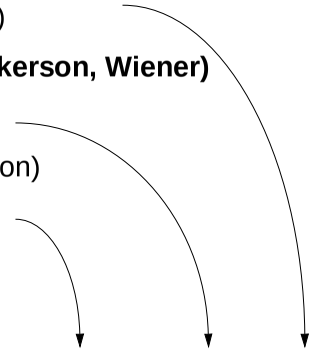
4. Objektorientierter Entwurf

- OMT (Rumbaugh et. al.)
- **RDD (Wirfs-Brock, Wilkerson, Wiener)**
- OOD (Booch)
- OOA/OOD (Coad/Yourdon)
- Objectory (Jacobsen)
- OOSA (Embley)
- Shlaer-Mellor
- OOAD (Martin)
- ....



UML
(Unified Modeling Language

Entwicklungszyklus

- Analyse:
  Beschreibung des Problembereichs - was das System tun soll - keine Implementierungsentscheidungen
- Entwurf:
  Beschreibung der Lösung (Datenstrukturen, Algorithmen, Prioritäten)
- Implementierung

Unterschiedliche Methoden zeigen unterschiedliche Gewichtung und Abhängigkeiten

OO Analyse und Entwurf

Warum OO Entwicklungszyklus (statt nur Programmierung)? Herkömmliche Entwicklungsmethoden vor allem funktional orientiert

- Nachvollziehbarkeit für Betroffene (traditionell: funktionale Spezifikation - komplexe Umsetzung beim Entwurf)
- Leichtere Integration zwischen verschiedenen Teilbereichen desselben Entwurfs (Teamarbeit)
- Konzeptuelle Integrität (Brooks - Mythical Man Month): Komplexität wird reduziert, Zuverlässigkeit des Entwicklungsprozesses steigt

OO Analyse & Entwurf (2)

Effekte

- Mehr Arbeit in die Analyse - Verwendung komplexer und ausdrucksstarker Modelle
  - Problem der oo Problembeschreibung
- Betonung auf Struktur
  - stabiler als Funktion in Entwurf und Implementierung
  - Datenkapselung

OO Analyse & Entwurf (3)

- Verwischung der Grenzen zwischen Entwicklungsphasen
  - Vorteil: keine konzeptuelle Umstellung
  - Detaillierungsgrad ändert sich über die Phasen hinweg, aber nicht die Konzepte
- Iterativ statt sequentiell
  - Da Fortschritt im wesentlichen in Detaillierung => Wechsel zwischen Phasen leichter möglich
  - Problem: Meilensteine

Geschichte

1990 Booch, Object-Oriented Design (OOD), abgeleitet von früheren Arbeiten über Ada. Überwiegend Notation, wenig Vorgangsweise

1990 Coard/Yourdon OOA/OOD, Wirfs-Brock et.al. (RDD)

1991 Rumbaugh et.al. OMT (Object Modeling Technique). Anlehnung an Semantische Datenmodelle, eigenes dynamisches und "funktionales (Datenfluß-) modell

1992 Objectory (Jacobs**o**n) Use Cases, wenig detailliertes Objektmodell

1994/95 Booch/Rumbaugh/Jacobson: UML

Object Modeling Technique (OMT)

J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lorensen:

"Object-Oriented Modeling and Design",

Prentice Hall, Englewood Cliffs, New Jersey, 1991

- Ursprung: als konzeptionelles Modell im relationalen Datenbankentwurf (*CACM 4/88*)
- DSM (= Data Structure Manager) als objektorientierte Sprache, die Objekte und Beziehungen zwischen den Objekten explizit unterstützt (*OOPSLA 87/89*)
- Basis in der semantischen Datenmodellierung führt zu Anlehnung des Objektmodells an ER-Modell

UML und Softwareentwurf

- UML: Wird nicht mehr als "Methode" verkauft, sondern als Notation.
- Viele verschiedene Diagrammarten, wenig Strukturierung in der Anwendungsentwicklung
- Grundlegende Vorgangsweise:
  1. Ziele formulieren
  2. Aktoren identifizieren
  3. Use Cases modellieren
  4. Objektmodellierung
  5. Ablaufmodellierung

UML-Diagrammarten

- Systemgrenzen und wesentliche Funktionen: Use Cases (Anwendungsfälle), Actors (Aktoren, Akteure)
- KonkreteUse-Case-Situationen:Interaktionsdiagramme
- Statische Struktur des Systems:Klassendiagramme
- Objektverhalten: Zustandsübergangsdiagramme
- Implementierungsarchitektur:Komponentendiagramme und Einsatzdiagramme (Deployment Diagrams)

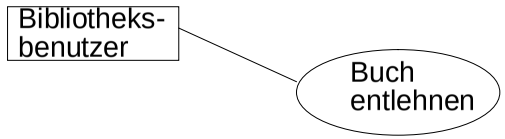
Einschub: Rational-Folien

Use Cases

- "Typischer" Anwendungsfall, beschreibt Menge von *Szenarien*
- Stichwortartige, ausschnittsweise Beschreibung von Teilen des Geschäftsprozesses, der durch die Anwendung unterstützt werden soll.
- Nicht Design- oder gar Implementierungsbeschreibung!
- Eher Beschreibung des Soll-Zustands
- Nicht durchexerzieren bis zum letzten Detail (Jacobson: für 10-Personenjahr-Projekt etwa 20 Use Cases; andere erwarten etwa 100).
- Wesentliches Mittel zur Analyse und Anforderungsermittlung

Use Cases (2)

Use-Case-Diagramme:



Typische Eintragungen in Use-Case-Beschreibung:

- Vorbedingungen für Systemzustand
- Nachbedingung (späteren Systemzustand)
- Beschreibung (z.B. numerierte Einzelschritte)
- Varianten (kleinere Abweichungen, z.B. erste Entlehnung durch diesen Benutzer

Use Cases (3)

- Regeln (Geschäftsregeln, Gültigkeitseinschränkungen etc.)
- Dienste (Services): Operationen und Objekte, die benötigt werden (teilw. erst später aufgefüllt)
- Ansprechpartner, offene Fragen
- Dialogbeispiele
- beteiligte Diagramme (z.B. Sequenz- und Kollaborationsdiagramme).

Entwurfskriterien

- Alle funktionalen Anforderungen durch Use Cases abgedeckt
- Jeder Use Case kommuniziert mit einem Aktor
- eindeutige, aussagekräftigen Namen
- richtige Granularität (zu kompliziert: aufteilen, sehr ähnlich: zusammenlegen)
- verständlich für Benutzer

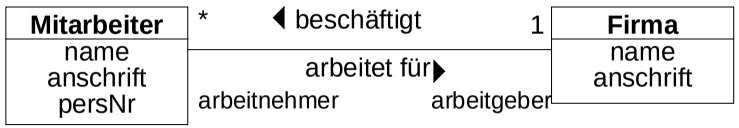
Objektmodell

benutzt "normale" objektorientierte Konzepte und Konzepte semantischer Datenmodelle

1. Assoziation - allgemeine Beziehung zw. Objekten
2. Aggregation - *partOf* -Beziehung zw. Objekten
3. Vererbung: Spezialisierung / Generalisierung

- *Attribute* von Objekten enthalten nur primitive Datenwerte, die zur Beschreibung des Zustands von Objekten verwendet werden: Zahlen, Strings
- Attributwerte sind keine Objekte und haben keine Identität

Assoziation - Beispiel



- *Assoziationen* beschreiben eine Menge von Links mit gleicher Struktur und Bedeutung
- Enden einer Assoziation: Rollen (*Roles*). **Achtung**: Nicht verwechseln mit Rollenkonzept von Sciore (Kap. Vererb.)
- Explizites Ordnen möglich

Assoziationen und Links

- *Links* sind Verbindungen/Beziehungen zwischen Objektinstanzen (Analogie: Tupel in einem relationalen System stellen Beziehungen zwischen Datenwerten dar)
- *Assoziationen* beschreiben eine Menge von Links mit gleicher Struktur und Bedeutung (Links sind Instanzen von Assoziationen).
  Beispiel: Personen *sind-angestellt-bei* Firmen
- Alle Links in einer Assoziation verbinden Objekte der gleichen Klasse
- Assoziationen sind unsymmetrisch, aber nicht gerichtet (keine Seite wird bevorzugt)

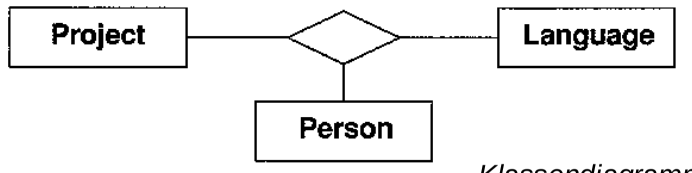
Rollen

- Eine Rolle, angewandt auf ein Objekt, bedeutet ein Objekt oder eine Menge von Objekten am anderen Ende der Assoziation
  (Implementierungsebene: Rollen durch Attribute/Referenzen dargestellt)
- Rollennamen sind notwendig und sollten eindeutig sein, wenn verschiedene Assoziationen dieselben Klassen betreffen

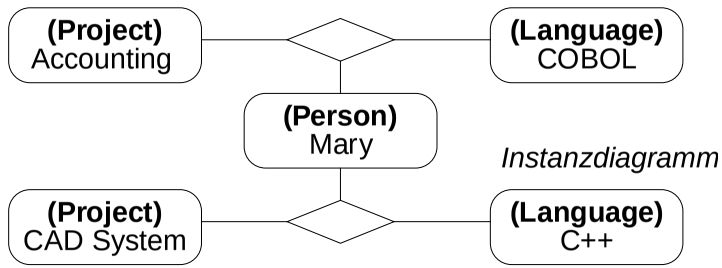
Warum Assoziationen

- Assoziationen drücken Beziehungen aus, die über einzelnen Objekten stehen, unabhängig von der Existenz dieser Objekte (andernfalls: Aggregation, siehe später)
- Darstellung von Assoziationen durch Referenzen auf Objekte ist eine Implementierungstechnik
- Benutzung von Referenzen bedeutet, daß inkonsistente Updates möglich sind, wenn Referenzen in beide Richtungen vorhanden
  Soll bei Löschen eines Objekts das Objekt an anderem Ende gelöscht werden?
- Problem: soll ein Objekt wissen, welche Objekte es verwenden?

Ternäre Assoziation

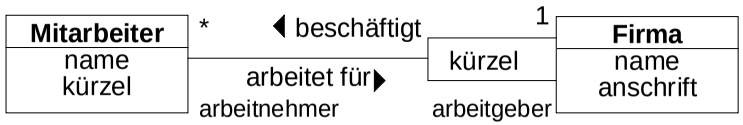


Klassendiagramm



Instanzdiagramm

Qualifizierte Assoziation



Qualifizierung bedeutet Angabe eines Attributs, das die Auswahl am anderen Ende einer Assoziation reduziert

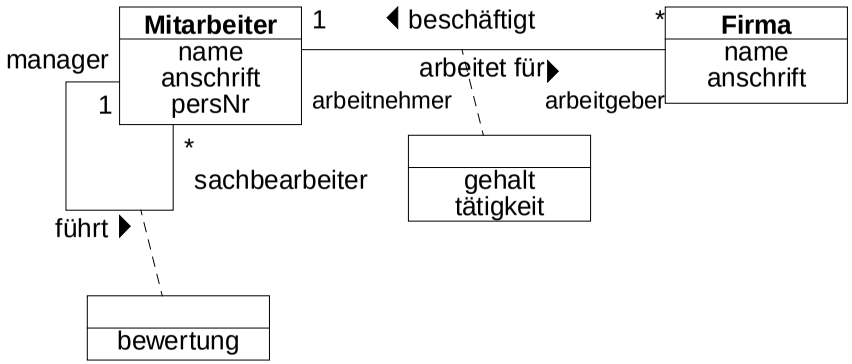
Zugriff immer ausschließlich über das qualifizierende Attribut

Assoziationsattribute

besonders oft (aber nicht ausschließlich) bei n:m-Assoziationen

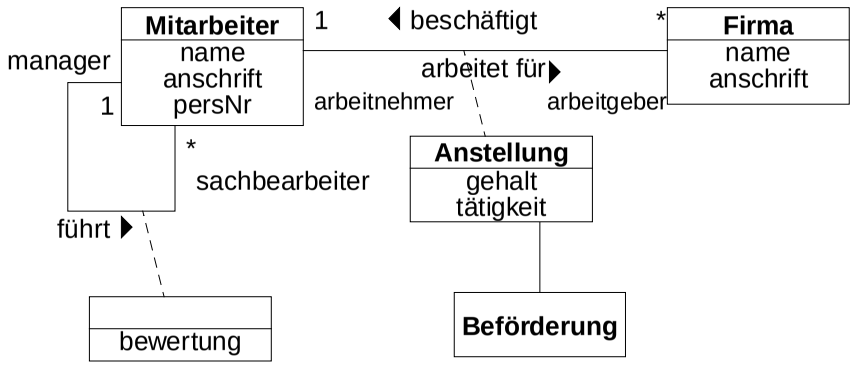
Darstellung von Assoziationsattributen durch Objektattribute ist bei 1:n-Beziehungen und 1:1-Beziehunge möglich, erfordert aber Modelländerung, wenn sich die Kardinalität der Beziehung ändert.

Assoziationsattribute - Beispiel



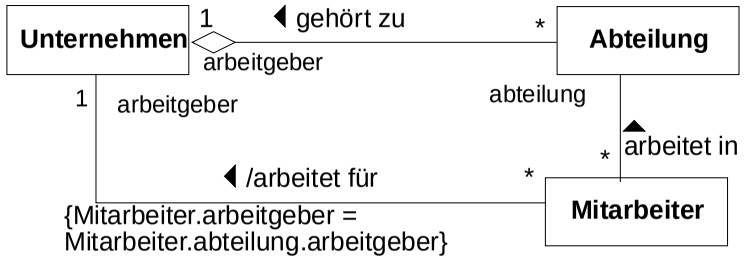
sogenannte “degenerierte Assoziationsklasse” (kein eigenständiges Objekt, kein Name)

Assoziation als Klasse



- Ausführen von Operationen auf Links
- Einbinden von Links in Assoziationen

Abgeleitete Assoziation

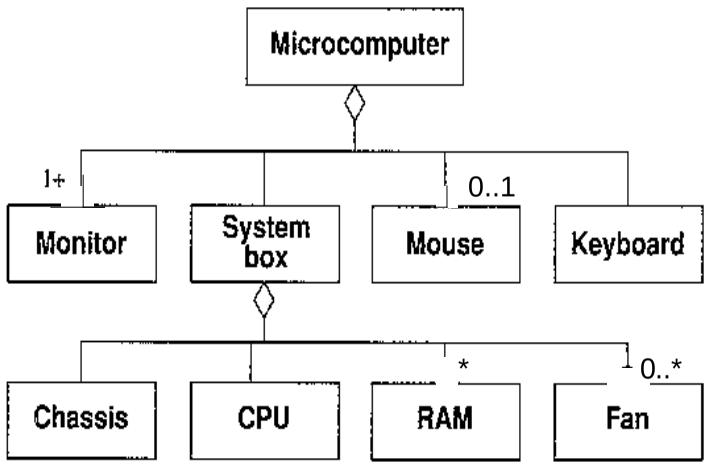


bei Bedarf berechnet

Aggregation

- Part-Of-Beziehung
- Eine Aggregationsbeziehung immer nur zwischen zwei Klassen (erlaubt unterschiedliche Kardinalitätsangaben)
- beliebig tief schachtelbar, möglicherweise rekursiv (vgl. Programmstrukturen: zusammengesetzte Statements)
- Auswahlkriterien gegenüber Assoziationen:
  - Wirklich eine Teile-Beziehung?
  - Werden Operationen automatisch an Teile weitergegeben? (*Propagierung*, z.B. Kopieren eines Dokuments bedingt Kopieren der Absätze)
  - Automatische Weitergabe von Werten nach oben?

Aggregation - Beispiel

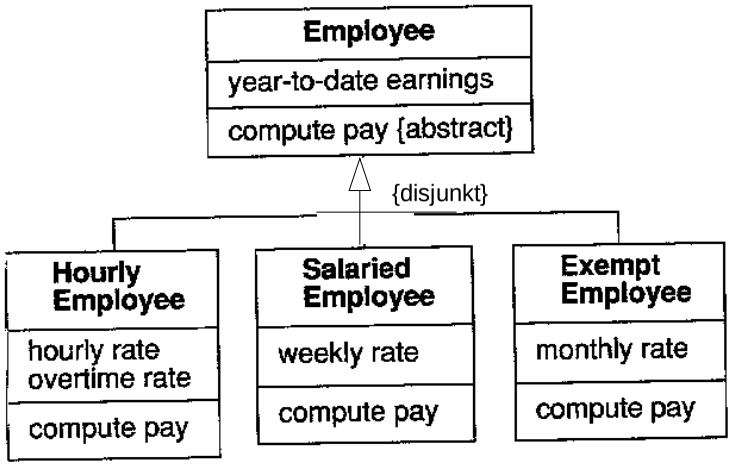


Komposition

Extremfall der Aggregation

Gekennzeichnet durch Existenzabhängigkeit

Spezialisierung/Generalisierung

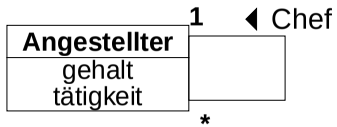


Vererbung

- verschiedene Kriterien: overlapping, disjoint, incomplete
- *Diskriminator*: Maßgeblicher Aspekt für die Unterteilung in Subklassen, kann explizit aufgelistet werden.
  z.B.: Anstellungstyp (Arbeiter), Antriebsart (Fahrzeuge)
- Mehrfachvererbung möglich

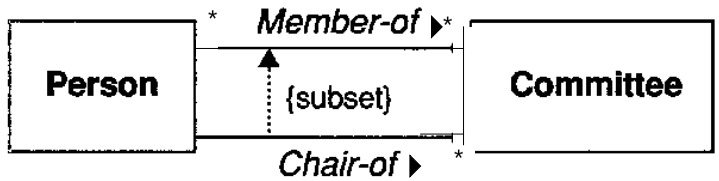
Constraints

- Angabe von (nahezu beliebigen) Konsistenzbedingungen



{gehalt <= Chef.gehalt}

Assoziationen mit Bedingungen



Beispiel: So nicht!

- Verschiedene Klassen werden durch eindeutige ID's (z.B. laufende Nummern) identifiziert
- Personen arbeiten für bis zu 3 Firmen
- Autos können Personen, Banken, oder Firmen gehören:

Person
name
alter
firma1 ID
firma2 ID
firma3 ID
person-ID
adresse

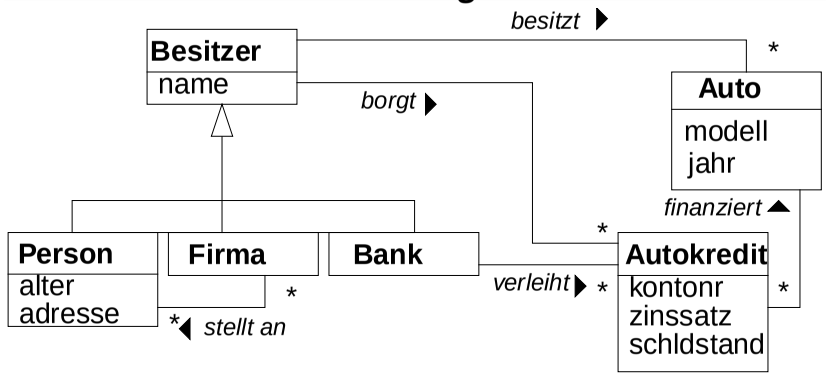
Auto
besitzer-ID
auto-ID
bestz-typ
modell
jahr

Autokredit
auto-ID
kundenart
kunden-ID
kontonr
bank-ID
zinssatz
schldstand

Firma
name
firmen-ID

Bank
name
bank-ID

Richtig



- Gewisse Erweiterungen: mehrere Kredite für ein Auto, mehrere Autos pro Besitzer etc.

Tips fürs Objektmodell

- Möglichst einfach
- Gute Namenswahl ist wichtig (und schwierig)
- Keine Pointer oder Referenzen - immer als Assoziationen
- Genau überprüfen, ob mehrstellige Assoziationen nicht in mehrere binäre zerlegt werden können (evt. mit qualifiern und Link-Attributen)
- Kardinalität muß nicht von Anfang an perfekt sein - später überprüfen
- Wo möglich, Qualifier verwenden

Tips fürs Objektmodell (2)

- Vererbungshierarchien nicht zu tief schachteln
- 1:1-Assoziationen sind verdächtig. Oft entweder optional (d.h. 0 ist möglich) oder mehrfach
- Fast immer sind Iterationen nötig, um alles richtig hinzubekommen
- Reviews durch andere sind sinnvoll
- Immer dokumentieren. Das Objektmodell beschreibt **was**, aber nicht **warum**
- Nicht alle Konstrukte müssen in jedem Modell vorkommen

Gliederung

- **Modul**: Gruppen von Klassen, die eine bestimmte Perspektive oder einen bestimmten Teil des Modells beschreiben.

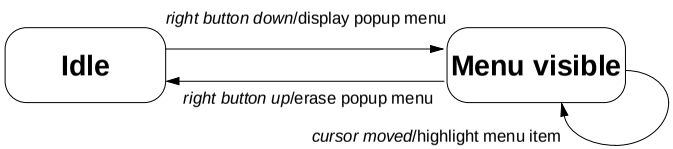
  z.B.: Installation, Elektrik, Lüftung, Personal als verschiedene Module eines Gebäudemodells
- Kopplung zwischen Modulen: Eine oder mehrere Klassen kommen in beiden vor
- Entwurfsrichtlinien: Mehr Verbindungen innerhalb eines Moduls als außerhalb (Einschätzung))
- Notation: Modulname auf dem Diagramm anführen (that's all)

Gliederung (2)

- **Blatt** (*Sheet*): Zerlegung eines Modells, so daß es auf eine Seite paßt
- nur ein Modul pro Blatt, sonst keine Einschränkungen
- Notation: Jedem Blatt eine eindeutige Nummer oder Name, so daß es auf anderen Dokumenten referenziert werden kann.

Zustandsdiagramm

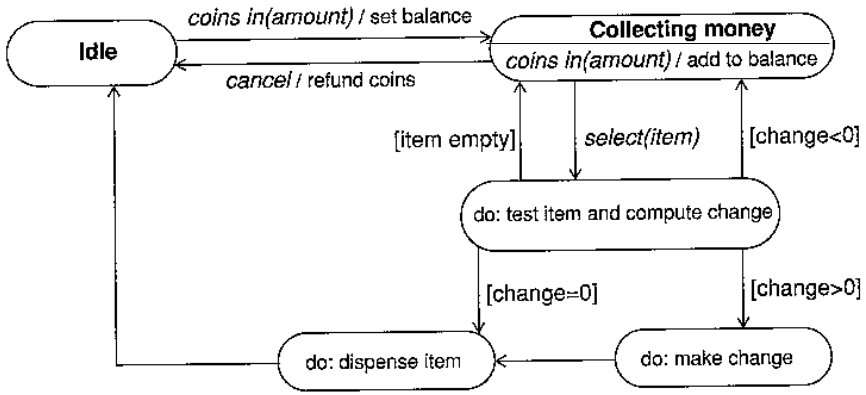
- Ereignis (*Event*): Ein Vorkommnis, Änderung des Systemzustands
- Zustand (*State*): Abstraktion für Attributwerte und Links eines Objekts
- Zustandsdiagramm: Zustände, die durch Ereignisse ineinander übergeführt werden (Transitionen)



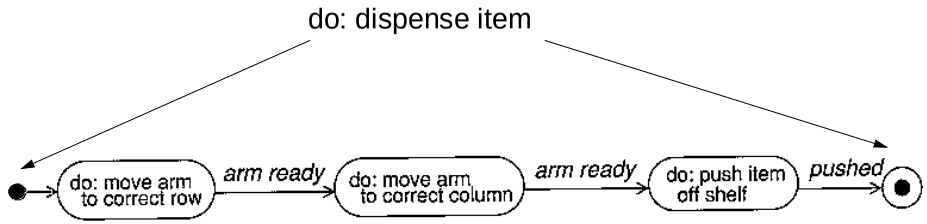
Zustandsdiagramm (2)

- Ein Zustandsdiagramm beschreibt die zeitlichen Abhängigkeiten im Verhalten **einer** Objektklasse
- Bedingungen beeinflussen die Transitionen
- Transitionen bedeuten Ausführung von Operationen (**Aktionen** - atomar), Zustände evt. Ausführung von **Aktivitäten** (länger anhaltend)
- Zustandsübergänge werden durch Operationen implementiert, und können Messages an andere Objekte auslösen
- Verschiedene Ereignisse können denselben Operationen (denselben empfangenen oder auch gesendeten Messages) entsprechen

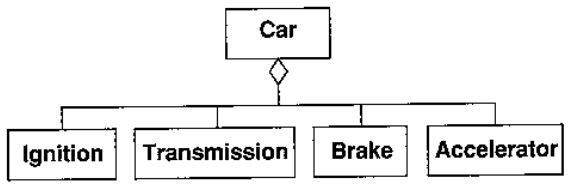
Zustandsdiagramm für Verkaufsautomat



Geschachteltes Zustandsdiagramm



Kommunikation zwischen Diagrammen

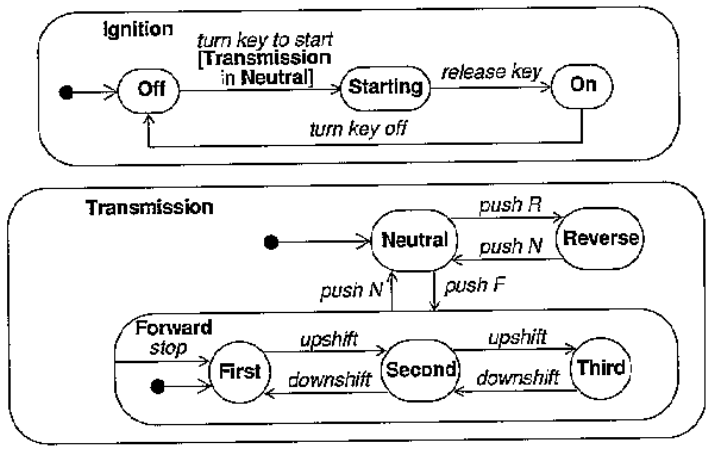


Ein komplexes System wird aufgebaut aus den Zustands-diagrammen der Objekte darin

Das Senden von Nachrichten (Schlüsselwort send) legt die Kommunikation zwischen diesen Diagrammen fest

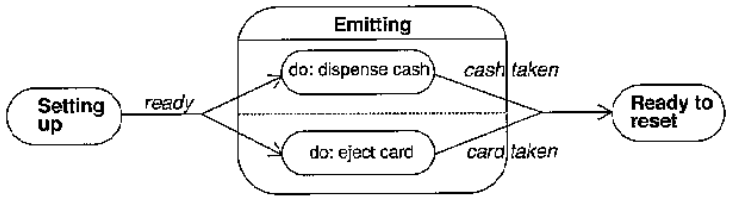
Sonderfall: Aggregation bedeutet Parallelität derZustands-diagramme der einzelnen Teile

... und zugehörige Zustandsdiagramme



Synchronisierung im Zustandsdiagramm

Beispiel aus “ABS - Automatischer Bankschalter”



Dynamisches Modell und Subklassen

- Zustandsdiagramme sind Beschreibungen der zulässigen Wertkonstellationen von Objektattributen
- Einschränkung des Verhaltens (semantische Sicht der Vererbung): zusätzliche Zustandsdiagramme, die sich nur auf neu hinzugekommene Attribute beziehen

Entwurfsprozeß

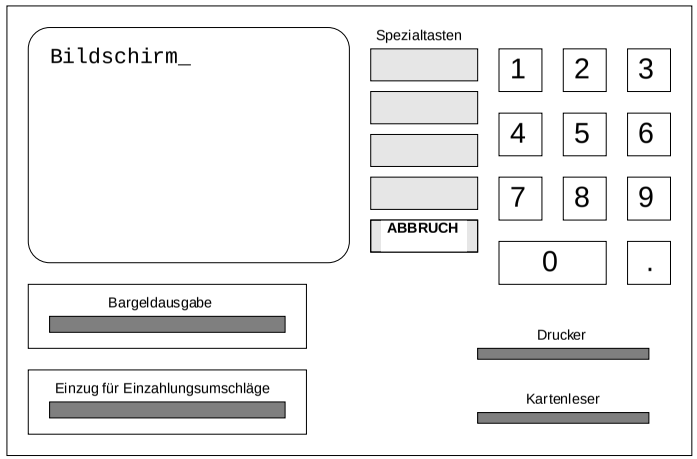
Möglicher Entwurfsprozeß, angelehnt an OMT (da UML keinen definiert)

1. Analyse
2. Systementwurf
3. Objektentwurf

➥ am Beispiel eines Bankomaten

- Analyse: Use Cases
  Objektmodell
  Dynamisches Modell

ABS - der Automatische Bankschalter



Analyse (1)

1. Problemspezifikation:

Die zu entwerfende Software soll ein Netzwerk von Bankschaltern, die einem Bankenkonsortium gehören, unterstützen. Es gibt automatische Bankschalter (ABS), die mit allen Banken des Bankenkonsortiums kommunizieren, und von Menschen bediente Bankschalter, die genau einer Bank zugeordnet sind. Jede Bank hat einen eigenen Computer zur Verwaltung ihrer Bankkonten und zur Durchführung der darauf arbeitenden Transaktionen. Von Menschen bediente Bankschalter sind direkt an einen Bankcomputer angeschlossen, sodaß Kassiere Konto- und Transaktionsdaten direkt eingeben können. ABS sind an einen Zentralcomputer angeschlossen, der wiederum mit den Bankcomputern kommuniziert. Ein ABS liest Bargeldkarten, interagiert mit Benutzern, kommuniziert mit dem Zentralcomputer, gibt Geld aus und druckt Quittungen. Das System muß über entsprechende Aufzeichnungs- und Sicherheitsmaßnahmen verfügen und parallele Zugriffe auf dasselbe Konto korrekt behandeln. Die Software für die einzelnen Bankrechner wird von den Banken selbst zur Verfügung gestellt. Die Kosten werden je nach Anzahl der Kartenträger auf die Banken aufgeteilt.

Analyse (2)

1. erstelle Use Cases

- Use-Case-Erstellung üblicherweise über Szenarien
- Achtung: üblicherweise nur ein kleiner Teil der möglichen Szenarien ausgearbeitet (sonst enormer Papierberg)
- Use-Case-Diagramm
- Typische Use-Case-Beschreibung: Akteur, auslösendes Ereignis, Voraussetzungen, Beschreibung des normalen Ablaufs, Kennzeichnung von Problemstellen, Ausnahmen

Analyse (2a)

1. Entwickle ein dynamisches Modell:
   - Erstelle Szenarios typischer Interaktionssequenzen
   - Identifiziere Ereignisse zwischen Objekten und erstelle ein *Sequenzdiagramm* für jedes Szenario
   - Erstelle ein *Kollaborationsdiagramm* für das System
   - Entwickle ein Zustandsdiagramm (ZD) für jede Klasse, die relevantes Verhalten zeigt.
   - Überprüfe die Konsistenz und Vollständigkeit der Ereignisse zwischen Zustandsdiagrammen

dynamisches Modell = ZDe + globales Ereignisfluß-D.

Achtung: Erstellung von dynamischem und Objektmodell verlaufen teilweise parallel

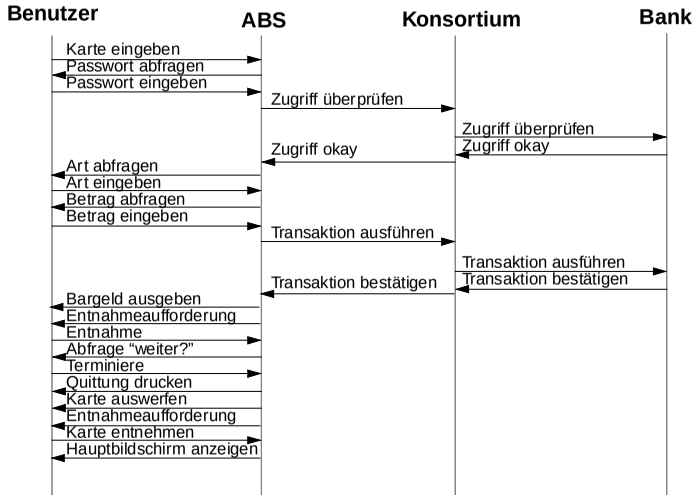
Szenario (1)

- Der ABS fordert den Benutzer zur Eingabe einer Karte auf; der Benutzer führt eine Karte ein.
- Der ABS zieht die Karte ein und liest die Seriennummer der Karte.
- Der ABS verlangt die Eingabe eines Passworts; der Benutzer gibt ein: “1234”.
- Der ABS läßt Seriennummer und Passwort beim Konsortium vergleichen; das Konsortium läßt die Daten bei Bank “39” überprüfen und gibt eine Akzeptanzbestätigung an den ABS.
- Der ABS fordert den Benutzer zur Auswahl einer Transaktion auf (Abhebung, Einzahlung, Überweisung, Kontoabfrage); der Benutzer wählt Abhebung.
- Der ABS erfragt den gewünschten Betrag; der Benutzer gibt 1000 öS ein.

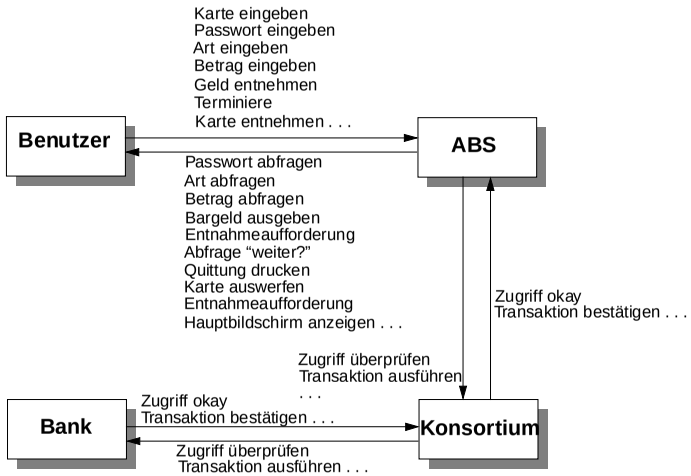
Szenario (2)

- Der ABS überprüft, ob die Transaktion im Rahmen der festgesetzten Grenzen ist und fordert das Konsortium auf, die Transaktion auszuführen; das Konsortium übergibt die Anforderung an die Bank, die schließlich die Transaktion bestätigt und den neuen Kontostand zurückliefert.
- Der ABS gibt das Bargeld aus und fordert den Benutzer auf das Geld zu entnehmen; der Benutzer entnimmt das Geld.
- Der ABS fragt den Benutzer, ob er fortfahren will; der Benutzer antwortet mit “nein”.
- Der ABS druckt eine Quittung, gibt die Bargeldkarte aus und fordert den Benutzer auf, diese zu entnehmen; der Benutzer entnimmt Quittung und Karte.
- Der ABS fordert einen Benutzer zur Eingabe der Karte auf

Dynamisches Modell - Sequenzdiagramm



Dynamisches Modell - Kollaborationsdiagramm



Analyse (3)

2. Erzeuge ein Objektmodell:
   - Identifiziere Objektklassen
   - Führe ein Data Dictionary mit Beschreibungen der Klassen, Attribute und Assoziationen (evt. CRC-Karten)
   - Erzeuge Assoziationen zwischen Klassen
   - Erzeuge Attribute für Klassen und Verbindungen
   - Teste Zugriffspfade mittels Szenarios. Iteriere.
   - Gruppiere Klassen in Module

Objektmodell = Objektmodelldiagramm + Data Dictionary

Klassenkandidaten (1)

Software | Banken-netzwerk | Kassier | ABS | Konsortium | Bank

Bank-computer | Konto | Transaktion | Schalter | Kontodaten | Transaktions-daten

Zentral-computer | Bargeld-karte | Benutzer | Bargeld | Quittung | System

Vorgangs-überwachung | Sicherheits-maßnahmen | Zugang | Kosten | Kunde

Transaktions-protokoll | Kommunikations-kanal

Auswahl von Kandidaten

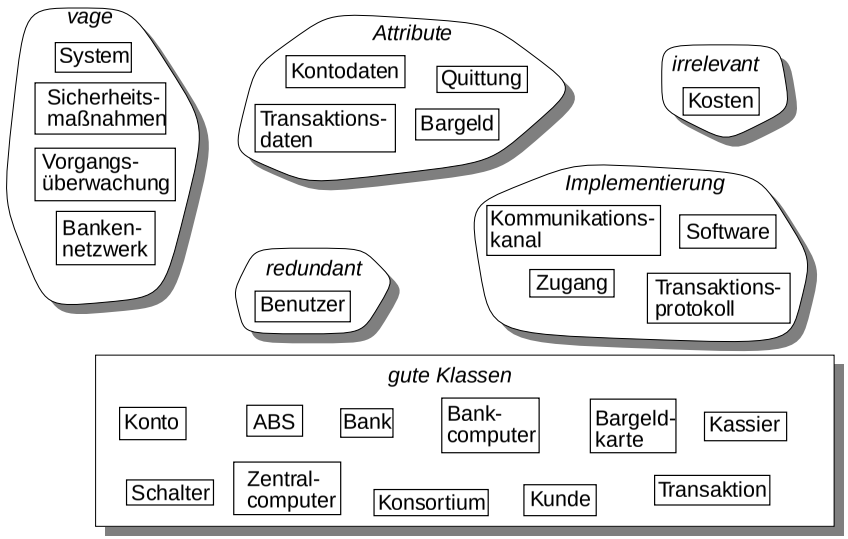
Bestimmte Klassen gehören nicht ins Modell

- Redundante Klassen: wenn ähnliche Konzepte, dann den [applikationsabhängig] besseren Namen behalten (Kunde? Passagier? Benutzer?)
- Irrelevante Klassen: hängt vom Kontext ab (z.B. Beruf eines Kunden ist für Theaterkartenreservierung unwichtig, Beruf eines Angestellten aber eventuell relevant)
- Vage Klassen: Vorgangsüberwachung
- Attribute: Kontodaten

Auswahl von Kandidaten (2)

- Operationen: Ist “Anruf” eine Folge von Aktionen, oder ein explizit vorhandenes Objekt mit Eigenschaften?
- Rollen: Klassennamen sollen allgemeines Wesen ausdrücken, nicht Funktion in speziellem Kontext (Besitzer? Person? Angestellter?)
- Implementierungskonstrukte: Kommunikationskanal

Klassenkandidaten (2)



Data Dictionary

Konto: Ein einzelnes Konto bei einer Bank, auf dem Transaktionen durchgeführt werden können. Es gibt Kontos verschiedener Typen (mindestens: Giro, Spar). Ein Kunde kann mehrere Konten haben,

ABS: Eine Maschine, an der durch eine Kundenkarte legitimierte Kunden ihre finanziellen Transaktionen selbst durchführen können. Der ABS kommuniziert mit dem Benutzer, um Transaktionsdaten aufzunehmen, sendet diese Daten zum Zentralcomputer, wo sie validiert werden und die Transaktionen ausgeführt werden. Der ABS gibt ggf. Bargeld an den Kunden aus. Wir nehmen an, daß der ABS nicht unabhängig vom Netz operieren muß.

Bank: Eine Finanzinstitution, die Konten für Kunden führt und an die Kunden Karten ausgibt, die diese zum Zugang zu ihren Konten am ABS-Netzwerk berechtigen. **...**

Bilden von Assoziationen

- Verbphrasen oder Partizipien (benachbart, enthalten, fährt, arbeitet für, Teil von)
- Anfang: keine Unterscheidung zwischen Assoziationen und Aggregation (kommt später)
- Überprüfen von Zugriffspfaden als Test für fehlende Assoziationen.
  Welche Abfragen (Queries) führe ich möglicherweise auf einem Objekt aus?

  Bsp.: Bei welcher Bank hat ein Kunde sein Konto?
  Bei welchem Schalter wurde eine Transaktion ausgeführt?

Eliminieren von Assoziationen

- Darstellung von Ereignissen statt permanenten Beziehungen: Kunde gibt Kontonummer ein
- Ternäre Assoziationen oft zerlegbar (*Firma zahlt Gehalt an Person, Kassier gibt Transaktion für Konto ein*)
- Ableitbare Assoziationen (*Großelternteil*)
  - Mehrfache Assoziationen zwischen denselben Klassen können Redundanz anzeigen
  - Das muß aber nicht der Fall sein (Bsp.: Komitee-Mitglieder, Komitee-Vorsitzende weiter vorn im Kapitel)
- Qualifier ausnützen!

Analog für Attribute

Attribute sind selten vollständig separat beschrieben (z.B. die Farbe des Autos)

- unabhängige Existenz wichtig -> Objekt (z.B. Chef - Objekt, Gehalt - Attribut). Ist 'Stadt' Attribut oder Objekt? Abhängig von Anwendung
- Qualifiers: Attribute, die vom Kontext abhängig sind. Häufig Namen (z.B. Abteilungsname)
- Objekt-Identifier nicht explizit auflisten, außer sie sind im Problembereich vorhanden: Kontonummer
- Link-Attribute
- interne Werte

Anderes Beispiel

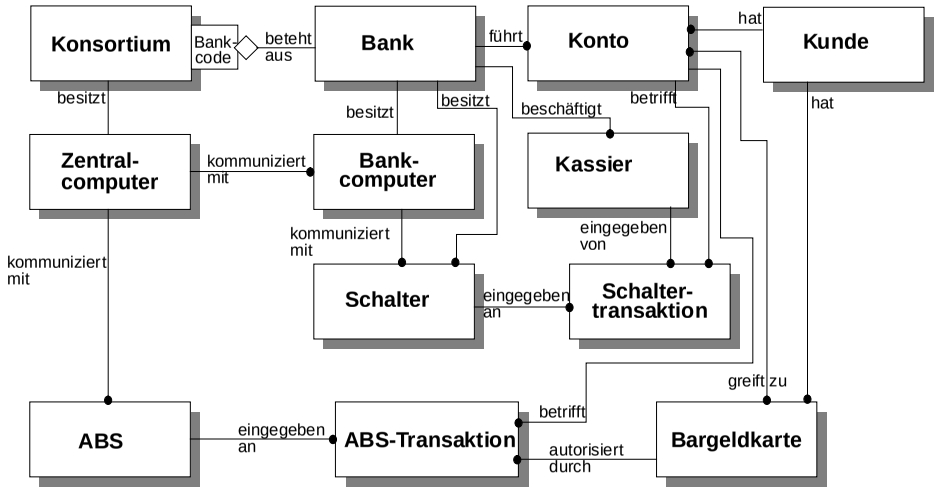
Finden von Klassenkandidaten:

1. Variante: Zu modellieren ist ein gerichteter Graph, bestehend aus Knoten und Kanten. Es gibt einen eindeutigen Einstiegsknoten. Von einem Knoten können mehrere Kanten ausgehen.

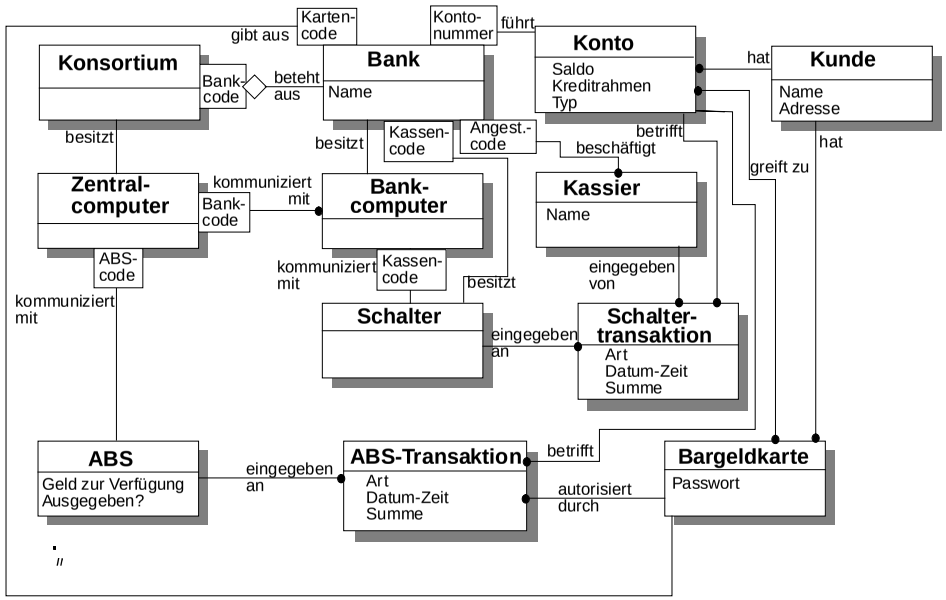
2. Variante: Kanten sollen eine Bezeichnung haben

3. Variante: Der Graph speichert die Anzahl seiner Knoten .

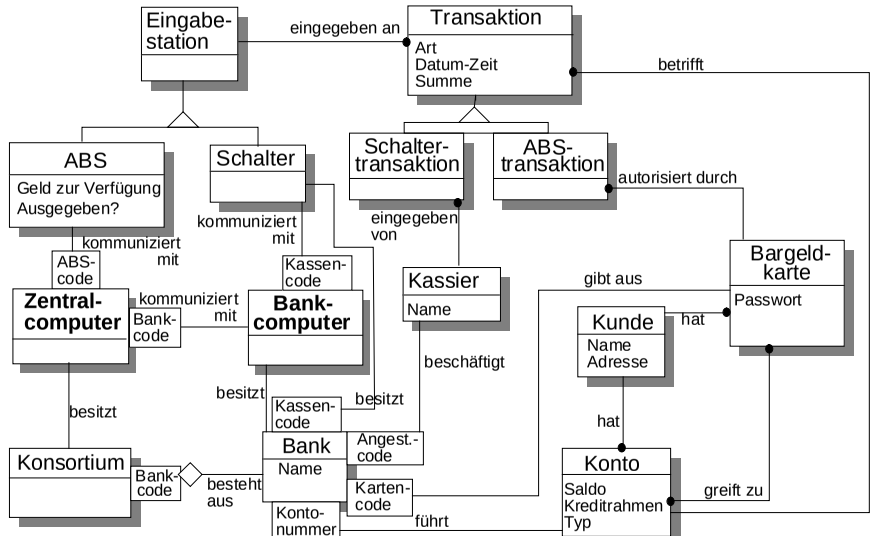
Klassendiagramm



Einführung von Attributen



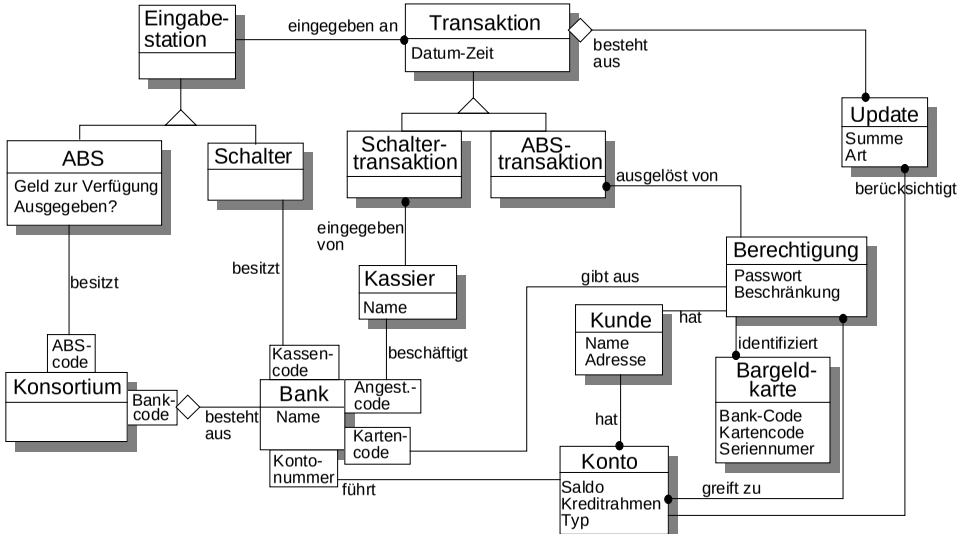
Berücksichtigung von Vererbung



Weitere Verfeinerung und Korrektur

- asymmetrische Assoziationen -> analoge neue Klassen
- Aufteilen von Klassen mit mehreren unterschiedlichen Aufgabenbereichen
- Aufteilen bei Problemen mit der Generalisierung
- Doppelte Assoziationen -> Oberklasse einführen
- Rolle besonders wichtig -> vielleicht neue Klasse (Anstellung - Angestellter)
- Assoziationen: Fehlende Zugriffspfade, Redundanz, Mangel an Operationen, die eine Assoziation verwenden
- Fehlen von Zugriffspfaden für Operationen
- Umwandeln von Attributen in Qualifier

Verfeinertes Klassendiagramm



Forts. dynamisches Modell-Ereignisse

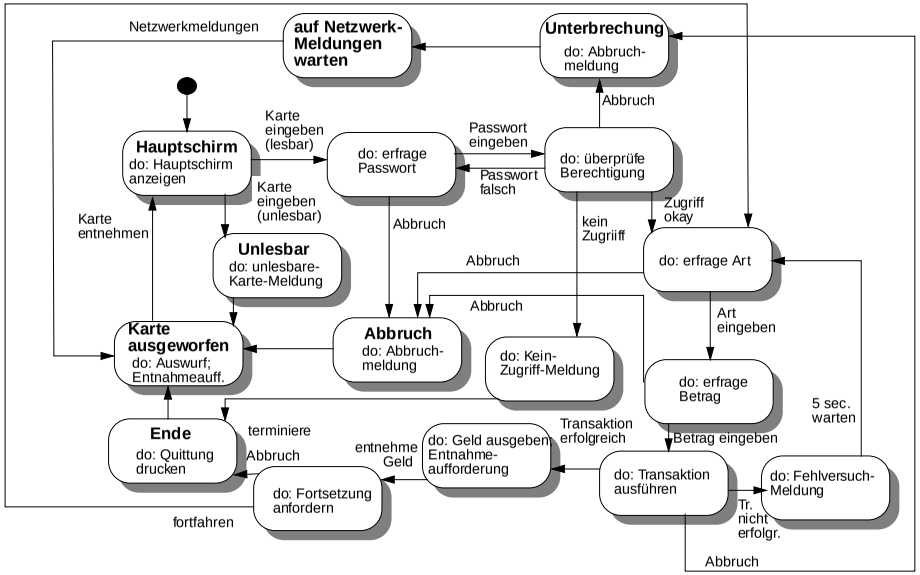
- Signale, Eingaben, Entscheidungen, Interaktionen mit der Außenwelt, Informationsübermittlung zwischen Objekten,...
- Unterscheidung zwischen verschiedenen Klassen,abhängig von Parametern

  z.B: Drücken der Return-Taste ist ein anderer Event als Drücken einer numerischen Taste, auch wenn beide auf derselben Tastatur liegen.
  z.B: Password eingeben ist unabhängig vom Wert
- keine objektinternen Vorgänge

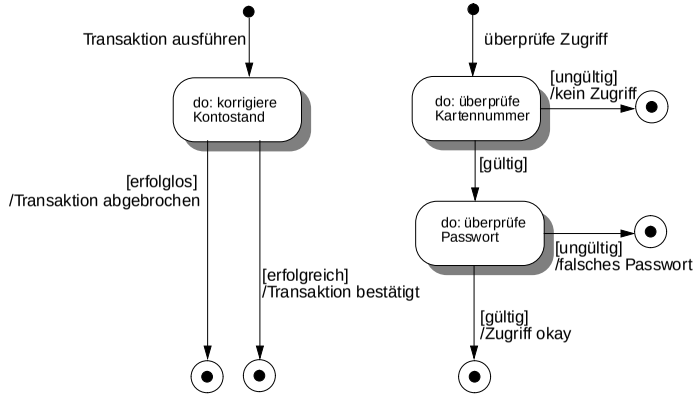
Zustandsdiagramm

- Ausgangspunkt: Event Traces
- Betrachte einen Event Trace, eingeschränkt auf ein Objekt
- Erstelle einen Pfad, dessen Kanten mit den Ein- und Ausgangskanten dieser Klassen in dem Event Trace übereinstimmen
- Füge die Event Traces von alternativen Szenarien ein
- Finde Schleifen
- Mit zunehmender Erfahrung wird die Notwendigkeit, Event Traces aufzustellen, geringer

Zustandsdiagramm für die Klasse ABS



Zustandsdiagramm für die Klasse Bank



Analyse (5a)

2. Überprüfe, iteriere und verfeinere die Modelle:

- Füge in der Konstruktion des dynamischen Modells gefundene Operationen in das Objektmodell ein. Zeige nicht alle Operationen in der Analysephase, um das Objektmodell nicht zu überladen. Zeige nur die wichtigen Operationen.
- Überprüfe, ob die Klassen, Assoziationen, Attribute und Operationen auf dem gegebenen Abstraktionsniveau vollständig sind. Vergleiche die Modelle mit der Problembeschreibung und der Kenntnis des Problembereichs. Teste die Modelle durch Szenarien.

Analyse (5b)

- Entwickle verfeinerte Szenarien (inklusive Fehlerbedingungen, o.ä) als Variationen der Basisszenarios. Benutze diese “Was-wäre-wenn-Szenarios” zur Überprüfung der Modelle.
- Wiederholung

Analysedokument = Problembeschreibung + Objektmodell + dynamisches Modell

Iteration

- Überprüfung und Korrektur
- Suche nach fehlenden Objekten
  - Aufteilen von inhaltlich getrennten Attributen/Assoziationen
  - redundante Assoziationen deuten auf fehlende Oberklassen
- Suche nach überflüssigen Klassen
  - zuwenig Attribute/Operationen/Assoziationen
- Suche nach fehlenden Assoziationen
  - Fehlende Zugriffspfade

Systementwurf (1-5)

1. Unterteile das System in Subsysteme
2. Identifiziere Parallelität
3. Weise Subsysteme Prozessen und Prozessoren zu
4. Wähle Strategien (bzgl. Datenstrukturen, Files, Datenbanken) zur Implementierung von Datenspeichern
5. Identifiziere globale Resourcen und lege Zugriffsmechanismen darauf fest

Systementwurf (6-8)

6. Wähle einen Ansatz, um den Kontrollfluß zu implementieren:
7. Beachte Grenzfälle
8. Setze Prioritäten für Kompromisse

Systementwurfsdokument = Struktur der Basisarchitektur des Systems + strategische Entscheidungen

Kontrollfluß (1)

- Repräsentiere den Systemzustand durch die aktuelle Stelle im Programmablauf

```
do forever
   Hauptschirm anzeigen
   Karte lesen
   repeat
      Password erfragen
      Password lesen
      Berechtigung überprüfen
   until Berechtigung ok or Abbruch
   repeat
      Transaktionsart erfragen
   ...
```

- Implementiere direkt einen endlichen Automaten
- Implementiere parallele Prozesse

Objektentwurf (1)

- *Detaillierung* und fallweise *Erweiterung* des Modells

1. Ermittle Operationen für das Objektmodell aus den anderen Modellen:
   - Definiere (je nach Implementierung des Kontrollflusses) eine Operation für jedes Ereignis des dynamischen Modells (Aktionen, Aktivitäten)

Objektentwurf (2)

2. Entwirf Algorithmen für die Operationen:
   - Wähle Algorithmen, die minimale Implementierungskosten erfordern
   - Wähle angemessene Datenstrukturen
   - *Falls notwendig, definiere (bzw. verwende) neue interne Klassen und Operationen*

     z.B. Klassen für Graphikfunktionen (vordefiniert oder speziell für die Anwendung geschrieben)
   - Weise die Zuständigkeit für Operationen zu, die nicht eindeutig zu einer Klasse gehören

Objektentwurf (3)

3. Optimiere Zugriffspfade:
   - Füge zusätzliche (redundante) Assoziationen ein, um den Zugriff auf Daten bequemer zu machen

Suche alle japanischsprechenden Angestellten (Annahme: 1000 Ang., 10 Fähigkeiten, 5 sprechen jap.)
Lösung bei häufiger Abfrage: *Indexassoziation*

Objektentwurf (3-4)

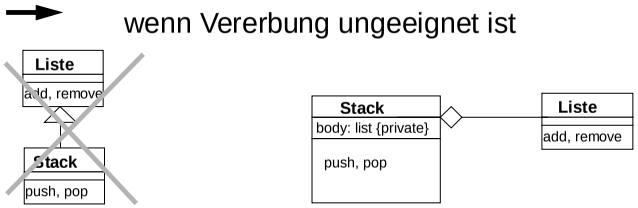
- Weg dorthin: Operationen untersuchen, welche Assoziationen sie traversieren (in beide Richtungen; in eine?)
- Wie stark fächert die Operation aus? Wieviele Treffer entstehen typischerweise?
- Überarbeite die Berechnungen im Hinblick auf größere Effizienz (z.B. Gruppieren von mehrfachen-Zugriffen auf dieselbe Assoziation)
- Speichere abgeleitete Werte, um die Berechnung komplexer Ausdrücke zu vermeiden

4. Implementiere den Kontrollfluß nach der im Systementwurf festgelegten Strategie

Objektentwurf (5)

5. Überarbeite die Klassenstruktur, um Vererbung besser auszunutzen:

- Gruppiere Klassen und Operationen um
- Verlagere gemeinsames Verhalten von Gruppen von Klassen in gemeinsame Superklasse
- Benutze Delegation, um das Verhalten einer Klasse von mehreren Klassen unterstützt zu realisieren, wenn Vererbung ungeeignet ist



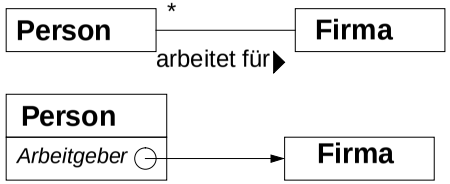
Objektentwurf (6)

6. Entwirf die Implementierung von Assoziationen:

- Analysiere die Traversion von Assoziationen
- Implementiere jede Assoziation als eigenständiges Objekt oder als objektwertiges Attribut einer oder beider Klassen der Assoziation

Implementierung von Assoziationen

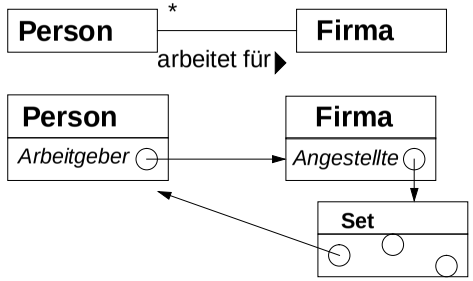
- Abhängig von der Zugriffs- (Traversions-) richtung
- Einbahn: darstellbar durch Pointer




```
Object subclass: #Person
    instanceVariables: 'arbeitgeber'
    ...
```

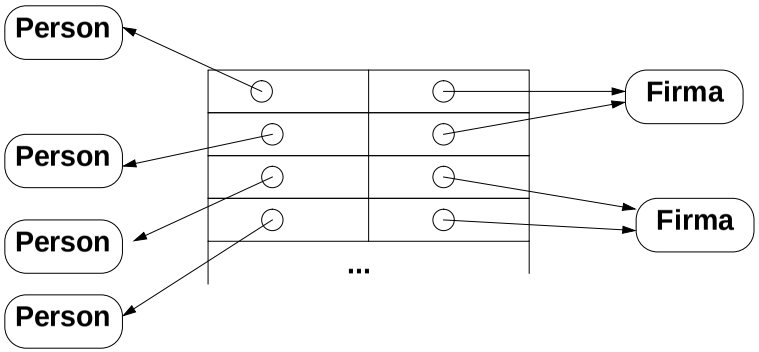

Implementierung von Assoziationen (2)

- 1:n
  - Attribut in eine Richtung, Absuchen in der anderen Richtung, wenn nötig (nur wenn andere Richtung selten)
  - Attribute in beide Richtungen



Implementierung von Assoziationen (3)

- Eigenes Assoziationsobjekt (auch für m:n-Beziehungen)



- komplexere Varianten davon für Qualifier

Objektentwurf (7-8)

7. Lege die genaue Repräsentation der Objektattribute fest
8. Fasse Klassen und Assoziationen zu Modulen zusammen

Entwurfsdokument = detailliertes Objektmodell + detailliertes dynamisches Modell