PiSoftware
www.pisoftware.de

Objektorientierte Softwareentwicklung

Analyse, Design und Programmierung

Ralf Pichocki

15. Auflage
2002

PiSoftware
www.pisoftware.de

15. Auflage, 2002

Bahnhofstraße 190
D-45770 Marl-Sinsen

Tel +49 2365 880480
Fax +49 2365 880482
info@pisoftware.de
http://www.pisoftware.de

Bearbeitung:
Gestaltung:
Druck:

Inhaltsverzeichnis

Einführung

Der Objektorientierte Ansatz nimmt in der Analyse- und Designphase von Software-Projekten rasch an Bedeutung zu. Zur Beschreibung objektorientierter Systeme ist die mittlerweile genormte „Unified Modeling Language“ (UML) geeignet, eine grafische Notationsweise, mit der alle Phasen, nämlich Analyse, Design und Programmierung, abgedeckt werden können.

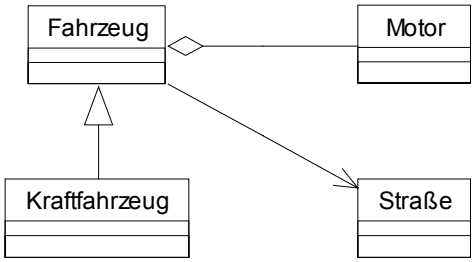
Objektorientierte Vorgehensweise

- Problembeschreibung
- Analyse:

  Objekte des Problembereiches

  sowie ihre Beziehungen untereinander als konzeptionelles Modell der Anwendung

  Beispiel: **Kraftfahrzeug** *ist* ein **Fahrzeug**, *hat* einen **Motor** und *nutzt* eine **Straße**

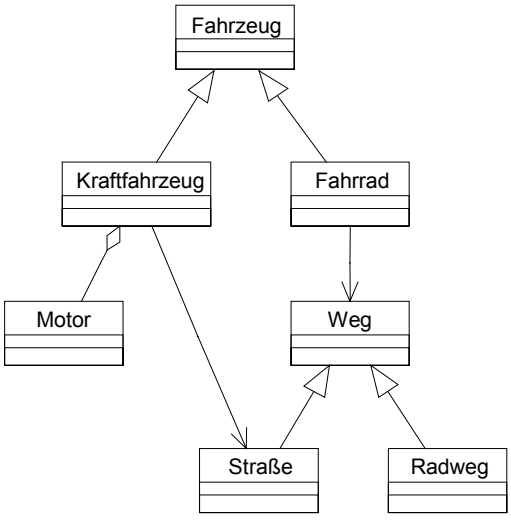


- Design:

  DV-technische Umsetzung, insbesondere mit den Zielen

  Wiederverwendbarkeit und Wartbarkeit

  Beispiel: : **Fahrrad** *ist* ein **Fahrzeug** und *nutzt* einen **Weg** (**Straße** oder **Radweg**)



- Programmierung:

  Umsetzung der Ergebnisse von Analyse und Design mit Hilfe objektorientierter oder objektbasierter Programmiersprachen, -tools und –systeme

  Quellcodebeispiel in Programmiersprache JAVA

```
public class Radweg extends Weg {
      //...
}

public class Fahrrad extends Fahrzeug {
      Weg fahrtStrecke = null;
      //...
}
```

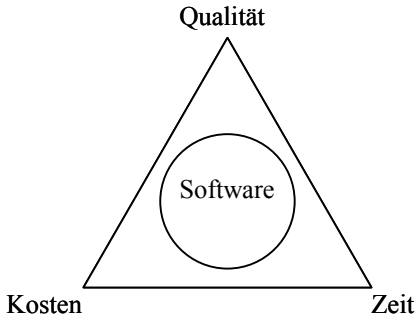
- Verschiedene Methoden: Peter Coad/Edward Yourdon, Grady Booch und andere

Überblick

- Objektorientierte Vorgehensweise in Analyse und Design, Methodenüberblick
- Prinzipien der objektorientierten Analyse
  - ❖ Identifikation von Objekten und Klassen
  - ❖ Identifikation ihrer Beziehungen untereinander (Vererbung, Assoziation, Aggregation)
  - ❖ Finden und Festlegen dynamischer und kommunikativer Objektbeziehungen
  - ❖ Beschreibung von Funktionen und Attributen der Objekte und Klassen
- Prinzipien des objektorientierten Designs
  - ❖ Modellierung elementarer Systembausteine
  - ❖ Abkapselung von Systemteilen, Definition von Schnittstellen
  - ❖ Wartbarkeit und Wiederverwendbarkeit
- Beispiele und Übungen

Grundlagen der Softwareentwicklung

Software-Entwicklung im Spannungsfeld von Qualität – Kosten – Zeit



Qualitätskriterien

- Funktionserfüllung
- Zuverlässigkeit 😐
- Robustheit ☺
- Erweiterbarkeit ☺
- Wiederverwendbarkeit ☺
- Kompatibilität 😐
- Portabilität 😐
- Benutzerfreundlichkeit
- Effizienz
- Wartbarkeit ☺

Probleme der Software-Entwicklung

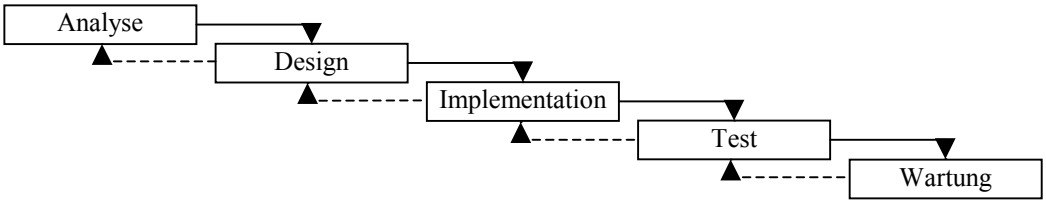
- Ende der 60er Jahre: „GOTO-Krise“, Lösung: strukturierte Programmierung
- Ende der 80er Jahre: „Wiederverwendbarkeits-Krise“, Lösung: Objektorientierung

Prinzip der Wiederverwendbarkeit

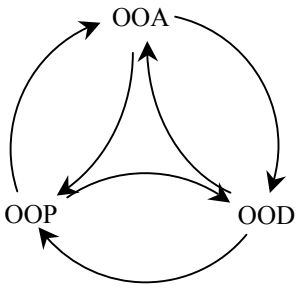
- Prozedurale Pogrammierung: Standarfunktionen und Module
- Objektorientierte Pogrammierung: Klassen und Klassenbibliotheken

Vorgehensmodelle

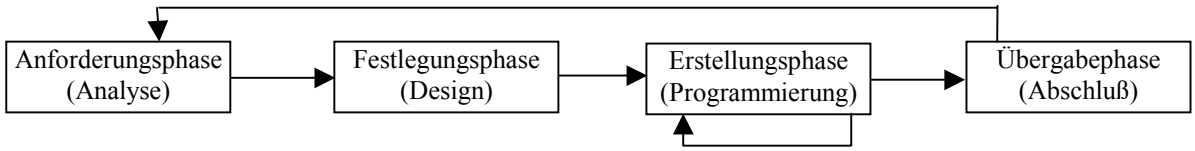
- Wasserfallmodell



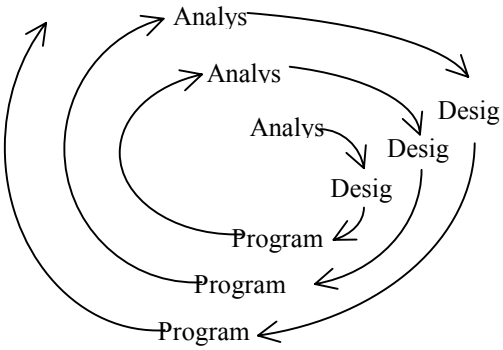
- Baseballmodell



- 4-Phasen-Modell



- Spiral-Modell



- Anwendungsfallgetriebene architekturzentrierte iterativ-inkrementelle Entwicklung

Objektorientierte Vorgehensweise

- Neue Art Probleme zu verstehen und Lösungen zu finden
- Blick auf Objekte als Einheit von Funktionen und Daten

  Beispiel **Auto.heizen()** und **Haus.heizen()** sind verschieden
- Softwaresysteme als Menge miteinander kommunizierender Objekte,

  wobei auch Klassen Objekte sein können
- Betonung auf Datenabstraktion, Informationskapselung und Wiederverwendbarkeit
- Neue Methoden für Analyse, Design und Programmierung

Eigenschaften und Ziele

Eigenschaften

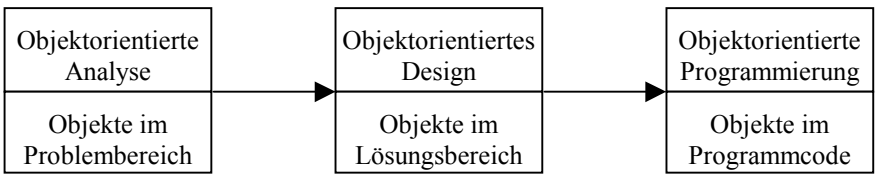
- Bessere Abbildung der realen Welt
- Wiederverwendbarkeit bereits entwickelter Module
- Einfache Modifikation und Wartung (Kapselung, Schnittstellen)
- Schnell verfügbare Prototypen, Entwicklung inkrementell und iterativ
- Besonders geeignet für offene Client/Server-Lösungen

Ziele

- Reduktion des Aufwandes bei Entwicklung und Wartung
- Wiederverwendbare Software-Bausteine und -Bibliotheken

Objektorientierter Lebenszyklus

- Analyse: Objekte im Problembereich
- Design: Objekte im Lösungsbereich
- Programmierung: Objekte im Programmcode

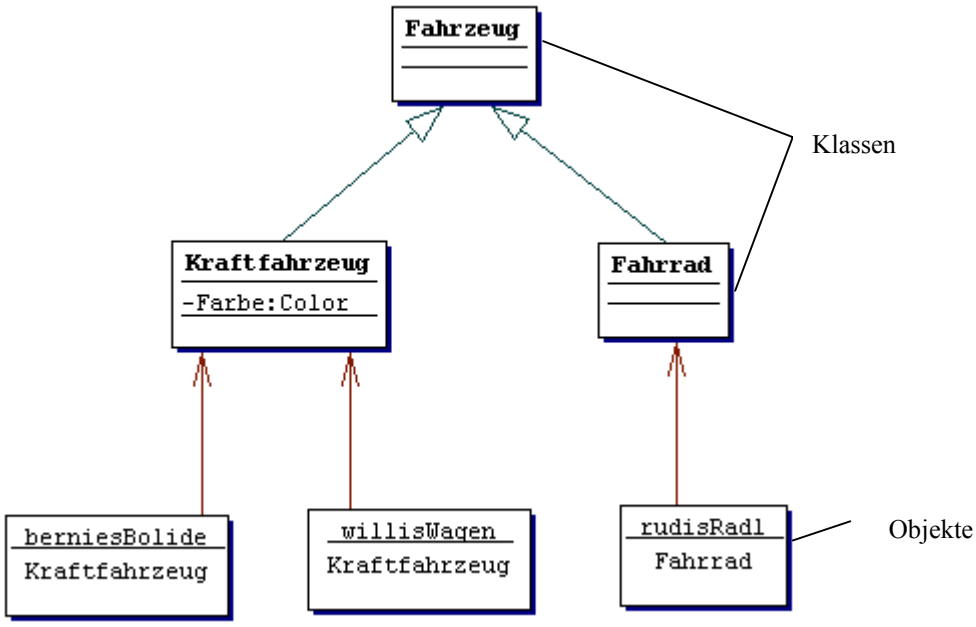


- Beschreibung aller Phasen mit Hilfe der UML ist möglich

Konzepte der objektorientierten Entwicklung

Objekte

- Funktionen
- Eigenschaften
- Eigenschaftswerte
- Identität



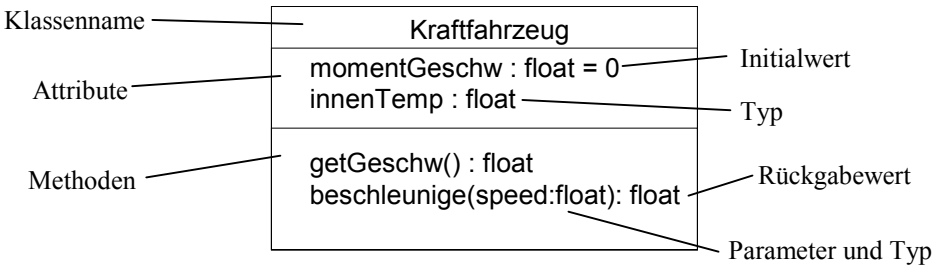
Klassen

- Gemeinsamkeiten
- Baupläne
- Datenabstraktion und Kapselung
- Klassifizierungsproblematik

Kraftfahrzeug
momentGeschw : float innenTemp : float
beschleunige(zielGesch : float) heize(zielTemp : float)

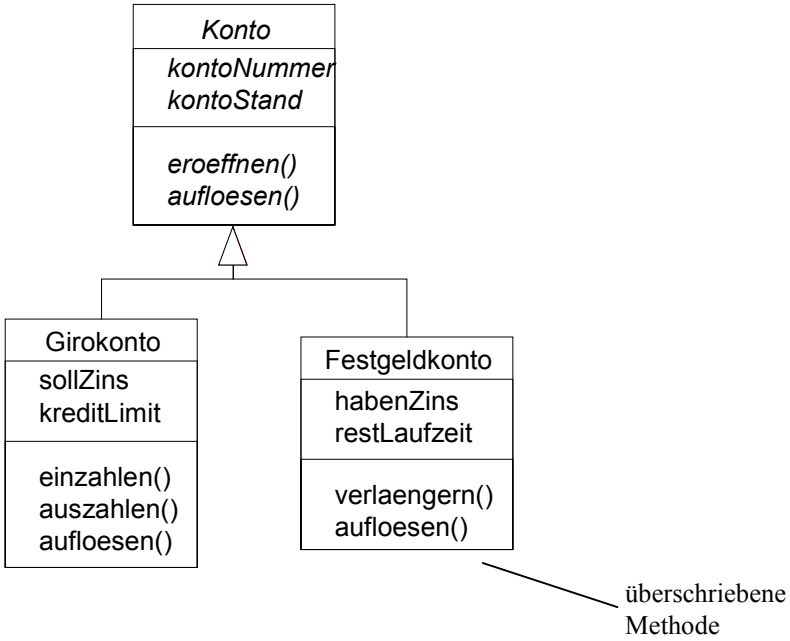
Haus
innenTemp : float
heize(zielTemp : float)

- Darstellung in der der UML mit Hilfe von Rational Rose 98



Vererbung

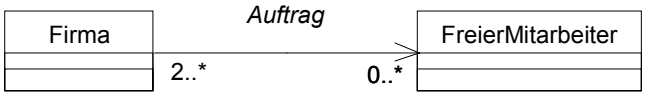
- spezielle Klassen
- erweiterte Funktionalität



- abstrakte Klassen

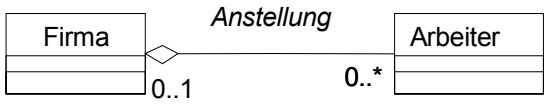
Assoziation

- „Nutzt“-Eigenschaft
- Kardinalität
- Rolle



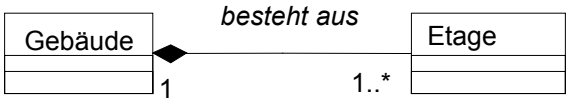
Aggregation

- „Hat“-Eigenschaft
- Kardinalität
- Kann- oder Muß-Beziehung
- Komponenten auch allein existenzfähig



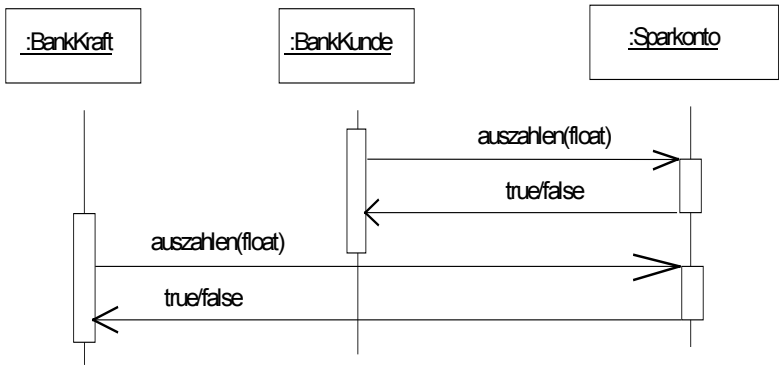
Komposition

- Form der Aggregation
- „Besteht aus“ -Eigenschaft
- Kardinalität
- Muß-Beziehung
- Komponenten allein nicht existenzfähig



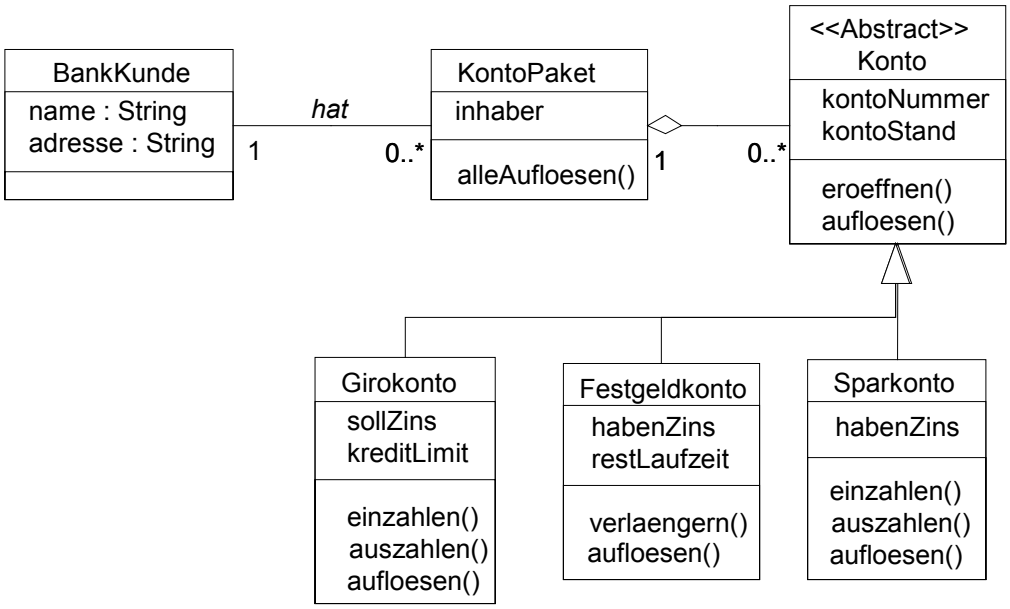
Nachrichten

- Senden entspricht Methodenaufruf
- Kommunikationspfad (Link) ist notwendig
- Sender *kennt* und *sieht* den Empfänger
- Entspricht Einschreiben *mit Rückantwort* aber *ohne Absender*



Polymorphie

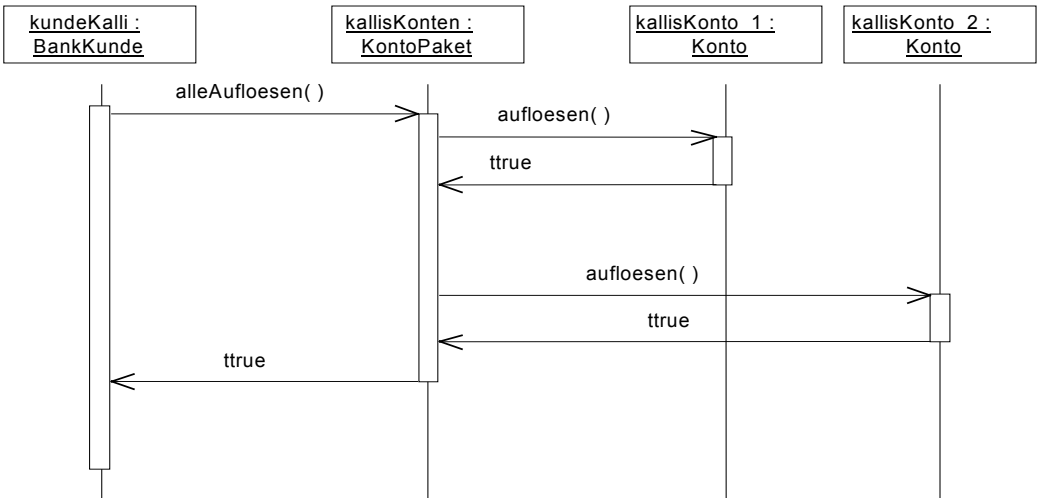
- Statische Polymorphie
- Dynamische Polymorphie



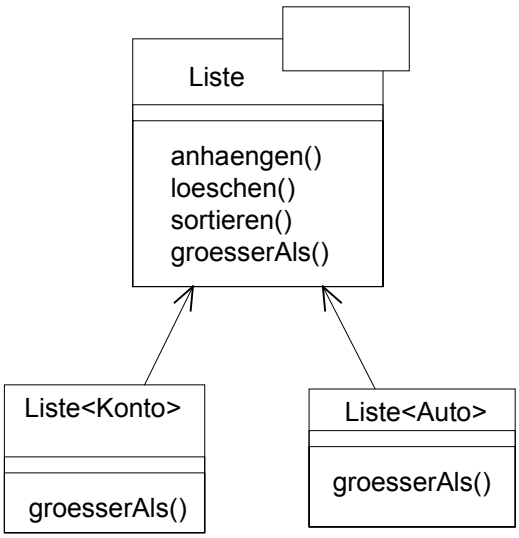
- Senden derselben Nachricht an Objekte verschiedener Klassen derselben Basisklasse
- trotzdem Aufruf der richtigen Methode

Generizität

- anderer Weg zur Wiederverwendbarkeit



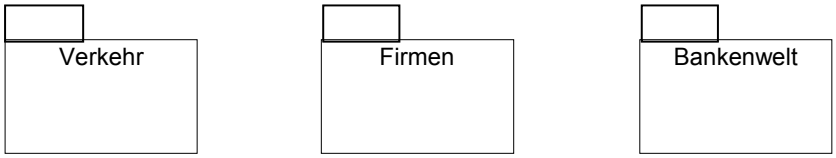
- hauptsächlich Collection-Klassen



- aber auch andere Klassen, die Objekte als Ganzes bearbeiten

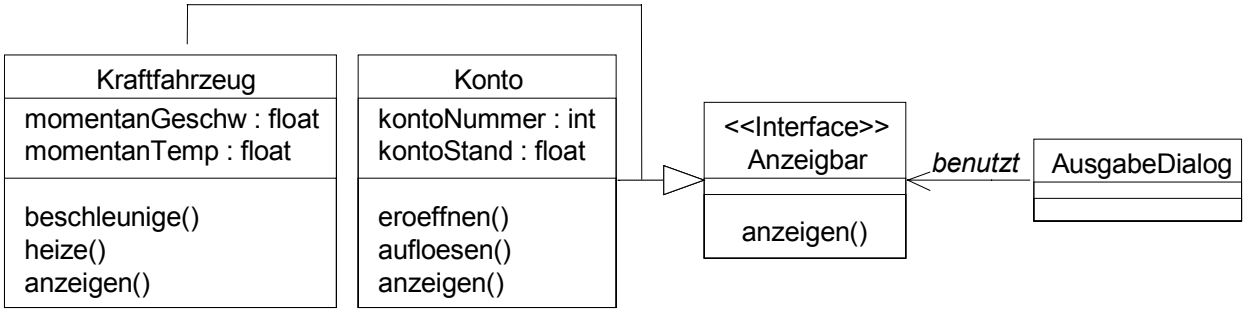
Subsysteme

- benannte, logische Zusammenfassungen zu Bibliotheken oder Modulen

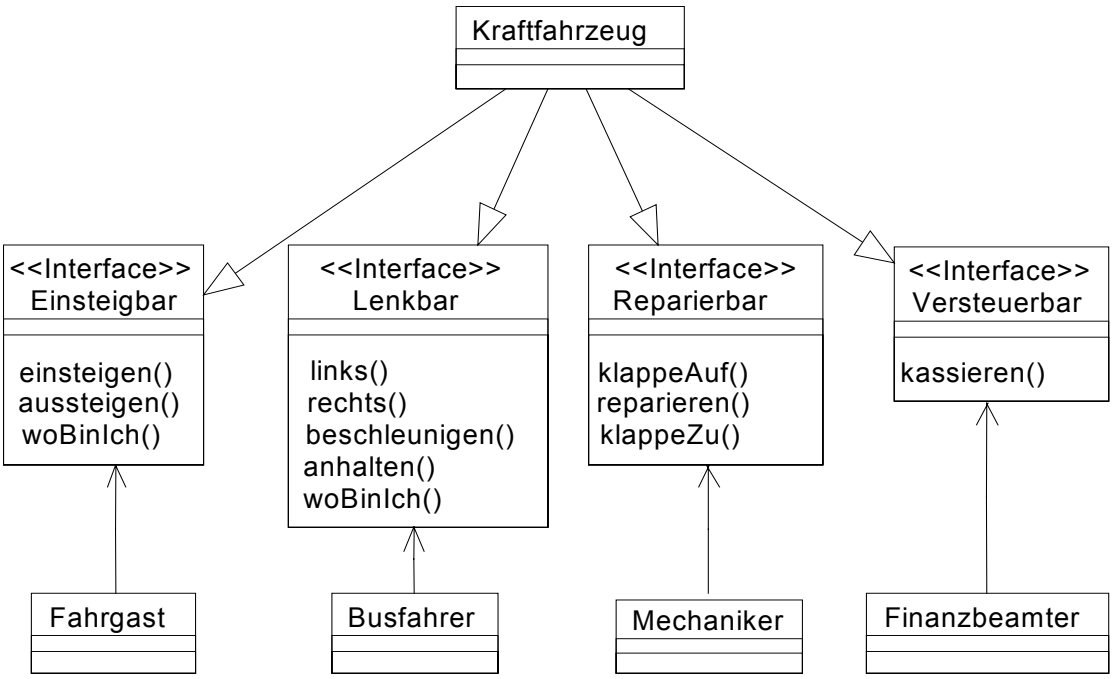


Schnittstellen

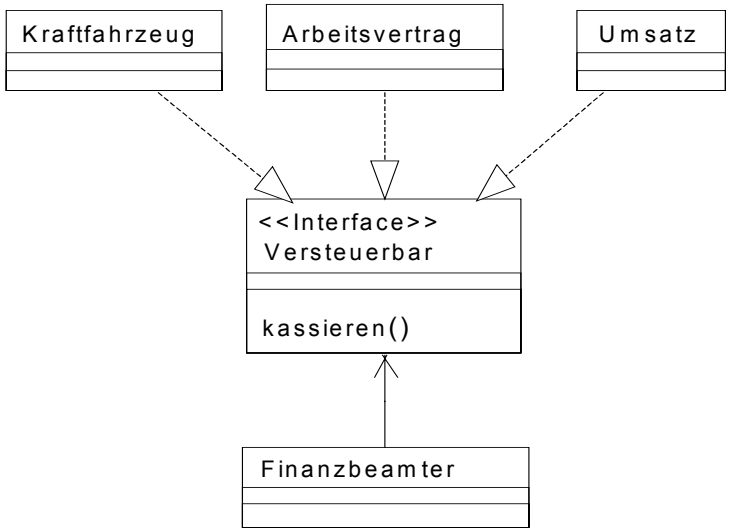
- Bereitstellung von Verhaltensmustern, die erst später implementiert werden
- Beispiel: Schnittstelle, für einheitlichen Zugriff auf Objekte verschiedener Klassen



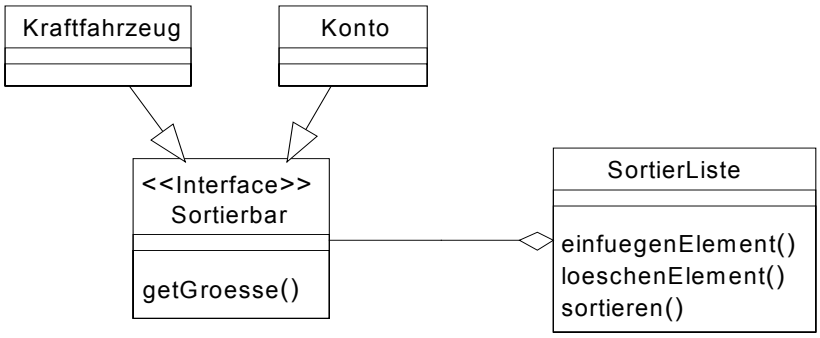
❖ Beispiel: Schnittstelle zum eingeschränkten Zugriff auf Objekte einer Klasse



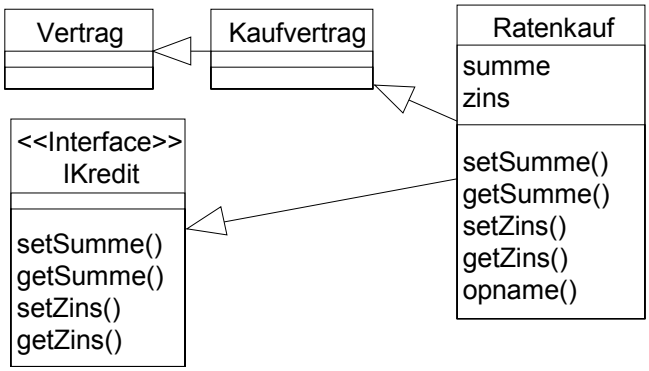
❖ Beispiel: Dieselbe Schnittstelle bietet einheitlichen Zugriff auf verschiedene Klassen



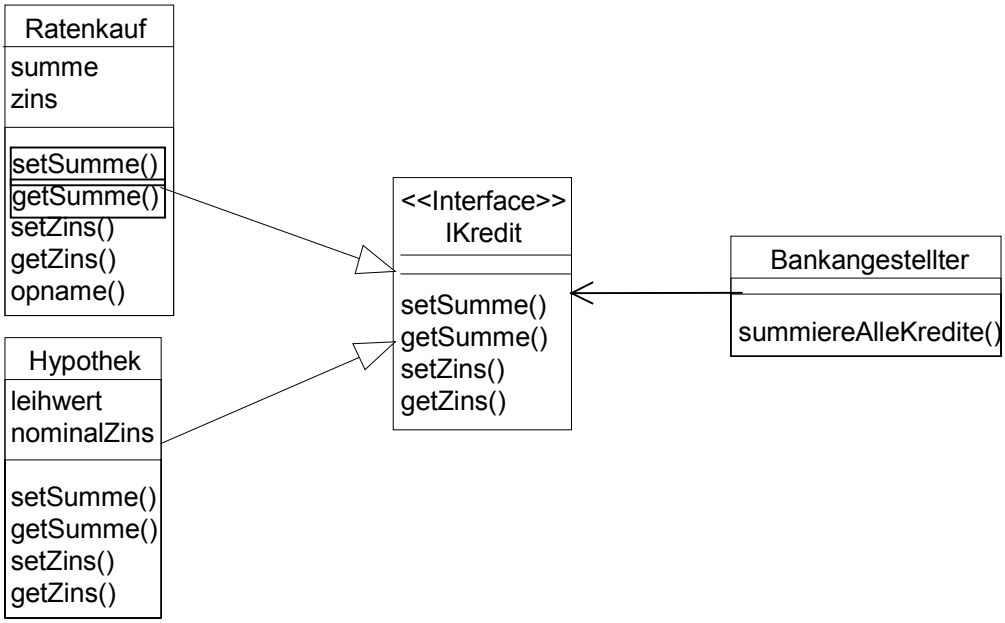
❖ Beispiel: Schnittstelle „Sortierbar“ statt Verwendung einer generischen Klasse



❖ Beispiel: Schnittstelle zur Simulation von Mehrfachvererbung



❖ Beispiel: Dieselbe Schnittstelle bietet einheitlichen Zugriff auf verschiedene Klassen



Objektorientierte Analyse

Ziele

- Probleme verstehen, Organisation aufdecken
- Komplexität auflösen, Dekomposition einer Anwendung
- Anforderungen beschreiben

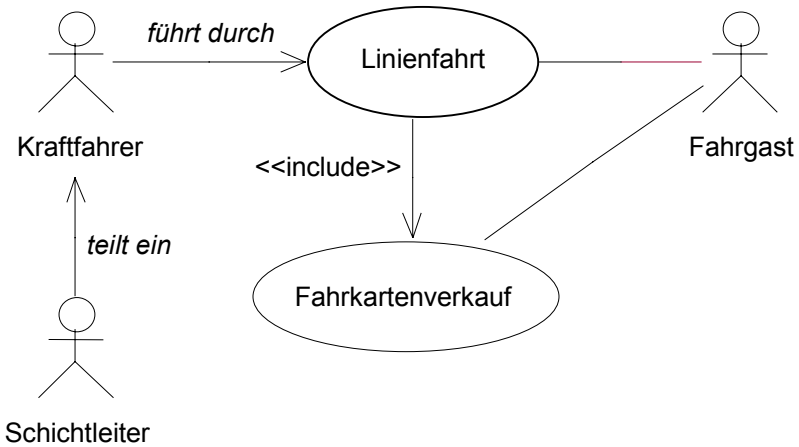
Vorgehensweise

- Vollständig dokumentiertes logisches Modell des relevanten Realweltausschnittes
- Darstellung des Anforderungsverhaltens in verschiedenen Modellen und Diagrammen
- Ermittlung der Abstraktionen, die den Anforderungen zugrunde liegen
- Objekte und Objektverhalten finden und beschreiben
- Statische Zusammenhänge zwischen Objekten finden (Beispiel: Bus – Motor)
- Dynamische Zusammenhänge zwischen Objekten finden (Beispiel: Bus – Fahrgast)

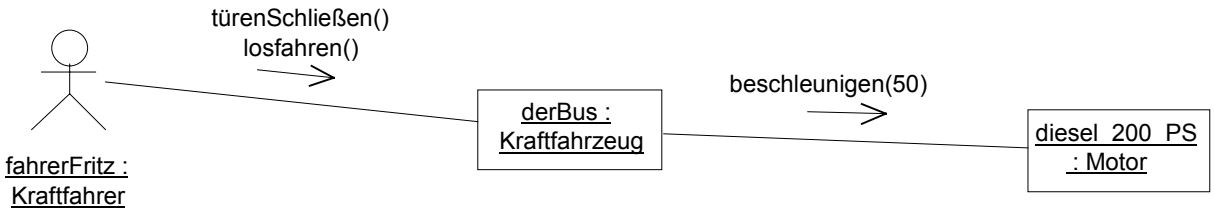
Ergebnisse

Statische Sicht:

- Objekte
- Klassen
- Strukturen
- Nachrichtenvebrindungen
- Darstellungsmittel: Anwendungsfalldiagramme



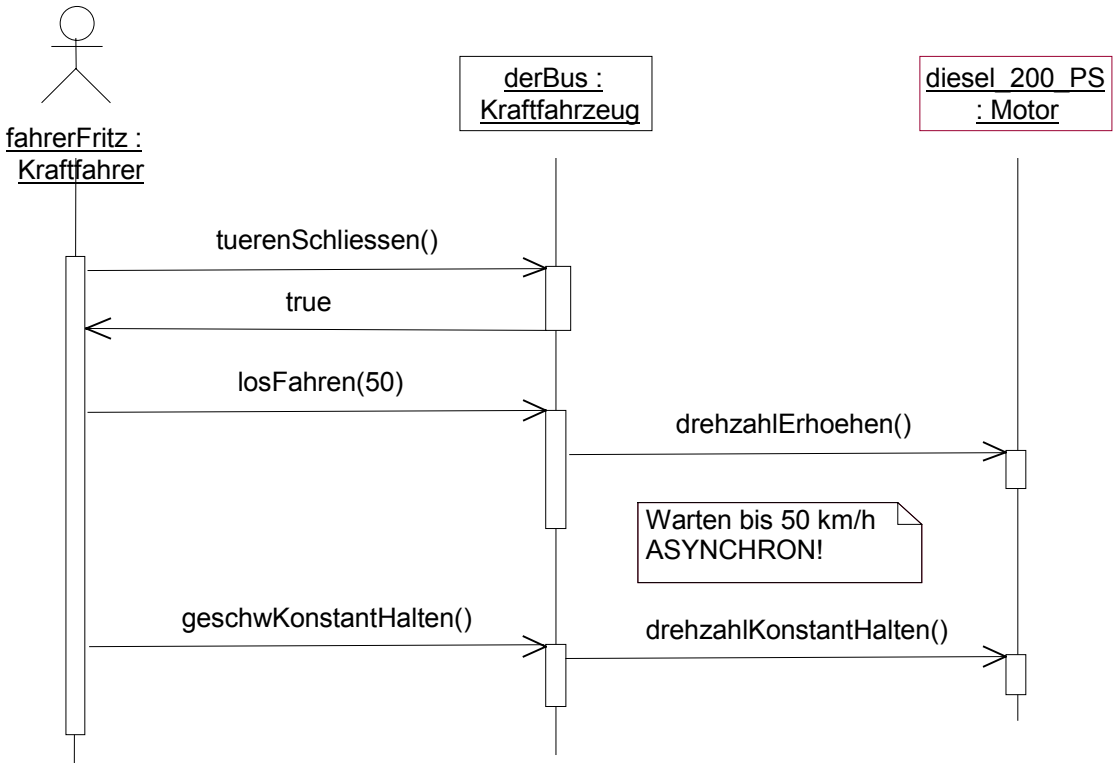
- Darstellungsmittel: Objektdiagramme
- Darstellungsmittel: Klassendiagramme
- Darstellungsmittel: Kollaborationsdiagramme



- Darstellungsmittel: Moduldiagramme

Dynamische Sicht:

- Objektverhalten
- Kommunikation
- Folgen von Methodenaufrufen
- Kontrollfluß
- Darstellungsmittel: Sequenzdiagramme



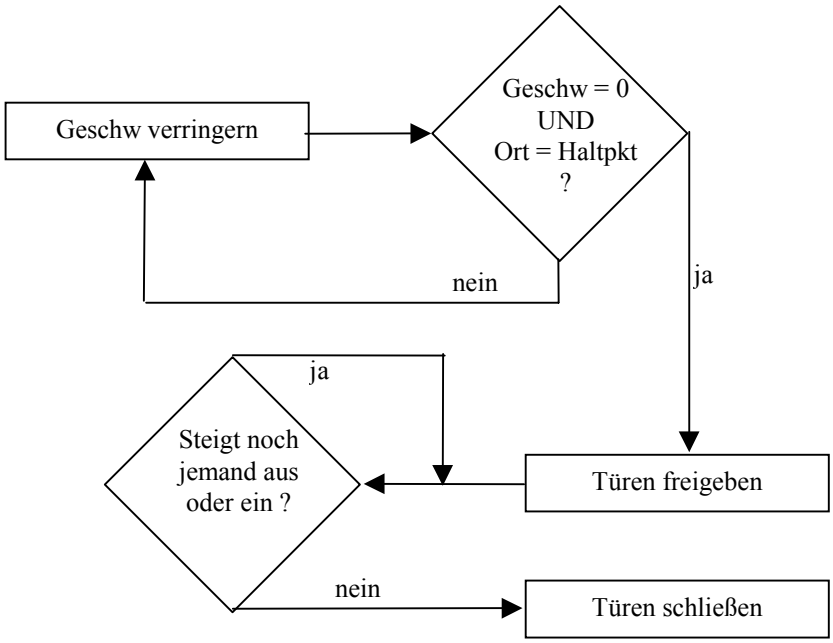
- Darstellungsmittel: Zustandsdiagramme

Funktionale Sicht:

- Algorithmen

```
Algorithmus „Bushaltestelle anfahren“:
Erreiche Haltebucht der Haltestelle,
dabei verringere Geschwindigkeit auf Null,
gib alle Türen frei,
warte bis niemand mehr ein- oder aussteigt,
schließe die Türen.
```

- Realisierung der Methoden
- Darstellungsmittel: Ablaufdiagramme



- Darstellungsmittel: Struktogramme
- Darstellungsmittel: Pseudocode

```
Pseudocode „Bushaltestelle anfahren“:
WHILE ( Geschw > 0 ) AND ( NOT isHaltestelleErreicht() )
      DO Geschw = MAX( Geschw – 10, 0 )
gibFreiTueren()
WHILE ( jemandSteigtEin() OR jemandSteigtAus() )
      DO warte( 1 )
schliesseTüren()
```

Statisches Modell

Vorgehensweise zur Modellierung

- Identifizieren von Objekten und Klassen
- Identifizieren von Verantwortlichkeiten und Attributen
- Identifizieren von Klassen- und Objektbeziehungen
- Identifizieren von Strukturen
- Identifizieren von Methoden und Nachrichtenverbindungen
- Zerlegen in Teilsysteme

Identifizieren von Objekten und Klassen

- Personen, Rollen, Orte, Dinge, Geräte, Einrichtungen
- Beschreibungen, Konzepte
- Ereignisse, Interaktionen
- Organisationen, Einheiten, Systemstrukturen
- Relevanz für das zu modellierende System (Realweltausschnitt!)
- Klassen stellen Dienste bereit, die für die Gesamtfunktionalität benötigt werden
- Klassen enthalten Infomationen, deren Speicherung innerhalb des Systems erforderlich ist
- Klassen werden durch mehr als ein Attribut beschrieben
- Klassen bilden zunächst nur den Problembereich ab
- Klassen enthalten zunächst keine Design- oder Implementierungskonstrukte

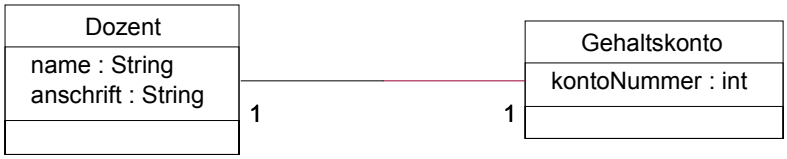
Identifizieren von Attributen und Verantwortlichkeiten

- Objekte einer Klasse haben gemeinsames Verhalten und Menge von Zuständen
- Objektverhalten definiert durch Methoden, die für Objekte aufgerufen werden können
- Zustand ist bestimmt durch die Werte seiner Eigenschaften
- Eigenschaften sind Attribute, aber auch Beziehungen zu anderen Objekten

- Prüffrage: Wie wird das Objekt allgemein und wie im Problembereich beschrieben?
- Prüffrage: Wie ist die Systemverantwortlichkeit des Objektes?
- Prüffrage: Welche Informationen werden von den Methoden innerhalb des Objektes benötigt?
- Attributspezifikation: Name, Datentyp und Wertebereich, Kurzbeschreibung

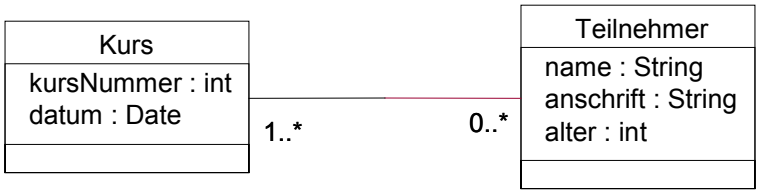
Identifizieren von Klassen- und Objektbeziehungen

- Spezifikation: Name der Klasse, beteiligte Klassen, Kardinalitäten. Kurzbeschreibung
- Objekte sind zur Erfüllung ihrer Aufgaben auf die Kommunikation mit anderen Objekten angewiesen
- Kommunikationspfad ist Verbindung (Link) zwischen zwei Objekten
- Kardinalität der Beziehungen
- Beispiele für Klassenbeziehungen:
  - Dozent - Gehaltskonto



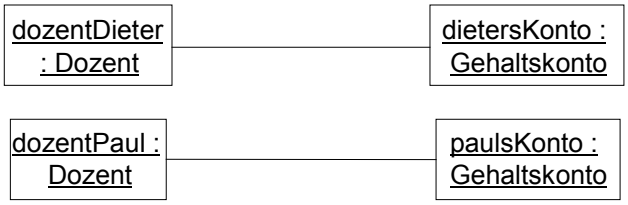
  - Kurstyp - Kurs

  - Kurs - Teilnehmer

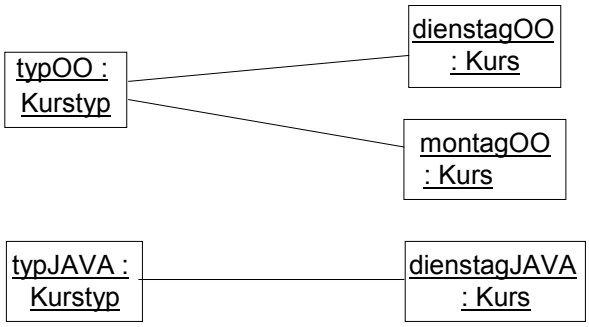


❖ Beispiele für Objektbeziehungen:

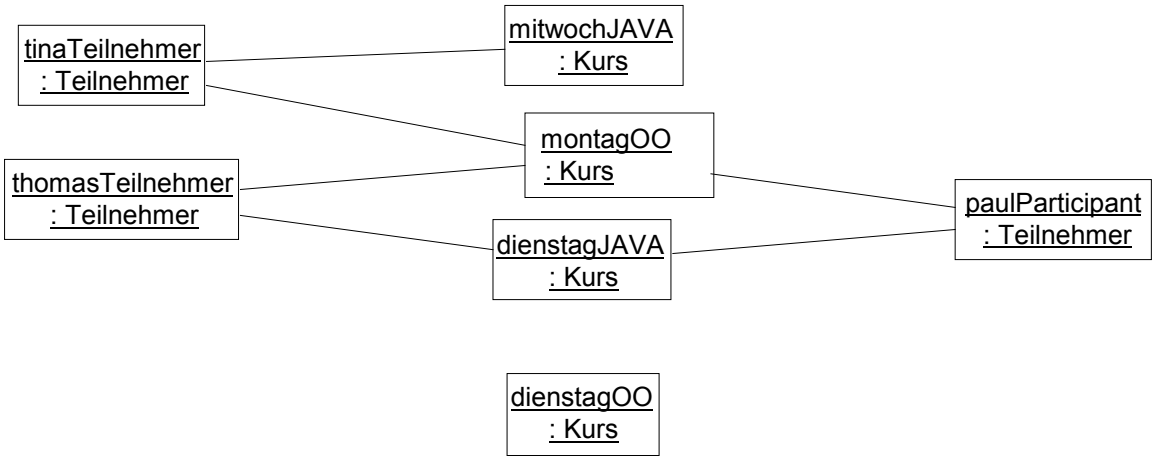
- dozentDieter – dietersKonto, dozentPaul – paulsKonto



- typOOKurs – montagOO, dienstagOO, typJAVA – dienstagJAVA



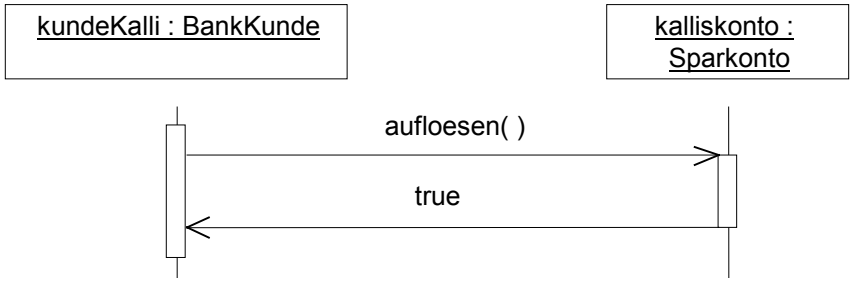
- Teilnehmer –Kurse



Identifizieren von Strukturen

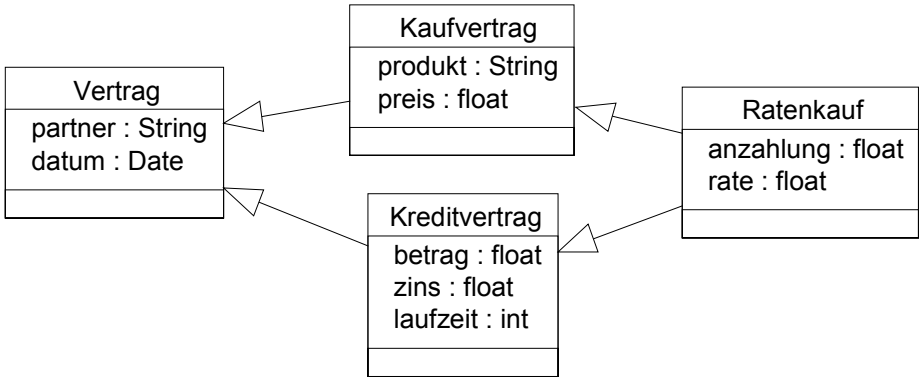
Vererbungstrukturen

- Objekte der abgeleiteten Klasse erben alle Attribute, Objektverbindungen und Methoden der Basisklasse
- Überall kann statt eines Objektes der Basisklasse auch eines der abgeleiteten Klasse verwendet werden. (Die Umkehrung gilt nicht!)
- Alle Nachrichten an Basisklassenobjekte können auch von Objekten einer abgeleiteten Klasse bearbeitet werden: da jedes *Konto* die Nachricht *gibStand()* verarbeiten kann, kann auch ein davon abgeleitetes *Sparkonto* diese Nachricht verarbeiten.
- Ein abgeleitetes Objekt kann die Objektverbindungen der Basisklasse nutzen, um selbst Nachrichten zu versenden: da jeder *Bus* einen Link zu seinem Motor hat, kann auch ein davon abgeleiteter *Linienbus* diesem die Nachricht *beschleunige()* senden.
- Beispiel:



Mehrfachvererbung

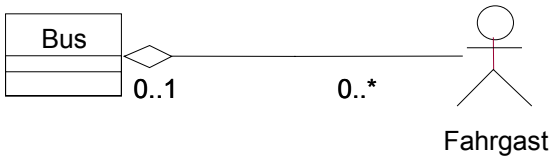
- Eine abgeleitete Klasse erbt von mehreren direkten Basisklassen
- Problem: Verschiedene Basisklassen haben gemeinsame Basisklasse, Doppelerbschaft
- Beispiel:



Aggregationsstruktur

- Ein Objekt enthält ein anderes Objekt oder mehrere andere Objekte
- Typen: Gesamtheit-Teil, Container-Inhalt, Gruppe-Mitglied, Verwendung
- Beispiele:
  - Gesamtheit-Teil

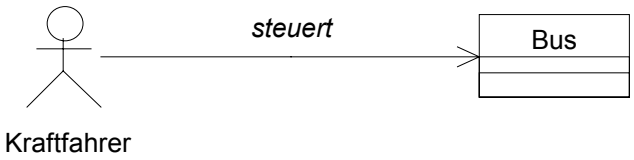
  - Container-Inhalt



  - Gruppe-Mitglied

Assoziationsstruktur

- Prüffrage: In welche Teile kann die Klasse im Problembereich zerlegt werden?
- Prüffrage: Werden die Komponenten innerhalb der Systemveatwortlichkeit benötigt?
- Prüffrage: Erfüllen diese die Kriterien an Klassen, die in das Modell gehören?
- Prüffrage: Haben die Teile keine ihrer Aufgaben an die Gesamtheit abgetreten?
- Beispiel:



Identifizieren von Methoden

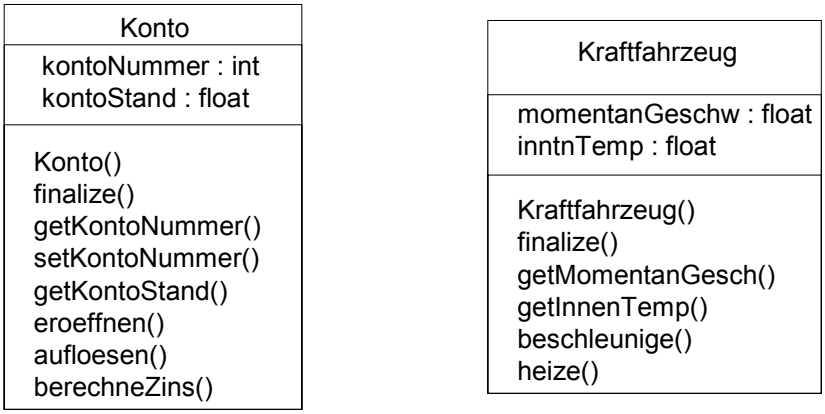
Konzept

- Sämtliche Aktivitäten eines objektorientierten Systems basieren auf der Kommunikation zwischen Objekten
- Ein Objekt fordert als Client die Dienste eines anderen Objektes als Server mittels Aussenden einer Nachricht an.
- Details über Methoden, Nachrichtenverbindungen und Reihenfolgen werden erst später festgelegt.
- Eine Nachricht kann auch als „Broadcast“ an alle Objekte einer Klasse gesendet werden
- Notwendig für Kommunikation: Objektbeziehung oder Gesamtheit-Teil-Verhältnis
- Allgemein kennt der Sender den Empfänger, nicht aber der Empfänger den Sender
- Nachrichtenversendung an sich selbst ist möglich und üblich: der *Bus*, der die Nachricht *anhalten()* empfängt, wird sich selbst die Nachricht *bremsen(0)* senden

Arten von Methoden

- Unterscheide implizite und explizite Methoden
- Implizite Methoden in der Analysephase nicht dargestellt, damit überschaubar
- Explizite Methoden stellen die echte Funktionalität dar, berechnen Werte

- In Klassen die Realwelt-Schnittstellen darstellen werden Methoden definiert die auf Eingaben oder Nachrichten externer Systeme warten



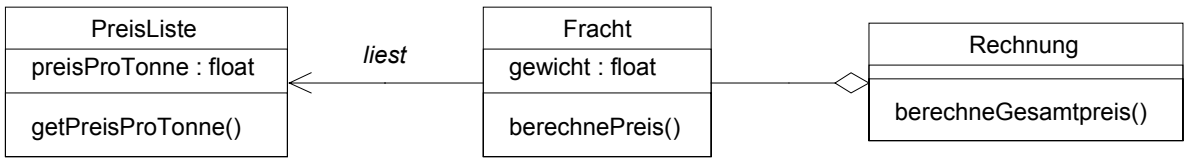
Implizite Methoden

- Konstruktoren: Erzeugung von Objekten
- Destruktoren: Zerstörung von Objekten
- Zugriffsfunktionen: *setAttribut*(), *getAttribut*, *isAttribut*(), *hasAttribut*()
- Verbindungsfunktionen

Explizite Methoden

- Beschreiben spezifische Verantwortlichkeiten einer Klasse
- Dienste, die ein Objekt dieser Klasse anbietet und zur Verfügung stellt
- Prüffragen: Welche Berechnungen sollen die Objekte bereitstellen?

  Gibt es Überwachungsaufgaben, die wahrgenommen werden müssen?



Spezifikation von Methoden

- Spezifikation einer Methode wird beim Empfänger bzw. Server vorgenommen
- Spezifikation:

  Name der Methode

  Typen und Namen der Argumente

Typ des Wertes

Beschreibung

- Beispiel:

```
Methoden der Klasse Fracht

Methode berechnePreis()
Argumente: keine
Rückgabewert: Fließkommazahl (float)
Beschreibung: Sendet die Nachricht getPreisProTonne()
an das Objekt vom Typ PreisListe. Berechnet den Preis als
Produkt von gewicht und dem erhaltenen PreisProTonne und
gibt diesen Wert zurück.

Methode ...
```

Identifizieren von Nachrichtenverbindungen

- Nachrichtenverbindung setzt einen Kommunikationspfad (Link) voraus
- Jede Verbindung (Pfeil vom Sender zum Empfänger) repräsentiert:

  angeforderte Methode, Argumente (Werte oder Referenzen), Resultat (Rückgabepfeil)
- Für einen Link können mehrere verschiedene Nachrichten angegeben werden
- Sequenzen von Nachrichten werden durch Nummern geordnet
- Spezifikation wird in der Klasse des Senders bzw. Clients vorgenommen
- Spezifikation:

  Name

  benötigte Methode

  Typen und Namen der Argumente

  Typ des Wertes

  Beschreibung
- Beispiel:

```
Nachrichtenenverbindungen der Klasse Fracht

Verbindung zur Klasse PreisListe
Benötigte Methode getPreisProTonne()
Argumente: keine
Rückgabewert: Fließkommazahl (float)
Beschreibung: Die Anforderung eines Fracht-Objektes an das
PreisListe-Objekt fordert den Tonnagepreis an, um daraus den
Preis für dieses Fracht-Objekt zu berechnen.

Verbindung zur Klasse...
```

Klassenspezifikation

- Klasse: Name, Spezialisierungen, Generalisierungen, Beschreibung
- Teile: Namen, Kardinalitäten, Strukturtypen, Beschreibungen
- Attribute: Namen, Datentypen, Beschreibungen
- Objektverbindungen: Beteiligte Objekte, Kardinalitäten, Beschreibungen
- Methoden: Name, Argumente (Namen/Typen), Werte, Beschreibungen
- Nachrichtenverbindungen: Beteiligte Objekte, Methoden, Argumente (Namen/Typen), Werte, Beschreibungen

Dynamisches Modell des Systems

Allgemeines

- Modelliert das Verhalten der Objekte
- Veränderungen der Attributwerte und Beziehungen im Zeitablauf
- Zustandsdiagramme modellieren Lebenszyklen von Objekten
- Zustandsübergänge sind immer Reaktionen auf Nachrichten

Ereignisse

- Ereignis ist Vorfall, der Attributwerte verändert und Zustandsänderung bewirken kann
- Ereignis ist immer mit einem Methodenaufruf verknüpft
- Externe Ereignisse treten auf bei Schnittstellenobjekt oder von externem System
  z.B. Taste wurde gedrückt, Timer-Objekt hat externes Ereignis erzeugt
- Interne Ereignisse treten auf bei Nachrichtenaustausch innerhalb des Systems
  z.B. Erzeugen und Löschen von Objekten

Zustände

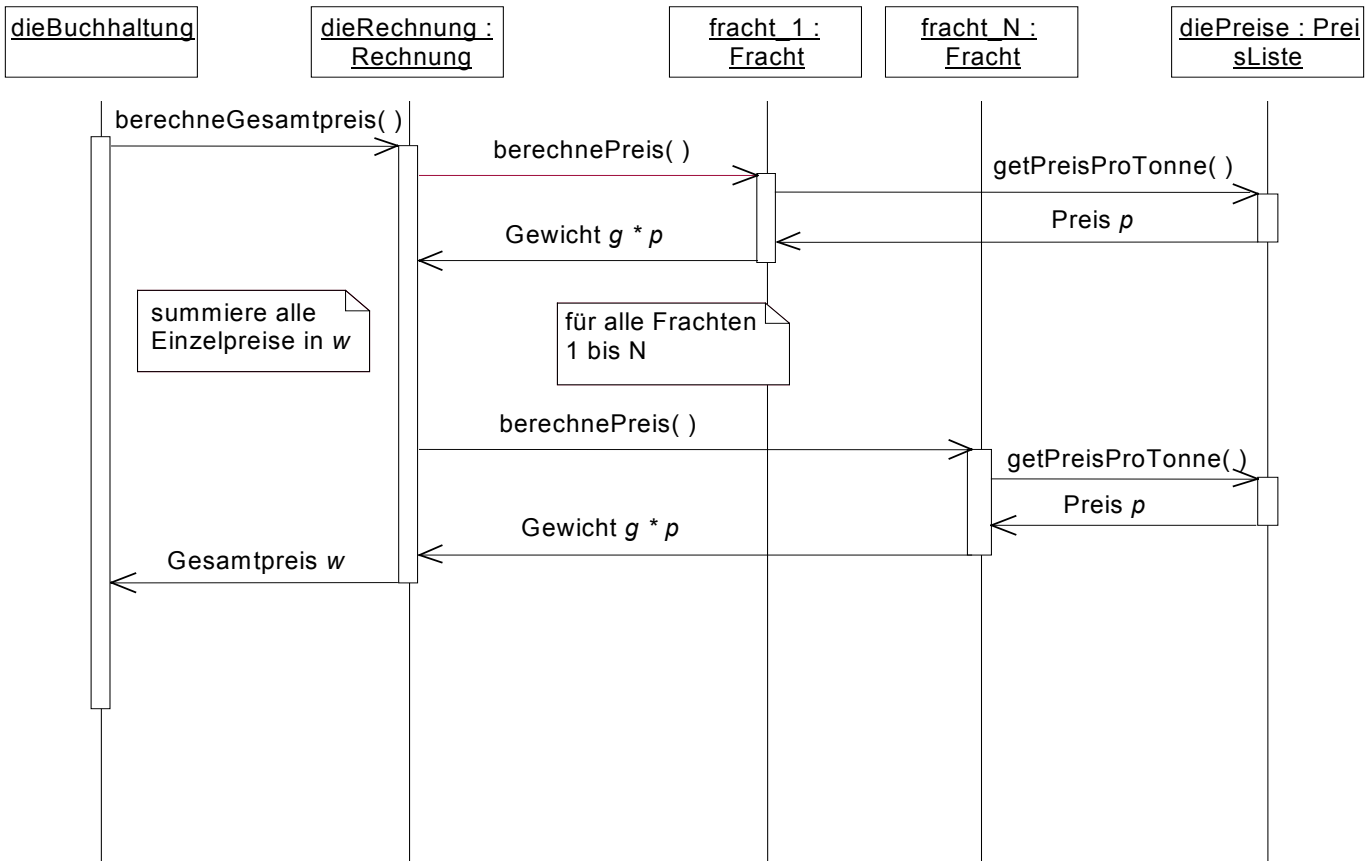
- Zustand eines Objektes wird bestimmt durch seine Attributwerte
- Zustände, die ein Objekt annehmen kann, werden in Zustandsspezifikation beschrieben
- Objekte ändern meistens ihren Zustand, um ihre Aufgaben erfüllen zu können

Szenarios

- Hypothetische Aufeinanderfolge von Ereignissen zur Darstellung der kausalen Zusammenhänge
- Jedes vorstellbare Ereignis, das eintreten kann, in mindestens einem Szenario erfasst
- Szenarios können als Skript formuliert werden.

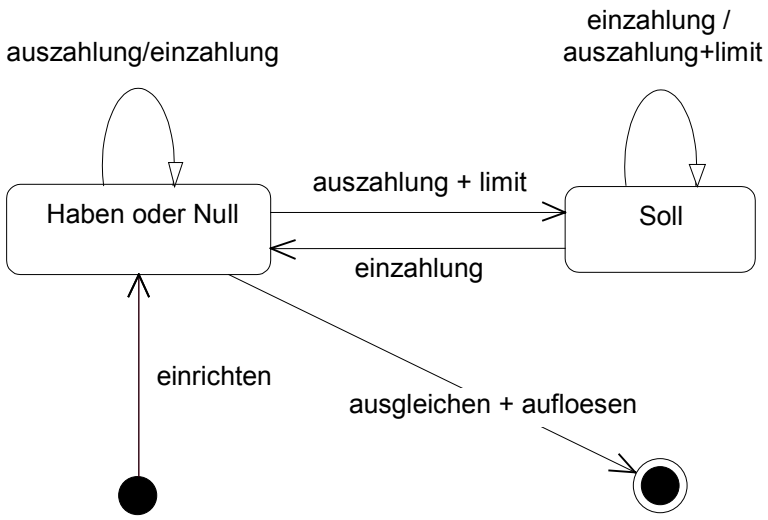
Ereignisfolgen

- Formalere und detailliertere Darstellung der Objektinteraktionen aus den Szenarios
- Erleichtern die Validierung der modellierten Struktur- und Verhaltensaspekte
- Pro Szenario ein Ereignisfolgediagramm
- Objekte durch vertikale Linien, Nachrichten durch Pfeile vom Sender zum Empfänger
- Beispiel: Vollständiges Sequenzdiagramm für die Methode *berechneGesamtpreis()* der Klasse *Rechnung*



Zustandsdiagramme

- Darstellung von Ereignissen, Zuständen, Aktivitäten und ihre Aufeinanderfolge für eine Klasse, für ein Teilsystem oder das ganze System
- Jeder Weg durch das Diagramm beschreibt ein Verhaltensmuster für ein Objekt
- Dem Verhalten in einem Szenario oder Ereignisfolgediagramm entspricht genau ein Weg durch das Zustandsdiagramm
- Zustandsdiagramme entstehen aufbauend auf die Ereignisfolgediagramme
- Beispiel Zustandsdiagramm



Funktionales Modell des Systems

- Algorithmische Beschreibung der identifizierten Methoden und Aktivitäten
- Entwurf erfolgt auf Klassenebene
- Hilfsmittel zur Spezifikation:

  Strukturierte Sprachen, Entscheidungsbäume und –tabellen, Ablaufpläne, Struktogramme, Pseudocode, mathematische Gleichungen

Objektorientiertes Design

- Nahtloser Übergang
- OOA: Modell zur Beschreibung der Systemverantwortlichkeiten
- OOD: Architekturmodell des zu implementierenden Systems
- Allgemeine taktische Vorgehensweise

Vorgehensweise

- Modellierung spezifischer Implementierung
  unter Berücksichtigung von Hard- und Softwareumgebung
- Statisches Modell wird erweitert um
  lösungsspezifische Attribute, Objektverbindungen, Methoden oder Klassen
- Iteration zwischen Problembereich (OOA) und Lösungsbereich (OOD)

Prüffragen:

- Welche Attribute sind nötig um Objektverbindungen zu implementieren?
- Welche Attribute sind nötig um Gesamtheit-Teil-Beziehungen zu implementieren?
- Sollen zur Effizienzsteigerung ableitbare Attribute modelliert werden?
- Welche zusätzlichen Klassen sind notwendig: Hilfsklassen, Benutzerschnittstelle?
- Sind persistente Objekte gefordert? Wenn ja, für welche Klassen?
- Findet Kommunikation nur synchron oder auch asynchron statt?
- Ist das System zentral oder verteilt zu realisieren?
  Wie werden im verteilten Fall Nachrichten weitergeleitet?
  Welche Architektur ist geeignet?
  Plattformunabhängigkeit: RMI, Beans
  Sprachenunabhängigkeit: COM, DCOM, ActiveX
  Plattform- und Sprachenunabhängigkeit: RPC, DCE, CORBA
- Fehlermöglichkeiten, Ausnahmebehandlungen insbesondere bei verteilten Systemen
- Datenschutz: Authentisierung, Authorisierung, Verschlüsselung
- Datensicherheit: Redundanz, Transaktionsverfahren

Ergebnisse der Designphase

Beschreibung der Architektur

- Klassen und Objektdiagramme der logischen Architektur
- Moduldiagramme der physikalischen Architektur
- Einteilung der Klassen in Klassenkategorien
- Einteilung der Module in Subsysteme

Beschreibung der gemeinsamen taktischen Vorgehensweise

- Fehlerbehandlung
- Speicherverwaltung
- Datenspeicherung
- Taskmanagement
- Transaktionskonzept
- Sicherheitskonzept

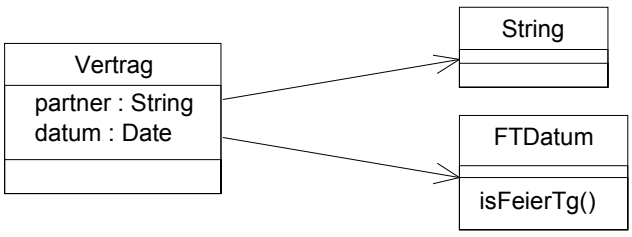
Entwicklungsplan

- Versionsplanung
- Einordnung der Szenarios und Funktionspunkte in Versionsfolge
- Aufgabenverteilung
- Risikoeinschätzung und Terminplanung
- Verifizierung der Architektur mit Hilfe von Prototypen

Elementare Systembausteine

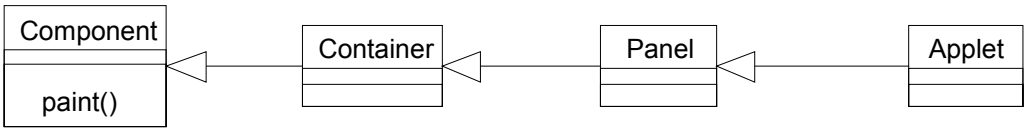
Problembereichskomponente

- Klassen und Strukturen gemäß der Analysephase
- Fortgeschrieben und ergänzt in der Designphase

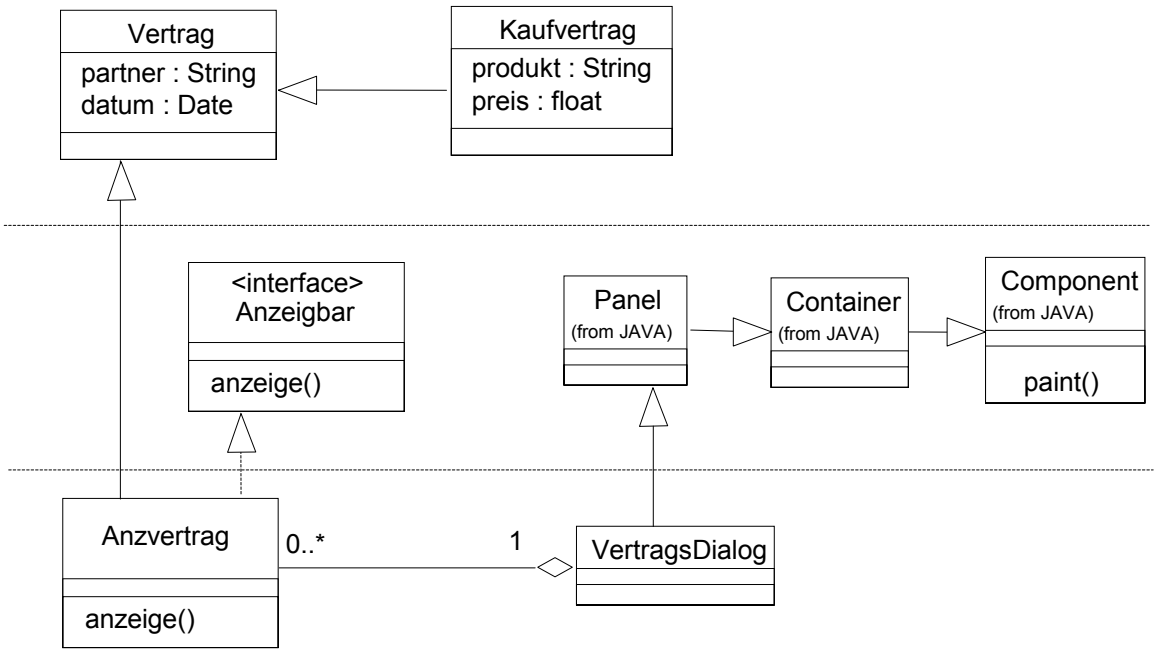


Kommunikationskomponente

- Bedienung des Systems durch den Benutzer
- Präsentation von Resultaten und Informationen
- Klassen und Objekte auf der Systemgrenze, stark werkzeug- und bibliotheksabhängig



- eigenständiges, klar abgegrenztes Subsystem
- Beispiel:



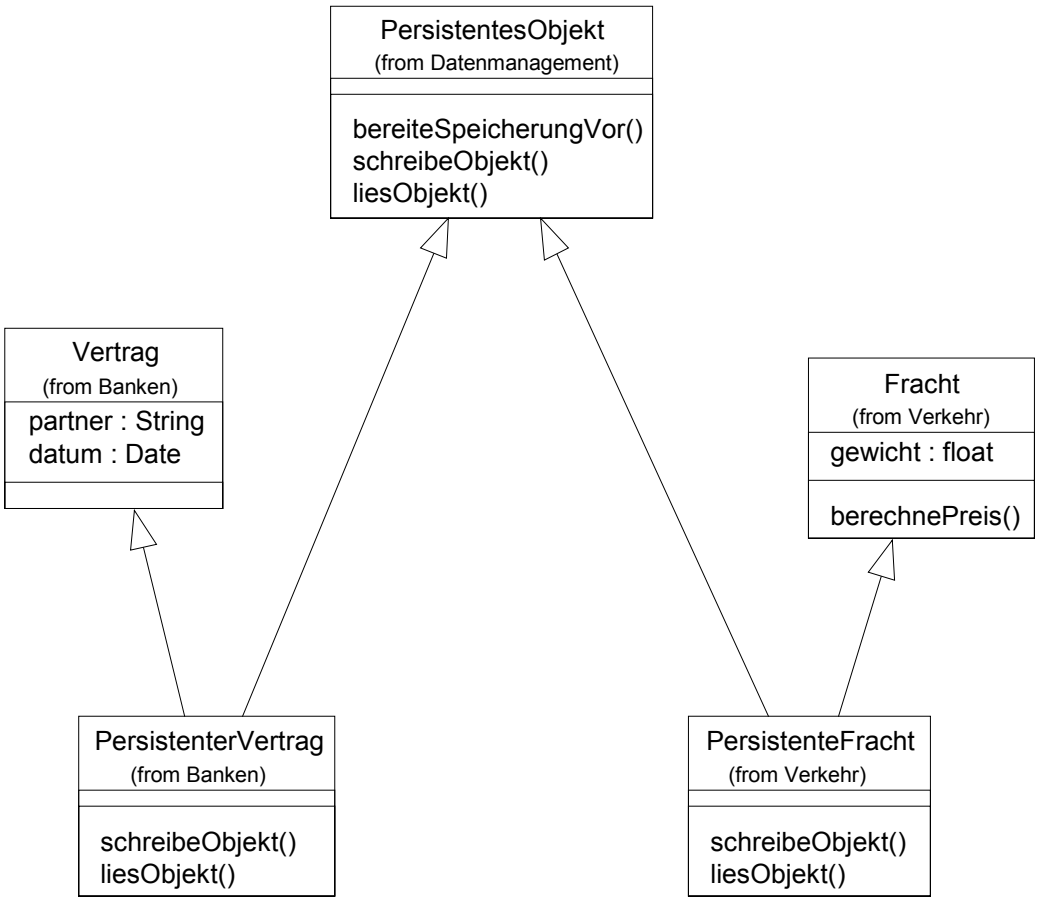
Datenmanagementkomponente

- Datenaspekte, Voraussetzung für Speicherung und Wiederauffinden von Objekten
- Einfluß haben Integrität und Konsistenz, aber auch Zugriffsgeschwindigkeit
- Ebenfalls isoliert zu betrachtendes Subsystem
- Technik Textdatei:

  unstrukturiertes Speichern in Datei

  in der Regel proprietäres Format

  zusätzliche abstrakte Basisklasse oder Schnittstelle (JAVA) für „persistente Objekte“



- Technik Relationale Datenbank:

  Speicherung in einer oder mehreren Tabellen

  Schutzmechanismen werden genutzt

  generische Klasse für Datenbankcollection

  Abbildung in die Relationen eines Datenbankschemas

  Verschiedene Verfahren:

  für jede Klasse eine Tabelle (ObjID, Attribut, ...)

  für jedes Attribut jeder Klasse eine Tabelle (ObjID, Attribut)

  insgesamt nur eine Tabelle (Klasse, ObjID, Attributname, Attribut)
- Technik Objektorientierte Datenbank:

  Nutzung der entsprechenden Erweiterung einer objektorientierten Programmiersprache

  Persistente Objekte unterscheiden sich nicht mehr von transienten Objekten

Taskmanagementkomponente

- Koordination der Problembereichsobjekte und ihrer Methoden
- Modellierung des nebenläufigen Verhaltens verschiedener Objekte sowie der asynchronen Objektkommunikation
- Klassen zum Starten und Beenden von Tasks und Prozessen
- Aktivitäten und Aktionen, Interprozesskommunikation und Prioritäten

Abkapselung von Systemteilen und Definition von Schnittstellen

- Modularisierung hat als Ziel kohäsive und lose gekoppelte Module
- Komplexität durch Modularisierung wesentlich verringert
- Module als Klassengruppen unter dem Aspekt der Wiederverwendbarkeit

Objektorientierte Programmentwicklung

Wartbarkeit

Formen von Wartung

- Fehlerbehebung und Mängelbeseitigung
- Weiterentwicklung und Erweiterung der Funktionalität

Bessere Wartbarkeit durch Objektorientierung

- Robustheit
- Erweiterbarkeit
- Wiederverwendbarkeit
- Abbildung ähnlich der realen Welt
- Verflechtung von Analyse und Design
- Kompatibilität

Wiederverwendbarkeit

Formen allgemeiner Wiederverwendbarkeit

- Code
- Komponenten aus Bibliothek
- Design-Muster
- Anforderungsspezifikationen

Formen objektorientierter Wiederverwendbarkeit

- Basisklassen
- Spezialisierte Klassen
- Mechanismen, sog. Design-Patterns
- Application-Frameworks
- Business-Objekte
- Dokumentkomponenten

Komponenten und Tools für objektorientierte Entwicklung

Klassenbibliotheken

- Problembezogen wiederverwendbare Klassen, z.B. Business-Objekte
- Anwendungsbezogen wiederverwendbare Klassen, z.B. Framework-Objekte
- Erweiterte Basisklassen, z.B. Grafikklassen, Bedienelemente
- Basisklassen, z.B. Datenstrukturen

Entwicklungs-Tools

- Standardisierte Notation, z.B. Unified Modelling Language (UML)
- Unterstützung des gesamten Entwicklungsprozesses (OOA – OOD – OOP)
- Grafisches Entwicklungssystem, textueller Editor mit Sprachenunterstützung
- Browser für Klassen- und Modulhierarchien
- Quellcodegenerator für die Zielsprache
- Reverse Engineering für Round-Trip-Entwicklung
- GUI-Builder, Compiler, Debugger
- Klassenbibliothekar für Verwaltung eigener und vorgefertigter Bibliotheken

Projektmanagement-Tools

- Konfigurationsmanagement mit Quelltext- und Versionskontrolle
- Projektübergreifende Informationen
- Projektplanung und –steuerung

Methodenüberblick

- Objektorientierte Analyse und objektorientiertes Design
  Peter Coad, Edward Yourdon, 1991
- Objektorientiertes Modellieren und Entwerfen
  Rumbaugh und andere, 1991
- Objektorientiertes Software-Engineering
  Jacobson und andere, 1992
- Objektorientierte Analyse und Design
  Grady Booch, 1994

Objektorientierte Analyse und objektorientiertes Design
Peter Coad, Edward Yourdon, 1991

Ziele

- Objektorientierte Analyse
- Objektorientiertes Design
- Top-down-Ansatz
- Integrierte Vorgehensweise durch objektorientierten Systementwurf
- Schließen der Lücke zwischen Analyse und Design

Objektorientierte Analyse

- Identifikation von Klassen und Objekten des abzubildenden Realweltausschnittes
- Identifikation von Strukturen
  - Generalisierung
  - Spezialisierung
  - Vererbungsprinzip
- Definition von Subjekten als Zusammenfassung von Objekten
- Attribute und Instanzenverbindungen definieren
  - Beachtung von Generalisierung und Spezialisierung
  - Attribute nur in der kleinsten gemeinsamen Oberklasse notieren
  - Instanzenverbindungen bedeuten, dass ein Objekt ein anders benötigt, um seine Aufgabe erfüllen zu können

- Definition von Methoden und Nachrichtenverbindungen

  Erzeugen und Löschen eines Objektes

  Lesen und Schreiben der Daten eines Objektes

  Berechnungen

Objektorientiertes Design

- Problembereichskomponente

  Verfeinerung der Ergebnisse der Analyse,

  z.B. im Hinblick auf Speicherverwaltung und Wiederverwendbarkeit

- Kommunikationskomponente

  Schnittstelle zwischen DV-System und Anwender erarbeiten

- Taskmanagement-Komponente

  Berücksichtigung zeitkritischer Vorgänge

  Hardwareeinsatz

- Datenmanagement-Komponente

  Zugriff auf und Manipulation von Daten

  datenbankunabhängiger Entwurf

Objektorientiertes Modellieren und Entwerfen
Rumbaugh und andere, 1991

Modelle

- Objektmodell
- Dynamisches Modell
- Funktionalitätsmodell

Objektmodell

- Identifizieren von Klassen und Objekten
- Anfertigen eines Data Dictionary
- Identifizieren von Beziehungen und Aggregationen
- Identifizieren der Attribute von Objekten und Beziehungen
- Einführung von Vererbungshierarchien zur Vereinfachung
- Sicherstellung von Zugriffpfaden für gewöhnliche Anfragen
- Wiederholung der bisherigen Schritte und Verfeinerung des Modells
- Gruppierung von Klassen in Modulen

Dynamisches Modell

- Entwerfen von Szenarien typischer Interaktionsfolgen
- Identifizieren von Ereignissen
- Entwurf einer Ereignisfolge für jedes Szenario
- Erstellen von Zustandsdiagrammen für jede Objektklasse
- Zuordnen von Ereignissen, Zuständen und Zustandsübergängen

Funktionalitätsmodell

- Identifikation der Ein- und Ausgabedaten
- Entwerfen von Datenflußdiagrammen
- Spezifikation der benötigten Funktionen
- Identifizieren von Beschränkungen
- Spezifizieren von Optimierungskriterien

Objektorientiertes Software-Engineering
Jacobson und andere, 1992

Modelle

- Anforderungsmodell
- Analysemodell

Anforderungsmodell

- Funktionale Anforderungen
  - Statische Struktur und statisches Verhalten
  - Dynamisches Verhalten
- Darstellungsform
  - Anwendungsfälle
  - Schnittstellenbeschreibungen
  - Problemdomäne
- Inhalte
  - Akteure
  - Anwendungsfälle
  - Systemabgrenzungen
  - Domänen-Objekte und Beziehungen

Analysemodell

- Fachliche Systemstruktur
  - Statische Struktur und statisches Verhalten
- Darstellungsform
  - Anwendungsfälle
  - Schnittstellenbeschreibungen
  - Problemdomäne
- Inhalte
  - Entity-Objekte
  - Interface-Objekte

Control-Objekte

Beziehungen zwsichen diesen Objekten

Objektorientierte Analyse und Design
Grady Booch, 1994

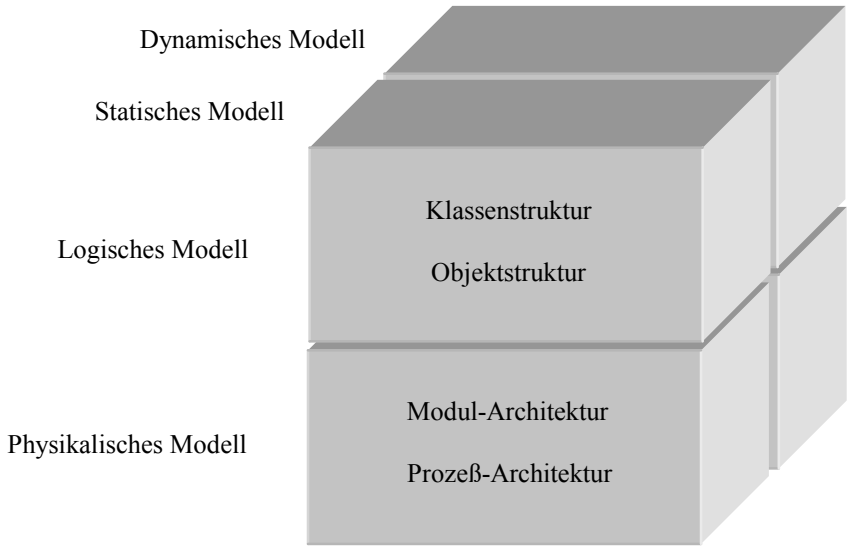
Macro Development Process

- Konzeptualisierung: Festlegen der Kernanforderungen
- Analyse: Entwicklung eines Modells für das gewünschte Sytemverhalten
- Design: Erzeugung einer Architektur für die Implementierung
- Evolution: Implementierung mit schrittweiser Verfeinerung
- Wartung: Verwalten der Entwicklung nach der Auslieferung

Micro Development Process

- Identifizieren von Klassen und Objekten einer bestimmten Abstraktionsebene
- Festlegen der Semantik dieser Klassen und Objekte
- Festlegen der Beziehungen zwischen diesen Klassen und Objekten
- Spezifizieren der Schnittstellen und Implementierung dieser Klassen und Objekte

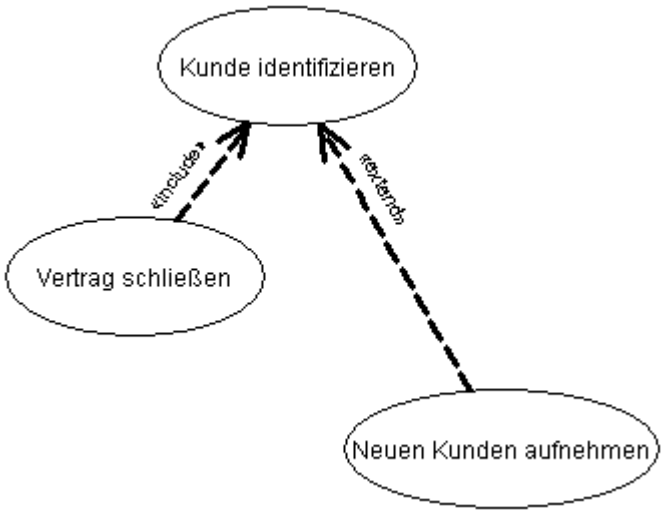
Dreidimensionales Modell



Anhang: Anforderungsanalyse

Anwendungsfalldiagramme

- Beschreibung einer typischen Interaktion eines Anwenders mit dem System
- Anwendungsfall entspricht etwa einem Geschäftsvorfall
- Keine Beschreibung des internen Verhaltens, keine funktionale Zerlegung
- Kein Designhilfsmittel sondern nur für die Anforderungsanalyse



- Textuelle Beschreibung zusätzlich zum Anwendungsfalldiagramm

  Nummer und Name des Anwendungsfalles sowie eine Kurzbeschreibung

  Auslöser sowie Vorbedingungen: Erwarteter Systemzustand vor dem Eintreten

  Ergebnisse und Nachbedingungen: Erwarteter Systemzustand nach dem Durchlaufen

  Nicht-funktionale Anforderungen: Design, Plattformfragen, Entwicklungsprioritäten

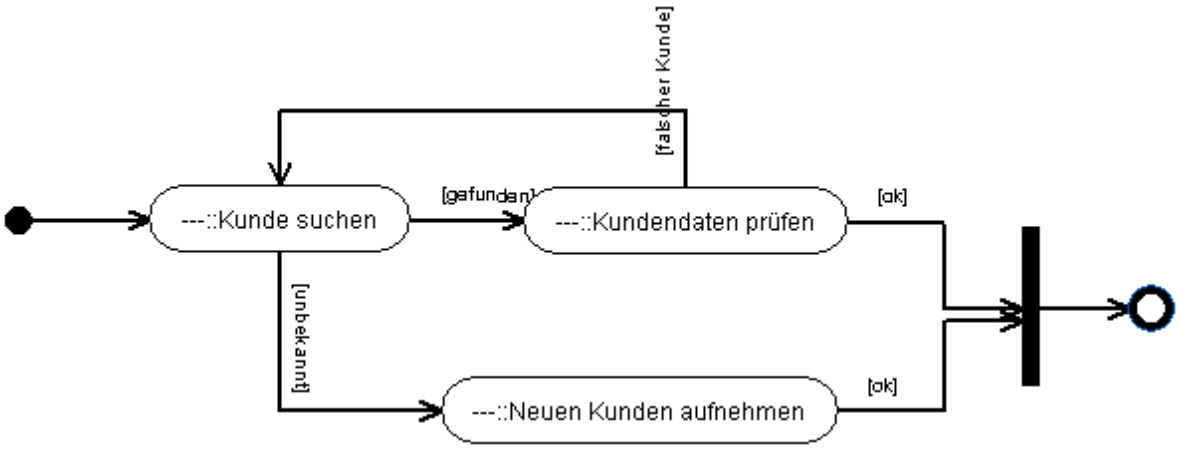
  Ablaufbeschreibung, gegliedert in Einzelschritte, jeder davon separat beschrieben

  Ausnahmen und Varianten: Abweichungen vom Normalfall, alternatives Verhalten

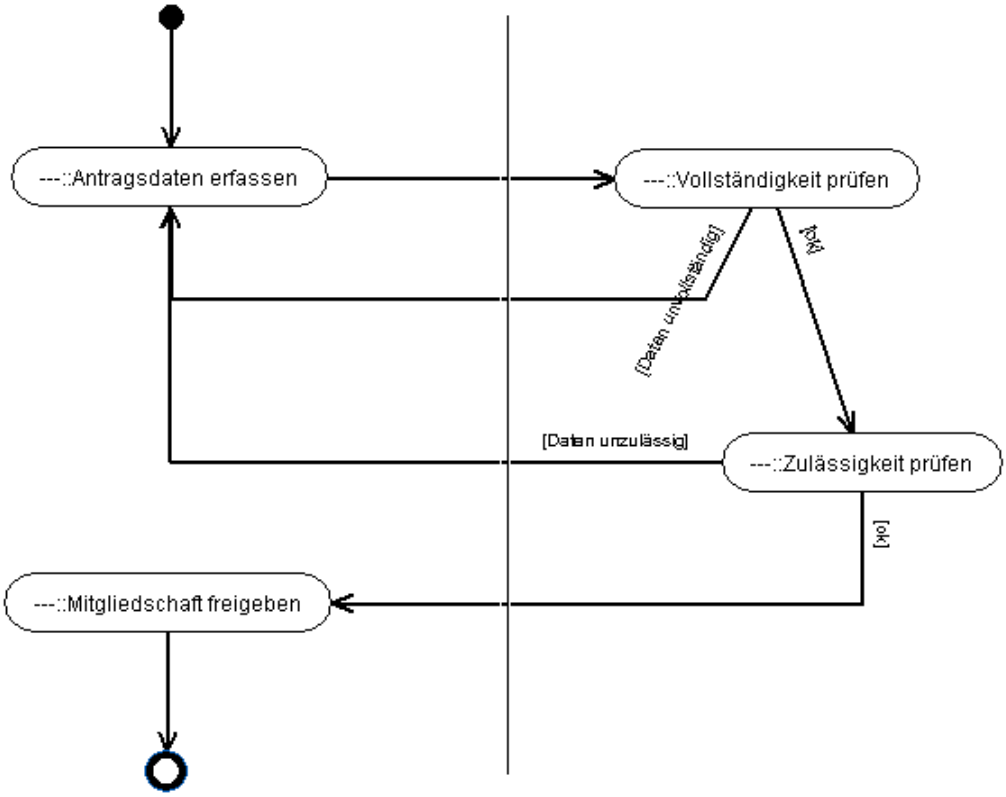
  Offene Punkte, Fragen, Dokumente, Referenzen
- Daumenregel: Pro Aufwandsjahr etwa fünf Anwendungsfälle
- Anwendungsfälle beschreiben, *was* das System leisten soll, nicht *wie* es das macht
- Prinzipiell auch nicht-softwareunterstützte Aktivitäten möglich

Aktivitätsdiagramme

- Verfeinerung der Anforderungen durch Modellierung von Einzelaktivitäten
- Einzelschritte der Anwendungsfälle sind Kandidaten für Einzelaktivitäten



- Zusätzliche Zuweisung der Verantwortlichkeiten
- Aufteilung in verschiedene „Swim Lanes“



Beispiel: Versicherungsvertrag

Anwendungsfallanalyse

- Anwendungsfalldiagramm

- Beschreibung des Anwendungsfalles „Vertrag schließen“

Beschreibung

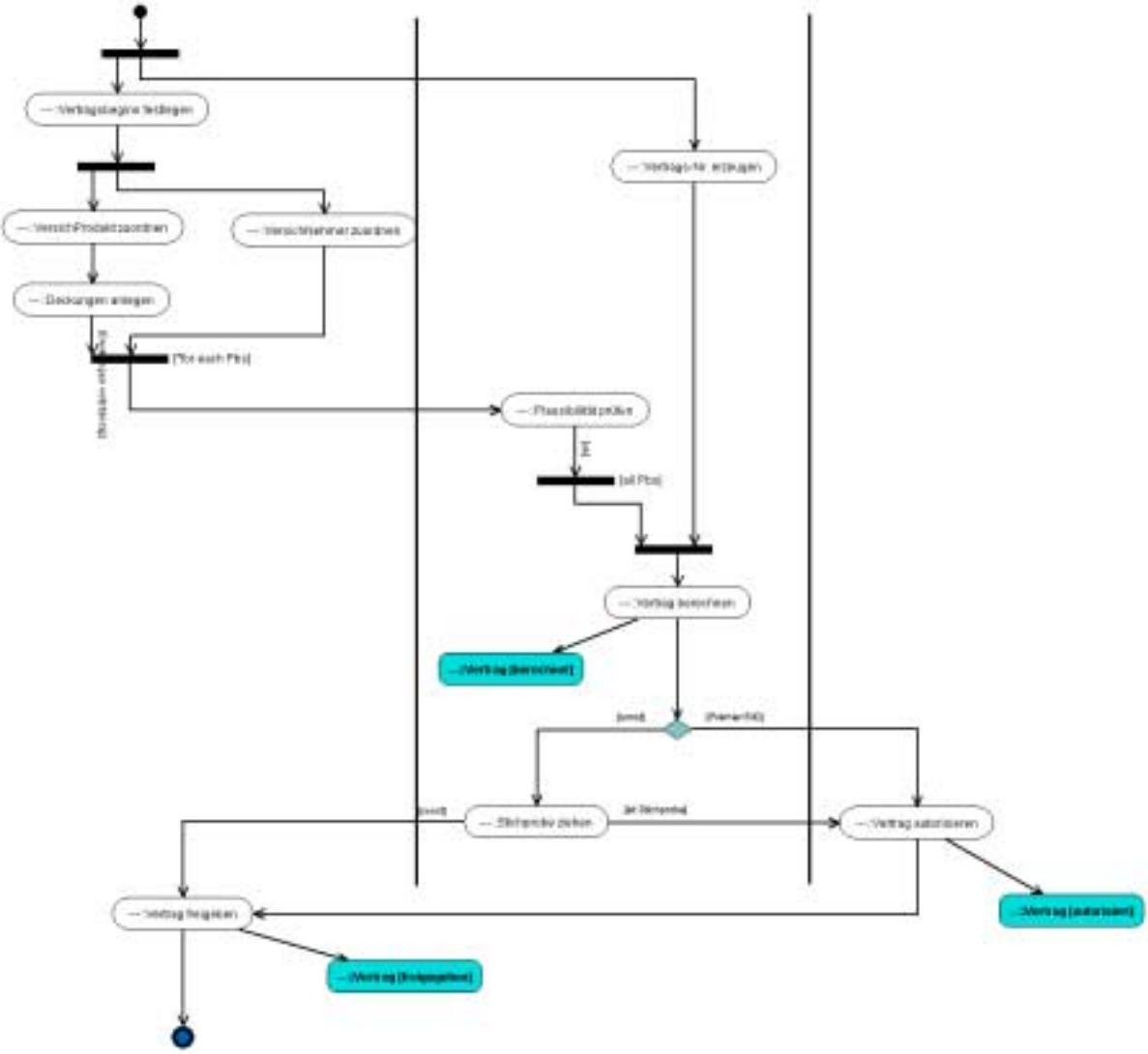
1. Beginn der Vertragslaufzeit eingeben
2. Produkt auswählen
   Der Anwender wählt ein Produkt aus, um es dem Vertrag zuzuordnen. Zur Auswahl werden ihm alle Produkte angeboten, die zum eingegebenen Vertragsbeginn gültig sind.
3. Vertrags-Nr. wird erzeugt
   Die Vertragsnummer wird automatisch erzeugt und eingeblendet.
4. *include* ⇒ Anwendungsfall „*Versicherungsnehmer zuordnen*“
5. Beitragszahler, Postempfänger und Leistungsempfänger festlegen
   [...]
6. [...]
7. *include* ⇒ Anwendungsfall „*Deckungen anlegen*“
8. Vollständigkeit und Plausibilität prüfen sowie berechnen
9. Freigeben

Variationen

4. Statt einen vorhandenen Versicherungsnehmer zu suchen und auszuwählen, wird ein neuer Versicherungsnehmer angelegt.
8. Ist die berechnete Prämie >500, muß der Vertrag speziell geprüft und autorisiert werden. *extend* ⇒ Anwendungsfall „*Vertrag autorisieren*“

Aktivitätenmodellierung

- Aktivitätsdiagramm



- Transitionen
- Synchronisation
- Darstellung von Objektzuständen

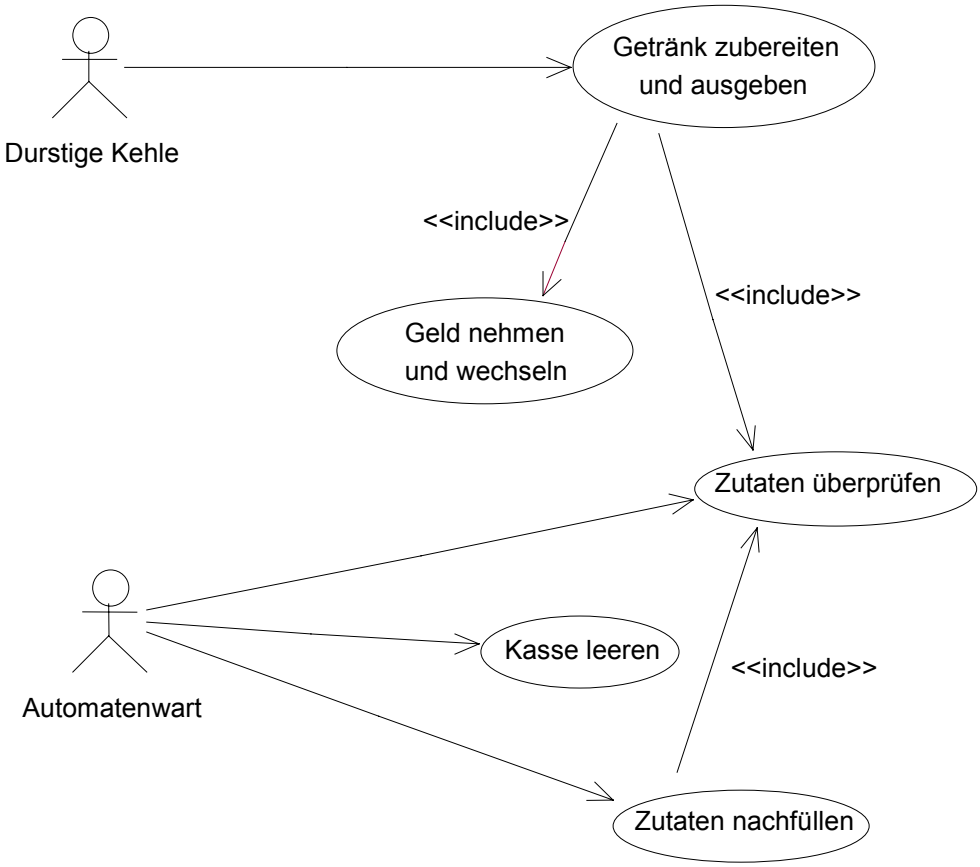
Beispiel: Getränkeautomat

Systembeschreibung

Das System beschreibt die Funktionsweise eines Getränkeautomaten für Warmgetränke (Kaffee und Kakao) einschließlich seiner Wartung (Nachfüllen von Zutaten)

- Münzeinwurf und –rückgabe
- Getränkeauswahl und –zubereitung
- Getränkeausgabe
- Leeren der Kasse und Auffüllen von Zutaten

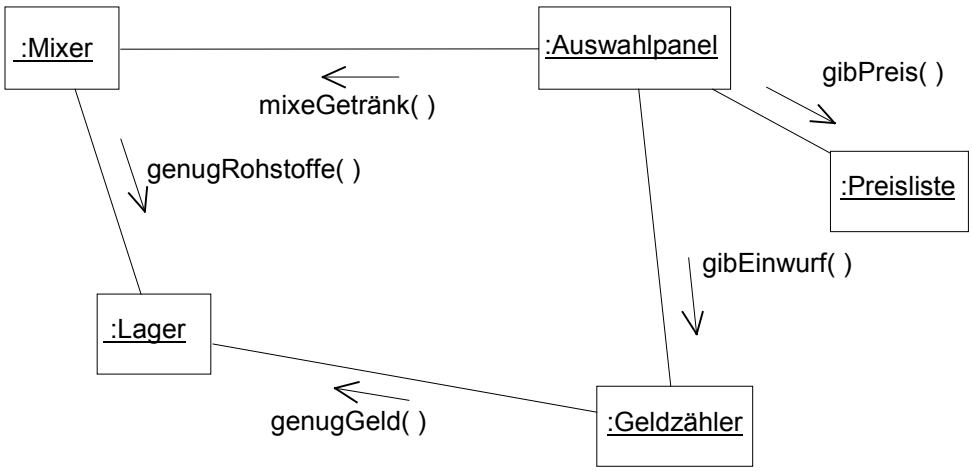
Anwendungsfalldiagramm



Kandidaten für Objekte, erster Ansatz

- Durstige Kehle, Automatenwart
- Rohstoffe, Becher, Wasser, Milch, Kaffepulver, Kakaopulver
- Geld, Groschen, Fuffziger, Markstücke
- Preisliste
- Automat, Kasse, Getränk
- Auswahlknöpfe
- Mixer, Lager, Geldzähler

Kollaborationsdiagramm



Kandidaten für Objekte, zweiter Ansatz

- Rohstofflager

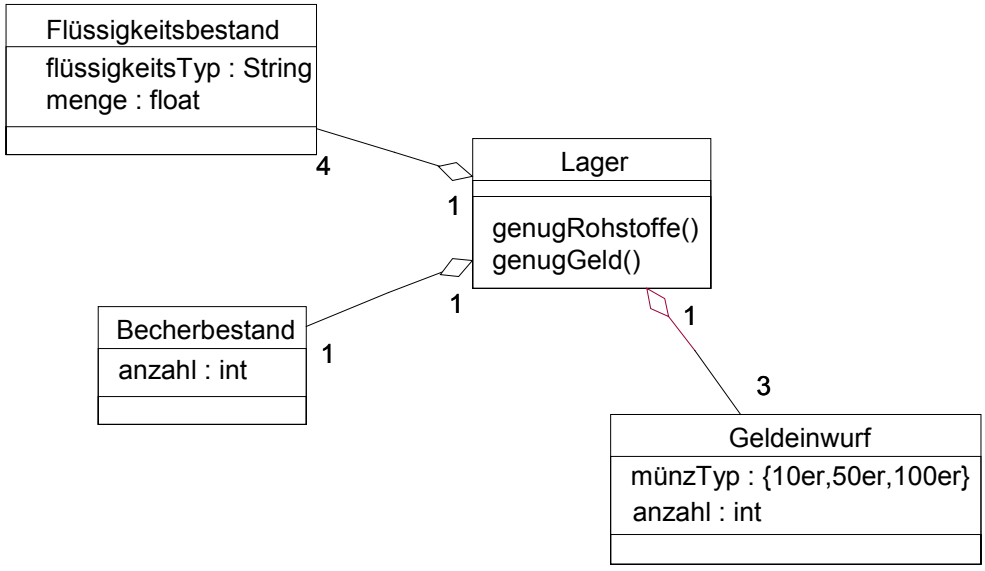
  Attribute: Becher, Wasser, Milch, Kaffepulver, Kakaopulver
- Geldzähler

  Attribute: Groschen, Fuffziger, Markstücke
- Preisliste
- Auswahlknöpfe
- Mixer

Entwurf der Klasse Lager

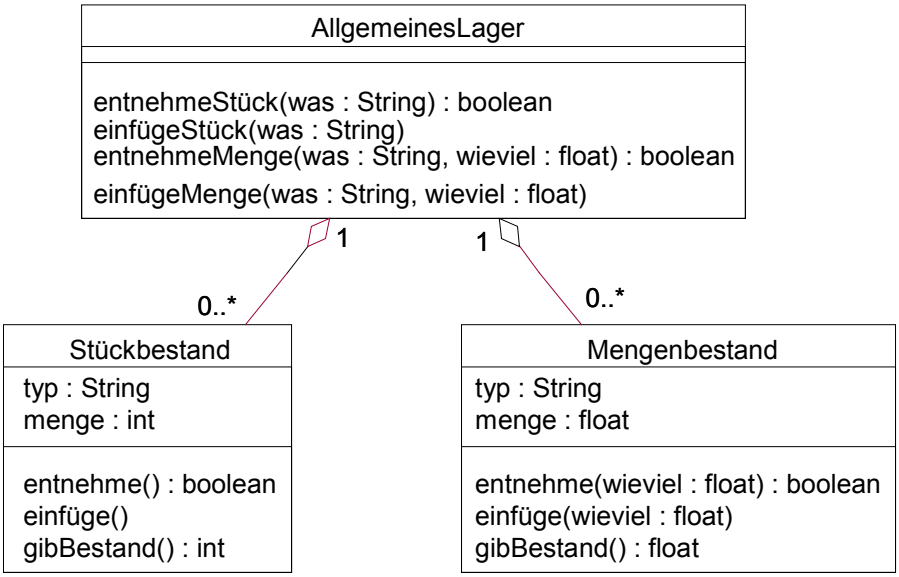
- Attribut: Becher, Ganzzahl, Zahl der vorhandenen Becher
- Attribut: Wasser, Fließkommazahl, Menge des vorhandenen Wassers
- Attribut: Milch, Fließkommazahl, Menge der vorhandenen Milch
- Attribut: Kaffepulver, Fließkommazahl, Menge des vorhandenen Kaffepulvers
- Attribut: Kakaopulver, Fließkommazahl, Menge des vorhandenen Kakaopulvers
- Methode: gibMengeZahl, Parameter: Typ, Rückgabe: Ganzzahl,
  liefert die Zahl der vorhandenen Becher
- Methode: gibMengeGramm, Parameter: Tyü, Rückgabe: Fließkommazahl,
  liefert die vorhandene Menge der als „Typ“ gewählten Zutat

Klassendiagramm der Klasse Lager und ihrer Hilfsklassen

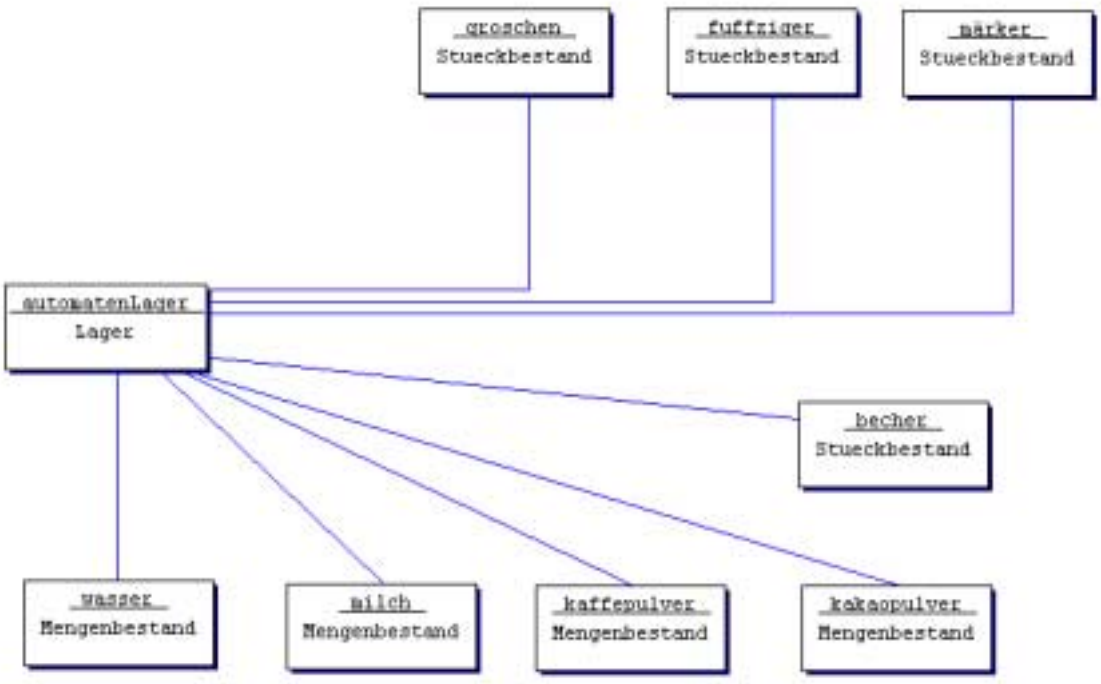


- Strenggenommen unterscheiden sich *Becherbestand* und *Geldeinwurf* nur sehr wenig
- Führe daher für beide eine gemeinsame Basisklasse ein
- Modelliere diese sofort so, daß sie später auch für andere Anwendungen nutzbar ist
- Modelliere enstprechend eine Basisklasse für den *Flüssigkeitsbestand*

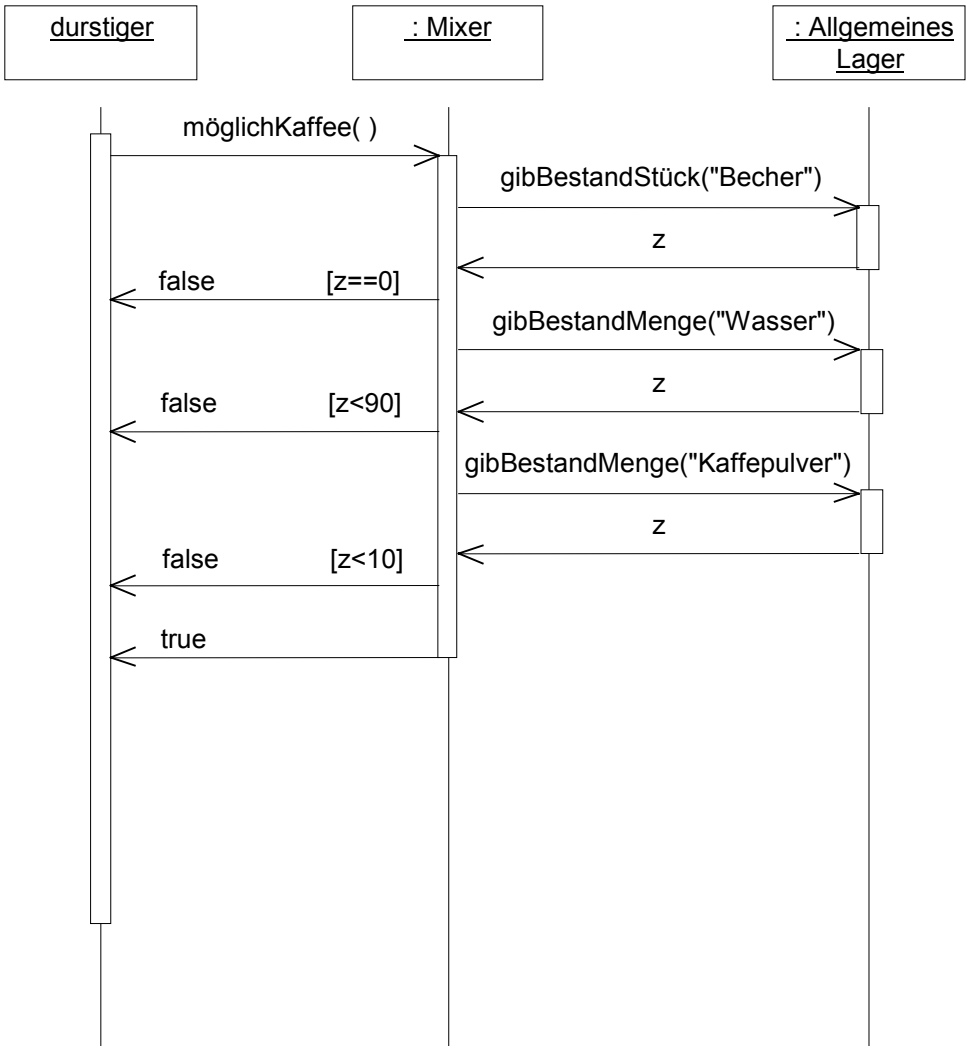
Klassendiagramm der allgemeinen Klasse Lager und ihrer Hilfsklassen



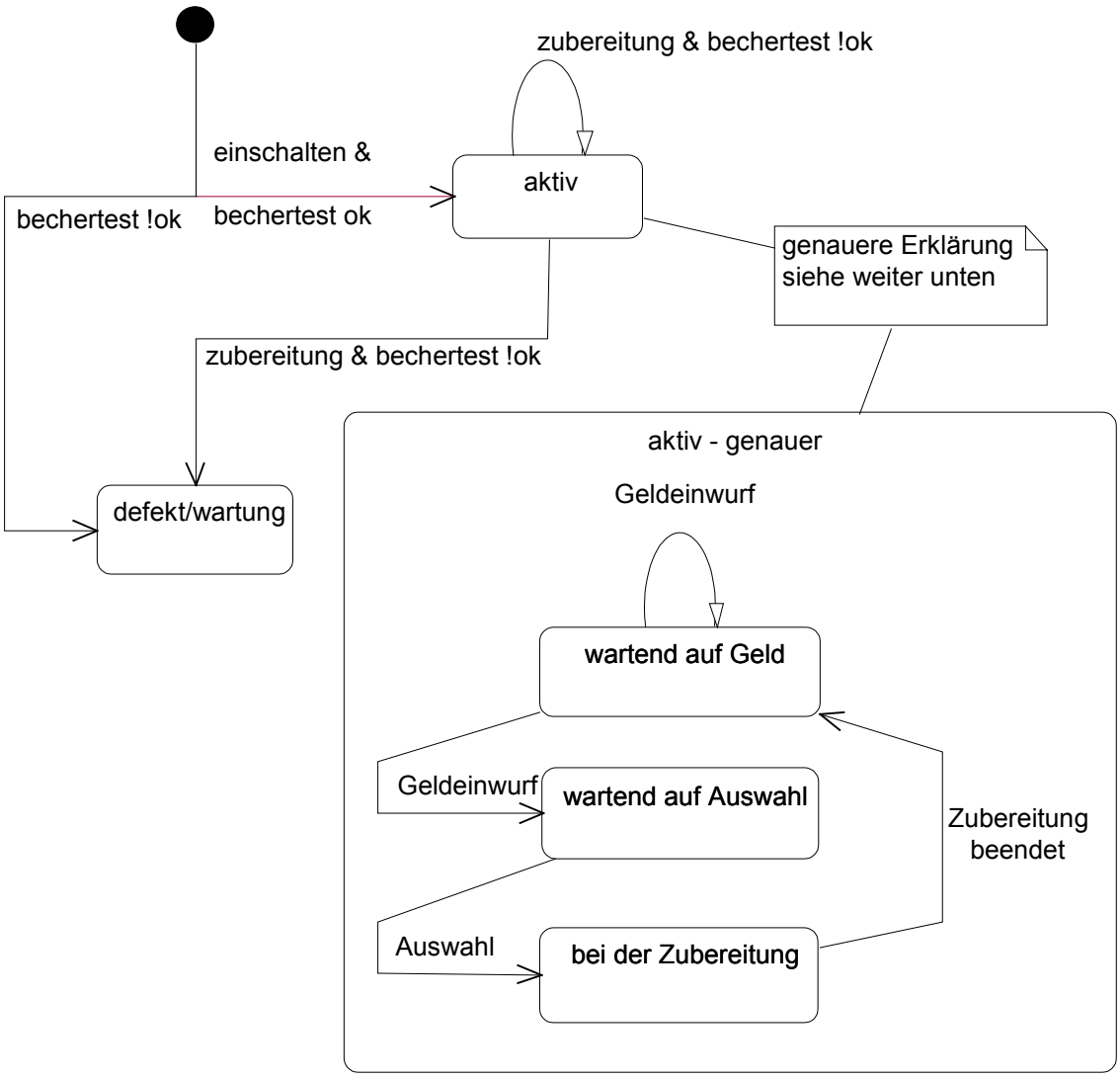
Objektdiagramm für das Lager des Getränkeautomaten



Sequenzdiagramm für die Nachricht* möglichKaffee() *an den Mixer

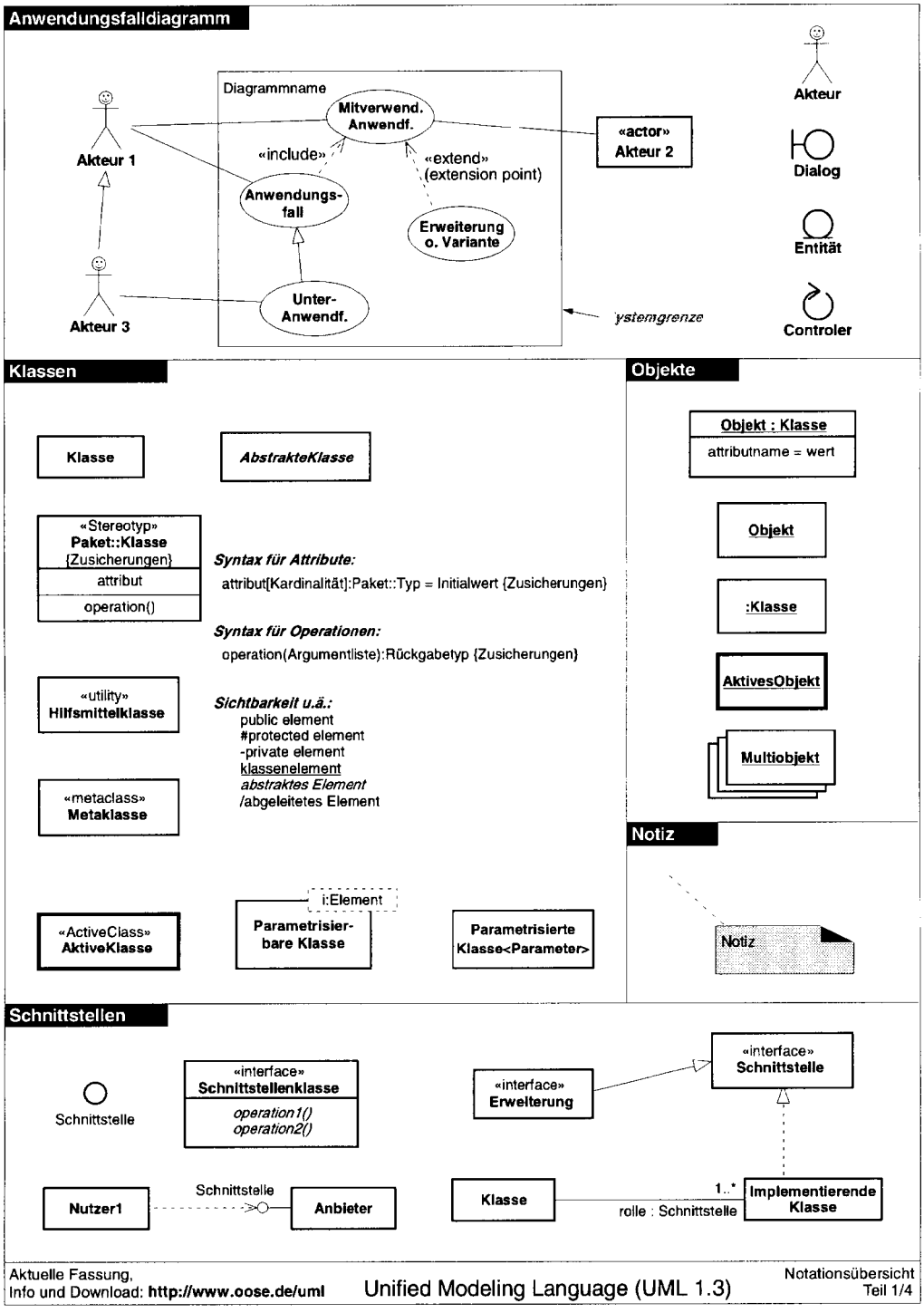


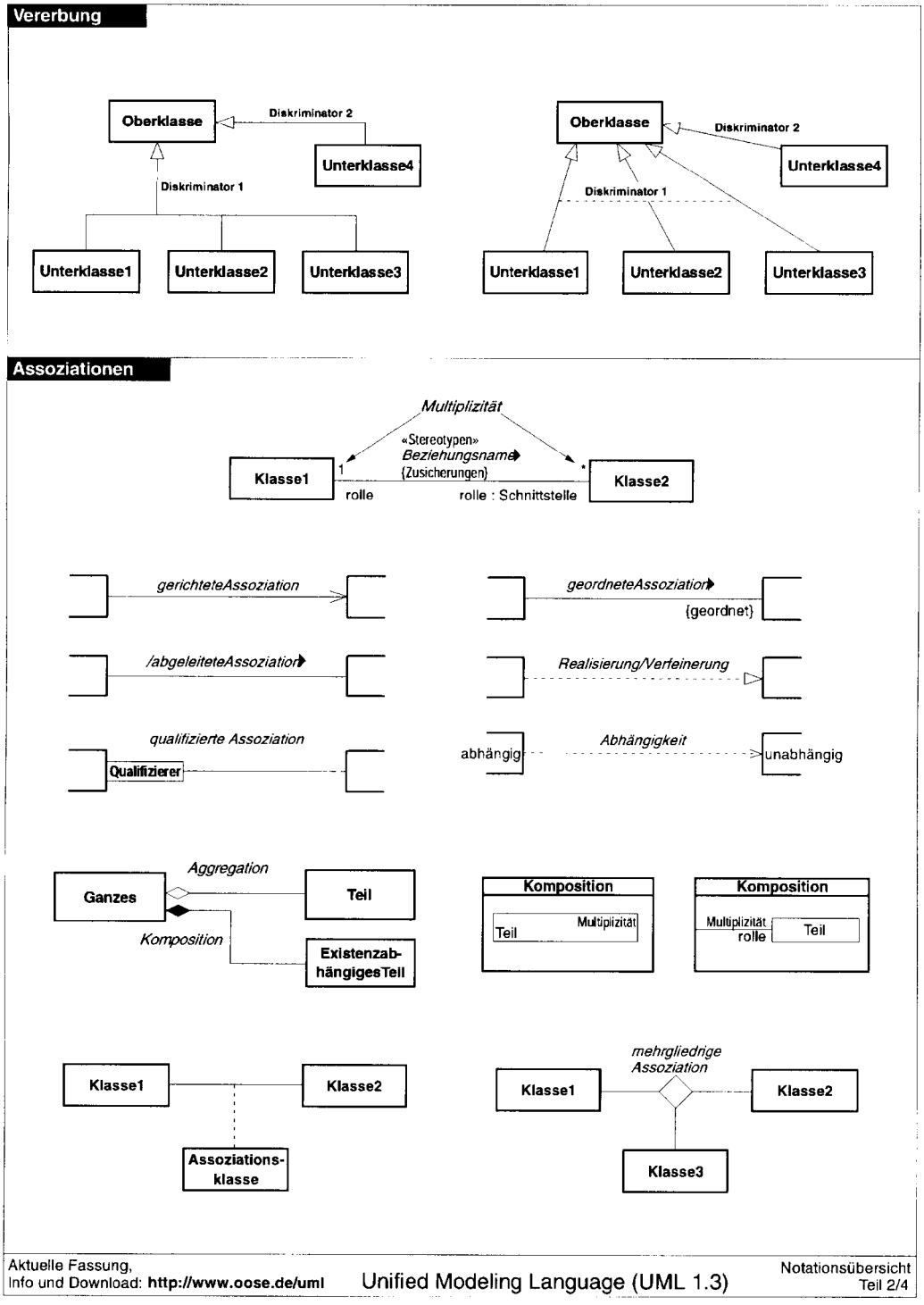
Zustandsdiagramm für den Getränkeautomaten

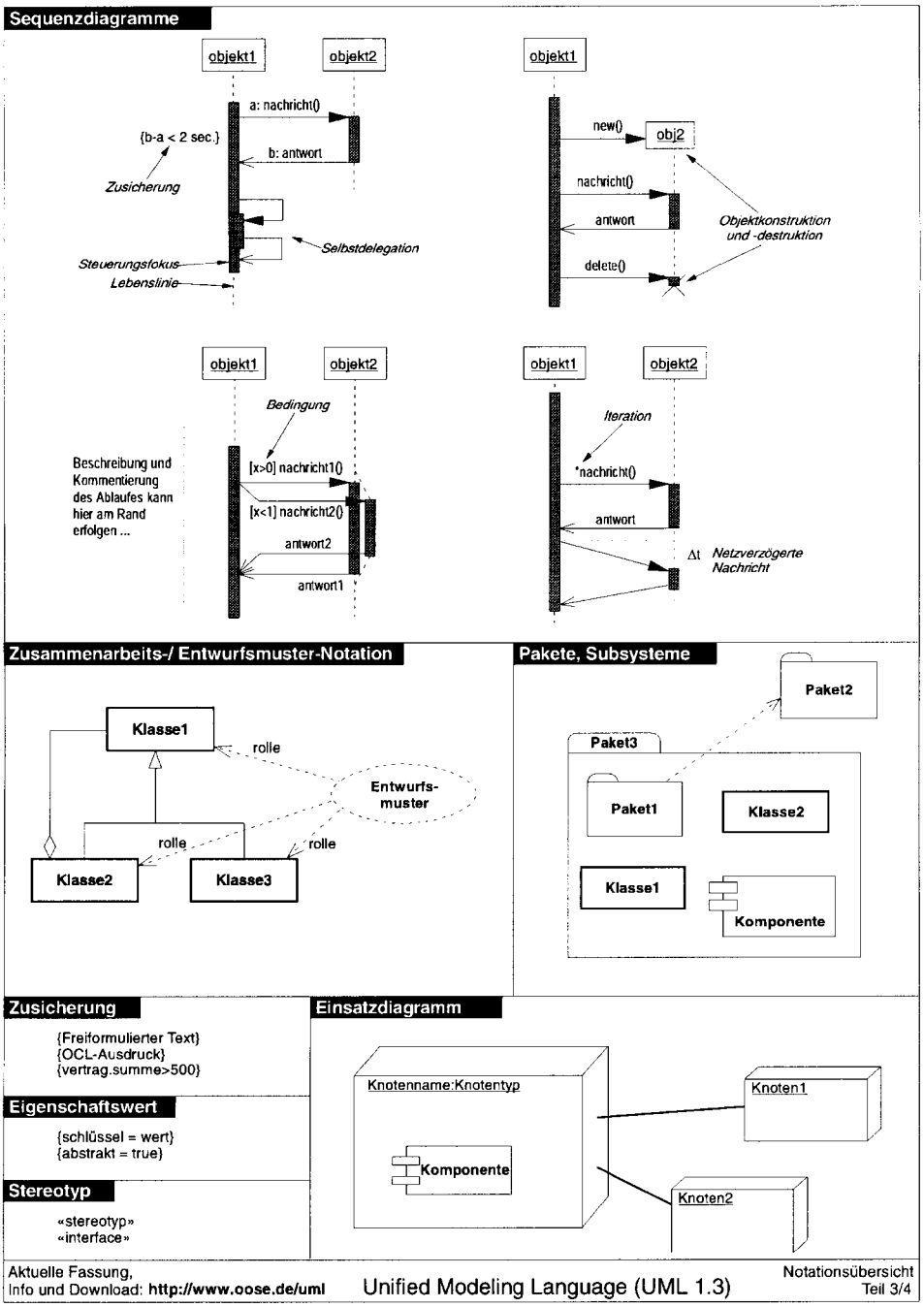


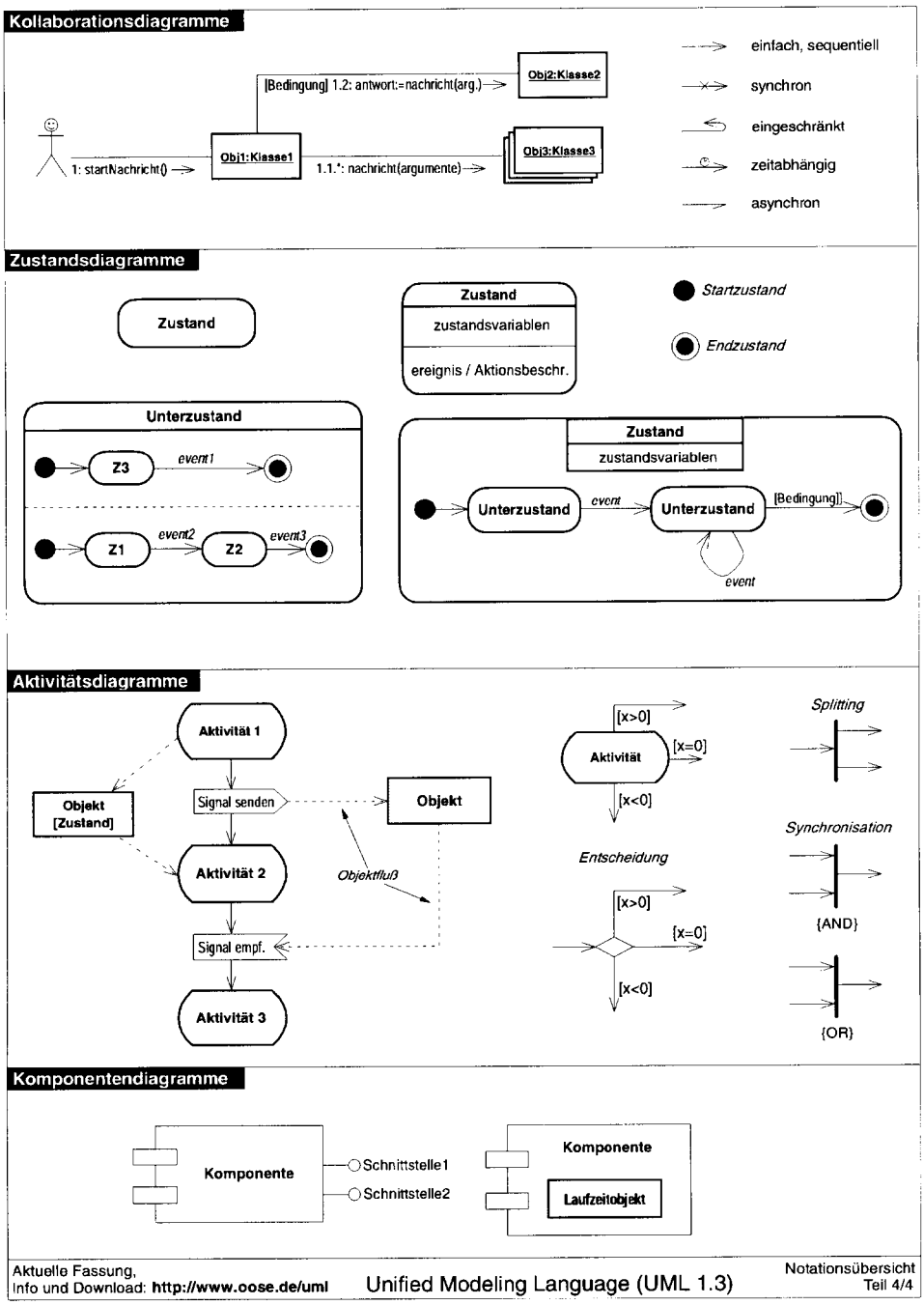
Unified Modelling Language

Notationsübersicht Unified Modelling Language (UML), entnommen aus [13]




Vererbung
Oberklasse
Diskriminator 2
Unterklasse4
Diskriminator 1
Unterklasse1
Unterklasse2
Unterklasse3
Oberklasse
Diskriminator 2
Unterklasse4
Diskriminator 1
Unterklasse1
Unterklasse2
Unterklasse3
Assoziationen
Multiplizität
«Stereotypen»
Beziehungsname
{Zusicherungen}
Klasse1
1
*
Klasse2
rolle
rolle : Schnittstelle
gerichteteAssoziation
geordneteAssoziation
{geordnet}
/abgeleiteteAssoziation
Realisierung/Verfeinerung
qualifizierte Assoziation
Qualifizierer
abhängig
Abhängigkeit
unabhängig
Aggregation
Ganzes
Teil
Komposition
Existenzab-
hängigesTeil
Komposition
Teil
Multiplizität
Komposition
Multiplizität
rolle
Teil
Klasse1
Klasse2
Assoziations-
klasse
mehrgliedrige
Assoziation
Klasse1
Klasse2
Klasse3
Aktuelle Fassung,
Info und Download: http://www.oose.de/uml
Unified Modeling Language (UML 1.3)
Notationsübersicht
Teil 2/4


Sequenzdiagramme
objekt1
objekt2
a: nachricht()
{b-a < 2 sec.}
b: antwort
Zusicherung
Selbstdelegation
Steuerungsfokus
Lebenslinie
objekt1
new()
obj2
nachricht()
antwort
Objektkonstruktion und -destruktion
delete()
objekt1
objekt2
Bedingung
Beschreibung und Kommentierung des Ablaufes kann hier am Rand erfolgen ...
[x>0] nachricht1()
[x<1] nachricht2()
antwort2
antwort1
objekt1
objekt2
Iteration
*nachricht()
antwort
Δt
Netzverzögerte Nachricht
Zusammenarbeits-/ Entwurfsmuster-Notation
Klasse1
rolle
Entwurfs-muster
rolle
rolle
Klasse2
Klasse3
Pakete, Subsysteme
Paket2
Paket3
Paket1
Klasse2
Klasse1
Komponente
Zusicherung
{Freiformulierter Text}
{OCL-Ausdruck}
{vertrag.summe>500}
Eigenschaftswert
{schlüssel = wert}
{abstrakt = true}
Stereotyp
«stereotyp»
«interface»
Einsatzdiagramm
Knotenname:Knotentyp
Komponente
Knoten1
Knoten2
Aktuelle Fassung,
Info und Download: http://www.oose.de/uml
Unified Modeling Language (UML 1.3)
Notationsübersicht
Teil 3/4


Kollaborationsdiagramme
einfach, sequentiell
synchron
eingeschränkt
zeitabhängig
asynchron
[Bedingung] 1.2: antwort:=nachricht(arg.)
Obj2:Klasse2
Obj1:Klasse1
Obj3:Klasse3
1: startNachricht()
1.1.*: nachricht(argumente)
Zustandsdiagramme
Zustand
zustandsvariablen
ereignis / Aktionsbeschr.
Startzustand
Endzustand
Unterzustand
Z3
event1
Z1
event2
Z2
event3
event
[Bedingung]
Aktivitätsdiagramme
Aktivität 1
Signal senden
Objekt
Objekt [Zustand]
Aktivität 2
Objektfluß
Signal empf.
Aktivität 3
[x>0]
[x=0]
[x<0]
Entscheidung
Splitting
Synchronisation
{AND}
{OR}
Komponentendiagramme
Komponente
Schnittstelle1
Schnittstelle2
Laufzeitobjekt
Aktuelle Fassung,
Info und Download: http://www.oose.de/uml
Unified Modeling Language (UML 1.3)
Notationsübersicht
Teil 4/4

Literaturverzeichnis

[1] Ken Barclay et al., Objektorientiertes Design mit C++, Prentice Hall, 1997

[2] Daniel J. Berg et al., Advanced Techniques for Java Developers, Wiley & Sons, 1999

[3] Grady Booch, Object-oriented Design, Benjamin/Cummings, 1991

[4] Rainer Burkhardt, UML – Unified Modeling Language, Addison-Wesley, 1998

[5] Peter Coad, Edward Yourdon, Object-orientied Design, Yourdon-Press, 1991

[6] Mark Coats et al., Using the CMOSpecification, Dr. Dobb's Journal, June 99

[7] Klaus-Peter Eckert, Objekt-Orientiertheit in offenen Systemen, Thomson, 1995

[8] Igor T. Hawryszkiewycz, Systemanalyse und –Design, Prentice Hall, 1995

[9] Gerti Kappel, Michael Schrefl, Objektorientierte Informationssysteme, Springer, 1996

[10] Robert C. Martin. Objektorientierte C++-Anwendungen, Prentice Hall, 1996

[11] Bertrand Meyer, Objektorientierte Softwareentwicklung, Hanser, 1988

[12] Udo Müller, C++-Implementierungstechniken, Thomson, 1997

[13] Bernd Oesterreich, Objektorientierte Softwareentwicklung, Oldenbourg, 1998

[14] Bernd Oesterreich, Erfolgreich mit Objektorientierung, Oldenbourg, 1999

[15] Objektorientierte Analyse und Design, Siemens Nixdorf Trainings Center, 1996

[16] Objektorientierte Programmierung, Siemens Nixdorf Trainings Center, 1996

[17] Dirk Riehle, Entwurfsmuster für Softwarewerkzeuge, Addison-Wesley, 1997

[18] Jeremy Rosenberger, CORBA, SAMS/Markt&Technik, 1998

[19] Günther Wahl, UML kompakt, OBJEKTspektrum 2/1998

[20] Rebecca Wirfs-Brock et al., Objektorientiertes Software-Design, Hanser, 1993

Stichwortverzeichnis