Software Engineering - Georg Kuschk – Mitschrift06

--- 02.12.2005 ---

5.1.c)

```
Betrag = 0
Preis = 0
Fahrkarte = null          *keine Fahrkarte gewünscht*
T=0                       *in sekunden*

while (true)
{
  if (gedrückte Taste == Kurzstrecke) then
    Fahrkarte = Kurzstrecke
    Preis = Kurzstreckenpreis
    T=0
  endif

  if (gedrückte Taste == Normal) then
        ....
  if (gedrückte Taste == Tageskarte) then
        ....

  if (eingegebene Geldmünze == 10) then
    Betrag += Geldmünze
    T=0
  endif

  ... für alle Münzen ...

  if (Betrag >= Preis) then
    Karte ausgeben
    T=0
    Betrag=0
    Fahrkarte=null
  endif

  warte(1s)

  if (T>=60) then
    Betrag zurückgeben
    T=0
    Betrag=0
    Fahrkarte=null
  endif
}
```

Zustandsautomaten

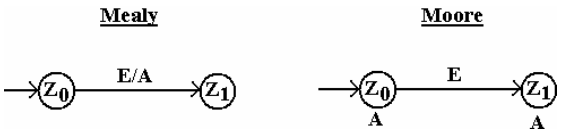
- Mealy-Automaten
  $M = (Q, \Sigma, \Delta, q_0, d, l)$
  Q : Zustandsmenge
  Σ : Eingabealphabet (endliche Menge)
  Δ : Ausgabealphabet
  q_0 : Startzustand
  d : Zustandsübergangsregeln $Q \times \Sigma \rightarrow Q$
  l : Ausgaberegeln $Q \times \Sigma \rightarrow \Delta$

- Moore-Automaten
  $M = (Q, \Sigma, \Delta, q_0, d, m)$
  m : Ausgaberegeln $Q \rightarrow \Delta$

- Akzeptor
  = spezieller Moore-Automat mit Finalzuständen

Darstellungsformen :

1.) markierte , gerichtete Graphen :



2.) Zustandsmatrix / Adjazenzmatrix :

Zustände \ Eingabe	e_0	e_1	e_2	...	e_m
q_0	...	...	...	...	...
q_1	...	...	m_{12}	...	...
...	...	...	...	...	...
q_n	...	...	...	...	...

Mealy : m_{ij} = Nachfolgezustand / Ausgabe

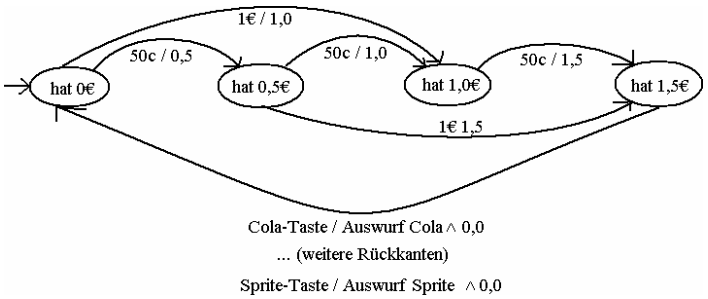
Moore : m_{ij} = Nachfolgezustand ; und die Ausgabe in eine neue Spalte a_i

3.) Zustandstabelle / Relationale Darstellung :

Aktueller Zustand	Eingabe	Ausgabe	Nachfolgezustand

5.2)

1.) gerichteter , markierter Graph :



2.) Zustandsübergangstabelle :

Akt. Zustand	Ereignis	Aktion	Nachfolgezustand
Hat 0€	50c	0,5	Hat 0,0€
Hat 0€	1€	1,0	Hat 1,0€
Hat 0,5€	50c	1,0	Hat 1,0€
Hat 0,5€	1€	1,5	Hat 1,5€
Hat 1€	50c	1,5	Hat 1,5€
Hat 1,5€	Cola-Taste	Auswurf Cola	Hat 0€
Hat 1,5€	Sprite-Taste	Auswurf Sprite	Hat 0€
Hat 1,5€	...	...	Hat 0€

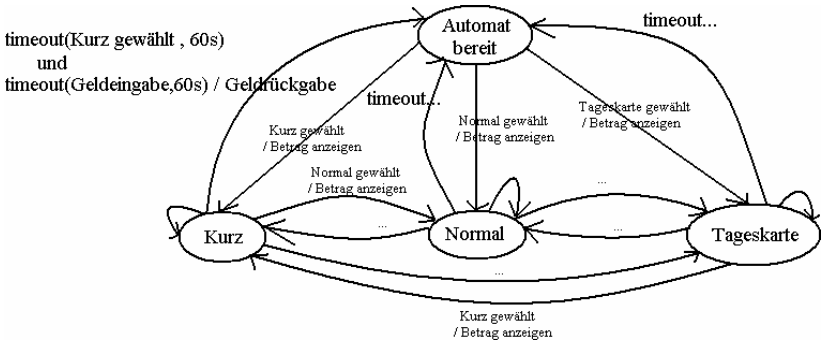
3.) Zustandsmatrix :

	50c	1€	Cola-Taste	Sprite-Taste
Hat 0€	Hat 0,5€ / 0,5	Hat 1,0€ / 1,0	-	-
Hat 0,5€	Hat 1,0€ / 1,0	Hat 1,5€ / 1,5	-	-
Hat 1€	Hat 1,5€ / 1,5	-	-	-
Hat 1,5€	-	-	Hat 0€ / Auswurf Cola AND 0,0	Hat 0€ / Auswurf Sprite AND 0,0

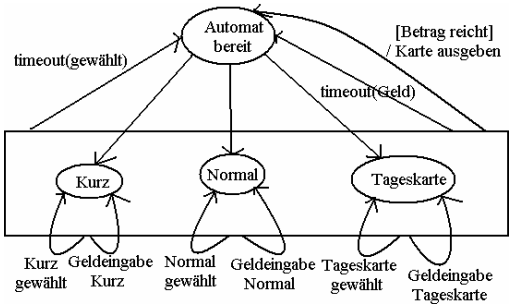
= Mealy

Bei Moore-Automaten die Ausgaben in extra Spalten abspalten

4.) Statechart :



5.) Hierarchischer Automat



Statecharts

Erweiterung von Mealy- und Moore-Automaten

- hybride Zustandsautomaten (Kombination aus beiden)
- bedingte Zustandsübergänge E [B] / A
- hierarchische Zustandsautomaten (mit disjunkten Zustandsmengen der Unterautomaten)
- Zustände mit Gedächtnis (Kreis mit H für 'history')
- nebenläufige Automaten (Produktautomat)

--- 09.12.2005 ---

5.3)

Gegebener Automat =

- endlicher Automat
- nebenläufiger Automat
- hierarchischer Automat mit Gedächtnis

5.3.a)

$$[A] \xrightarrow{a} \underset{\text{Startzustände}}{[C,F]} \xrightarrow{e} [D,F] \xrightarrow{g} [D,G] \xrightarrow{g} [D,I] \xrightarrow{g} \underset{\text{merken}}{[D,J]} \xrightarrow{b} [B] \xrightarrow{c} \underset{\text{gemerkt}}{[D,J]}$$

(beim nächsten Betreten im gemerkten Zustand)

5.3.b)

$$\underset{\text{Startzustand}}{[A]} \xrightarrow{a} [C,F] \xrightarrow{g} [C,G] \xrightarrow{g} [C,I] \xrightarrow{h} [C,G] \xrightarrow{b} [B] \xrightarrow{a} [A] \xrightarrow{a} \underset{\text{merken}}{[C,G]} \xrightarrow{a} [A] \xrightarrow{a} \underset{\text{da gemerkt}}{[C,G]}$$

Petri-Netze

- Modellierung , Analyse und Simulation von dynamischen Systemen und nichtdeterministischen Vorgängen
- markierter , gerichteter , i.A. zusammenhängender , bipartiter Graph

$P = (S, T, E, m, m_0)$

S : Stellen (-menge) = Zustände im System - Kreis

T : Transitionen (-menge) = Zustandsübergänge - Balken

E : gerichtete , markierte Kanten - Pfeil

m : Markenbelegung (Stellen mit Marken/tokens) , $M(S_i)$ = Anzahl der Marken der Stelle S_i

m_0 : Startmarkierung

Marken : Belegung der Stellen mit Objekten

(genauere Definition siehe Wikipedia)

Arten von Petri-Netzen :

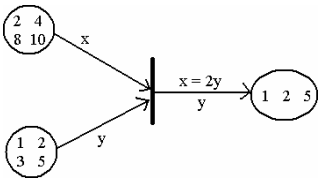
1.) einfaches Petri-Netz :
- Schaltregeln :
  - Transition t „kann feuern“ :
    Wenn jede (geladene) Eingabestelle mindestens eine Marke hat.
  - t „feuert“ :
    Jede Eingabestelle verliert eine Marke.
    Jede Ausgabestelle bekommt eine Marke.

2.) Bedingungs-Ereignis-Netz :
- höchstens eine Marke pro Stelle
- Schaltregeln :
  - t kann feuern :
    Alle Vorgänger müssen je eine Marke haben und alle Nachfolger keine.
  - t feuert :
    Wie bei einfachen Petri-Netzen.

3.) Stellen-Transitions-Netz :
- Jede Stelle hat eine Kapazität $K \in N \quad , (S \to N)$ (natürliche Zahlen)
  =maximale Anzahl der Marken je Stelle
- Notation : k=n bzw. keine Notierung entspricht $k = \infty$
- Jede Kante hat ein Gewicht $W : (S \times T) \cup (T \times S)$
  Notation : $O \overset{W(s,t)}{\to} |_t \quad bzw. \quad |_t \overset{W(t,s)}{\to} O$
- Schaltregeln :
  - t kann feuern
    Jede Eingangsstelle hat mindestens genauso viele Marken wie das Kantengewicht und die Kapazität jeder Ausgangsstelle darf nicht überschritten werden.
  - t feuert
    Aus jeder Eingangsstelle S_i werden $w(S_i, t)$ Marken entfernt und zu jeder Ausgangsstelle S_j werden $w(t, S_j)$ Marken hinzugefügt.

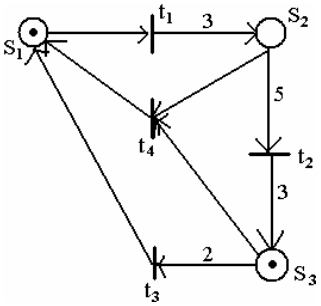
4.) Prädikat-Transitions-Netz :
- gefärbte Marken
- Transitionen : Schaltbedingungen mit Variablen
- gerichtete Kanten mit Variablennamen beschriftet (Notation siehe Beispiel)
- Schaltregeln :
  - t kann feuern
    Nur bei gefärbten Marken , welche die Schaltbedingungen erfüllen.
  - t feuert
    Verbraucht Marken , welche die Schaltbedingungen erfüllen und erzeugt neue gefärbte Marken (Schaltwirkung).

Beispiel für P/T-Netz :



Eigenschaften von Petri-Netzen :
- Erreichbarkeit
- Lebendigkeit (tote und lebendige Marken)
- Fairness
- Invarianten

--- 16.12.2005 ---

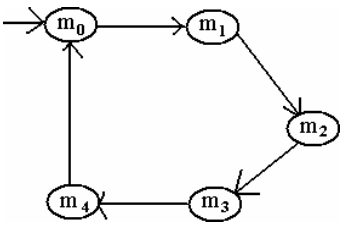


= Stellen-Transitions-Netz

Anfangsmarkierung : $m_0 = \begin{pmatrix} M(S_1) \\ M(S_2) \\ M(S_3) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

Erreichbarkeitsgraph :

$$\underset{m_0}{\begin{pmatrix}1\\0\\1\end{pmatrix}} \xrightarrow{t_1} \underset{m_1}{\begin{pmatrix}0\\3\\1\end{pmatrix}} \xrightarrow{t_4} \underset{m_2}{\begin{pmatrix}1\\2\\0\end{pmatrix}} \xrightarrow{t_1} \underset{m_3}{\begin{pmatrix}0\\5\\0\end{pmatrix}} \xrightarrow{t_2} \underset{m_4}{\begin{pmatrix}0\\0\\3\end{pmatrix}} \xrightarrow{t_3} \underset{m_5=m_0}{\begin{pmatrix}1\\0\\1\end{pmatrix}}$$



Der Graph ist zyklisch , also stark zusammenhängend. Jede Markierung ist somit von sich selbst aus erreichbar.
$\Rightarrow$ Alle m_i sind Heimatmarkierungen

Stelleninvariante :

ges.: $x \in N^3$, so dass $\forall j : m_j{}^T x = const$ für alle von m_0 aus erreichbaren Markierungen m_j

$$m_j = m_0 + \underbrace{C}_{\text{Adjazenzmatrix}} \cdot \underbrace{d_j}_{\text{Schaltvektor}}$$

$$\begin{aligned} m_j{}^T x &= m_0{}^T x + (C \cdot d_j)^T x = const \\ &= m_0{}^T x + \underbrace{d_j{}^T \cdot C^T x}_{=0} = const \end{aligned}$$

$$\Rightarrow C^T x = 0$$

Adjazenzmatrix :

$$C = \begin{pmatrix} -1 & 0 & 1 & 1 \\ 3 & -5 & 0 & -1 \\ 0 & 3 & -2 & -1 \end{pmatrix}$$

(Zeilen = Stellen S1,S2,S3 , Spalten = Transitionen t1,t2,t3,t4)

wegen obiger Forderung $C^T x = 0$ muss also gelten :

$$\begin{pmatrix} -1 & 3 & 0 \\ 0 & -5 & 3 \\ 1 & 0 & -2 \\ 1 & -1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Lösung des überbestimmten LGS ergibt die triviale Lösunge $x =$ Nullvektor
$\Rightarrow$ keine Stelleninvariante

Transitionsinvariante :

ges.: Menge von Transitionen , die eine Markierung m in sich selbst überführt.

$$m = m + C \cdot d \qquad , \; d \neq 0 \; , \; d \in N^4$$

D.h. $C \cdot d = 0$

$$M_i = \{\{t_1, t_4, t_1, t_2, t_3\}\} \qquad , \qquad d = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

- Transition t_1 feuert im Zyklus 2mal

(Jedes ganzzahlige Vielfache von d ist Lösung)

$$C \cdot d = \begin{pmatrix} -1 & 0 & 1 & 1 \\ 3 & -5 & 0 & -1 \\ 0 & 3 & -2 & -1 \end{pmatrix} \cdot \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

unterbestimmtes LGS lösen :

$\Rightarrow \quad d_1 = d_2$, d.h. $d_2 = s \in N \setminus \{0\}$

$d_1 = d_3 = d_4 = 2s$

$$\Rightarrow d = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \cdot s$$

beschränktes Netz : Anzahl der Markierungen ist beschränkt (Anzahl hier : $5 < \infty$)

Lebendigkeitsanalyse :
Netz ist lebendig , wenn Anfangsmarkierung lebendig ist

S/T-Netz ist lebendig , wenn :
- Jede Transition kommt in mindestens einer T-Invarianten der Basis D vor.
  (Basis : linear unabhängige , raumaufspannende Vektoren)
- Der Erreichbarkeitsgraph der mit $d = \sum_{d' \in D} d'$ konsistenten Schaltfolgen ist stark zusammenhängend.

$D =$ 1-dimensional (Gerade)

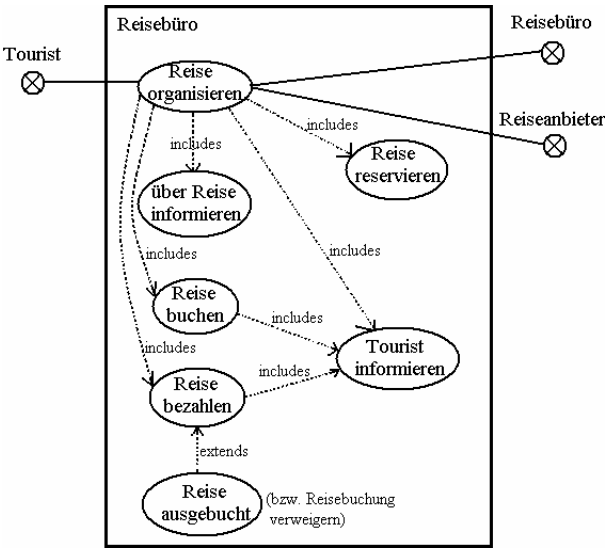
$$\Rightarrow d = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$\Rightarrow$ Beide obigen Bedingungen sind erfüllt , d.h. das Netz ist lebendig

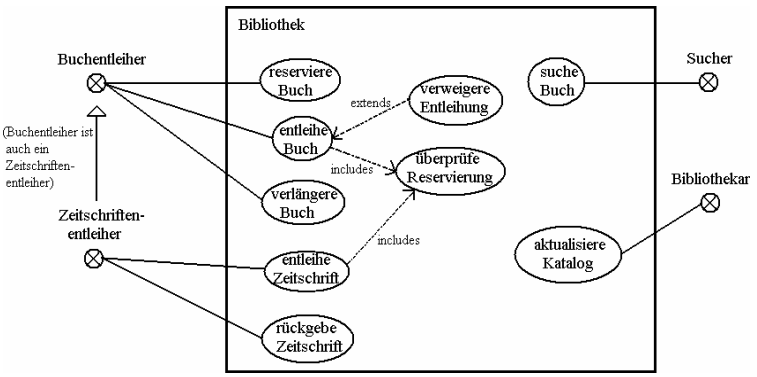
konservatives Netz :
Für alle Stellen existiert eine Stelleninvariante x mit $x_j > 0$

Hier : Da $x = 0 \quad \Rightarrow$ nicht konservativ

Anwendungsfalldiagramm :

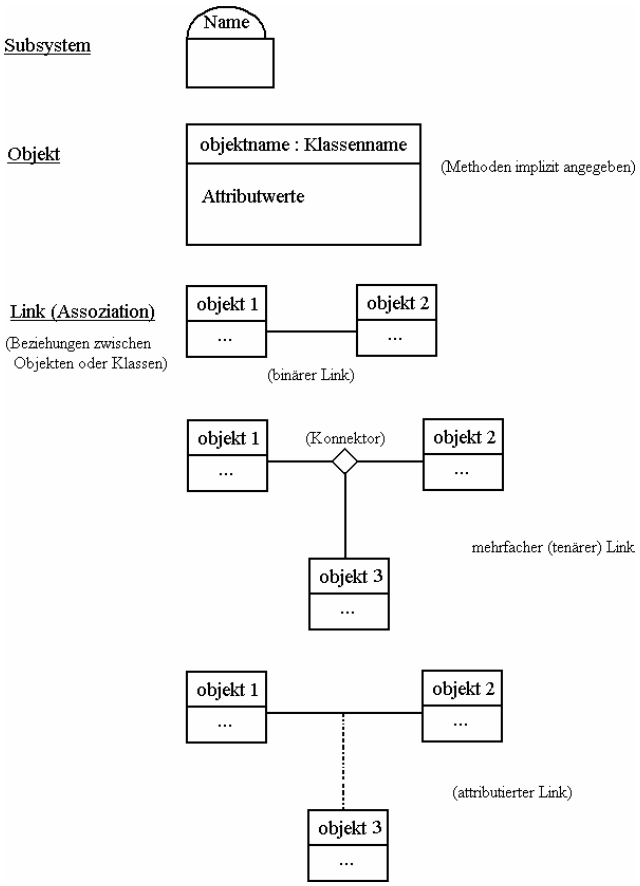


Linien = „wenn dann“ Beziehungen



UML

- Darstellung des OO-Modells
- Notationen :



Subsystem : z.B. Anwendungsdiagramme oder wieder Subsysteme
Objekte beginnen mit Kleinbuchstaben

- Rolle : Bedeutung des Objekts im Link (an den Link schreiben)

- Objektdiagramm :
  Alles Dargestellte = Objekte
  (Klassendiagramm , welches nur Objekte enthält)

- Objektmethode :

$$m(k_1\ n_1 : T_1, ..., k_m\ n_m : T_m) : r\{E\}$$

- $k_i \in \{in, out, inout\}$
- n_i Parameter
- T_i Typen
- r Rückgabetyp
- E Exceptions

-- 13.01.2006 --

OCL

- (weitere) Qualifizierung von Assoziationen , Eigenschaften der Objekte , der Klasse ...

- Jeder OCL-Ausdruck besitzt ein Bezugselement des UML-Diagramms (z.B. auf eine Klasse)

- Ein Bezugselement wird unterstrichen ode als Kontextangabe über den Ausdruck geschrieben

- context Bezugselement
  self : Referenz auf eine Instanz des Bezugselementes
  Die Bedingung bezieht sich auf alle Objekte dieser Klasse.

- Zugriff auf Objekte mittels Punktnotation :
  z.B. Eine Tagung dauert 1 bis 5 Tage :

```
context Tagung
self.Dauer>=1 Tag  and  self.Dauer<=5 Tage
```

  z.B. Anmeldedatum muss vor dem 10.02.2006 liegen :

```
context Tagung
if (self.Dauer.year > 2006) then false
  else (self.Datum.years = 2006) implies
    if (self.Datum.month > 2) then false
      else (self.Datum.month = 2) implies
        if (self.Datum.day > 10) then false
          else true
```

- Vergleiche , logische und arithmetische Operatoren stehen für die entsprechenden Typen zur Verfügung

- Bedingungen können als Invarianten geschrieben werden

- Klassifikation auch möglich für Vor- und Nachbedingungen einer Operation (Methode) oder Wächterbedingungen für einen Zustand

  z.B.

```
context Tagung
inv :: self.Dauer>=1   and   self.Dauer<=5

context Tagung : anmelden(ad.Datum)
pre : ad <= Anfangsdatum
```

- Zugriff auf verknüpfte Objekte entlang von Assoziationen möglich mittels :
  - Rollenname (des verknüpften Objekts)
  oder falls dieser fehlt :
  - mittels Name der Partnerklasse (aber jetzt mit Kleinbuchstaben beginnend)

  (an Exemplar der Quellklasse mittels Punktnotation angefügt)

  z.B. Bezeichnung für die Tagung eines Organisators

```
context Organisator
self.tagung
```

  z.B. Menge der Teilnehmer einer Tagung

```
context Tagung
self.teilnehmer
```

- Zugriffspfade beliebig lang
  z.B.

```
context Organisator
self.tagung.teilnehmer
```

- vordefinierte Funktionen für Kollektionen (Sammlungen)
  z.B. Anzahl der Elemente einer Kollektion

```
context Tagung
self.teilnehmer -> size >= 10
```

- Attribute und Assoziationen werden vererbt

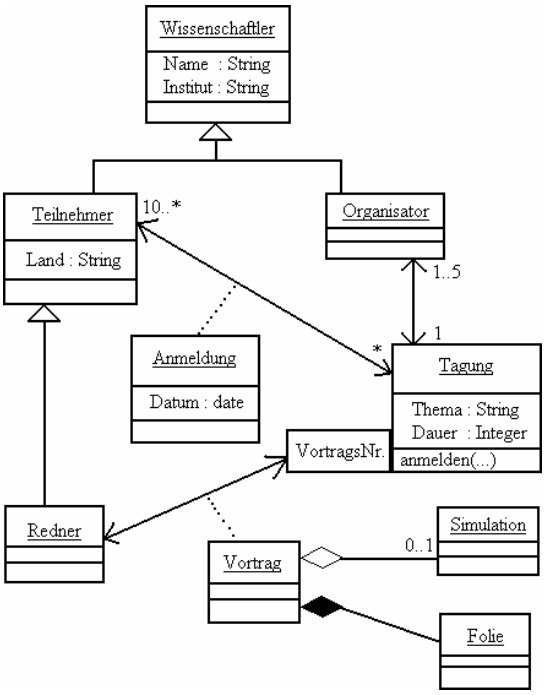
- Zugriff auch auf Assoziationsklassen möglich
  z.B. Ratschlag , höchstens 20 Folien für einen Vortrag zu verwenden
  context Redner
  self.vortrag.folie -> size <= 20

- Assoziationsklasse kann ihre Enden referenzieren
  z.B. Ein Redner darf nicht dem Institut der Organisatoren angehören
  context Teilnehmer
  not self.tagung.organisator.institut -> includes (self.redner.institut)

- Objektkollektorwerte (qualifizierende Attribute) werden in […] hinter das ausgewählte Objekt geschrieben
  z.B. Der erste Vortrag wird von einem Organisator gehalten.
  context Tagung
  self.organisator -> includes (self.redner [1])

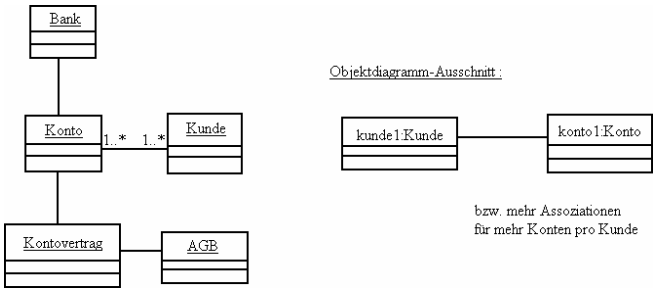
- Funktionen auf Kollektionen : collect / select
  z.B. context Tagung
  self.teilnehmer -> select (p:Teilnehmer , p.Land="Belgien")

- Quantoren-ähnliche Operationen : forAll / exists
  z.B. Alle Belgier brauchen keine Gebühr zu bezahlen.
  context Tagung
  self.teilnehmer->select (p.Teilnehmer | p.Land="Belgien").anmeldung -> forAll (a:Anmeldung | a.Gebuehr=0)



-- 20.01.2006 --

Klassendiagramm (Bank/Kunde)

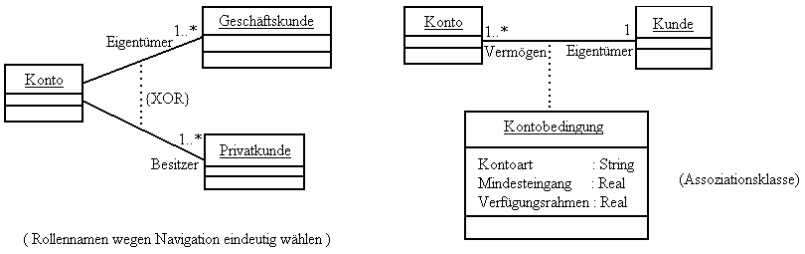


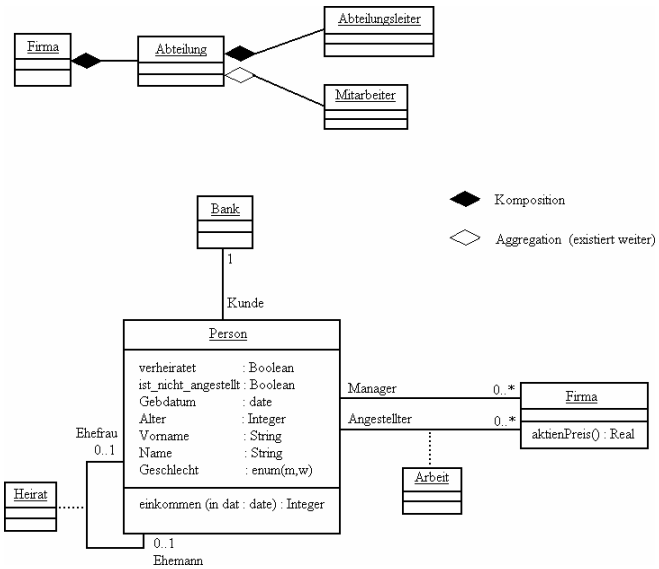
Die Leserichtung der Assoziationen wird auch mit einem ausgefüllten Pfeil und zugehörigem Verb angezeigt.

Zusicherungen :

in UML : { Konto.Kontovertrag = Kunde.Konto. .... }

in OCL : natürlichsprachlich möglich





Zusicherung: in OCL : {person1.Ehefrau ≠ person1.Ehemann}

-- 27.01.2006 --

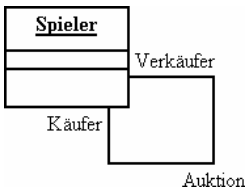
OCL
(Zusicherungen = Restriktionen)

- Alter einer Person soll nicht-negativ sein :
  context Person
  inv : self.Alter >= 0 (Invariante , immer erfüllt)

- Alter von verheirateten Personen soll mindestens 18 sein :
  context Person
  self.Ehefrau -> notEmpty implies self.Ehefrau.Alter >= 18 and
  self.Ehemann -> notEmpty implies self.Ehemann.Alter >= 18

- Eine Firma hat maximal 50 Angestellte :
  context Firma
  self.Angestellter -> size <= 50
  (self.Angestellter = Menge der Angestellten einer konkreten Firma)

- Eine Person ist entweder Ehefrau oder Ehemann (oder unverheiratet)
  context Person
  self.Ehefrau -> size = 0 or self.Ehemann -> size = 0
  (auch mit Empty / notEmpty möglich)



context s : Spieler
{ not (s.Käufer = s.Verkäufer) }

Bzw. wenn man zu "Spieler" ein Attribut „Name:String" hinzufügt :
{ not (s.Käufer.Name = s.Verkäufer.Name) }
pre : { „keine zwei gleichen Namen in der Klasse Spieler" }

- context Kredit
  self.Kreditnehmer -> includesAll (self.hypothek.sicherheit.besitzer)

  linke Seite :
  - konkreter Kredit
  - Menge der Personen , die den konkreten Kredit besitzen

  rechte Seite :
  - konkreter Kredit
  - Menge von Hypotheken , die zum konkreten Kredit gehören
  - Menge von Hypotheken , die zum konkreten Kredit gehören , die je 1 Haus als Sicherheit haben
  - Menge von Mengen , Besitzer der Häuser

Problem : Menge vs. Menge von Mengen z.B. {1,2,...} = { {1,2} , ... , {3,4} }
Lösung z.B. : Einfügen der Funktuion flatten nach besitzer : besitzer -> flatten()
Gegenstück : asSet() ?

$\Rightarrow$
Die Menge der Personen , die einen bestimmten Kredit haben , sind die Besitzer der Häuser , welche als Sicherheit der dem Kredit zugeordneten Hypotheken gelten.

- context Haus
  self.Wert >= self.hypothek -> collect(hoehe) -> sum()

  Der Wert eines Hauses ist größer gleich der Summe aller zugehörigen Hypotheken.

- context Kredit
  self.hoehe = self.hypothek -> collect(hoehe) -> sum()

  Die Kredithöhe entspricht dem Wert der zugehörigen Hypotheken.

Software-Architektur

- Entwurfsphase

- vorher : SA-Modell
  OOA-Modell (jeweils Lastenheft)

- Beschreibt die Struktur des SW-Systems durch
  - Systemkomponenten ,
  - ihre Beziehungen untereinander
  - und deren extern sichtbare Schnittstelle

- Art der Komponentenfestlegung :
  - Schichtenmodell
  - Komponentenmodell

Einflussfaktoren
- Produkteinsatz (d.h. vom konkreten Produkt)
- Randbedingungen (Qualitätssicherung , Sicherheit)
- Zielplattform

Client-Server-Architektur
logische Aufteilung

- Datenhaltung
- Anwendungslogik
- Präsentation (grafische Oberfläche)