (siehe später)



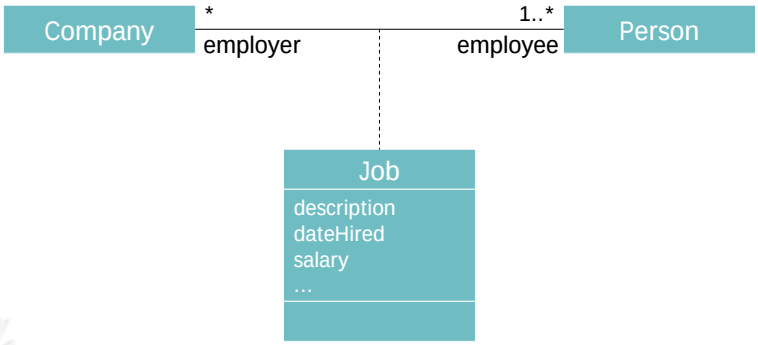
Interfaces in Beziehungen

- Nur eine Teilmenge der Klassenfunktionen in einer Beziehung sichtbar
  - Rollen werden über „Interface Specifier“ typisiert



Association Classes

- Beziehung selbst hat Eigenschaften



Constraints in Beziehungen

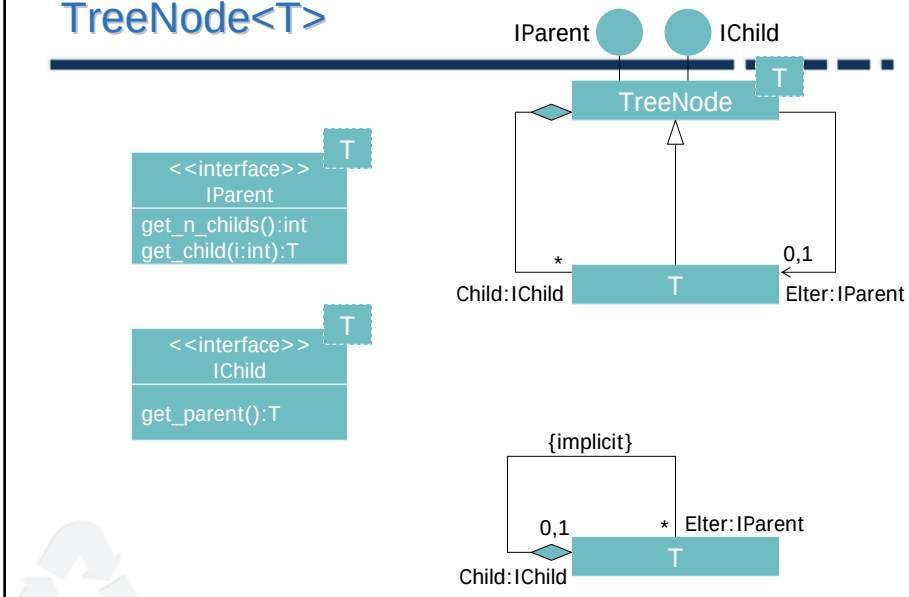
... allgemein

- implicit
  - Beziehung ergibt sich implizit (konzeptionell)
  - Beispiel: Beziehung der Basisklasse in Ableitungen

... für Enden einer Beziehungen

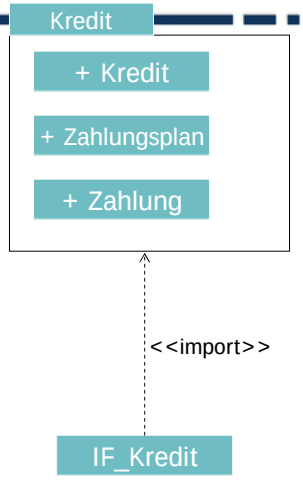
- ordered
  - Objekte einer Beziehungsmenge sind darin geordnet
- changeable
- addOnly
- frozen

TreeNode<T>



Packages revisited

- Konzeptionelle Gruppierung
- Zusätzliche Möglichkeiten
  - Sichtbarkeit (+,#,-)
  - Verschachteln
    - innere Packages sehen alles im umfassenden Package
  - Importbeziehungen
- Grundstereotypen
  - facade (Sicht auf anderes Package)
  - framework (Patterns)
  - stub
  - subsystem
  - system



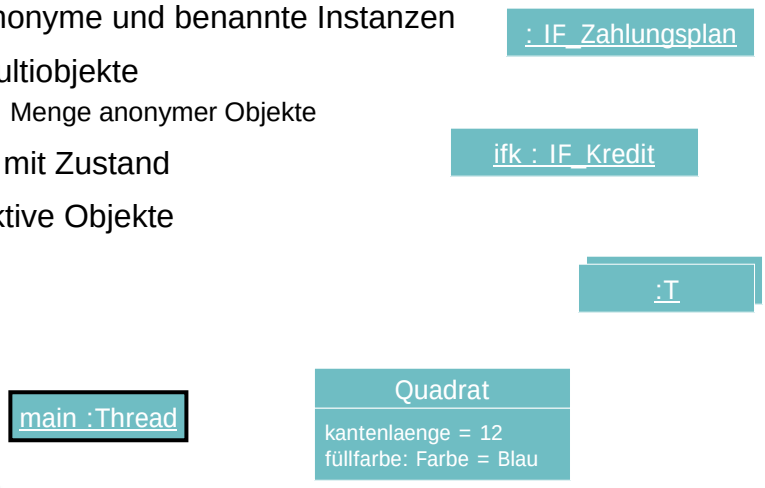
Objektdiagramm

- Illustration von Datenstrukturen, statische Schnappschüsse
- Ziel:
  - statische, logische Sicht
  - statische Prozeßsicht

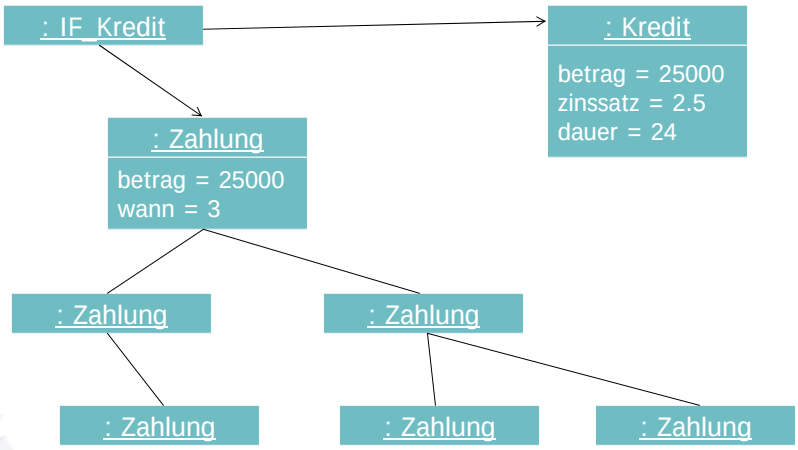
  aus der Perspektive realer oder prototypischer Fälle
- Elemente: Objekte, Beziehungen zwischen Objekten

Darstellung von Objekten

- Anonyme und benannte Instanzen
- Multiobjekte
  - Menge anonymer Objekte
- ... mit Zustand
- Aktive Objekte



und Objektbeziehungen



Hints and Tips

- A well-structured object diagram
  - is focused on communicating one aspect of a system's static design view or static process view.
  - represents one frame in the dynamic storyboard represented by an interaction diagram.
  - contains only those elements that are essential to understanding that aspect.
  - provides detail consistent with its level of abstraction: only attribute values and adornments should be exposed that are essential to understanding.
  - is not so minimalist as to misinform the reader about semantics that are important.

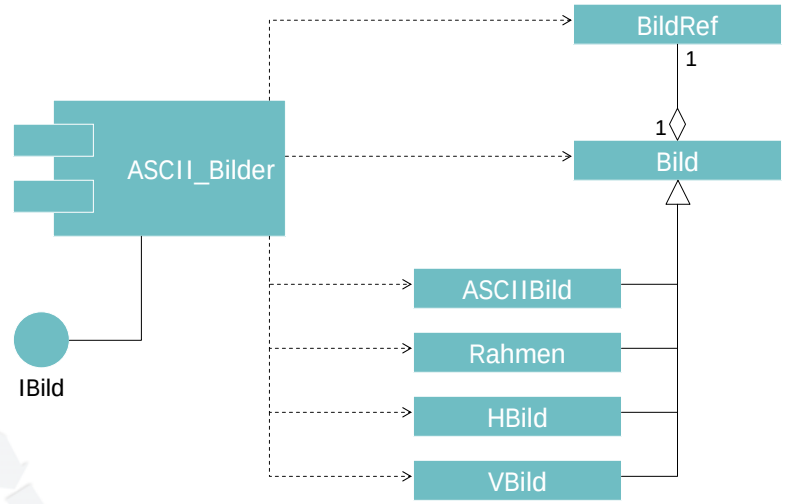
Hints and Tips (contd.)

- When drawing an object diagram
  - it should have a name that communicates its purpose.
  - its layout minimizes line crossings.
  - things that are semantically close are laid out spatially close.
  - notes and colors are used as visual vues to draw attention to important features of the diagram.
  - values, states, and roles of each object are included as necessary to communicate the intent.

Komponentendiagramm

- Ziel: Darstellung der statischen Implementierungssicht
- Elemente:
  - Komponenten
  - Beziehungen zwischen Komponenten (Uses, Depends, ...)
- Komponente abbildbar auf ein oder mehrere Klassen, Schnittstellen und Beziehungen
- Komponenten
  - exportieren Interfaces
  - importieren Interfaces

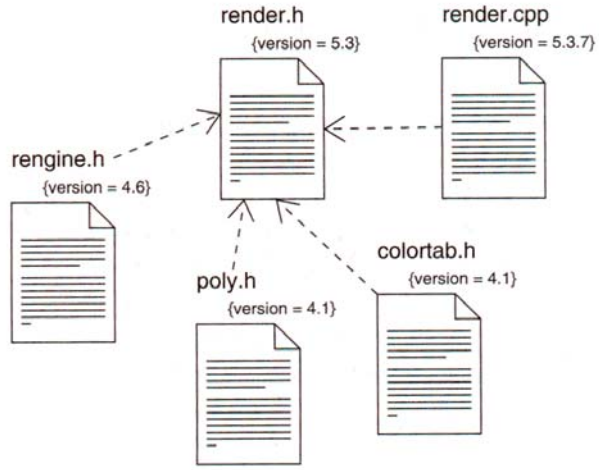
Beispiel



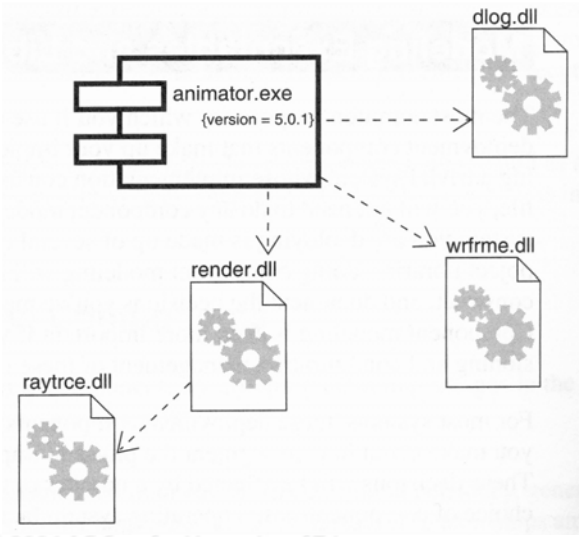
Komponententypen

- Deployment Component
  - Komponenten, die ein ausführbares System ergeben
  - Elemente: Dynamische Bibliotheken, Executables, ...
- Work Product Component
  - Elemente des Entwicklungsprozesses
  - Source Code, Datendateien, ...
- Execution Component
  - Entstehen bei der Ausführung eines Systems
  - Beispiel: COM+

Beispiel 2: Source Code

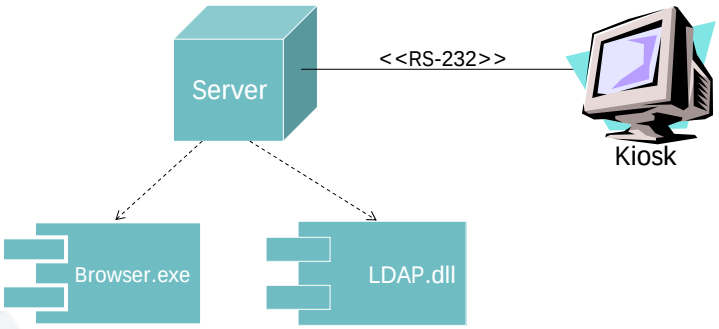


Beispiel 3: Executables & Libraries

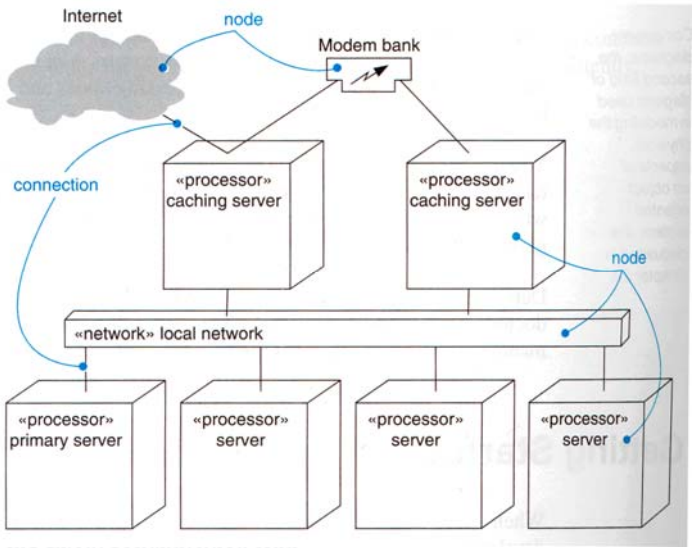


Deployment Diagram

- Ziel: Statische Sicht auf installiertes System
- Elemente: Knoten, Beziehungen zwischen Knoten

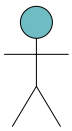


Beispiel



Use Case Diagram

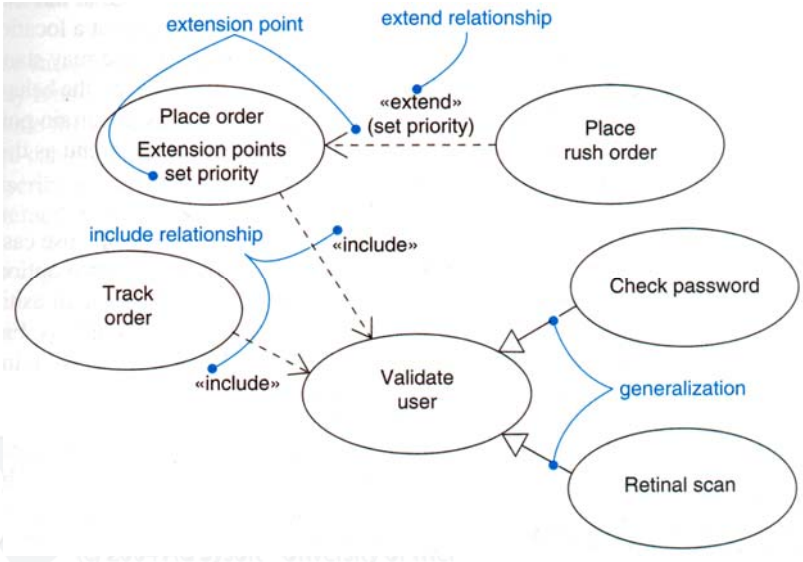
- Ziel
  - Interaktion des Systems mit der Umgebung
    - Benutzer
    - Bediener
    - Technische Prozesse
  - Dialog Entwickler mit Endbenutzer und Domänexperten
- Elemente
  - Use Cases, Aktoren (spezielle Klassen), Beziehungen



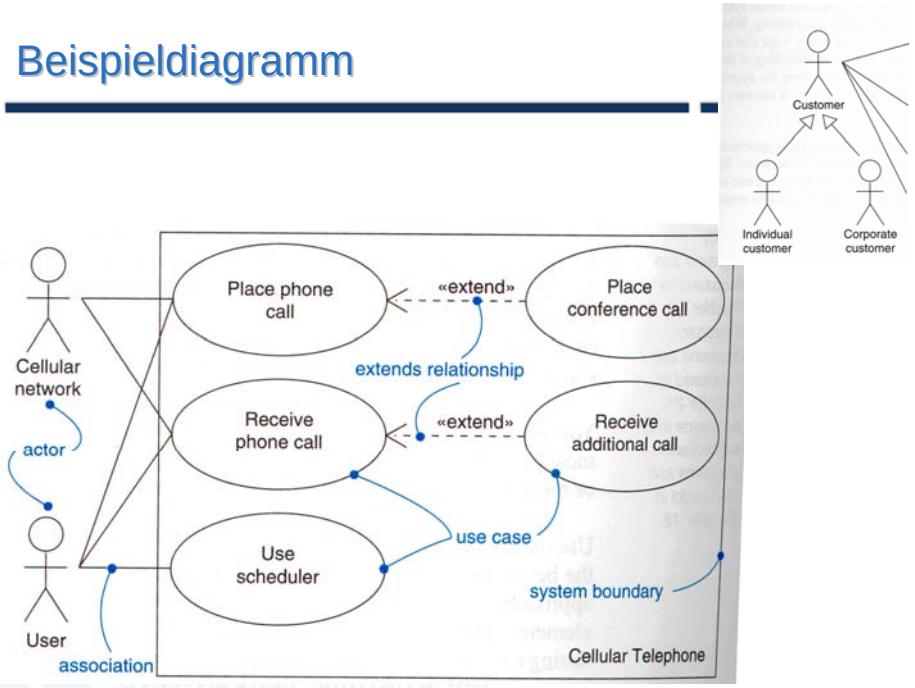
Aktor

Use Case

Beziehungen zwischen Use Cases

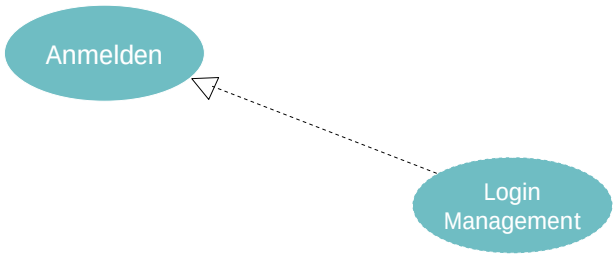


Beispieldiagramm



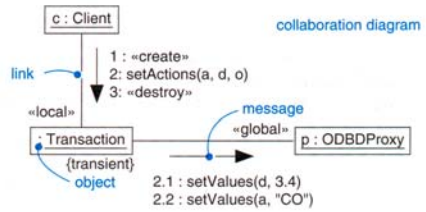
Use Case Realisierung

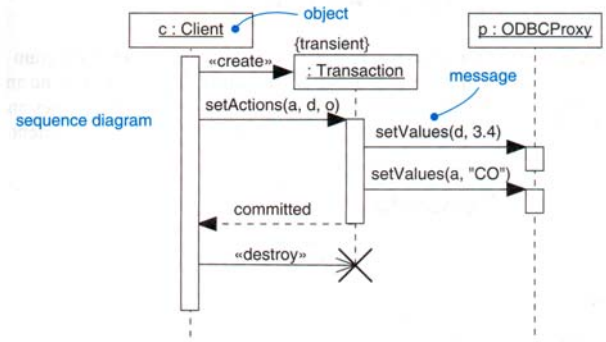
- Kollaborationen realisieren Uses Cases
- Use Cases können Attribute und Operationen haben



Sequence und Collaboration

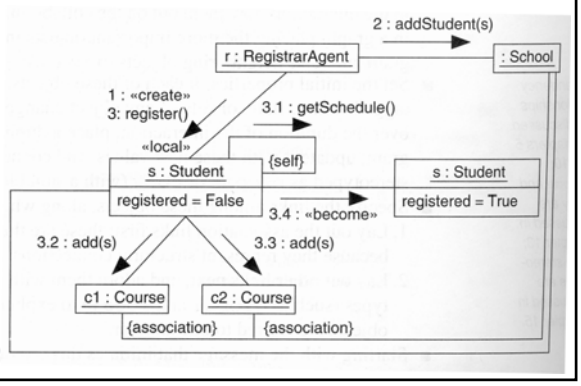
- Dynamische Abläufe mit Betonung
  - der zeitlichen Abfolge (Sequenz)
  - der Objekt-interaktion (Collaboration)
- Elemente
  - Objekte
  - Beziehungen
  - Nachrichten





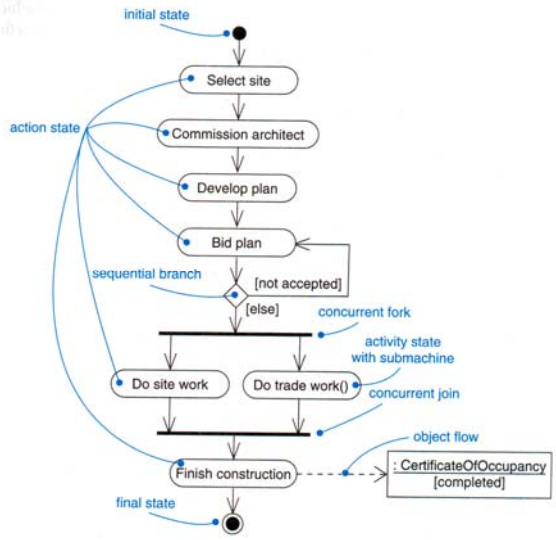
Zustandsänderungen

- Sequence
  - Objekt taucht mehrmals mit verschiedenen Zuständen auf
  - Transitionsstereotyp <<become>>
  - Zustandselement mehrfach auf Zeitlinie
- Collaboration
  - Zustandselement mehrfach inkl. <<become>>

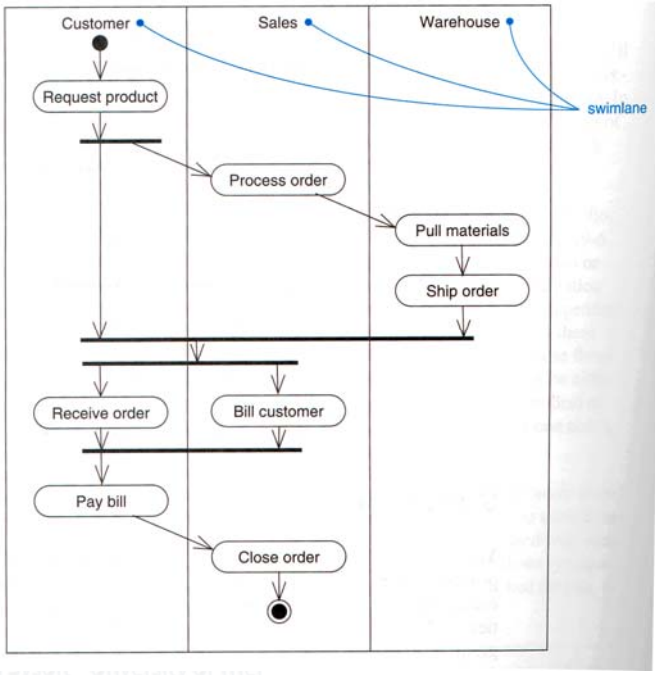


Activity Diagram

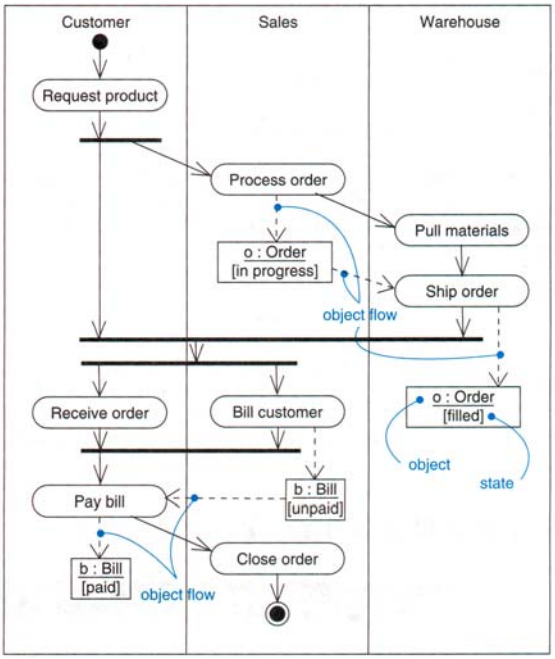
- Ziel:
  - Betonung des Kontrollflusses zwischen Objekten
  - Aktivitäten sind Subjekt (Objekte bei Sequence, Collaboration)
  - Dynamische Sicht



Schwimm-bahnen



Object Flow



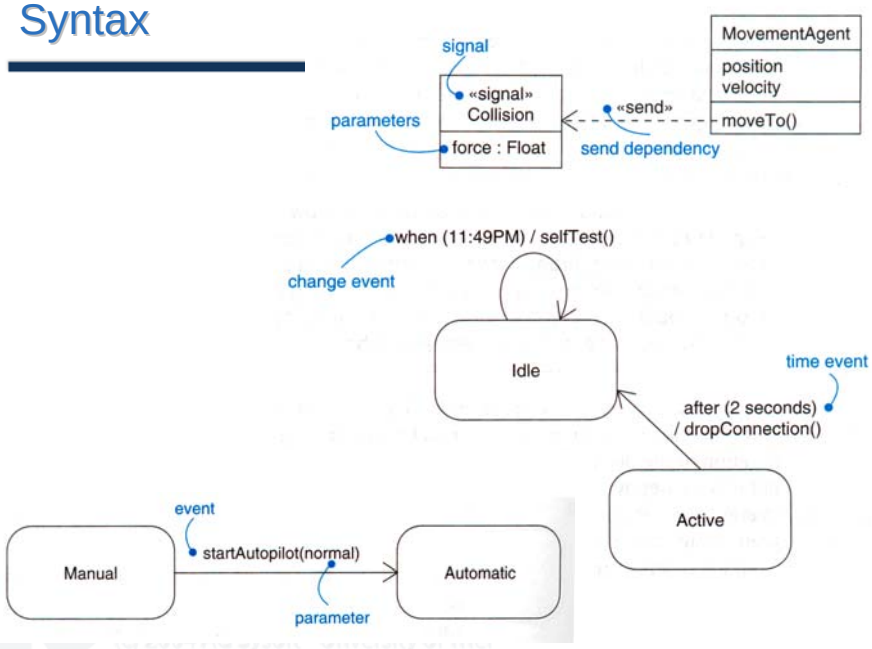
Statechart Diagram

- Ziel:
  - Modellierung des Verhaltens von Schnittstellen, Klassen und Kollaborationen
  - Dynamische Sicht
- Zustandsautomat bestehend aus
  - Zuständen
  - Transitionen
  - Ereignissen
  - Aktivitäten

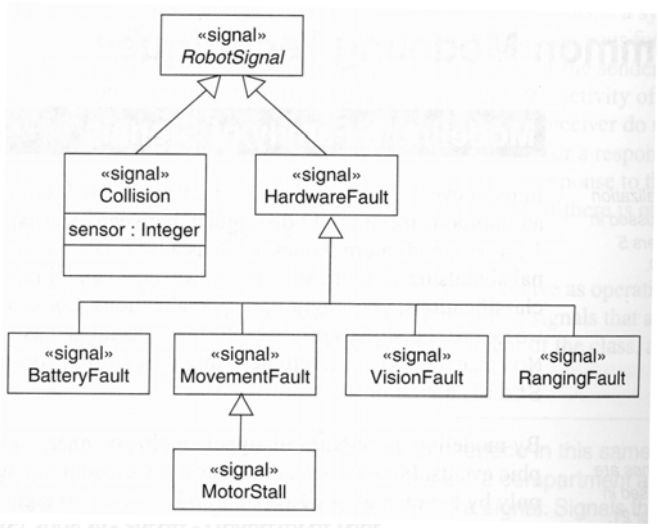
Ereignisse

- Externe Ereignisse
  - Zwischen System und Aktoren
- Interne Ereignisse
  - Zwischen Objekten innerhalb des Systems
- Ereignistypen
  - Signal: Asynchron, stereotype Klasse
  - Call: Aufruf und Ausführung einer Operation
  - Time: Verstreichen von Zeit
  - Change: Zustandswechsel / Bedingung erfüllt

Syntax

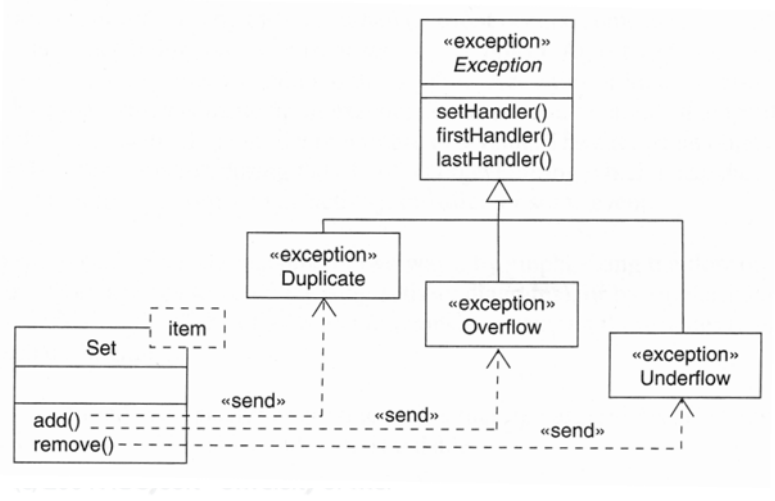


Signalfamilien

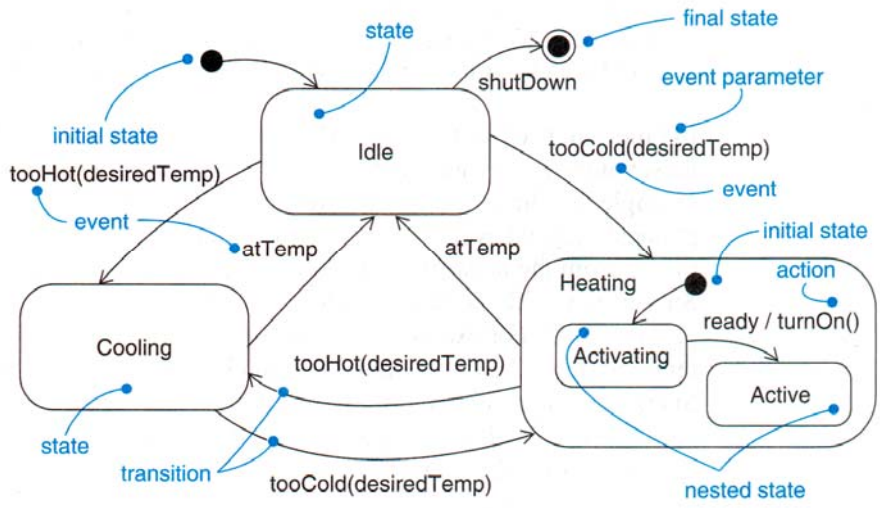


Exceptions

- Spezielle Signale

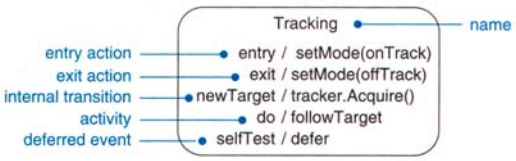


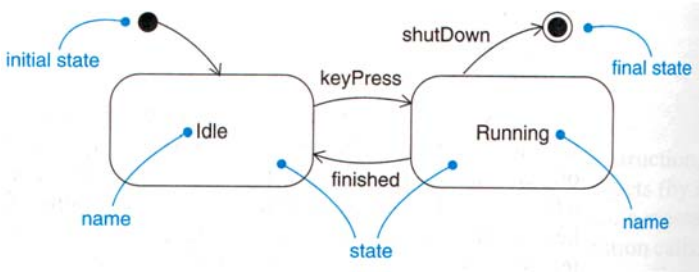
State Machine



Zustand

- Bestandteile
  - Name
  - Entry/Exit Aktionen
  - Interne Transitionen
  - Unterzustände

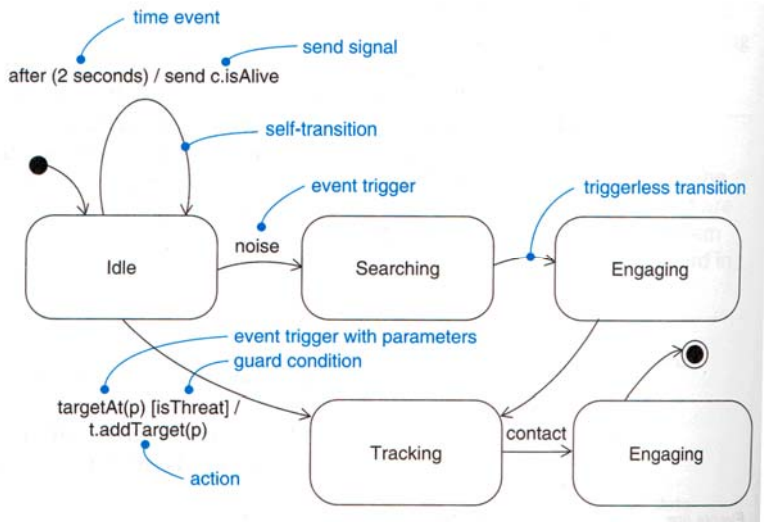




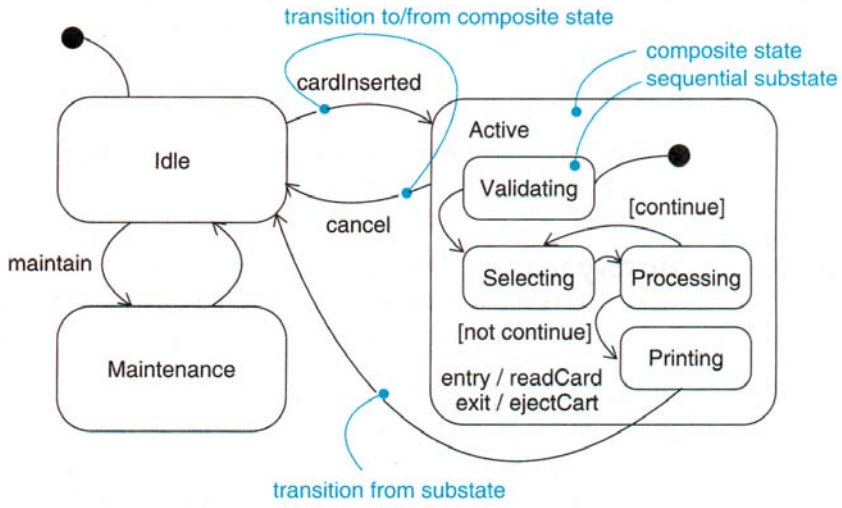
Transitionen

- Bestandteile
  - Quellzustand
  - Event Trigger: Ereignis, daß Transition potentiell auslöst
  - Guard: boolesche Bedingung für Auslösen bei Anliegen eines passenden Ereignisses
    - true: „kann“ feuern
    - false: Ereignis verpufft, falls keine andere Transition des Quellzustands mit diesem Ereignis feuert
  - Aktion: Atomare Berechnung
  - Zielzustand

Syntax

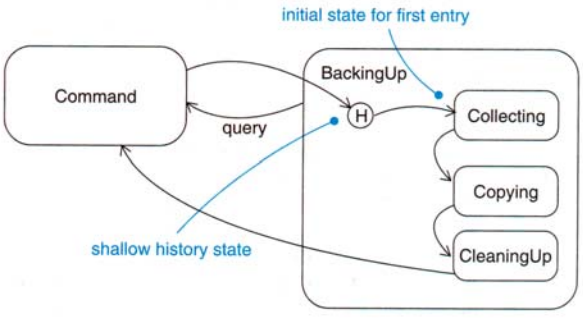


Beispiel



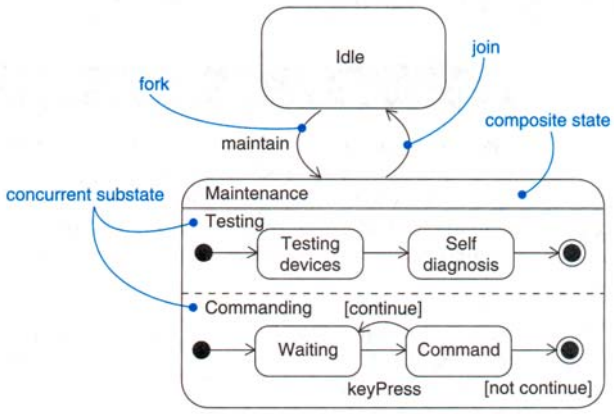
History States

- Ausgangspunkt: Transition betritt Zustand mit Unterzuständen
  - State Machine startet im inneren Startzustand
- Speicherung des letzten Substates vom vorherigen Aufruf

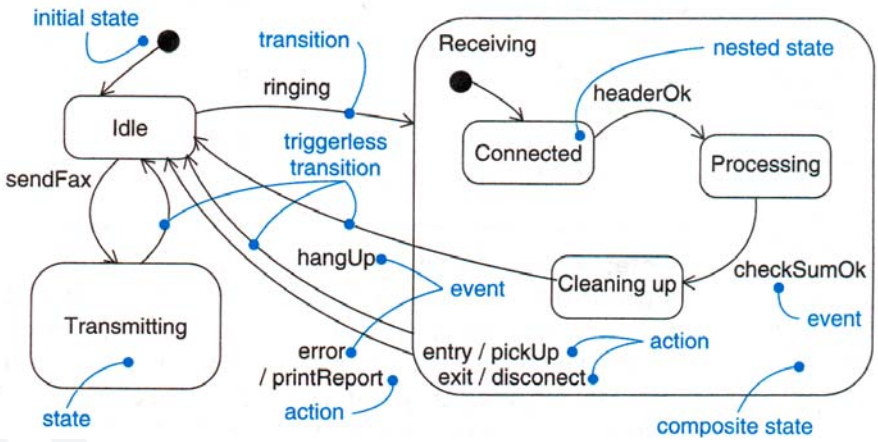


Konkurrente Substates

- Beim Eintritt einer Transition starten alle Substates in ihrem initialen Zustand



State Chart Diagram



Abstraktionsebenen

- Verschiedene Sichten
- Verschiedene Abstraktionsebenen
- Zwei Ansätze
  - Diagramme mit unterschiedlichen Detaillierungsgrad
  - Modelle mit unterschiedlichen Abstraktionsgrad