

MATERIALSAMMLUNG - FORMALE METHODEN: OCL UND ECLIPSE

Prof. Dr. Hans-Jürgen Buhl

Sommersemester 2012
Fachgruppe Mathematik und Informatik
FB C
Bergische Universität Wuppertal

Praktische Informatik
PIBUW - SS12
April 2012
6. Auflage, 2012
Praktische Informatik 02

Inhaltsverzeichnis

1	UML und SdV	39
1.1	Rekapitulation: UML-Klassendiagramme	39
1.1.1	Klassen und Objekte	39
1.1.2	Klassenspezifikation	40
1.1.3	Links und Assoziationen	41
1.1.4	Rollen und Assoziationsnamen	42
1.1.5	Multiplizitäten (Kardinalitäten)	43
1.1.6	Stereotypen	43
1.1.7	Tagged Values	44
1.1.8	Generalisierung, Spezialisierung und Vererbung	45
1.1.9	Abstrakte Klassen	45
1.1.10	Komposition / Aggregation	46
1.1.11	Qualifizierte Assoziationen/Qualified Associations	48
1.1.12	Assoziationsklassen	49
1.1.13	template classes	50
1.1.14	UML 2.0	50
1.1.15	Modell und Metamodell	50
1.2	Spezifikation einfacher Klassen nach Prinzipien der SdV	51
1.2.1	Ein einfaches Beispiel ...	51
1.2.2	Vor- und Nachbedingungen in OCL	52
1.2.3	Spezifikation durch Verträge	53
1.2.3.1	Methodenklassifikation in C++	54
1.2.3.2	Vertragspflichten/Vertragsnutzen	55
1.2.3.3	Beispiel-Codeverträge	56
1.2.3.4	Subcontracting/Untervertragswesen	60
1.2.3.5	Beispiel zum Subcontracting	61
1.2.3.6	Funktion invert (Invertieren einer Matrix)	64
1.2.3.7	Interface myDictionary::Put	64
1.2.3.8	Interface LoeseLGS	65
1.2.3.9	Interface Bruecke	66
1.2.3.10	Zusammenfaßung der SdV-Prinzipien	67
1.2.4	... und sein (OCL2-)Vertrag	68
1.3	Ein Beispiel aus dem industriellen Einsatz: Die Klasse java.awt.Color	69
1.3.1	Klassenspezifikation: java.awt.Color	69
1.3.2	Hinweise	71

2	OCL-Spezifikation von Klasseninterdependenzen	73
2.1	Abhängigkeiten der assoziierten Exemplare	73
2.2	size(), includes() und forAll() – Methoden-Verträge	76
2.3	Der Ergebnistyp von (Mehrfach-)Navigationen; collect() von Klassenfeatures	78
2.4	Assoziationsklassen	81
2.5	Qualifizierte Assoziationen	84
2.6	Andere Methoden für die Collections Bag/Set	87
2.7	Schleifen und Iteratoren	88
2.8	Andere Collections	89
2.9	Together und automatische Code-Erzeugung	90
2.10	Fallstudie: Person/Haus/Hypothek	93
2.11	Einige erste Hilfskomponenten	96
2.12	Alle Instanzen einer Klasse: allInstances()	100
2.13	Fallstudie Personenstandsdaten, Hilfsklassen: Adresse, BioDaten, Datum, Personenstand	
2.14	Fallstudie Modell Wohnanlage	112
2.15	Fortsetzung Fallstudie Person/Haus/Hypothek	116
2.16	Startwerte von Attributen und Ergebnisse von Observatoren	117
2.17	Virtuelle OCL Variablen / Operationen/OclHelper	117
2.18	Enumeration	117
2.19	Tuple (structures)	117
2.20	Typ-Konformität	119
2.21	Operator-Vorrangsregeln	120
2.22	oclIsUndefined()	120
2.23	Vordefinierte Operationen auf OclAny	121
2.24	Statusdiagramme in UML	121
2.25	Grundlegende Observatoren bei Existenz von Assoziationen	122
2.26	Modell Student/Universitaet/Pruefungsergebnisvermerk	122
2.27	pre-Zustand in Nachbedingungen	125
2.28	Contracts zum Modell Student/Universitaet/Pruefungsergebnisvermerk	126
2.29	UML Constraints in UML 1.x	127
2.29.1	or / xor	127
2.29.2	subset	128
2.30	OCL primitive type Real	129
2.31	OCLs dreiwertige Logik	132
2.32	Stil-Hinweise für OCL-Constraints	133
2.33	Einfache Beispielverträge und die geeignete Kontextwahl	134
2.34	OCL String als ADT	135
2.35	OCL in Together-Tools	136
2.36	OMGs C++ Language-Mapping für CORBA	137
2.37	Was ist ein UML-Modell: MOF	137
2.38	Metalevel2-Constraints = Wohldefinierte Regeln für Modelle	139
2.39	OCL-Beispiele	142
2.40	OCL-Fallstudie „Royal and Loyal Model“ (nach Jos Warmer)	143

Abbildungsverzeichnis

Tabellenverzeichnis

1.1	Verpflichtungen/Vorteile von Verträgen zwischen Komponentenanbieter und -benutzer	53
1.2	Pflichten - Nutzen von Kunden und Lieferanten	56
2.1	logische Operationen in OCL	77
2.2	Methoden für die Collection Set	88
2.3	Schleifen und Iteratoren	88
2.4	Collection Operationen mit verschiedenen Bedeutungen	89
	count.ocl	15
	subsequence.ocl	15
	DM_Euro.cc	27
1.1	Konstruktor-Methoden:	69
1.2	Query-Methoden	70
	FlugFlugzeugPerson.ocl	73
	listen/flug1.txt	74
	listen/flug3.txt	74
	listen/flug2.txt	74
	JobNeu.ocl	81
	JobNeu2.ocl	82
	JobNeu3.ocl	82
	qualWork.ocl	84
2.1	Klasse Bank mit qualifizierter Assziation	90
2.2	Enumeration Gender	91
2.3	Klasse Person	91
2.4	OCL-Constraints Datum	98
2.5	OCL-Constraints Person	98
2.6	Constraints Adresse	103
2.7	Constraints Biodaten	103
2.8	Constraints Datum	104
2.9	Constraints Person	106
2.10	Constraints Hochzeit	111
2.11	Constraints Haus()	112
	addEtage.ocl	113
	addEtage2.ocl	113
	addEtage2a.ocl	113
	addEtage3.ocl	113

Invs.oc1	113
Invs2.oc1	113
Invs3.oc1	113
Invs4.oc1	114
Invs5.oc1	114
SysInvs.oc1	114
SysInvs2.oc1	114
SysInvs3.oc1	114
SysInvs4.oc1	114
SysInvs5.oc1	114
Destr.oc1	114
Destr2.oc1	115
Destr3.oc1	115
Universitaet.oc1	126
Student.oc1	126
Immatrikulation.oc1	126
Pruefungsergebnisvermerk.oc1	127
GeschRgl.oc1	139
WFR.oc1	140
WFR2.oc1	141
TreueProgramm.oc1	144

Formale Methoden

4 V Di 12-14 HS 08

Do 12-14 HS 08

Einordnung: Master IT; Master Mathematik; Nebenfächer und Studienschwerpunkte Informatik anderer Studiengänge

Inhalt: Softwarequalität, Zusicherungen, Klassifizierung von Klassenmethoden; Programming by Contract; Vorbedingungen, Nachbedingungen und Invarianten; Contracts bei der Vererbung; formale Spezifikation mit OCL2 und Eclipse; Frame-Regeln; Fallstudien formaler Spezifikation.

Übungen zu Formale Methoden

2 Ü Do 16-18 D.13.11

Vorbemerkungen:

Formale Methoden

Inhalte:

1. Softwaregüte
2. Zusicherungen in Algorithmen:
Konstruktoren, Modifikatoren,
Observatoren und Destruktoren;
Ausnahmebedingungen
3. Methodik *Programming by Contract*:
Vorbedingungen, Nachbedingungen und Invarianten;
Softwareanbieter/Softwarenutzer
4. Startwerte, Vererbung von Klasseninvarianten,
Methodenvor- und -nachbedingungen
5. Formale Spezifikation (in OCL2):
UML-Klassendiagramme und *Constraints*
virtuelle Attribute und Methoden,
redundante Attribute und Methoden;
Constraints an Attribute, Methoden und
Assoziationen; Container-Typen; Frame-Regeln
6. Fallstudien von formal spezifizierter
Software (Algorithmen und Datenstrukturen)
7. Von der formalen Spezifikation zur (Prototyp-)Software

Modulziele:

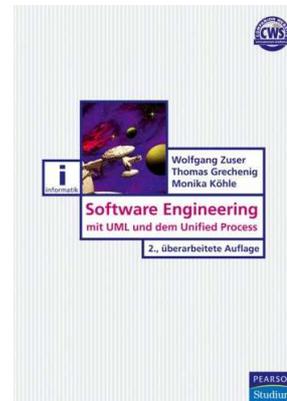
Die Studierenden lernen formale Software-Modelle lesen, verstehen und kritisieren, um formale Methoden als ein Kommunikationsmittel der Teammitglieder eines Software-Entwicklungsteams schätzen zu lernen. Sie entwickeln mit Hilfe der formalen Spezifikation Teilsysteme von realistischen Softwaremodellen selbst.

Literatur:

Wolfgang Zuser

Software Engineering
Mit UML und dem Unified Process
Gebundene Ausgabe - 464 Seiten
Pearson Studium
Erscheinungsdatum: Juni 2004

Auflage: 2., überarb. Aufl.
ISBN: 3827370906



Bernd Oestereich

Software Engineering Analyse und Design mit
UML 2.3
Objektorientierte Softwareentwicklung
Olderbourg Verlag München
Auflage: 9., aktualisierte und erweiterte Aufl.
ISBN: 978-3-486-58855-2



Harald Störrle

UML 2 für Studenten
Pearson Studium München
2005

Dan Pilone

UML 2.0 in a nutshell
O'Reilly
2005

Dan Pilone

UML 2.0 kurz und gut
O'Reilly
2. Auflage, 2006

OMG

UML Infrastructure

OMG Available Specification

Versionb 2.4.1

<http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF>

OMG

UML Superstructure

OMG Available Specification

Versionb 2.4.1

<http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>

OMG

Object Constraint Language

OMG Available Specification

Versionb 2.3.1

<http://www.omg.org/spec/OCL/2.3.1/PDF>

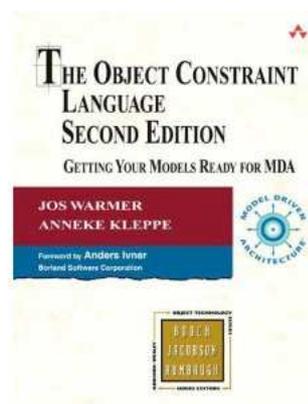
Jos Warmer

The Object Constraint Language Second Edition

Addison-Wesley

Erscheinungsdatum: 2003

ISBN: 0321179366



Tony Clark, Jos Warmer

Object Modeling with the OCL.

The Rationale behind the Object Constraint Language

<http://www.amazon.de/Object-Modeling-OCL-Rationale-Constraint/dp/3540431691>

ISBN: 3-540-43169-1

Scott W. Ambler

The Elements of UML 2.0 Style.

Cambridge University Press

2005

ISBN: 978-0-521-61678-2

Nimal Nisanke

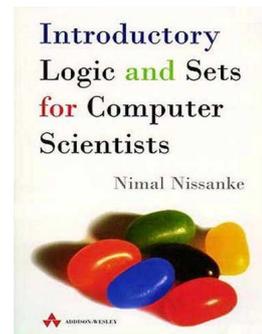
Introductory Logic and Sets for Computer Scientists.

Broschiert - 400 Seiten

Addison Wesley

Erscheinungsdatum: Oktober 1998

ISBN: 0201179571



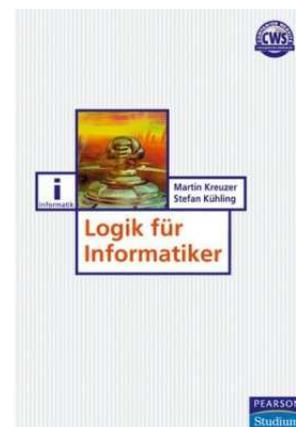
Martin Kreuzer, Stefan Kühling

Logik für Informatiker

Pearson Studium

Erscheinungsdatum: März 2006

ISBN: 3827372151



FOLDOC - Free-On-Line-Dictionary-Of-Computing

<http://wombat.doc.ic.ac.uk/foldoc/>



Search Home Contents Feedback Random

Enter a word or phrase in the box at the top of any page and click the **Search** button or hit Enter. You can try [other FOLDOC servers](#) if this one is slow for you. Please contact me before creating any kind of mirror of the dictionary.

[More help](#) - [Firefox extension for FOLDOC](#) - [Recent Updates](#)

Supported by [Imperial College Department of Computing](#)
Copyright © 1993 by Denis Howe. All Rights Reserved

<http://foldoc.org/>

14175 terms, 5126252 bytes
Last modified: 2006-01-20 02:30

Eine Suche bei FOLDOC zu **formal methods** und **specification** ergibt folgendes:

Formale Methoden / formal methods

<Mathematik Spezifikation> Mathematisch basierte Technik zur Spezifikation, Entwicklung und Verifikation von Software und Hardware Systemen.

Spezifikation / specification

Ein Dokument welches beschreibt, was ein System tun soll (nicht wie es das erledigen soll)!

Manchmal ist dazu auch ein exemplarisch zitierter Algorithmus sinnvoll.

Formale Methoden:

- Formale Semantik
- **Formale Spezifikation**
- Formale Verifikation
- Theorembeweisen
- Modellprüfverfahren (model checking)

Formale Spezifikationsprachen benutzen anfangs mathematische Formelschreibweise (Latex-Stil):

```

Queue = Qelem*
q0 = [ ]

ENQUEUE (e : Qelem)
ext wr    q : Queue
post     q =  $\overleftarrow{q} \frown [e]$ 

DEQUEUE() e : Qelem
ext wr    q : Queue
pre      q  $\neq$  [ ]
post      $\overleftarrow{q} = [e] \frown q$ 

ISEMPTY() r :  $\mathbb{B}$ 
ext rd    q : Queue
post     r  $\Leftrightarrow$  (len q = 0)

```

VDM, VDM++

Heute geht man aber immer mehr zu reiner ASCII (oder Unicode)-Syntax über:

```

public AddExpertToSchedule: Period * Expert ==> ()
AddExpertToSchedule(p,ex) ==
  schedule(p) := if p in set dom schedule
                 then schedule(p) union {ex}
                 else {ex};

```

and the RemoveExpertFromSchedule operation can be expressed as:

```

public RemoveExpertFromSchedule: Period * Expert ==> ()
RemoveExpertFromSchedule(p,ex) ==
  let exs = schedule(p) in
    schedule := if card exs = 1
                 then {p} <-: schedule
                 else schedule ++ {p |-> exs \ {ex}}

```

(Seite 15)

Overture

Benutzte freie UML2-/OCL-Tools

Hilfsmittel (Tools) zur formalen Spezifikation von OOP-Modellen mit Hilfe von OCL2:

Papyrus:

<http://www.papyrusuml.org>

oder in Kürze: <http://wiki.eclipse.org/MDT/Papyrus>

(Verfügbar (vorinstalliert) auf dem PI/IT-Ausbildungsclustern!)

Hinweis zur Installation auf dem eigenen Notebook:

Eclipse Modeling aktuell: Indigo

Help

Install Modeling Components
OCL-Tools

Papyrus

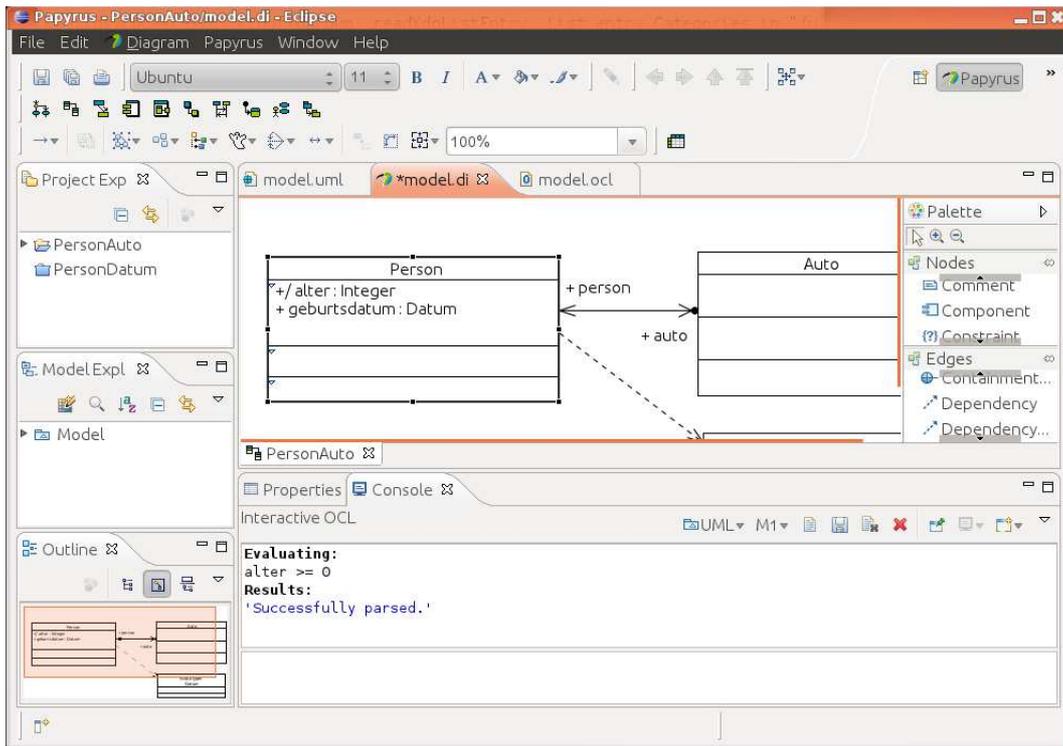
Help

Install new software
<http://download.eclipse.org/tools/cdt/releases/indigo> (p2 repository)

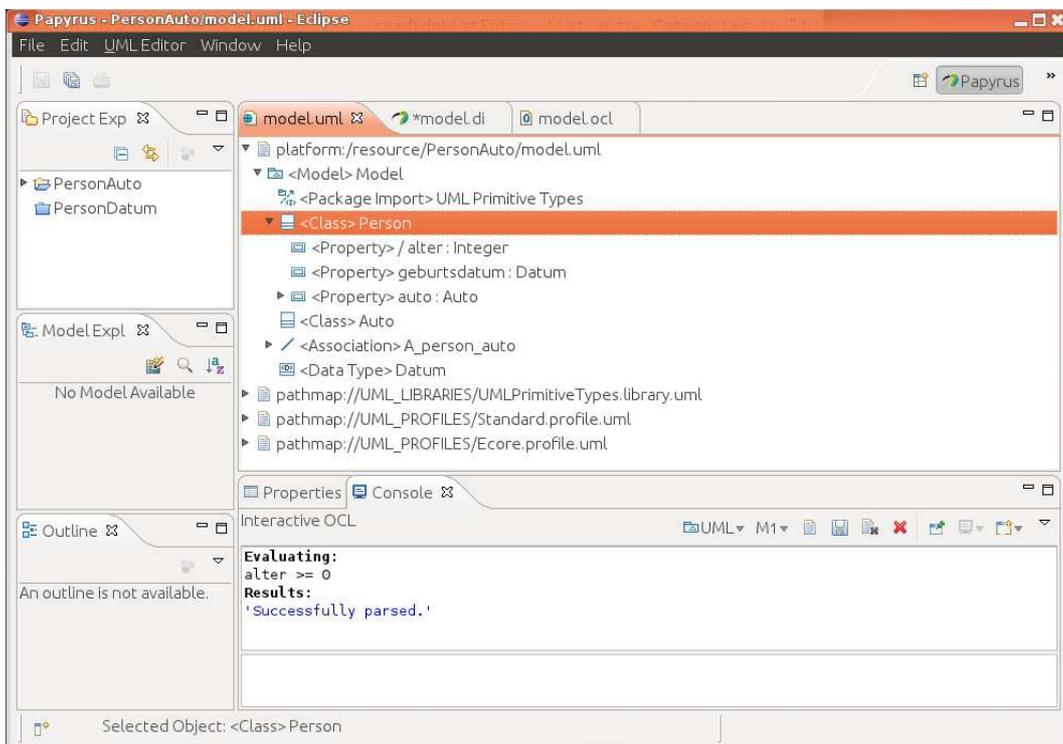
Help

Eclipse Marketplace
OCL
Dresen OCL
Install

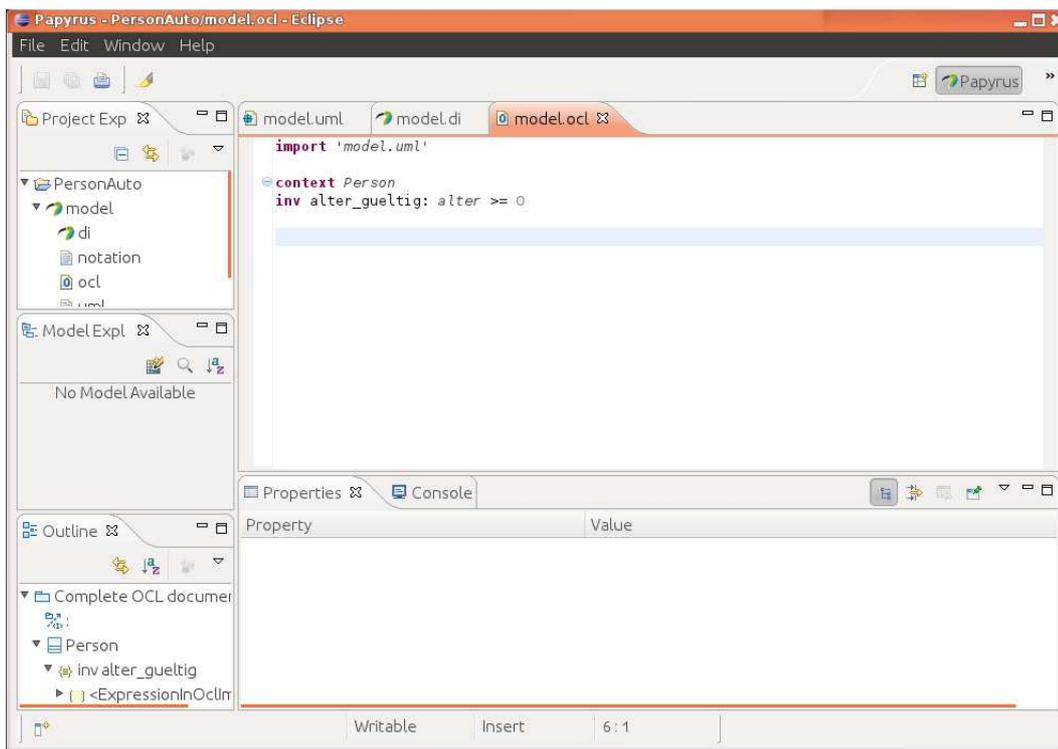
Erste Gehversuche: UML (mit Papyrus)



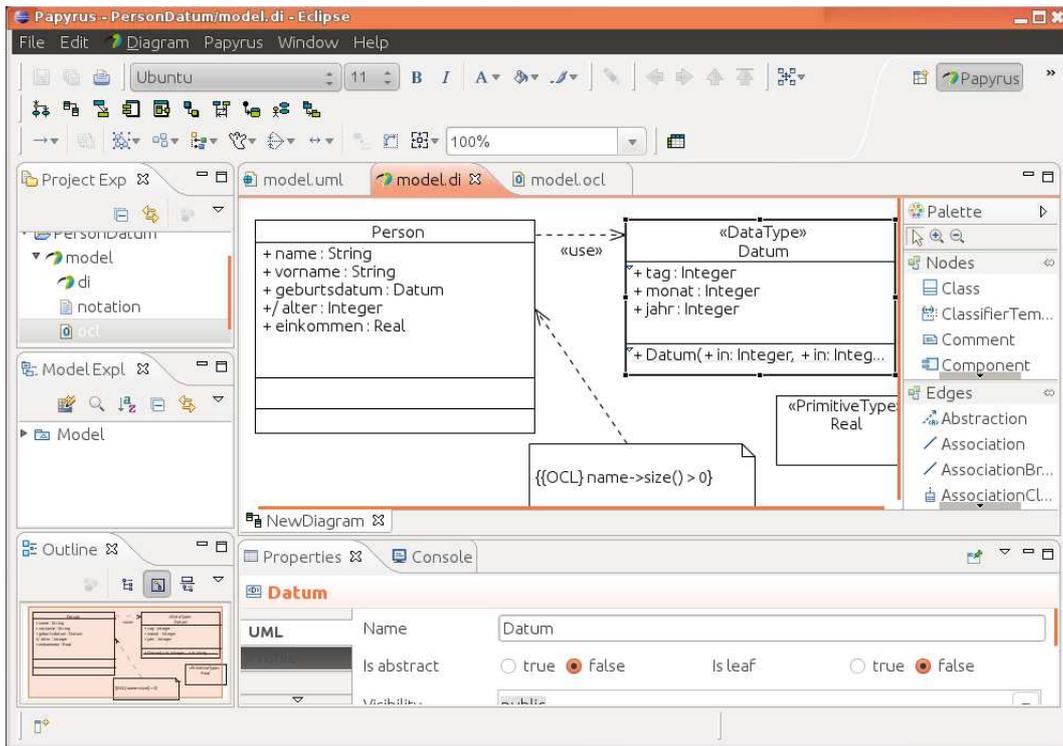
und OCL mit den OCL-Tools von Papyrus:



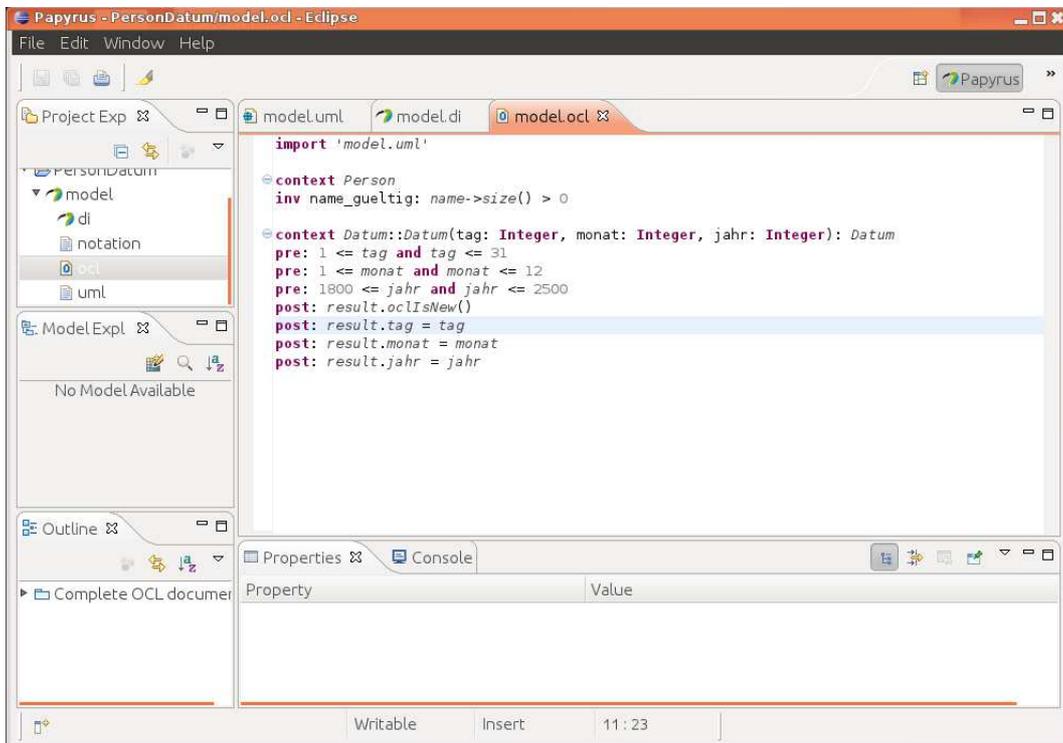
Besser mit allen Constraints in einer model.ocl-Datei:



Datum mit einem Konstruktor (static, isquery):



und vollständiger OCL-Spezifikation;



Einleitende Bemerkungen

Softwareprobleme:

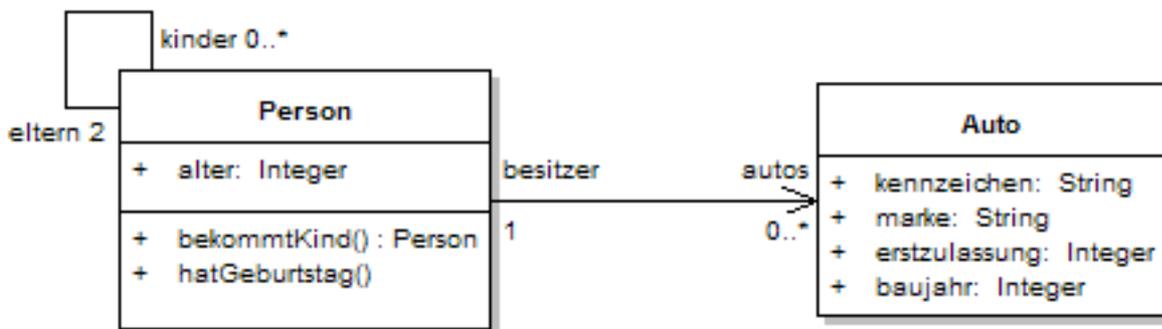
TomTom-Navigator und Schaltjahre
<http://catless.ncl.ac.uk/Risks/26.77.htmlsubj11>

Visa network down

...

Die „Object Constraint Language“

http://de.wikipedia.org/wiki/Object_Constraint_Language
und einige erste „Constraints“ zur Spezifikation fehlerfreier Software



natürlichsprachig (und dadurch evtl. mehrdeutig):

Das Alter einer Person ist nicht negativ.
Eine Person ist jünger als ihre Eltern.
Nach einem Geburtstag ist eine Person um genau 1 Jahr älter.
Eine Person hat höchstens 2 Eltern.
...

und formal:

```
context Person inv: self.alter >=0
context Person inv: self.eltern->forall(e|e.alter > ...)
context Person::hatGeburtstag() post: self.alter=self.alter@pre+1
context Person inv: self.eltern->size()<=2
```

...

OMG-Dokumentation von OCL2: <http://www.omg.org/spec/OCL/2.3.1/PDF>

- **Formale Methoden**

sind logikbasierte Techniken für die
Spezifikation,
Entwicklung und
Verifikation
von SW- und Hardwaresystemen.

- Die **Spezifikation** eines Systems ist ein Dokument, das beschreibt, was ein System tun soll (nicht wie es das tun soll).

- **Beispiele für entsprechende Beschreibungen:**

- a) Eine Funktion kann **implizit** (durch Angabe von Eigenschaften) spezifiziert werden:

$\begin{aligned} &max(s : \mathbb{N}_1\text{-set})m : \mathbb{N}_1 \\ &pre\ card\ s \neq 0 \\ &post\ m \in s \wedge \forall x \in s. m \geq x \end{aligned}$
--

- b) Eine Funktion kann **explizit** (durch Angabe eines Algorithmus) spezifiziert werden:

$\begin{aligned} &min(r : Real) : Real \\ &post: \text{if self} \leq r \text{ then result} = \text{self else result} = r \text{ endif} \end{aligned}$

- **Explizite Spezifikationen** sind immer im Sinne *einer* exemplarischen Beschreibung aufzufassen (denotationell). Alle Implementierungen des Softwaresystems, die zu dieser Spezifikation äquivalente Ergebnisse liefern sind zulässig.
- Ideal wären eigentlich immer implizite Spezifikationen (warum?), jedoch sind explizite (formale) Spezifikationen besser als gar keine oder nur umgangssprachliche Spezifikationen, da man hier nachlesen kann, was *genau* der Zweck einer Methode ist, zum Beispiel (im OCL-Handbuch):

Spezifikation `Collection(T)::count()`

```
context Collection(T)::count(object : T) : Integer
  post: result = self->iterate( elem; acc : Integer = 0 |
    if elem = object then acc + 1 else acc endif)
  ...
```

(Vergleiche Seite 157 der OCL-Spezifikation.)

Weitere Spezifikationshilfsmittel sind Verträge (bestehend aus Vor- und Nachbedingungen):

- **Vor-/Nachbedingungen**

```
context Sequence(T)::subSequence(lower : Integer ,
                                upper : Integer) : Sequence(T)
  pre : 1 <= lower
  pre : lower <= upper
  pre : upper <= self->size()
  post: result->size() = upper - lower + 1
  post: Sequence{lower .. upper}->forAll( index |
    result->at(index - lower + 1) = self->at(index))
  ...
```

(Vergleiche Seite 166 der OCL-Spezifikation.)

Sie erlauben die eindeutige Verantwortlichkeitszuordnung: Im Fehlerfall Vorbedingung verletzt (Aufrufender verantwortlich), Nachbedingung bei eingehaltener Vorbedingung verletzt (Software-Produzent verantwortlich).

„Der zusätzliche Aufwand, etwas formal zu beschreiben, muß eine Rechtfertigung haben. Nur zu formalisieren, um eine formale Spezifikation zu erhalten, ist keine Rechtfertigung. Eine formale Spezifikation ist auch nicht um jeden Preis und für alle Teile eines Systems sinnvoll. Nichtsicherheitskritische Teile müssen nicht unbedingt formal beschrieben werden.“
(Sergio Montenegro: *Formale Methoden in der Softwareentwicklung Heute und Morgen*)

- **Beispiele zu Softwareproblemen/Spezifikationsmängeln:**

- Euro-Panne bei der Deutschen Bank 24 (Update)** 4.1.2002

Geldautomaten der Deutschen Bank 24 müssen sich wohl an den Euro erst noch gewöhnen. Wer Anfang Januar Euro-Beträge von Geldautomaten dieser Bank bezogen hat, durfte sich am heutigen Freitag wundern, dass ihm die Bank das 1,95-fache vom Konto abgebucht hat. Offensichtlich haben die Bank-Computer an Stelle der maßgeblichen Euro-Summe irrtümlich mit dem Zahlenwert des umgerechneten DM-Betrags gerechnet.

Verunsicherte Kunden erfuhren zunächst nur, dass sogar die Angestellten der Bank dem Problem zum Opfer gefallen sind. Mit der Hoffnung auf hilfreichere Informationen mussten sie sich jedoch vorerst gedulden. Erst gegen elf Uhr konnten die Ansprechpartner an der Telefonhotline für etwas Beruhigung sorgen: „Das Problem ist bekannt, die falschen Buchungen werden automatisch zurückgezogen und korrigiert“.

Inzwischen fand die Bank heraus, dass bei einem nächtlichen Datenverarbeitungs-lauf einige Tausend der insgesamt etwa 1,5 Millionen angefallenen Kontobewegungen durch einen Programmfehler falsch bearbeitet worden sind. Theoretisch hätten zwar auch herkömmliche Barabhebungen am Bankschalter betroffen sein können, doch zufällig drehte es sich bei den fehlerhaften Buchungen tatsächlich nur um Abhebungen von Geldautomaten, hieß es bei der Deutschen Bank 24. Das erklärt auch, warum bei anderen Banken, die gebührenfreies Abheben von denselben Geldautomaten wie die Deutsche Bank 24 ermöglichen, keine vergleichbaren Fehler aufgetreten sind.

Markus Block, Sprecher der Deutschen Bank 24, erklärte gegenüber heise online, alle falschen Buchungen würden bis zum Samstag korrigiert sein, sodass kein Kunde finanzielle Nachteile zu erwarten habe. (hps/c't)

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/23747>

Computer-Panne ließ die Telefone abstürzen

<http://www.wz-newsline.de/index.php?redid=181930> 30.10.2007

Düsseldorf. Am Tag nach dem teilweisen Zusammenbruch des Telefonnetzes war bei der Telekom in Düsseldorf Ursachenforschung angesagt. Wie sich herausstellte, war das Aufspielen einer neuen Softwareversion auf einen Vermittlungscomputer die Ursache der Störung.

In der Landeshauptstadt – ausgerechnet noch im Telekomgebäude an der Nobelmeile Königsallee – steht der besagte Server. Betroffen waren in erster Linie die Telefonate von Konkurrenzanbietern wie Arcor, die die Gespräche aus ihren lokalen Netzen über den Düsseldorfer Server ins bundesweite Telekomnetz einleiten.

Durch den Zusammenbruch des Vermittlungscomputers mussten die Gespräche über Server in Hamburg und Stuttgart umgeleitet werden. Dadurch wurden die Netze überlastet – auch Gespräche im Telekomnetz kamen dann nicht mehr zustande oder wurden falsch vermittelt. ...

Den Fehler zu finden, war nicht ganz einfach. „Er war zunächst nicht regional einzugrenzen“, sagt Wendtland. Wie sich dann herausstellte, war der Düsseldorfer Server schuld am Desaster. Man hatte gestern eine neue Software-Version auf diesen Rechner aufgespielt, die fehlerhaft sein muss. „Wir haben dann den Rechner komplett neu aufgesetzt“, sagt der Telekom-Sprecher. Das heißt: Die Software wurde komplett gelöscht und die ältere, stabile Version wieder installiert.

Gegen 20 Uhr war die Störung so wieder beseitigt.

Jetzt wird mit dem Hersteller der Software nach dem genauen Fehler gesucht. Aber auch die Stromversorgung des Servers wird überprüft. Spannungsschwankungen könnten den Ausfall auch verursacht haben.

<http://catless.ncl.ac.uk/Risks>

<http://catless.ncl.ac.uk/Risks/24.88.html#subj3.1>

Neuaufgabe desselben Szenarios: 30.09.2009

Computerprobleme legen Check-in-System der Lufthansa lahm

<http://www.heise.de/newsticker/meldung/Computerprobleme-legen-Check-in-System-der-Lufthansa-lahm-798193.html>

Computerprobleme haben an diesem Morgen bei der Fluggesellschaft Lufthansa dazu geführt, dass Passagiere zeitweise kein Gepäck aufgeben und nicht einchecken konnten. Auslöser des Problems ist nach Angaben eines Lufthansa-Sprechers ein Update des zentralen Check-in-Systems in Kelsterbach bei Frankfurt am Main. Nach dem Update seien die Server nicht wie gewünscht hochgefahren. Das führte dazu, dass Passagiere wie früher üblich händisch mit Bordkarten einchecken mussten.

Die Probleme setzten heute Morgen um 3.46 Uhr ein, nachdem das Update vorgenommen worden war. Die Server in Kelsterbach sind für die weltweite Abwicklung von Check-ins zuständig. Die Folge waren Flugverspätungen und -streichungen. Interkontinentalflüge von Deutschland aus seien nicht ausgefallen, erklärte der Lufthansa-Sprecher. Mittlerweile sei das Check-in-System wieder hochgefahren worden. Allerdings würden noch nicht alle Applikationen laufen, daher gebe es noch Probleme.

Die Fluggesellschaft ist noch dabei, die genaue Ursache der Probleme zu klären. Sie hofft, diese im Laufe des Vormittags in den Griff zu kriegen. Die Lufthansa bittet ihre Kunden, sich auf der Website über ihren gebuchten Flug zu informieren. Alternativ können sie im Lufthansa-Callcenter ...

Und wiederum: 21.04.2009

Netzausfall legt Millionen Handys lahm

http://www.rp-online.de/wirtschaft/news/unternehmen/T-Mobile-Chef-entschuldigt-sich_aid_699292.html

T-Mobile-Chef entschuldigt sich

(RP) Alle T-Mobile Kunden können wieder telefonieren. Wie die Telekom mitteilte, ist die bundesweite Störung im Handy-Netz behoben. T-Mobile-Chef Georg Pölzl entschuldigte sich bei allen Kunden. Am Dienstag konnten Millionen von T-Mobile-Nutzern wegen eines Computerproblems stundenlang nicht telefonieren. Die Panne löste bei vielen Verärgerung aus. ...

Grund für den Ausfall sei ein Software-Fehler bei einem Server, dem Home Location Register, gewesen. Die betroffene Technik sorgt dafür, dass eine Verbindung zwischen Mobilfunkstation und der zugehörigen Rufnummer hergestellt wird. Dort werden die Telefonnummern den einzelnen SIM-Karten zugeordnet.

Ein Sprecher des Unternehmens verglich die Funktion des Servers zuvor mit der eines Pförtners. Ohne den sei es weder möglich in das T-Mobile-Netz hinein, oder hinauszutelefonieren. Wie es zu dem Serverausfall kommen konnte, ist noch unklar.

Probleme über Probleme:

http://de.wikipedia.org/wiki/Programmfehler#Folgen_von_Programmfehlern

- **1982 stürzte ein Prototyp des F117 Kampffjets ab**, da bei der Programmierung die Steuerung des Höhenruders mit der des Seitenruders vertauscht worden war.
- **Zwischen 1985 und 1987 gab es mehrere Unfälle** mit dem medizinischen Bestrahlungsgerät Therac-25. Infolge einer Überdosis, die durch fehlerhafte Programmierung und fehlende Sicherungsmaßnahmen verursacht wurde, mussten Organe entfernt werden, drei Patienten verstarben aufgrund der Überdosis.
- **Am 25. Februar 1991 verfehlte eine Patriot-Rakete** in Saudi-Arabien wegen eines Registerüberlaufs eine Scud-Rakete, und diese zerstörte daraufhin eine Armeebarracke, wobei es zu 28 Toten kam.
- **Am 12. März 1995 kam es wegen eines um wenige Byte zu klein bemessenen Stapelspeichers** in der Software eines Hamburger Stellwerks, bei dem auch das Ersatzsystem aus Sicherheitsgründen abgeschaltet wurde, zu massiven Verzögerungen im bundesweiten Zugverkehr.
- **Am 4. Juni 1996 wurde der Prototyp der Ariane-5-Rakete** der Europäischen Raumfahrtbehörde eine Minute nach dem Start in vier Kilometern Höhe gesprengt, weil der Programmcode, der von der Ariane 4 übernommen worden war und nur für einen von der Ariane 4 nicht überschreitbaren Bereich (Beschleunigungswert) funktionierte, die Steuersysteme zum Erliegen brachte, als eben dieser Bereich von der Ariane 5, die stärker als die Ariane 4 beschleunigt, überschritten wurde. Dabei war es zu einem Fehler bei einer Typumwandlung gekommen, dessen Auftreten durch die verwendete Programmiersprache Ada eigentlich hätte entdeckt und behandelt werden können. Diese Sicherheitsfunktionalität ließen die Verantwortlichen jedoch abschalten:
The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error.
Der Schaden betrug etwa 370 Millionen US-Dollar.
- **1999 verpasste die NASA-Sonde Mars Climate Orbiter** den Landeanflug auf den Mars, weil die Programmierer das falsche Maßsystem verwendeten - Pfund x Sekunde statt Newton x Sekunde. Die NASA verlor dadurch die Sonde.
- Zum Jahreswechsel 1999 / 2000 kam es in einigen wenigen Programmen zum Jahr-2000-Problem. Die meisten Fehler wurden jedoch schon vorher durch Patches behoben.

- **Bei Toll Collect kam es 2003** unter anderem wegen der fehlenden Kompatibilität von Softwaremodulen zu drastischen Verzögerungen mit Vertragsstrafen und Einnahmeausfällen in Milliardenhöhe.
- **Am 8. Oktober 2005 führte im russischen Plessezsk** ein Programmfehler zum Fehlstart einer Trägerrakete und zum Verlust des Satelliten CryoSat.
- **Anfang November 2005 konnte an der Tokioter Börse** wegen eines Programmfehlers stundenlang kein Handel betrieben werden. Auch in den nachfolgenden Wochen gab es viele fehlerhafte Wertpapierordern, die in einem Fall sogar einen finanziellen Schaden von über 300 Millionen Dollar ausmachte. Der Präsident der Börse, Takuo Tsurushima, trat daraufhin von seinem Amt zurück.
- **Im Oktober 2007 kamen zehn Angehörige** der südafrikanischen Armee aufgrund eines Programmfehlers in einem vollautomatisierten 35-mm-Flakgeschütz ums Leben.
- **04.07.2005. Begleitet von großem Werberummel** hat die NASA den Kometen Tempel1 beschossen. Nun zeigen die Daten: Getroffen hat sie gut, gelernt hat sie wenig. Ein Softwarefehler hat dazu geführt, dass die ersten - und besten - Bilder des Zusammenpralls im Datenspeicher des Begleitsatelliten von späteren Aufnahmen überschrieben wurden.
- **Chaos an Hannovers Geldautomaten.** Computerprobleme haben am Samstag alle 240 Geldautomaten der Sparkasse in der Stadt und Region Hannover lahm gelegt. Die Fusion der Stadt- und Kreissparkasse sollte am Wochenende auch technisch umgesetzt werden. Beim Hochfahren eines Server habe sich ein Fehler eingeschlichen, so dass die Geldautomaten nicht mehr funktionierten. Die Sparkasse öffnete stadtdessen fünf Filialen, damit Kunden etwa in Einkaufszonen Bargeld abheben können.
- **Berliner Magnetbahn.** Fünf - Null, tippt der Operator in die Tastatur und erwartet, daß die Magnetschwebbahn auf 50 Stundenkilometer beschleunigen würde. Doch nichts geschah. Wieder tippt er fünf - null und vergaß diesmal nicht die „Enter“-Taste zu betätigen, mit der die Daten erst in den Rechner abgeschickt werden. Die insgesamt eingegebene Tastenfolge „fünf - null - fünf - null“ interpretiert der Rechner als Anweisung, auf unsinnige 5050 Stundenkilometer zu beschleunigen. Dies konnte die Bahn zwar nicht, aber immerhin wurde sie so schnell, daß sie nicht mehr rechtzeitig vor der Station gebremst werden konnte. Es kam zum Crash mit Personenschaden – so geschehen vor zwei Jahren bei einer Probefahrt der Berliner M-Bahn. Vernünftigerweise hätte die den Computer steuernde Software die Fehlerhaftigkeit der Eingabe „5050“ erkennen müssen. ...
- **19.06.2004. DaimlerChrysler-Rückrufaktion** von 10.000 Mercedes-Benz-Modellen wegen fehlerhafter Kraftstoff-Abschaltung durch Softwarefehler der Dieselsteuergeräte.

- **Excel 2007 verrechnet sich beim Multiplizieren:** Von einer Tabellenkalkulation sollte man eigentlich erwarten können, dass sie das Einmaleins beherrscht. Doch darauf kann man sich in Excel 2007 nicht verlassen. Wie Blogger Brad Linder berichtet, verrechnet sich Microsofts aktuelle Excel-Version im Umgang mit reellen Zahlen: Sie liefert bei der Multiplikation von 850 mit 77,1 statt des korrekten Resultats 65.535 den runden, aber falschen Wert 100.000. Der Fehler betrifft auch andere Multiplikationen wie $10,2 * 6425$ oder $40,8 * 1606,25$, deren Ergebnis eigentlich 65.535 lauten sollte. Siehe: <http://blogs.msdn.com/excel/archive/2007/09/25/calculation-issue-update.aspx>

- **RISKS: 10. November 2009. Subject: Apostrophe in Your Name? You Can't Fly!**

This is the stuff of nightmares - not to mention enormous frustration and possible stomach ulcers. If you have an apostrophe in your name - like many of Irish descent do - you may find it impossible to board an airplane in the coming months. Why? Because airline computers can't print an apostrophe on the boarding pass, the name on your boarding pass will not exactly match the name on your driver's license or passport. And beginning next year, the two must match or you don't fly. And they call this progress.

- **November 1994: Pentium-FDIV-Bug.** Fehlerhaftes Microprogramm im Pentium führt zu falschen Divisionsergebnissen: „leichter Genauigkeitsverlust bei Gleitkomma-Divisionen mit bestimmten Operanden-Paaren“. Intel kündigte zunächst an, nur CPUs von Anwendern tauschen zu wollen, die darlegen konnten, dass sie von dem Fehler betroffen seien. Der Fehler werde bei einem Normalanwender statistisch nur alle 27000 Jahre einmal auftreten. Am 20. Dezember kündigt Intel ein umfassendes Austauschprogramm für alle betroffenen CPUs an.

Weitere interessante Fundstellen:

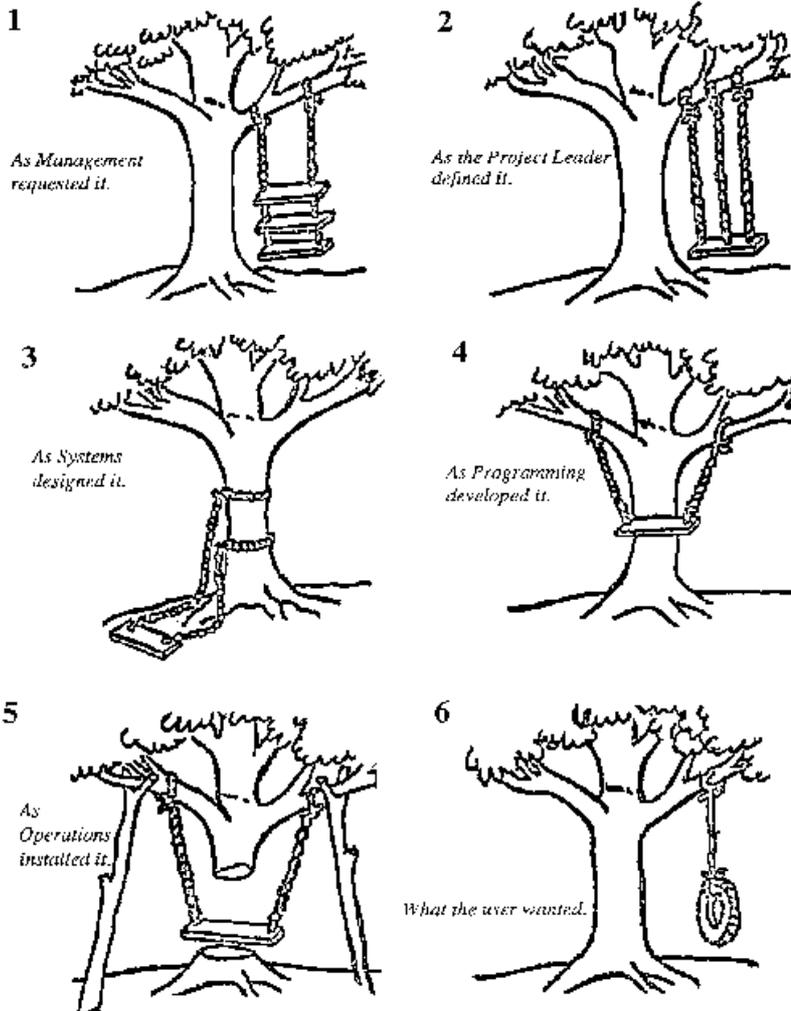
- [20 Famous Software Disasters](#)
- [Software bugs in the data reservoir](#)
- [The top 10 IT disasters of all time](#)
- [Kleine BUGs, große GAUs](#)
- [Top 25 Most Dangerous Programming Errors](#)

Ein kleines Kompendium zu Bugs: <http://de.wikipedia.org/wiki/Programmfehler>

Aktuelles

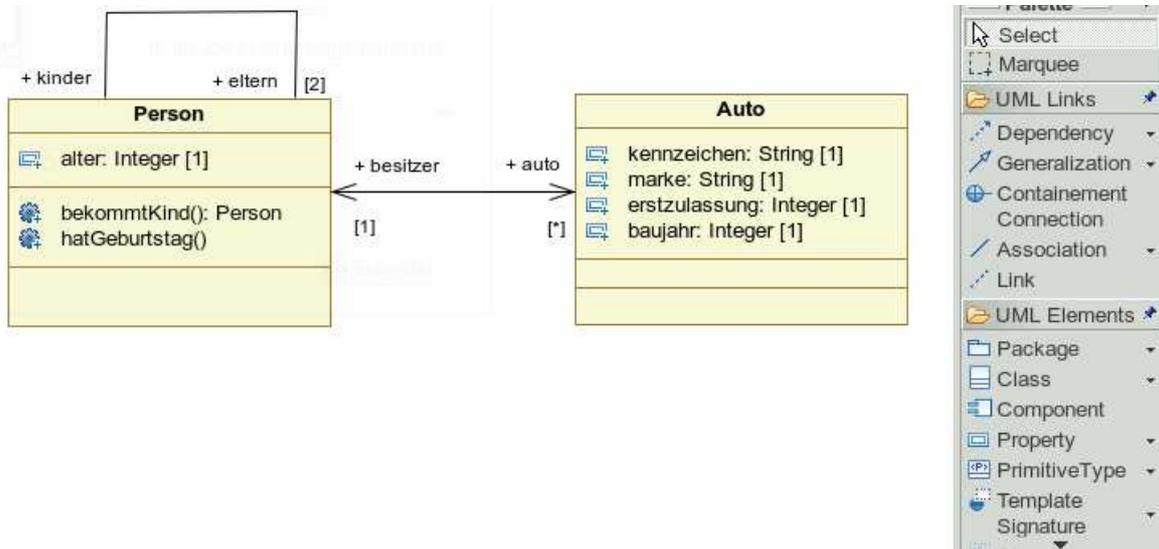
... im Spotlight (<http://catless.ncl.ac.uk/Risks>):

- Therac-25 läßt grüßen, siehe auch Überbestrahlung
- Toyota-Rückrufaktion, siehe auch „Toyota uncontrolled acceleration“
- London's stock exchange crashes again



- OCL-Beispiele (Wikipedia)

Eingabe der zugehörigen UML-Modelle in Papyrus:



und erste Constraints:

DefaultDiagram

Properties

PersonAuto::Person

General

Profile

Comments

Constraints

Appearance

Advanced

Constraints:

- AlterNichtNegativ -> <Class> Person
- AlterVonEltern -> <Class> Person

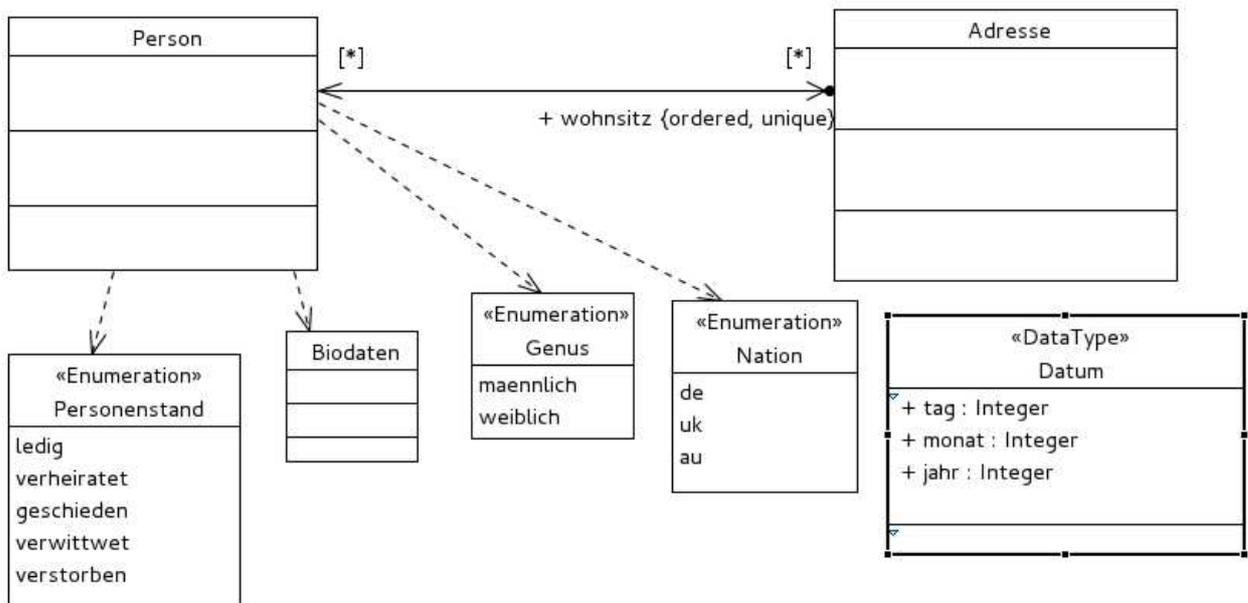
Level: M1 Language: OCL

eltern->forAll(e | e.alter >= alter + 8)

Successfully parsed. Evaluate

Ein Modell im Umfeld **Buch/Bibliothek/Autor** und die vielfältigen Assoziationen ... sowie Constraints.

Siehe auch die Beispiele in der **Object Constraint Language Specification** und:



- Softwareprobleme vermeidende Spezifikationen: Die Benutzung von mit Einheiten versehenen Zahlenwerten, am Beispiel der Datei DM_Euro.cc

Euro
– Wert : double
– Euro() + Euro(dw : DM) + Euro(e : const Euro &) + Euro(w : double) + ZeigeWert() : double

Abbildung 0.1: Die Klasse Euro

DM
– Wert : double
– DM() + DM(ew : Euro) + DM(d : const DM &) + DM(w : double) + ZeigeWert() : double

Abbildung 0.2: Die Klasse DM

als instantiierbare Kinder einer abstrakten Klasse *Waehrung*.

Eine Anwendung:

Original mit anonymer Geldeinheit (double)

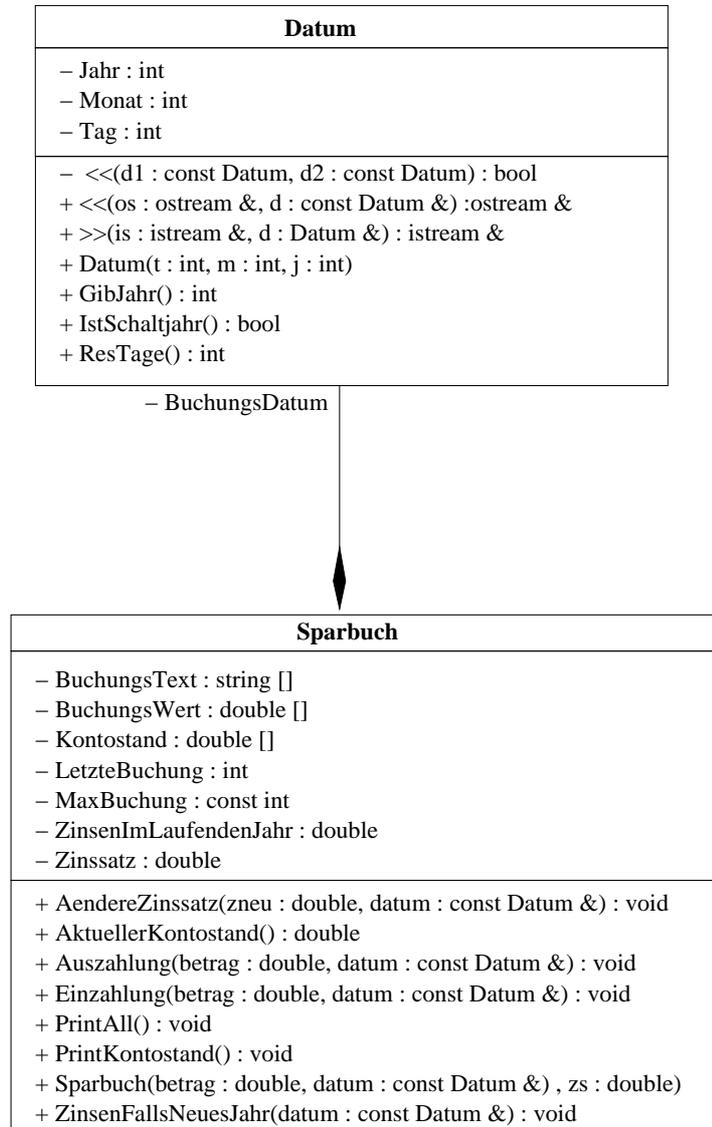


Abbildung 0.3: Die Klassen Datum und Sparbuch

Und besser: Klasse Sparbuch mit Klasse DM und Klasse EURO:

```
////////////////////////////////////  
// Datei:   DM_Euro.cc  
// Version: 1.1  
// Zweck:   DM und Euro  
// Autor:   Holger Arndt  
// Datum:   23.05.2001  
////////////////////////////////////  
  
#include <iostream>  
#include <iomanip>  
  
using namespace std;  
  
class DM;  
  
class Euro  
{  
private:  
    double Wert;  
public:  
    Euro() : Wert(0.0) {};  
    Euro(double w) : Wert(w) {};  
    Euro(const Euro &e) : Wert(e.Wert) {};  
    Euro(DM dw);  
    double ZeigeWert() const { return Wert; };  
};  
  
class DM  
{  
private:  
    double Wert;  
public:  
    DM() : Wert(0.0) {};  
    DM(double w) : Wert(w) {};  
    DM(const DM &d) : Wert(d.Wert) {};  
    DM(Euro ew) : Wert(ew.ZeigeWert() * 1.95583) {};  
    double ZeigeWert() const { return Wert; };  
};  
  
Euro::Euro(DM dw)  
{  
    Wert = dw.ZeigeWert() / 1.95583;  
};
```

```

}

void DruckeEuroBetrag(const Euro &e)
{
    cout << "Geldbetrag: " << setiosflags(ios::fixed) <<
        setprecision(2)
        << e.ZeigeWert() << " Euro" << endl;
}

int main()
{
    Euro b1(12.3);
    Euro b2(14.12);
    DM b3(1.23);
    Euro b4;
    Euro b5(b1);

    DruckeEuroBetrag(b1);
    DruckeEuroBetrag(b2);
    DruckeEuroBetrag(b3);
    DruckeEuroBetrag(b4);
    DruckeEuroBetrag(b5);

    return 0;
}

```

- Einheiten und Dimensionen in neueren Programmiersprachen:
Arbeite nicht mit dimensionslosen skalaren Attributen sondern mit **Maßeinheiten** (units) und **Dimensionsrechnung**:

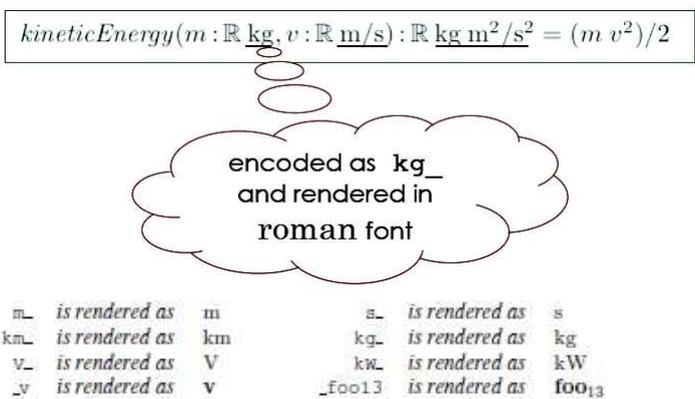
– http://h20219.www2.hp.com/Hpsub/downloads/50gWorking_with_units.pdf

```

:36_m*15_yd
:UBASE(ANS(1))
540_(m*yd)
493.7760_m^2
EDIT VIEW STACK RCL PURGE CLEAR

```

- Units and Dimensions in **Fortress**:



Vgl. <http://www.ipl.t.u-tokyo.ac.jp/~takeichi/attachments/Fortress.pdf>

```

v : ℝ m/s = (3 meters + 4 meters)/5 seconds

```

Correct

```

v : ℝ m/s = (3 meters + 4 seconds)/5 seconds

```

static error

```

v : ℝ m/s = (3 meters + 4 meters)/5

```

static error

```

kineticEnergy(3.14 kg, 32 f/s in m/s)

```

unit conversion

- Units and Dimensions in der Programmiersprache **F#**

```

let gravityOnEarth = 9.81<m/s^2> // Beschleunigung
let heightOfDrop = 3.5<m> // Laenge
let speedOfImpact = sqrt(2.0 * gravityOnEart * heightOfDrop)

```

Automatische Einheiten-Dimensionsrechnung in C++

```
#include <complex>
#include <iostream>

#include <boost/typeof/std/complex.hpp>

#include <boost/units/systems/si/energy.hpp>
#include <boost/units/systems/si/force.hpp>
#include <boost/units/systems/si/length.hpp>
#include <boost/units/systems/si/electric_potential.hpp>
#include <boost/units/systems/si/current.hpp>
#include <boost/units/systems/si/resistance.hpp>
#include <boost/units/systems/si/io.hpp>

using namespace boost::units;
using namespace boost::units::si;

quantity<energy>
work(const quantity<force>& F, const quantity<length>& dx)
{
    return F * dx; // Defines the relation: work = force *
                   distance.
}

int main()
{
    /// Test calculation of work.
    quantity<force>    F(2.0 * newton); // Define a quantity of
    force.
    quantity<length>  dx(2.0 * meter); // and a distance,
    quantity<energy>  E(work(F,dx)); // and calculate the work
    done.

    std::cout << "F = " << F << std::endl
               << "dx = " << dx << std::endl
               << "E = " << E << std::endl
               << std::endl;

    /// Test and check complex quantities.
    typedef std::complex<double> complex_type; // double real and
    imaginary parts.

    /// Define some complex electrical quantities.
    quantity<electric_potential, complex_type> v = complex_type
    (12.5, 0.0) * volts;
    quantity<current, complex_type>           i = complex_type
    (3.0, 4.0) * amperes;
    quantity<resistance, complex_type>        z = complex_type
    (1.5, -2.0) *
```

```

    ohms;

    std::cout << "V   = " << v << std::endl
              << "I   = " << i << std::endl
              << "Z   = " << z << std::endl
              // Calculate from Ohm's law voltage = current *
              // resistance.
              << "I * Z = " << i * z << std::endl
              // Check defined V is equal to calculated.
              << "I * Z == V? " << std::boolalpha << (i * z == v)
              << std::endl
              << std::endl;
    return 0;
}

```

produziert folgende Ausgabe:

```

F   = 2 N
dx  = 2 m
E   = 4 J

V   = (12.5,0) V
I   = (3,4) A
Z   = (1.5,-2) Ohm
I*Z = (12.5,0) V
I*Z == V? true

```

Boost.Units benutzt **Metafunktionen**, um `quantity<.>`-Werte mit automatischer Dimensionsanalyse zu ermöglichen:

```

quantity<length> L = 2.0*meters;           // quantity
    of length
quantity<time>   E = 14.5*seconds;        // quantity
    of time
// mit:
// template<class Unit, class Y = double> class quantity;
//

```

Conversions

Pool von vordefinierten Konstanten
alphabetische Liste der Grundeinheiten

Meßungenauigkeiten und Fehlerfortpflanzung:

```
quantity<length , measurement<double> >  
  u(measurement<double>(1.0,0.0)*meters) ,  
  w(measurement<double>(4.52,0.02)*meters) ,  
  x(measurement<double>(2.0,0.2)*meters) ,  
  y(measurement<double>(3.0,0.6)*meters);
```

mit den Ergebniswerten (Fehlerbalken):

```
x+y-w      = 0.48(+/-0.632772) m  
w*x        = 9.04(+/-0.904885) m^2  
x/y        = 0.666667(+/-0.149071) dimensionless
```

...

```
w*y^2/(u*x)^2 = 10.17(+/-3.52328) m^-1  
w/(u*x)^(1/2) = 3.19612(+/-0.160431) dimensionless
```

- **Weitere Beispiele zu Spezifikationsmängeln:**

- **PC-Problem lässt Walmart-Kunden in den USA dreifach zahlen**

Ein Computer-Problem hat dazu geführt, dass 800.000 Karten-Transaktionen bei Walmart-Filialen in den ganzen USA doppelt oder dreifach verbucht wurden. Aufgetreten sei der Fehler beim Transaktions-Dienstleister First Data. US-Medien zitieren die First-Data-Sprecherin Staci Busby: "Die mehrfachen Mastercard- und Visa-Buchungen haben wir wieder zurückgenommen, vor Dienstag sind diese aber nicht ausgeführt. Jeder, der am 31. März bei Walmart eingekauft hat, sollte seine Abrechnung noch einmal überprüfen."

Zu Details des Problems könne sie nichts sagen; klar sei jedoch, dass nur Walmart-Kunden davon berührt seien. Betroffene Kunden würden von First Data kontaktiert, versprach die Firmensprecherin, zudem sei eine kostenlose Info-Hotline geschaltet. (tol/c't)

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/46278>

- **US-Bezahlsystem mit öffentlichen Kreditkartendaten**

Durch einen primitiven Fehler auf den Webseiten des amerikanischen Bezahl-Dienstleisters PaySystems waren tausende von Kundendatensätzen einschließlich Kreditkartendaten zugänglich. Jeder PaySystems-Kunde konnte dabei die Daten anderer Kunden einsehen und sogar ändern.

PaySystems bietet an, Bezahlvorgänge zu widerrufen. Dabei wird diesem Vorgang eine Transaktionsnummer zugewiesen, die beim Aufruf der zugehörigen Informationen als Parameter in der URL auftaucht. Durch Ändern dieses Parameters konnte man beliebige Transaktionen anderer Kunden abrufen und anschließend über eine zweite URL auch deren Adresse und Kreditkartendaten.

myPaySystems

>> ORDER DETAILS

Order Number: 82821

Order Date:
Jan 23, 2004

Refund Date:
Jan 23, 2004

Vendor Company Name:
Med-E-Ville

Vendor's Customer Support Line (if available):

Website:
rxatcost.net

Product SKU:
Bonus Offer: 36 +8 viagra

Product Description:
36 + 8 Generic Viagra 100 mg

Besonders erschreckend war auch die Art und Weise, wie die Firma auf die Sicherheitslücke reagiert hat. Ein c't-Leser entdeckte das Problem zufällig und unterrichtete PaySystems unverzüglich. Als nach einer Woche nichts passierte, wendete er sich an heise Security. Auf unsere Nachfragen antwortete PaySystems prompt, dass man den Hinweis zur Kenntnis genommen habe und an der Beseitigung des Problems noch arbeite. Auf weitere Nachfragen, warum man die Seiten nicht unverzüglich gesperrt habe, kam keine Antwort mehr. Mittlerweile ist diese Lücke zwar geschlossen, aber die Daten standen – selbst nachdem PaySystems über das Problem informiert war – noch mindestens eine Woche ungeschützt im Netz.

Das Ausmaß des Problems lässt sich nur schwer abschätzen. Aber die Tatsache, dass die Transaktionsnummern sequenziell vergeben wurden und mehrere Stichproben sofort zum Erfolg führten, lässt darauf schließen, dass hunderte solcher Transaktionen zugänglich waren. Über welchen Zeitraum die Daten so offen im Netz standen, können wir nicht beurteilen. Nachdem PaySystems unsere diesbezüglichen Nachfragen ignoriert hat, rechnen wir nicht damit, dass der Dienstleister seine Kunden auf die mögliche Gefährdung der Kreditkartendaten hinweist. (ju/c't)

Link zu diesem Artikel bei heise-online:
<http://www.heise.de/newsticker/meldung/45566>

- Report: Wurm Lovsan nicht Schuld an Blackout 2003

Eine amerikanisch-kanadische Untersuchungskommission der Energieaufsichtsbehörde (FERC) ist zu dem Ergebnis gekommen, dass der Wurm Lovsan/MSBlaster nicht der Verursacher des gigantischen Stromausfalls im Nordosten der USA im vergangenen Jahr war. Beim Blackout 2003 waren 50 Millionen Amerikaner zeitweise ohne Strom. Da zeitgleich der Wurm im Internet die Runde machte und Millionen von Windows-Rechnern infizierte

oder lahmlegte, lag der Schluss nahe, Lovsan könne zum Ausfall beigetragen haben. Immerhin greifen Energieerzeuger schon seit längerem auf Windows für ihre Managementsysteme zurück. Anzeige

Im Februar dieses Jahres wurde aber bekannt, dass ein Softwarefehler eines Unix-Systems zur Überwachung und Steuerung von Stromnetzen beim Erzeuger FirstEnergy den Ausfall begünstigte. Durch den Fehler wurden Alarmer und Meldungen nicht mehr an das Kontrollpersonal weitergeleitet. Damit war es nicht mehr möglich, Gegenmaßnahmen zu ergreifen: Der Ausfall einer Versorgungsleitung führte zum Zusammenbruch des gesamten Stromverbundes.

Der Fehler des Managementsystems sei aber laut Untersuchungsbericht weder auf Cyberattacken durch Al-Quaida noch durch Würmer oder Viren zurückzuführen. Grundlage der Ermittlungen waren Befragungen von Mitarbeitern, Telefonmitschnitte und Berichte von Behörden und Geheimdiensten. Allerdings habe man nicht die Logdateien von Netzwerkgeräten, Firewalls und Intrusion-Detection-Systemen ausgewertet, die eventuell tiefere Hinweise gegeben hätten.

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/46328>

- **Software-Fehler verursachte US-Stromausfall 2003**

Acht Staaten im Nordosten der USA und Teile Kanadas blieben im August des vergangenen Jahres für fünf Tage ohne Strom. Insgesamt waren 50 Millionen Menschen betroffen. Schuld am Blackout war nach Angaben von SecurityFocus ein Softwarefehler des Managementsystems zur Überwachung und Steuerung von Stromnetzen beim Erzeuger FirstEnergy.

Das betroffene System XA/21 stammt von General Electric und ist bei Erzeugern weit verbreitet. Der Fehler wurde nach einem mehrwöchigen intensiven Code-Audit gefunden und soll bisher nur beim großen Blackout aufgetreten sein. Nach Angaben des Sprechers von FirstEnergy löste eine besondere Kombination von Ereignissen und Alarmen den Fehler aus, woraufhin das System seinen Dienst einstellte. Der kurz darauf einspringende Backup-Server versagte ebenfalls, da er mit der Zahl der bereits aufgelaufenen, aber nicht verarbeiteten Meldungen überfordert war.

In der Folge nahm das System auflaufende Alarmer nicht mehr entgegen und meldete sie nicht an das Bedienpersonal weiter. Hinzu kam, dass den Betreibern nicht einmal auffiel, dass ihr System bereits versagt hatte. Eine Stunde lang soll die Kontrollstation veraltete Daten angezeigt haben. Bei auftretenden Störungen blieb zwangsläufig die Reaktion aus.

Normalerweise koppelt ein Stromerzeuger sein Netz bei größeren Ausfällen von den anderen Stromnetzen ab, um weitere Schäden durch Überlast zu

vermeiden. Somit bleibt ein Problem regional begrenzt. Da die Alarme aber nicht registriert wurden, reagierten die Operatoren nicht.

FirstEnergy will nun seine XA/21-Systeme gegen die Produkte eines Wettbewerbers austauschen. Das North American Electric Reliability Council (NERC) hat eine Richtlinie herausgegeben, in der Maßnahmen beschrieben sind, Vorfälle wie am 14. August zu vermeiden. Unter anderem wird darin FirstEnergy aufgefordert, bis zum Austausch ihrer System alle notwendigen Patches für XA/21 zu installieren.

Da sich der Zeitpunkt des Blackouts und der Ausbruch des Wurms Lovsan/Blaster überschneiden, gab es Vermutungen, der Wurm könnte den Ausfall verursacht haben. Auch warnte das CERT/CC Anfang August davor, dass Lovsan Unix-Systeme mit Distributed Computing Environment (DCE) angreift und zum Absturz bringen kann. XA/21 ist ein EMS/SCADA-System (Supervisory Control and Data Acquisition), das auf Unix mit X-Windows basiert. Sicherheitslücken gibt es hier reichlich. Somit kann zukünftig nicht ausgeschlossen werden, dass Würmer, die den Weg in ein Kontrollzentrum gefunden haben, solche Systeme beeinflussen können.

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/44621>

- **US-Sicherheitsexperten fordern bessere Ausbildung für Softwareentwickler**

Die National Cyber Security Partnership (NCSP) fordert in ihrem aktuellem "Security Across the Software Development Life Cycle" eine bessere Ausbildung der Entwickler. Der Bericht befasst sich insbesondere mit dem Lebenszyklus von Software. Sicherheit müsse sich über die gesamte Lebensspanne eines Software-Produktes erstrecken. Jeder Abschnitt der Spanne, angefangen vom Design und Spezifikation, über die Implementierung und Tests bis hin zum Patch-Management soll unter den Gesichtspunkten der IT-Sicherheit bearbeitet werden.

Die Arbeitsgruppe hat zur Definition entsprechender Empfehlungen vier Untergruppen gebildet, die sich mit Schulung von Entwicklern und Anwendern, Softwareprozessen und Patchen beschäftigen. Die vierte Gruppe – Incentive Subgroup – will ein Programm erarbeiten, um Herstellern das Entwickeln von sicherer Software schmackhaft zu machen. Dazu sollen Preisverleihungen und Zertifizierungen gehören. Daneben stellt man auch die Idee vor, die Sicherheit einzelner Softwaremodule als Messlatte für die weitere Karriere der jeweiligen Entwickler heranzuziehen.

Die vergangenes Jahr gegründete Arbeitsgruppe hat sich die Verbesserung der Cyber Security der US-amerikanischen Informationsinfrastruktur zum Ziel gesetzt. Mitglieder sind diverse Sicherheitsexperten aus Forschung, Lehre und

Industrie, sogar Vertreter der National Security Agency finden sich in der Gruppe. Die Vorsitzenden der Gruppe sind Ron Moritz von Computer Associates und Scott Charney von Microsoft. Ähnliche Ziele wie die NCSP verfolgen die Cyber Security Industry Alliance (CSIA) und der Global Council of CSOs

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/46241>

- Softwarefehler plagt Mercedes-Diesel

Software-Bugs plagen die User nicht etwa nur, wenn sie vor dem Computer am Schreibtisch sitzen oder mit Mobilrechnern unterwegs sind. Internet-Zugang, Navigationsrechner oder multimediale Konsolen lassen das Auto zum IT-Problemfeld werden – darüber hinaus aber kämpfen Automobil-Elektroniker mittlerweile mit immer komplexeren computergestützten Steuerungssystemen und deren Software und damit auch mit den Bugs dieser Software. Jüngstes Beispiel: Wegen eines Softwarefehlers ruft DaimlerChrysler rund 10.000 Transporter der Mercedes-Benz-Modelle Vito und Viano mit Dieselmotoren zurück. In Deutschland sollen rund 3.000 Fahrzeuge betroffen sein.

Ursache des Rückrufs ist ein Bug in der Software, mit der die Dieselesteuergeräte ausgerüstet sind. Sie aktivieren in Situationen, in denen dies eigentlich nicht vorkommen sollte, die Kraftstoffabschaltung, wodurch der Motor ausgeht. Betroffen seien Fahrzeuge mit Dieselmotoren, die zwischen November 2003 und April 2004 hergestellt wurden. Die Kunden würden durch die Servicestellen von Mercedes-Benz direkt angeschrieben, erklärte der Konzern; die Fahrzeuge erhielten eine fehlerbereinigte Software.

Link zu diesem Artikel bei heise-online:

<http://www.heise.de/newsticker/meldung/48403>

Weitere Links:

- [Code Contracts for .NET](#)
- [10 der bekanntesten Software-Fehler](#)

natürlichsprachige informelle versus formaler Spezifikation
(mit Beispielen in der Spezifikationsprache Z)
formale Spezifikation in Z

<i>Abs</i>
<i>BirthdayBook</i> <i>BirthdayBook1</i>
$known = \{ i : 1..hwm \bullet names(i) \}$ $\forall i : 1..hwm \bullet$ $birthday(names(i)) = dates(i)$

(aus dem Z Reference Manual)

Seite 27 (Natural Language) in: Introduction to Modelling

1 UML und SdV

1.1 Rekapitulation: UML-Klassendiagramme

1.1.1 Klassen und Objekte

<http://de.wikipedia.org/wiki/Klassendiagramm>



Abbildung 1.1: Eine Klasse

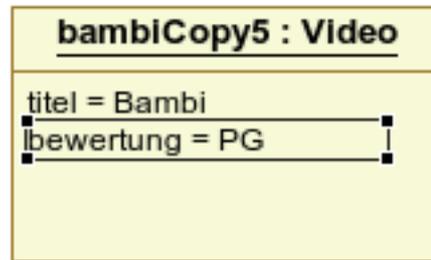
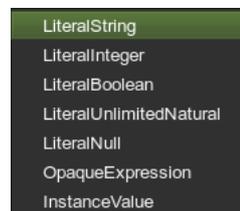


Abbildung 1.2: Ein Objekt dieser Klasse (InstanceSpecification)

<<primitive>> UML-Datentypen:

Boolean
String
Integer
UnlimitedNatural
Real



Siehe Kapitel 12 von <http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/>

1.1.2 Klassenspezifikation

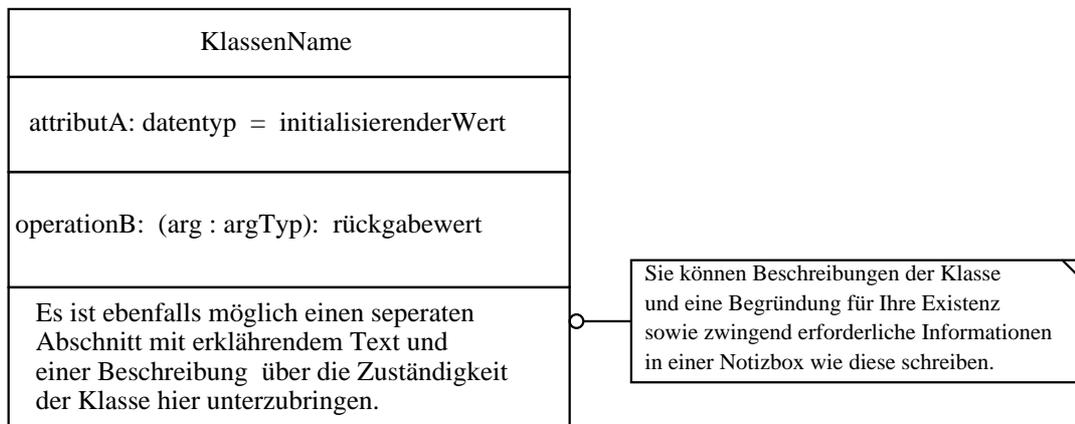


Abbildung 1.3: Spezifikation einer Klasse

klassenName

Normale Schrift = konkrete Klasse

kursiveSchrift **oder** << abstract >> = abstrakte Klasse

(kursive Schriften sind nicht bildschirmfreundlich; benutzen Sie die Stereotyp-Notation)

Klassen- oder Instanzenattribute

Normale Schrift = Instanzen-Bereich

Unterstrichen **oder** \$ = Klassenobjekte (\$ ist kein UML-Standard)

in der Regel mit kleinem Anfangsbuchstaben

Methoden/Operationen

Für abstrakte Methoden benutzen Sie = 0 oder << abstract >>

(=0 ist kein UML-Standard)

in der Regel mit kleinem Anfangsbuchstaben

Attribut- und Methodensichtbarkeit

+ public (öffentliche Sichtbarkeit)

- private (private Sichtbarkeit)

protected (geschützte Sichtbarkeit)

~ package

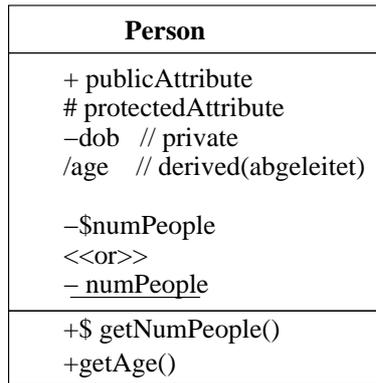


Abbildung 1.4: Eine Klasse: Person

- Das Attribut **age** ist abgeleitet.
- Die Anzahl der Instanzen der Klasse **Person** (numPeople) ist ein Attribut der Klasse **Person** selbst und nicht von einer Instanz der Klasse. Diese wird als statisches Klassen-Attribut (class static member variable) bezeichnet. Sie arbeitet wie eine globale Variable der Klasse. Manchmal wird als alternative Schreibweise für Klassenattribute und deren Verhalten das \$ Zeichen verwendet.

1.1.3 Links und Assoziationen

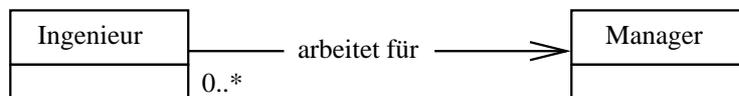


Abbildung 1.5: Assoziationen verbinden Klassenexemplare

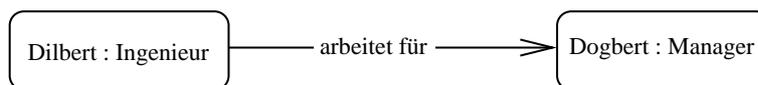


Abbildung 1.6: Assoziationen verbinden Klassenexemplare

1.1.4 Rollen und Assoziationsnamen

Rolle

Benannte Instanzen einer Klasse die an das anderen Ende der Assoziation geschrieben werden, gewöhnlich ein Substantiv. Werden automatisch als Attribut in der Ausgangsklasse der Assoziation realisiert. Rollennamen sollten in der Regel mit kleinem Buchstaben beginnen.

Assoziationsname

Benennt die Assoziation selbst; erfordern zuweilen einen Pfeil, der die Richtung der Assoziation anzeigt; gewöhnlich Verben oder Verbschlagworte.

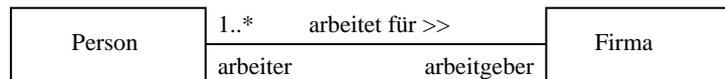


Abbildung 1.7: Rollen in Klassen

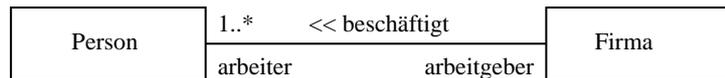
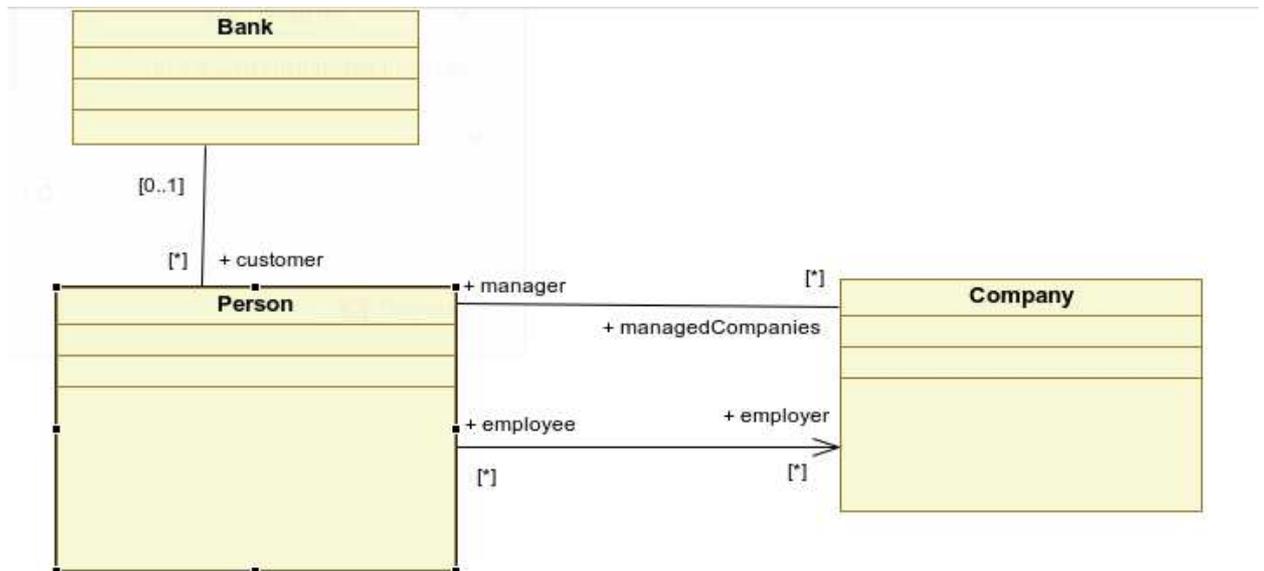


Abbildung 1.8: Rollen in Klassen (Fortsetzung)

Einige Beispiele:



1.1.5 Multiplizitäten (Kardinalitäten)

- Multiplizitäten beschreiben die Anzahl der Instanzen am Assoziationsende.
- Beispiele:

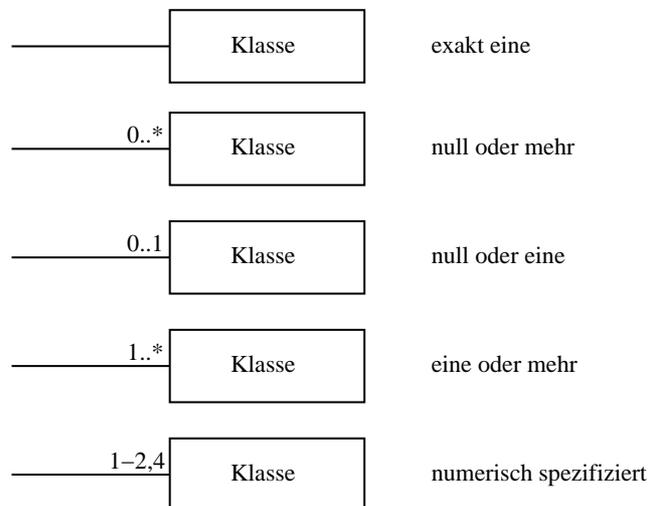


Abbildung 1.9: Multiplizität

Anmerkung: n und * können anstelle von 0..* verwendet werden.

1.1.6 Stereotypen

Stereotypen

Eine konventionelle Kategorisierung für modellierende Entitäten:

- Sie werden oft bei Klassen, Assoziationen und Methoden angewendet.
- Sie bieten einen Weg, UML zu erweitern; sie dienen zur Definition eigener, für spezielle Probleme modellierter Elemente.
- Einige Stereotypen werden von CASE-Werkzeugen (CASE tool generator) erkannt.

Es gibt zwei Wege, Stereotypen darzustellen:

- Benutzen Sie normale UML-Elemente, mit dem Stereotypnamen zwischen << und >>.
- Benutzen Sie eigene eindeutige Icons.

Beispiele:

```
<< abstract >>, << interface >>, << exception >>,
<< instantiates >>, << subsystem >>, << extends >>,
<< instance of >>, << friend >>,
<< constructor >>, << thread >>, << uses >>,
<< global >>, << create >>, << invent your own >>
```

Wir werden die folgenden Stereotypen im Sinne von SdV/DbC benutzen:

```
<< constructor >>
<< destructor >>
<< basic observator >>
<< derived observator >>
<< modifier >>
```

Andere gebräuchliche Stereotypen sind:

```
<< interface >>
<< utility >>
<< local >>
<< parameter >>
<< delegate >>
<< ... >>
```

[http://de.wikipedia.org/wiki/Stereotyp_\(UML\)](http://de.wikipedia.org/wiki/Stereotyp_(UML))

1.1.7 Tagged Values

- Tagged Values sind ein weiterer Mechanismus, UML zu erweitern: Er erlaubt es, dem Modell neue Eigenschaftsspezifikationen hinzuzufügen (Name = Wert).

Gebräuchliche Beispiele für **tagged values** sind:

- {Autor = (Dave,Ron)}
- {Versionsnummer = 3}
- {Location = d:\Location\uml\examples}
- {Location = Node: Middle Tier}

1.1.8 Generalisierung, Spezialisierung und Vererbung

- **Arbeitnehmer** generalisiert **Manager** und **Ingenieur**.
- **Ingenieur** spezialisiert **Arbeitnehmer**.
- **Manager** ist eine **Art/Sorte** von **Arbeitnehmer**.
- **Manager** und **Ingenieur** erben die Schnittstellen von **Arbeitnehmer** und in diesem Fall auch einige Implementierungseinzelheiten.

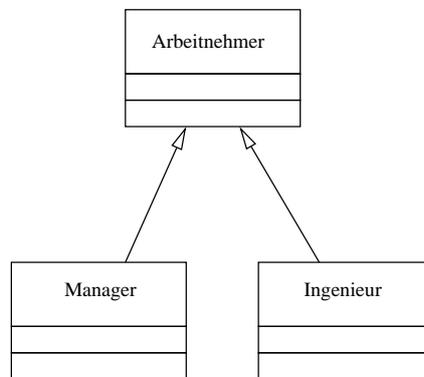


Abbildung 1.10: Generalisierung, Spezialisierung und Vererbung

1.1.9 Abstrakte Klassen

- Eine Generalisierung ohne vollständige Implementierungsspezifikation.
- Sie wird in UML mit dem Stereotyp `<< abstract >>` angezeigt.
- In C++ werden alle **pure virtual** Methoden = 0 deklariert.
- In Java wird sie mit dem Schlüsselwort "abstract" gekennzeichnet
- Ein **Interface** ist wie eine abstrakte Klasse, aber ohne jede Implementierung.

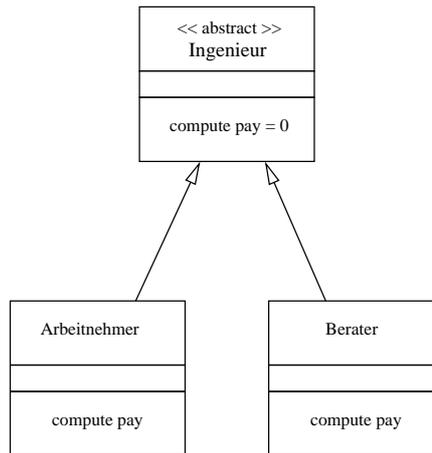


Abbildung 1.11: Abstrakte Klassen

1.1.10 Komposition / Aggregation

Das Rautenzeichen wird für verschiedene Eigenschaften / Konzepte eingesetzt.

- Teil- / Ganzes-Beziehung (am häufigsten verwendet)
- Hat - ein
- Hat - eine Sammlung - von
- Ist zusammengesetzt - aus

Beachten Sie, wie die Zeit die Kardinalitäten beeinflussen kann: Ein Auto kann viele Fahrer haben, aber zu einem bestimmten Zeitpunkt, kann es nur einer fahren.

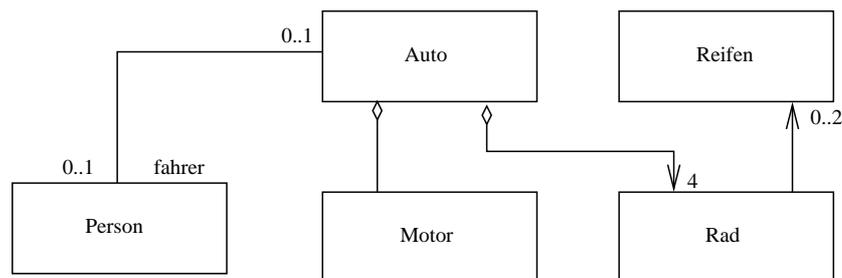


Abbildung 1.12: Komposition / Aggregation

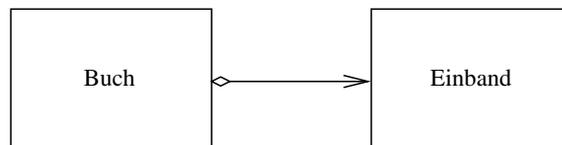
Komposition:

- UML benutzt ein ausgefülltes Rautensymbol für eine **Komposition**.
- Das leere Rautensymbol beschreibt eine **Aggregation**.
- Eine **Komposition** ist eine stärkere Assoziation als eine **Aggregation**. Der Unterschied besteht darin, dass bei einer **Komposition**, ein Teil nie mehr als ein Ganzes ist und das ein Teil und ein Ganzes immer einen gemeinsamen Lebenszyklus/Lebenszeit haben.
- In folgenden Beispiel sind **Zeilen** ein fester und permanenter Bestandteil des **Layouts**, aber die Anzahl der Zeichen in jeder Zeile verändert sich zur Lebenszeit des Layout-Exemplars.



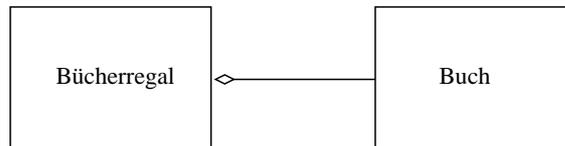
Abbildung 1.13: Komposition zwischen Layout und Zeile

- Das Objekt **Zeile** ist ein Teil vom Objekt **Layout**, sodass Zeilen erzeugt werden, wenn ein Layout erzeugt wird und Zeilen zerstört werden, wenn ein Layout zerstört wird. **Zeile** hat keine selbstständige Existenz.
- Beispiel: Ein Buch besteht aus Seiten (pages) und einem Einband (cover).



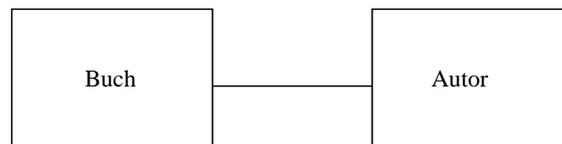
Aggregation:

- Instanzen der Klasse Buch existieren unabhängig von Objekt Bücherregal, aber Objekt Bücherregal hat Kenntnis von seinen Instanzen der Klasse Buch.



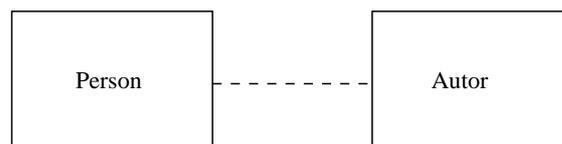
Assoziation:

- Ein Objekt der Klasse Buch hält eine halb-permanente Referenz zu einem Objekt der Klasse Autor ohne jede einschränkende Semantik.
- Beispiel: Bücher haben einen Autor



Dependancy:

- Instanzen der Klasse Person haben vorübergehende Beziehungen zu Instanzen der Klasse Autor
- Beispiel: Eine Person liest ein Buch, dann gibt sie es einem Freund.



1.1.11 Qualifizierte Assoziationen/Qualified Associations

- Sie werden benutzt, damit Instanzen einer Klasse, die in einer "ein zu viele"-Beziehung zu einer anderen Klasse B stehen, über einen eindeutigen Identifizierer schnell auf die Instanzen von B zugreifen zu können.
- Qualifizierte Assoziationen sind für gewöhnlich mit einer Art "Wörterbuch" ausgestattet (auch als assoziative Felder bekannt), etwa ein **Hash Table** oder einer **TreeMap**.

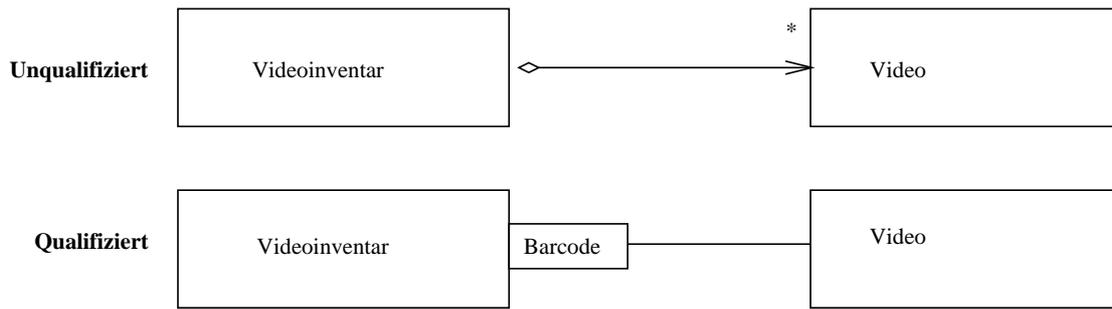


Abbildung 1.14: Qualifizierte Assoziation

1.1.12 Assoziationsklassen

Assoziationen benötigen manchmal eigene Attribute.

- Im folgenden Beispiel ist ein Arbeitsvertrag eine Assoziationsklasse für die "arbeitet für"-Assoziation.
- **Anmerkung:** Die Semantik der Assoziationsklasse (so wie sie modelliert wurde) zeigt an, dass für jedes Personen/Firma-Paar, exakt ein Arbeitsvertrag existiert. Somit beschreibt dieses Modell, dass eine Person nicht zu zwei unterschiedlichen Zeiten für dieselbe Firma arbeiten kann.
- **Anmerkung:** Der Stereotyp <<Geschichte>> erklärt den Zeitaspekt der Beziehung: Er besagt, dass eine Person über die Zeit für viele Firmen arbeiten kann, aber zu einer bestimmten Zeit immer nur für keine (0) oder eine (1) Firma arbeitet.

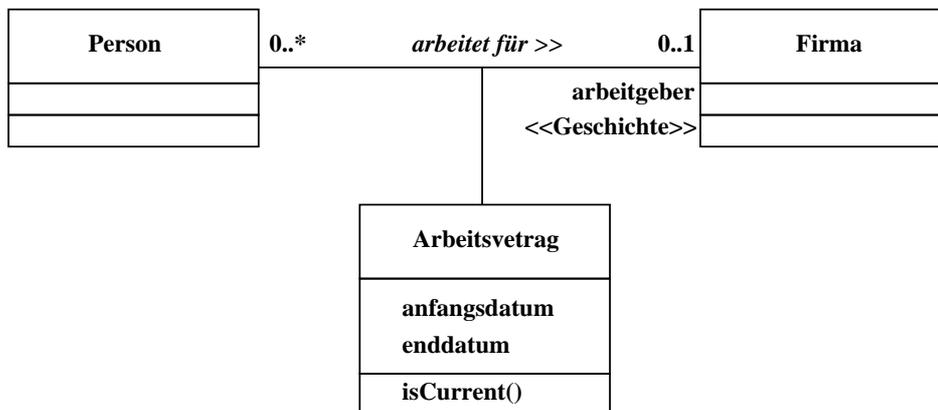


Abbildung 1.15: Assoziierte Attribute

- Unterstützt Ihr UML-Tool keine Assoziationsklassen, sollte man folgendes Work-around benutzen.

- Beachten Sie dabei die Änderung in der assoziierten Kardinalität und die Tatsache das die "arbeitgeber"-Assoziation nun abgeleitet ist ("/").

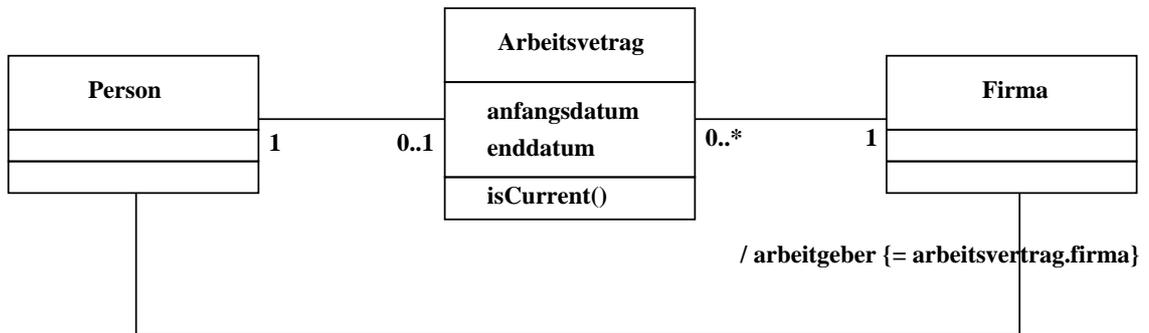


Abbildung 1.16: Assoziiertes Attribut (Fortsetzung)

1.1.13 template classes

<http://www.csci.csusb.edu/dick/samples/uml1b.html#Templates>

<http://tfs.cs.tu-berlin.de/lehre/SS04/GRA-IOSEP/Folien/bjoern.pdf>

How to use templates in UML models

1.1.14 UML 2.0

http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_classdiagram.html

<http://www.jeckle.de/files/umltutorial.pdf> (Seite 16ff.)

Assoziationen im Klassendiagramm

1.1.15 Modell und Metamodell

UML User-Modell und Metamodell (Seite 9f., 19f.)

4-Schichten-Architektur von UML (Seite 10)

1.2 Spezifikation einfacher Klassen nach Prinzipien der SdV

1.2.1 Ein einfaches Beispiel ...

KEY, VALUE

```
mydictionary
-----
- keys: vector<KEY>*
- values: vector<VALUE>*
- count: unsigned int
-----
/* basic queries */
+ get_count() : unsigned int
+ has(k: const KEY &) : bool
+ value_for (k: const KEY &) : VALUE

/* constructors */
+ << constructor >> mydictionary()
+ << constructor >> mydictionary(
    s: const mydictionary<KEY,VALUE>&)
+ << destructor >> ~mydictionary(): null

/* disable assignmet operator */
- = (s: const mydictionary<KEY,VALUE>&):
    mydictionary<KEY,VALUE>&

/* derived queries */
/* ... */

/* modifiers */
+ put(k: const KEY &, v: const VALUE &): null
+ remove(k: const KEY &): null
```

Klassifikation der Methoden in

- grundlegende Abfragen (Queries/Observatoren)
- abgeleitete Abfragen (Queries/Observatoren)
- Aktionen (Modifikatoren)
- Konstruktoren/Destruktoren

Siehe dazu zum Beispiel:

Spezifikation durch Vertrag — eine Basistechnologie für eBusiness

Das typische Aussehen von Verträgen zwischen Nutzer und Lieferant von mydictionary:

Wann darf der default-Konstruktor benutzt werden?

Wann darf der Kopierkonstruktor benutzt werden?

Welche Wörterbücher erzeugen sie jeweils?

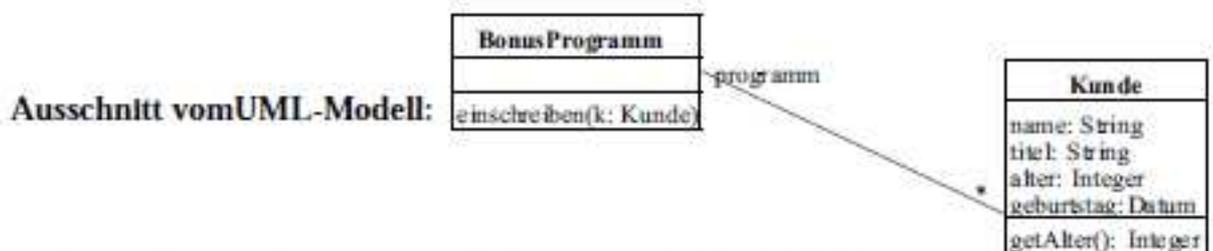
Wann darf `remove(k)` benutzt werden?

Darf `put(k,v)` nur im Falle `not has(k)` benutzt werden? Was geschieht, wenn für einen schon vorhandenen Schlüssel `put(k,v)` benutzt wird?

...

1.2.2 Vor- und Nachbedingungen in OCL

OCL-Manual Seite 8f.



```
BonusProgramm::einschreiben(k:Kunde)  
pre: not kunde->includes(k)  
post: kunde = kunde@pre->including(k)
```

(aus: http://www.bruegge.in.tum.de/teaching/ss01/Info2/vorlesung/fohlen/03a_Vertraege_4.pdf)

1.2.3 Spezifikation durch Verträge

http://de.wikipedia.org/wiki/Design_by_contract

(SdV, *Design by Contract*¹, *Programming by Contract*) ist eine Methode zur Spezifikation der dynamischen Semantik von Softwarekomponenten mit Hilfe von Verträgen aus erweiterten booleschen Ausdrücken. SdV basiert auf der Theorie der abstrakten Datentypen und formalen Spezifikationsmethoden. Spezifizierte Komponenten können Module, Klassen oder Komponenten im Sinne von Komponententechnologien (wie Microsofts COM, .NET oder Suns EJB) sein. Verträge ergänzen das Kunden-Lieferanten-Modell:

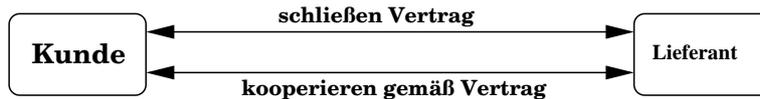


Abbildung 1.17: Kunden-Lieferanten-Modell

Grundlegend für die Vertragsmethode ist das **Prinzip der Trennung von Diensten in Abfragen und Aktionen** (*command-query separation*):

- **Abfragen** geben Auskunft über den Zustand einer Komponente, verändern ihn aber nicht. Sie liefern als Ergebnis einen Wert. Die Abfragen einer Komponente beschreiben ihren abstrakten Zustand.
- **Aktionen** verändern den Zustand einer Komponente, liefern aber kein Ergebnis. Die Aktionen einer Komponente bewirken ihre Zustandsveränderungen.

Diesem Prinzip folgend sind seiteneffektbehaftete Funktionen als Dienste zu vermeiden².

SdV	VERPFLICHTUNGEN	VORTEILE
Benutzer der Klasse	delegiert nur bei erfüllter Vorbedingung	kommt in den Genuß der garantierten Nachbedingung und Invarianten
Anbieter der Klasse	erfüllt die Nachbedingung (oder löst Exception aus)	braucht Vorbedingung nicht überprüfen; kann sich auf deren Einhaltung verlassen

Tabelle 1.1: Verpflichtungen/Vorteile von Verträgen zwischen Komponentenanbieter und -benutzer

¹„Design by Contract“ ist ein Warenzeichen von Interactive Software Engineering.

²In bestimmten Fällen, z.B. bei Fabrikfunktionen, können Seiteneffekte sinnvoll sein. Solche Funktionen sind nicht als Spezifikatoren verwendbar und sollten entsprechend gekennzeichnet sein.

1.2.3.1 Methodenklassifikation in C++

- const-Methoden (Abfragen/Queries/Observatoren) teilt man in wesentliche und abgeleitete solche ein.
- Die wesentlichen Observatoren erlauben eine vollständige Spezifizierung des Zustands eines Klassenexemplars.
- Sie (und nur sie) werden nicht durch Nachbedingungen spezifiziert. Sie dienen vielmehr dazu, abgeleitete Observatoren und Modifikatoren (das sind nicht-const-Methoden) in ihren Nachbedingungen näher zu bestimmen.
- Dazu werden die abgeleiteten Observatoren durch eine Nachbedingung unter Benutzung einer oder mehrerer wesentlicher Observatoren spezifiziert.
- Modifikatoren werden durch eine Nachbedingung unter Benutzung aller wesentlicher Observatoren spezifiziert, um den exakten Zustand des Exemplars am Ende des Modifikatoraufrufs anzugeben.
- Verzichte (evtl.) in Nachbedingungen von Modifikatoren darauf, explizit zu spezifizieren, was sich nicht ändert (in der Annahme, dass alles nicht explizit genannte als *ungeändert* zu gelten hat). Leider ist nicht immer klar, was *ungeändert* zu bedeuten hat: Mindestens dann sollten Frameregeln (Rahmenbedingungen) explizit spezifizieren, was nach Aufruf des Modifikators *gleich* ist wie vorher.
- Explizite Spezifikation aller Rahmenbedingungen können bei programminterner Überprüfung der Nachbedingungen fehlerhafte Implementierungen aufdecken!
- Schreibe für jede Methode eine Vorbedingung mit Hilfe von
 - Abfragen und
 - Bedingungen an Methodenparameter.

Hier (bei den Vorbedingungen) dürfen auch abgeleitete Abfragen, die eventuell effizienter sein können als eine sonst nötige Kombination mehrerer wesentlicher Abfragen, benutzt werden.

- Sorge dafür, dass bei Erfülltsein der Vorbedingungen auf jeden Fall die Nachbedingungen ebenfalls erfüllt sind (oder — in Ausnahmefällen — eine Exception ausgelöst wird).
- Sorge dafür, dass die Abfragen in Vorbedingungen effizient berechnet werden (evtl. durch Hinzufügen weiterer effizienter abgeleiteter Abfragen). Vergesse nicht, die evtl. hinzugefügten neuen abgeleiteten Abfragen durch Nachbedingungen (und Vorbedingungen) zu spezifizieren.
- Nutze Invarianten um die Abhängigkeit von Methoden zu spezifizieren (Konsistenzbeziehungen).

- Untersuche alle Abfragen paarweise auf Redundanzen und formuliere solche explizit als Invarianten.
- Wann immer Abfrage-Ergebnisse oder Methoden-Parameter eingeschränkte Wertebereiche besitzen, formuliere dies explizit in Form von
 - Vorbedingungen,
 - Nachbedingungen
 oder
 - Invarianten.
- Schreibe die Nachbedingungen von virtuellen (also überschreibbaren) Methoden immer in der Form


```
Vorbedingung implies Nachbedingung
```

 (oder `(!Vorbedingung) || Nachbedingung`), um die Redefinition in Kindklassen konfliktfrei zu ermöglichen.

1.2.3.2 Vertragspflichten/Vertragsnutzen

Ein Grund für die strikte Trennung von Abfragen und (reinen) Aktionen ist, dass Abfragen als **Spezifikatoren** dienen, d.h. als Elemente von Verträgen. **Verträge** setzen sich aus Bedingungen folgender Art zusammen:

- **Invarianten** einer Komponente sind allgemeine unveränderliche Konsistenzbedingungen an den Zustand einer Komponente, die vor und nach jedem Aufruf eines (public) Dienstes gelten. Formal sind Invarianten boolesche Ausdrücke über den Abfragen der Komponente; inhaltlich können sie z.B. Geschäftsregeln (business rules) ausdrücken.
- **Vorbedingungen** (preconditions) eines Dienstes sind Bedingungen, die vor dem Aufruf eines Dienstes erfüllt sein müssen, damit er ausführbar ist. Vorbedingungen sind boolesche Ausdrücke über den Abfragen der Komponente und den Parametern des Dienstes.
- **Nachbedingungen** (postconditions) eines Dienstes sind Bedingungen, die nach dem Aufruf eines Dienstes erfüllt sind; sie beschreiben, welches Ergebnis ein Dienstaufruf liefert oder welchen Effekt er erzielt. Nachbedingungen sind boolesche Ausdrücke über den Abfragen der Komponente und den Parametern des Dienstes, erweitert um ein Gedächtniskonstrukt (`@pre`), das die Werte von Ausdrücken vor dem Dienstaufruf liefert.

Verträge legen Pflichten und Nutzen für Kunden und Lieferanten fest. Die Verantwortlichkeiten sind klar verteilt:

Der Lieferant garantiert die Nachbedingung jedes Dienstes, den der Kunde aufruft, falls der Kunde die Vorbedingung erfüllt. Eine verletzte Vorbedingung ist ein Fehler des Kunden, eine verletzte Nachbedingung oder Invariante (bei erfüllter Vorbedingung) ist ein

Fehler des Lieferanten. Die Verträge spezifizieren also eindeutig die Verantwortlichkeit bei Auftreten eines Fehlers.

	KUNDE	LIEFERANT
PFLICHT	Die Vorbedingung einhalten.	Die Nachbedingung herstellen und die Invariante erfüllen.
NUTZEN	Ergebnisse/Wirkungen nicht prüfen, da sie durch die Nachbedingungen garantiert sind. Bei Methodenaktivierung werden die Anweisungen ausgeführt, die die Nachbedingungen herstellen und die Invarianten erhalten	Die Vorbedingungen nicht prüfen; sie sind durch den Vertrag garantiert und Mehrfachüberprüfungen sollten vermieden werden.

Tabelle 1.2: Pflichten - Nutzen von Kunden und Lieferanten

Schwache Vorbedingungen erleichtern den Kunden die Arbeit, starke Vorbedingungen dem Lieferanten. Je schwächer die Nachbedingungen sind, umso freier ist der Lieferant und umso ungewisser sind die Kunden über das Ergebnis/den Effekt. Je stärker die Nachbedingungen sind, umso mehr muß der Lieferant leisten.

1.2.3.3 Beispiel-Codeverträge

... in der C++-Macro-Erweiterung **Gnu Nana**
(siehe <http://savannah.gnu.org/projects/nana/>)
... für eine Klasse **vektor**:

- friend-Funktion **Norm()** (abgeleitete Abfrage/Query/Observator)

```

double Norm(const vektor& v)
{
    REQUIRE(v.invariant());
    // ...
    // double qsum = ...
    ENSURE(approximatelyEqualTo(qsum, S(int k=v.lo(), k<=v.hi
        ( ),k++,
                                v(k)*v(k)), 2.0));
    ENSURE(approximatelyEqualTo(result*result, qsum, 2.0));
    return result;
}

```

- Methode `normalize()` (Modifikator ohne Rückgabewert (void))

```

void vektor::normalize()
DO
    REQUIRE(Norm(*this)!=0.0);
    ID(vektor value_old(*this));
    ...
    ENSURE(approximatelyEqualVekTo(result*n, value_old,
        2.0));
    ENSURE(approximatelyEqualTo(Norm(result), 1.0, 2.0)
        );
END

```

- i-ter Einheitsvektor (statische Klassenmethode)

```

vektor vektor::ei(int n, int i)
{
    REQUIRE((n>=1) && (1<=i) && (i<=n));
    ...
    ENSURE(result.lo()==1);
    ENSURE(result.hi()==n);
    ENSURE(E1(int k=result.lo(), k<=result.hi(), k++,
        result(k)!=0.0));
    ENSURE(result(i)==1.0);
    ENSURE(result.invariant());
    ...
}

```

- Konstruktor

```

vektor::vektor(const double x[], int n) : low(1), high(n)
{
    REQUIRE((n>=1) && (x!=0));
    REQUIRE("x[] hat mindestens n Komponenten");
    ...
    ENSURE(lo()==1 && hi()==n);
    ENSURE(A(int k=lo(), k<=hi(), k++, (*this)(k)==x[k-lo()])
        );
END

```

- Modifikator

```

void vektor::changeValueAt(int i, double x)
DO
    REQUIRE((lo()<=i) && (i<=hi()));
    ...

```

```

    ENSURE((*this)(i)==x);
    ENSURE("alle anderen Komponenten von *this ungeändert");
END

```

Überlegen Sie sich einen expliziten Nichtänderungsvertrag für „alle anderen Komponenten“ von `*this` (Frame-Bedingung).

- `operator!=` (abgeleitete Abfrage)

```

bool vektor::operator!=(const vektor& w) const
DO
    REQUIRE(w.invariant());
    ...
    ENSURE(result == ((hi()-lo())!=(w.hi()-w.lo())) ||
           E(int k=lo(), k<=hi(), k++, (*this)(k)!=w(k-lo()+w.lo
           ()))));
    ...
}

```

Auszüge aus dem [Nana 2.5 Manual](#):

```
void REQUIRE (exprn) Macro
```

Called at the beginning of each method to check its precondition.

```
void ENSURE (exprn) Macro
```

Called at the end of each method. This checks the postcondition for a method.

```
typeof(exprn) S (init,condition,next,exprn) Macro
```

Sum the values generated by `exprn` for all values given by `'for(int;condition;next)'`. The type of the value returned is given by the type of the `exprn`.

```
bool E1 (init,condition,next,exprn) Macro
```

There exists only one value generated by `'for(int;condition;next)'` for which the `exprn` is true.

```
bool A (init,condition,next,exprn) Macro
```

For all values generated by `'for(int;condition;next)'` the `exprn` must be true.

```
bool E (init,condition,next,exprn) Macro
```

There exists at least one value for `exprn` generated by `'for (int;condition;next)'` which is true.

We also provide support for referring to previous values of variables in postconditions.

The ID macro is used to create variables to save the old state in.:
void ID (Text decln) Macro

The programmer should provide a class method called 'invariant' which returns 'true' if the object is consistent, 'false' otherwise.

```
DO // checks the class invariant + {  
...  
END // check the class invariant + }
```

Benutzt wurden zusätzlich die Hilfsfunktionen:

```
static bool approximatelyEqualTo(double left ,  
                                double right ,  
                                double factor)  
{  
    return fabs(left - right) <=  
        std::numeric_limits<double>::epsilon( ) *  
        factor * std::max(fabs(left), fabs(right));  
}
```

```
template <class T>  
set<T> operator+(const set<T>& s , const T& e) {  
    set<T> result( s ) ;  
    result.insert( e ) ;  
    return result ;  
}
```

1.2.3.4 Subcontracting/Untervertragswesen

Es gelten folgende Regeln bei der Vererbung (von is-a-Methoden):

- a) Vorbedingungen können in einer Kindklasse (Untervertrag) abgeschwächt werden oder müssen gleich sein.
- b) Nachbedingungen in einer Kindklasse (Untervertrag) müssen (im Falle des Erfüllseins der Vorbedingung der Elterklasse) gleich oder stärker sein als diejenigen der Elterklasse.
- c) Invarianten in der Kindklasse (Untervertrag) müssen ebenfalls gleich oder stärker als die der Elterklasse sein.

Dann ist ein echtes *Subcontracting* realisiert.

Bemerkung: Es reicht die Kindnachbedingung im Falle des Eintreffens der Eltervorbedingung gleich oder stärker als die Elternachbedingung zu realisieren. Im Falle „Kindvorbedingung **and not** Eltervorbedingung“ darf die Kindnachbedingung frei gewählt werden.

$$\begin{array}{l} \text{Invariante}_{\text{Kindklasse}} = \text{Invarinte}_{\text{Elterklasse}} \text{ and } \dots \\ \text{Vorbedingung}_{\text{Kindmethode}} = \text{Vorbedingung}_{\text{Eltermethode}} \text{ or } \dots \\ \text{Nachbedingung}_{\text{Kindmethode}} = \begin{cases} \text{Nachbedingung}_{\text{Eltermethode}} \text{ and } \dots & , \text{ falls } \text{Vorbedingung}_{\text{Eltermethode}} \\ \text{beliebig} & , \text{ sonst} \end{cases} \end{array}$$

1.2.3.5 Beispiel zum Subcontracting

Contract in nana:

```
class name_list {
...
public:
    //////////// basic queries:
    unsigned int get_count() const; // number of items in stack
    bool has(const string& a_name) const;
...
    //////////// (pure) modifiers:
    virtual void put(const string& a_name); // Push a_name
                                           // into list
}
void name_list::put(const string& a_name) // Push a_name
                                           // into list

DO
    REQUIRE(/* name not in list */ !has(a_name));
    ID(set<string> contents_old(begin(),end()));
    ID(int count_old = get_count());
    ID(bool not_in_list = !has(a_name));
...
    ENSURE(has(a_name));
    ENSURE( (!not_in_list) || (get_count() == count_old + 1));
    ID(set<string> contents(begin(),end()));
    ENSURE( (!not_in_list) || (contents == contents_old + a_name
        ));
END
```

und ein möglicher *is-a*-Subcontract:

```
//////////////////////////////// child class relaxed_name_list //////////////////////////////////
//////////////////////////////// (more user friendly) //////////////////////////////////
class relaxed_name_list : public name_list{
    ////////////////////////////////// (pure) modifiers: (redefined)

    virtual void put(const string& a_name);    // Push a_name
                                              //into list
    ...
}
void relaxed_name_list::put(const string& a_name)    // Push
a_name                                              //into list

DO
  REQUIRE(/* nothing */ true); // usable without conditions
  ID(set<string> contents_old(begin(),end()));
  ID(int count_old = get_count());
  ID(bool not_in_list = !has(a_name));
  ...
  ENSURE(has(a_name));
  ENSURE((!not_in_list) || (get_count() == count_old + 1));
  ENSURE(not_in_list || (get_count() == count_old));
  ID(set<string> contents(begin(),end()));
  ENSURE(not_in_list || (contents == contents_old));
  ENSURE((!not_in_list) || (contents == contents_old + a_name)
  );
END
```

Die Regeln des Contracting/Subcontracting, Zusammenfassung:

Klassen-Invarianten (Gültige Attributwertkombinationen/Objekte der Klasse)

- schränken Werte von Attributen ein, trennen gültige von ungültigen Exemplaren einer Klasse
- spezifizieren Redundanzen (vgl. Day/Month/Year, Count/IsEmpty, ...)

Methoden-Vorbedingungen (an Attribute und Parameter)

- schränken den Bereich ein, in dem die Methode erfolgreich sein muß, benutzt werden darf

Methoden-Nachbedingungen (an Attribute und Parameter)

- spezifizieren (formal) das Ergebnis der Methode (das **was**, nicht das **wie**)

Was vor und nach jeder Methode gelten muß (in Form von **Hoare-Tripeln** notiert):

<p>Konstruktor: $\{VB_{Parameter}\}$ Konstruktor $\{Inv \text{ and } NB_{Konstruktor}\}$</p> <p>Destruktor: $\{Inv\}$ Destruktor $\{-\}$</p> <p>Jede andere (öffentliche) Methode M: $\{VB_M \text{ and } Inv\}$ M $\{NB_M \text{ and } Inv\}$</p>
--

Weitere Subcontracting-Beispiele

aus: http://www.cse.yorku.ca/course_archive/2004-05/F/3311/sectionA/22-InheritDBCgen.pdf

1.2.3.6 Funktion invert (Invertieren einer Matrix)

Beispiel in **Eifel**:

Ursprüngliche Definition:
invert(epsilon:REAL) is - - Invert matrix with precision epsilon
 require epsilon >= 10⁽⁻⁶⁾
 ...
 ensure abs ((Current * inverse) - Identity) <= epsilon
end

Redefinition: (Untervertrag)
invert(epsilon:REAL) is - - Invert matrix with precision epsilon
 require else epsilon >= 10⁽⁻²⁰⁾
 ...
 ensure then abs ((Current * inverse) - Identity) <= (epsilon/2)
end

1.2.3.7 Interface myDictionary::Put

Ein Vertrag zwischen Kunde und Unternehmer (in **Cleo** spezifiziert) laute:

```
INTERFACE myDictionary[Keys, Values]
ACTIONS
  Put(IN k:Keys, IN v:Values)
    PRE
      NOT Has(k)
    POST
      Has(k)
      ValueFor(k) = v
      Count = OLD(Count)+1
  .
  .
  .
```

Kann er durch den Unternehmer allein nicht zeitgerecht erfüllt werden, so kann sich dieser eventuell folgendermaßen aus seiner Notlage befreien: Ein anderer Unternehmer biete den folgenden Vertrag an:

```

INTERFACE myDictionary[Keys, Values]
ACTIONS
  Put(IN k:Keys, IN v:Values)
    PRE
      TRUE
    POST
      Has(k)
      ValueFor(k)= v
      NOT OLD(Has(k)) IMPLIES Count = OLD(Count)+1
      OLD(Has(k))      IMPLIES Cout = OLD(Count)
.
.
.

```

1.2.3.8 Interface LoeseLGS

```

----- LoeseLGS-Elter
ACTIONS
  LoeseLGS( IN A : Matrix,
            IN b : Vektor,
            OUT x : Vektor )
    PRE
      NOT Det(A) = 0
    POST
      || A * x - b || <= EPSILON

```

sowie ein Subcontract:

```

----- LoeseLGS-Kind
ACTIONS
  LoeseLGS( IN A : Matrix,
            IN b : Vektor,
            OUT x : Vektor )
    PRE
      TRUE
    POST
      NOT Det(A) = 0 IMPLIES || A * x - b || <= EPSILON
      Det(A) = 0     IMPLIES "x ist eine Minimalstelle von || A * x - b ||"

```

1.2.3.9 Interface Bruecke

INTERFACE Fussgaengerbruecke

----- Fussgaengerbruecke

QUERIES

MaxLast : REAL

AktLast : REAL

INVARIANTS

MaxLast >= 7500

AktLast <= MaxLast

ACTIONS

ueberquereBruecke(IN gew : REAL,
OUT Guthaben : INTEGER)

PRE

gew + AktLast <= MaxLast

gew <= 200

Guthaben >= 2

POST

AktLast = OLD(AktLast) + gew

Guthaben = OLD(Guthaben) - 2

verlasseBruecke(IN gew : REAL)

...

sowie ein Subcontract:

INTERFACE Autobruecke

----- Autobruecke

QUERIES

MaxLast : REAL

AktLast : REAL

INVARIANTS

MaxLast >= 800000

AktLast <= MaxLast

ACTIONS

ueberquereBruecke(IN gew : REAL,
OUT Guthaben : INTEGER)

PRE

gew + AktLast <= MaxLast

gew <= 20000

Guthaben >= 20

POST

AktLast = OLD(AktLast) + gew

OLD(gew) <= 200 IMPLIES Guthaben = OLD(Guthaben) - 2

NOT OLD(gew) <= 200 IMPLIES Guthaben = OLD(Guthaben) - 20

verlasseBruecke(IN gew : REAL)

...

Aufgabe:

Überlege Contracts und Subcontracts im Umfeld:

- Kunde/Stammkunde
- Firmenkonto/Privatkundenkonto
- Vereinsmitglied /Vorstandsmitglied
- ...

1.2.3.10 Zusammenfassung der SdV-Prinzipien

1. Observatoren (und nur diese) haben einen Ergebniswert; sie ändern den Objektinhalt nicht!
Modifikatoren haben **keinen** Ergebniswert.
2. Unterscheide: "grundlegende Observatoren" von
3. "abgeleiteten Observatoren". Jeder abgeleitete Observator hat eine Nachbedingung, die auf die grundlegenden Observatoren zurückgreift.
4. Für jeden Konstruktor/Modifikator schreibe eine Nachbedingung, die die Werte aller grundlegenden Observatoren am Ende einer Methode festlegt.
5. Für jeden Observator und jeden Konstruktor/Modifikator schreibe notwendige Vorbedingungen.
6. Schreibe für jede Klasse eine Invariante, die die sich nicht ändernden Merkmale der Objekte beschreibt (also **gültige** und **ungültige** Objekte unterscheidet).
7. Die grundlegenden Observatoren sind ein minimaler Methodensatz, der dazu dient den Zustand eines Exemplars vollständig zu charakterisieren. Sie haben außer Konsistenzbeziehungen zu anderen Methoden **keine** Nachbedingungen.

1.2.4 ... und sein (OCL2-)Vertrag

```
package mydictionary

context mydictionary
inv: keys != 0
inv: values != 0
inv: count >= 0
inv: keys->size() = values->size()
inv: keys->size() = count
/* Inv. der priv. Attribute fuer das Implementierungs-Team */
/* basic observators */
context mydictionary::get_count() : Integer
body: count
/* post: result = KEY->select(k | has(k))->size() */
context mydictionary::has(k : KEY) : Boolean
post consistentWithCount: get_count()=0 implies not result
context mydictionary::value_for(k: KEY): VALUE
pre: has(k)
/* constructor */
context mydictionary::mydictionary()
post: get_count() = 0
context mydictionary::mydictionary(s : mydictionary) // const &
post: s.get_count() = self.get_count()
/* post: KEY->forall(k | s.has(k) implies s.value_for(k) = self
    .value_for(k)) */
/* modifcator */
context mydictionary::put(k: KEY, v: VALUE)
pre: not has(k)
post: has(k)
post: get_count() = get_count@pre() + 1
post: value_for(k) = v
/* post: KEY->forall(kl | has@pre(kl) implies value_for@pre(kl)
    = value_for(kl)) */
context mydictionary::remove(k: KEY)
pre: has(k)
post: not has(k)
post: get_count() = get_count@pre() - 1
/* post: KEY->forall(kl | has@pre(kl) implies (kl = k or
    value_for@pre(kl) =
    value_for(kl)))
    */

endpackage — mydictionary
```

1.3 Ein Beispiel aus dem industriellen Einsatz: Die Klasse `java.awt.Color`

<code>java.awt.Color</code>
<code>- int redness</code> <code>- int blueness</code> <code>- int greenness</code> <code>- int opaqueness</code>
<code><<constructor>></code> <code>+ Color(int r, int g, int b)</code> <code>+ Color(float r, float g, float b, float a)</code>
<code><<query>></code> <code>+ int getRed()</code> <code>+ Color darker()</code> <code>+ Color brighter()</code> <code>...</code>

Abbildung 1.18: Die Standard Farbklassse: `java.awt.Color`

Was sagt Ihnen dieses Klassendiagramm? Was sagt es nicht?

1.3.1 Klassenspezifikation: `java.awt.Color`

Invarianten: (Für jedes Farbobjekt `c`)

$0 \leq \text{redness}(c) \leq 255$ **and** $0 \leq \text{greenness}(c) \leq 255$ **and**
 $0 \leq \text{blueness}(c) \leq 255$ **and** $0 \leq \text{opaqueness}(c) \leq 255$

Verträge der Konstruktor-Methoden:

Listing 1.1: Konstruktor-Methoden:

```
public Color(int r, int g, int b)
  pre: 0 <= r <= 255 and 0 <= g <= 255 and 0 <= b <= 255
      —(throws IllegalArgumentException)
  — modifies: redness, greenness, blueness, opaqueness
  post: redness = r and greenness = g and blueness = b
       and opaqueness = 255

public Color(float r, float g, float b, float a)
  pre: 0.0 <= r <= 1.0 and 0.0 <= g <= 1.0 and 0.0 <= b <=
      1.0 and 0.0 <= a <= 1.0
      —(throws IllegalArgumentException)
  post: redness = r*255 and greenness = g*255 and
       blueness = b*255 and opaqueness = a*255
```

Constructors throwing exceptions

Verträge der Query-Methoden und Modifikatoren:

Listing 1.2: Query-Methoden

```
public int getRed()
  post: result == redness

public Color darker()
  post: result.redness == redness*0.7
       and result.greenness == greenness*0.7
       and result.blueness == blueness*0.7
       and result.opaqueness == 255

public Color brighter()
  post: (redness / 0.7) > 255 implies result.redness == 255
       and (redness / 0.7) <= 255 implies result.redness ==
         redness / 0.7
       and (greenness / 0.7) > 255 implies result.greenness == 255
       and (greenness / 0.7) <= 255 implies result.greenness ==
         greenness / 0.7
       and (blueness / 0.7) > 255 implies result.blueness == 255
       and (blueness / 0.7) <= 255 implies result.blueness ==
         blueness / 0.7
       and result.opaqueness == 255
...

```

Bemerkung: Im Sinne des „Programming by Contract“ sollte der wesentliche Teil der Spezifikation „völlig“ implementierungsunabhängig durchgeführt werden. Das heißt:

- kein Zugriff auf private Attribute bzw. Methoden
- keine Vorwegnahme der zu benutzenden Algorithmen
- Alle Nachbedingungen sollten mit Hilfe der *basic Queries* formuliert werden.
- Alle Vorbedingungen sollten mit Hilfe der *derived Queries* formuliert werden.

Bemerkung: Die Spezifikation mittels OCL geschieht aber nicht **nur** für den benutzenden Programmierer, sondern auch als Hilfe innerhalb des Implementierungsteams. **Hier** sollte natürlich auch auf implementierungsabhängige Einzelheiten Bezug genommen werden können. Zum Beispiel sollten die **basic Queries** selbst mittels Nachbedingungen spezifiziert werden, die aber nur dem Implementierungsteam sichtbar sein sollten.

1.3.2 Hinweise

1. Spezifiziere wo immer nötig implementierungsspezifische Entscheidungen (meist in der Form $\langle \rangle 0$)
2. Stelle sicher, daß die in den Vorbedingungen benutzten Observatoren „effizient“ arbeiten. Falls nötig, füge zusätzliche abgeleitete schnell arbeitende Observatoren hinzu (virtuelle, nur zur Spezifikation benötigte Methoden).
3. Wenn ein abgeleiteter Observator als Attribut implementiert wird, sollte die Klasseninvariante entsprechend erweitert werden.
4. Um die Neuimplementierung virtueller Methoden zu unterstützen sollte **jede** Nachbedingung einer virtuellen Methode durch Ihre Vorbedingung abgeschirmt sein.

Zu C++:

- Konstante Methoden sind (reine) Observatoren.
- Nichtkonstante Methoden sollten keine Ergebnisse liefern; bei Beendigung muß neben der Nachbedingung auch die die Klasseninvariante erfüllt sein.
- Direkter modifizierender Zugriff auf Attribute ist gefährlich (warum?) und sollte deshalb nicht erlaubt sein.

2 OCL-Spezifikation von Klasseninterdependenzen

2.1 Abhängigkeiten der assoziierten Exemplare

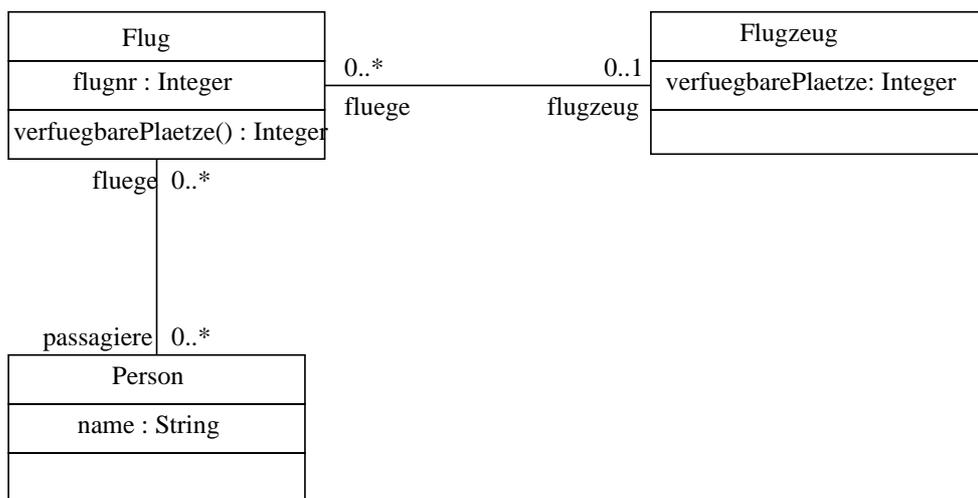


Abbildung 2.1: assoziierte Exemplare

body: zur Spezifikation der Ergebnisse von query-Operationen:

```
context Flug :: verfügbarePlaetze () : Integer
```

```
body: self.flugzeug.verfügbarePlaetze
```

```
context Flug
```

```
inv konsistenteBidir1 : flugzeug.fluege -> includes (self)
```

```
context Flugzeug
```

```
inv konsistenteBidir2 : fluege.flugzeug = Bag{self}
```

```
context Flug
```

```
def: momentanVerfügbarePlaetze () : Integer =  
    verfügbarePlaetze () - passagiere -> size ()
```

Durch `def`: eingeführte Methoden oder Attribute sind für OCL-Zwecke bestimmte Hilfsobjekte, die im Klassen-Gültigkeitsbereich als `<<OclHelper>>`-Objekte nutzbar sind. Das Modell kann zum Beispiel in C++ implementiert werden durch:

```
class Flug{
  Flugzeug* flugzeug;
  int      flugnr;
  vector <Person *>
    passagiere;
  int verfuegbarePlaetze ();
}

class Person{
  string name;
  vector <Flug *> fluege;
}

class Flugzeug{
  vector <Flug *> fluege;
  int verfuegbarePlaetze;
}
```

Abbildung 2.2: Implementierungsbeispiel

Momentaner Status etwa:

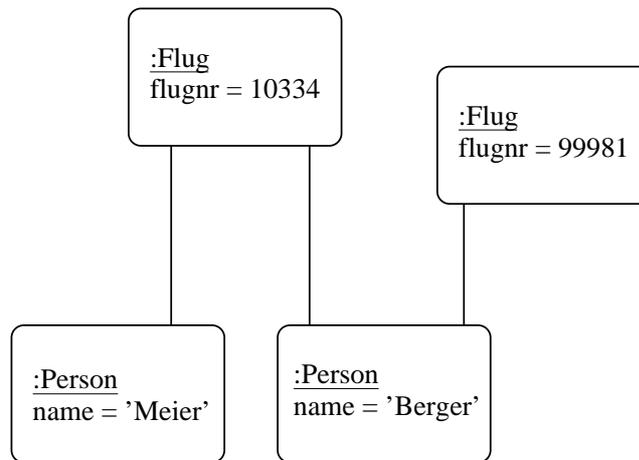
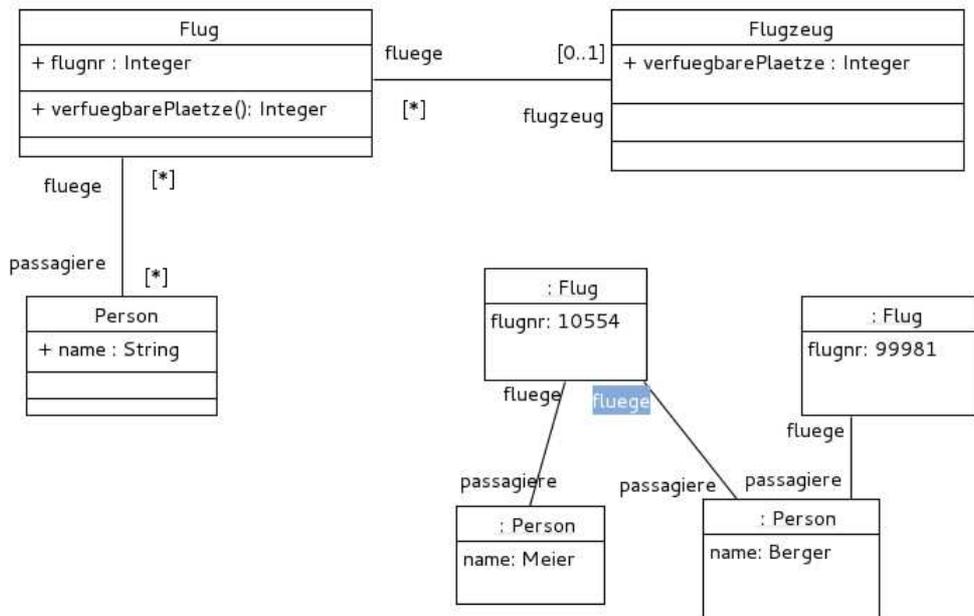


Abbildung 2.3: Zustand/Schnappschuß (Objektdiagramm)

In Papyrus:



Bemerkungen:

Beachte die nicht normgerechten Doppelpunkte statt Gleichheitszeichen bei der Spezifikation der Attributwerte der Instance-Specification.

Benutze `instance specification`, `Classifier`, `slot`, `defining feature`, `LiteralInteger`, ... und `Instance Specification link`,

Types of Collection

- **Set:**
 - Non-ordered, unique
 - Example: **Set** {1, 2, 5, 88}
- **OrderedSet:**
 - Ordered, unique
 - Example: **OrderedSet** {'apple', 'grape', 'orange'}
- **Bag:**
 - Non-ordered, non-unique
 - Example: **Bag** {1, 2, 5, 88, 5}
- **Sequence:**
 - Ordered, non-unique
 - Example: **Sequence** {2, 3, 4, 5}
 - **Sequence** {2..5}

(Formal Modelling and Analyzing a Flash File System)

2.2 size(), includes() und forAll() – Methoden-Verträge

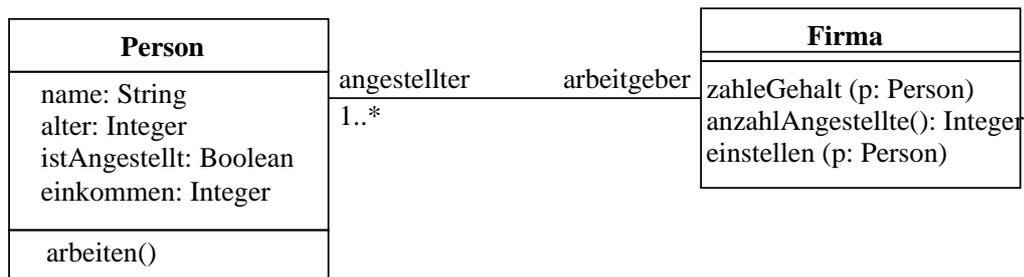


Abbildung 2.4: Modell Person-Firma

Zu diesem Klassendiagramm liege der folgende, in OCL formulierte Vertrag vor:

```

context Person
  inv: alter >= 0

context Person::arbeiten()
  pre: istAngestellt and alter >= 13

context Firma::zahleGehalt(p: Person)
  pre: angestellter->includes(p)

context Firma::anzahlAngestellte() : Integer
  pre: angestellter->forAll(p | p.arbeitgeber = Bag{self})
  post: result = angestellter->size()

context Firma::einstellen (p: Person)
  pre: (p.alter >= 13) and not p.istAngestellt and angestellter->excludes(p)
  post: ((angestellter->size()) = angestellter@pre->size() + 1)
        and p.istAngestellt and angestellter->includes(p)

context Person
  inv: if istAngestellt then
        einkommen >= 300
      else
        einkommen < 300
      endif

context Person
  def: spitzname:String = 'Angestellter'
  
```

Vergleiche auch Abschnitt 7.5.4 des OCL 2.3.1 Handbuchs.

or	if -
and	then -
xor	else -
not	endif
=	
<>	
implies	

Tabelle 2.1: logische Operationen in OCL

2.3 Der Ergebnistyp von (Mehrfach-)Navigationen; collect() von Klassenfeatures

- Navigation durch eine Assoziation mit Vielfachheit 1 liefert ein Objekt des Assoziationsendtyps.
- Das Navigieren durch (mehrere) Assoziationen liefert beim ersten Erreichen einer nicht-1 Rolle ein Objekt des Typs
 - Bag oder
 - Sequence oder
 - OrderedSet oder
 - Set

je nachdem, ob die Rolle die Eigenschaft

- { - }
- { ordered }
- { ordered, unique }
- { unique }

hat.

- Zu einer 0..1-Rolle `rollenname` sollte man nur geschützt durch `rollenname->notEmpty()` `implies rollenname...` voranschreiten.
- Weiternavigation nach Erreichen eines Container-Objekts liefert einen Bag, falls die letzte Navigation zu einer *nicht* als { ordered } gekennzeichneten Rolle führt oder die Vielfachheit 1 hat.
- Ist die letzte Rolle bei der Weiternavigation nach zwischenzeitlichem Erreichen eines Containerobjekts als { ordered } gekennzeichnet, so ist das Navigationsergebnis eine Sequence.
- Jede (explizite oder implizite) `collect()`-Operation (vergleiche Abschnitt 7.7.2 des OCL 2.3.1 Handbuchs) liefert einen Bag, falls die Quelle ein Set oder Bag ist. Sie liefert eine Sequence, falls die Quelle eine Sequence oder ein OrderedSet ist:
Im Kontext von Firma ist

```
self.angestellter->collect(alter)
self.angestellter->collect(p | p.alter)
self.angestellter->collect(p : Person | p.alter)
```

ein Bag. Benötigt man die *Menge* der Alter aller Angestellten, so muß explizit typgewandelt werden:

```
self.angestellter->collect(alter)->asSet()
```

Da collect()-Operationen sehr häufig benötigt werden, können sie in abkürzender Schreibweise ähnlich wie Mehrfachnavigationen benutzt werden:
`self.angestellter.alter->asSet()`

Papyrus kann dazu benutzt werden, den Typ einer OCL-Subexpression zu erfragen:
 Welchen Typ hat zum Beispiel fluege im context Flugzeug?

The screenshot displays the Papyrus UML editor interface. At the top, there are tabs for 'model.uml', 'model.di', and 'model.ocl'. The main workspace shows a class diagram with the following elements:

- Class Flug:** Attributes include '+ flugnr : Integer' and '+ verfuegbarePlaetze(): Integer'.
- Class Person:** Attribute includes '+ name : String'.
- Associations:**
 - Flug to Person: Multiplicity [*] on both ends.
 - Flug to an unnamed class: Multiplicity [0..1] on the unnamed class end, [*] on the Flug end.
- Instance Diagram:**
 - Instance : Flug with attribute flugnr: 10554.
 - Two instances of : Person, both with attribute name: Meier.
 - Lines connect the : Flug instance to both : Person instances, labeled 'passagiere'.
 - Lines connect the : Flug instance to the unnamed class, labeled 'fluege'.

Below the diagram is the 'Interactive OCL' console. It shows the following text:

```

Cannot find operation (= (Flugzeug)) for the type (Set(Flug))

Evaluating:
fluege =
self
Results:
Cannot find operation (= (Flugzeug)) for the type (Set(Flug))

fluege = self
    
```

model.uml model.di model.ocl

```

classDiagram
    class Flug {
        +flugnr : Integer
        +verfuegbarePlaetze(): Integer
    }
    class Flugzeug {
        +verfuegbarePlaetze : Integer
    }
    class Person {
        +name : String
    }
    Flug "0..1" -- "*" Flugzeug : fluege
    Flug "*" -- "*" Person : passagiere
    Flug "1" -- "1" Person : fluege
    Flug "1" -- "1" Person : passagiere
  
```

Interactive OCL UML M1

Evaluating:
 fluege.flugzeug
 = self

Results:
 Cannot find operation (= (Flugzeug)) for the type (Bag(Flugzeug))

fluege.flugzeug = self

2.4 Assoziationsklassen

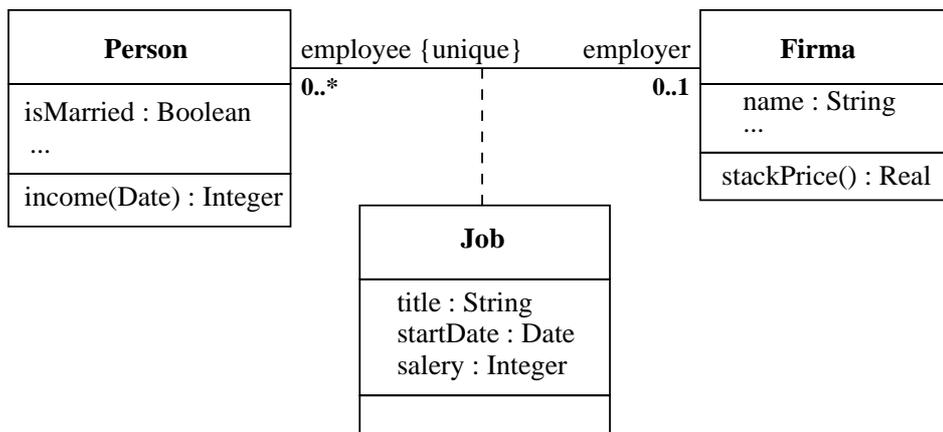


Abbildung 2.5: Assoziationsklasse Job

wird als Workaround implementiert als:

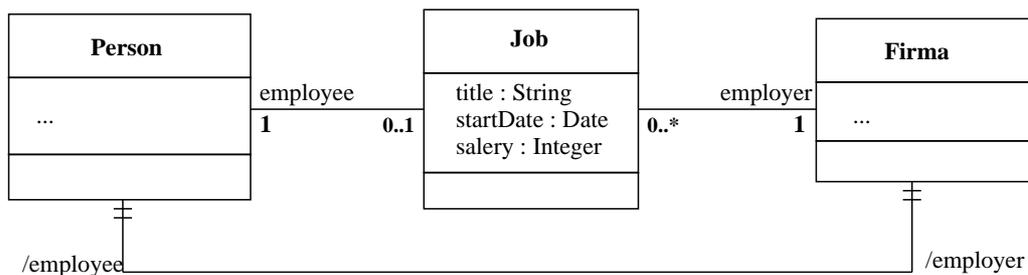


Abbildung 2.6: Assoziationsklasse im Workaround

Hier ist zu benutzen:

```
context Firma
... job.employee ...
```

statt original

```
context Firma
... employee ...
```

Deshalb wird in der Workaround-Klasse Firma die abgeleitete Rolle `employee` als

```
context Firma :: employee : Set(Person)
derive : job.employee -> asSet()
```

eingeführt.

Analog für die Klasse Person:

```
context Person :: employer : Firma  
derive: job.employer
```

Alternativ könnte man die <<OclHelper>>-Attribute

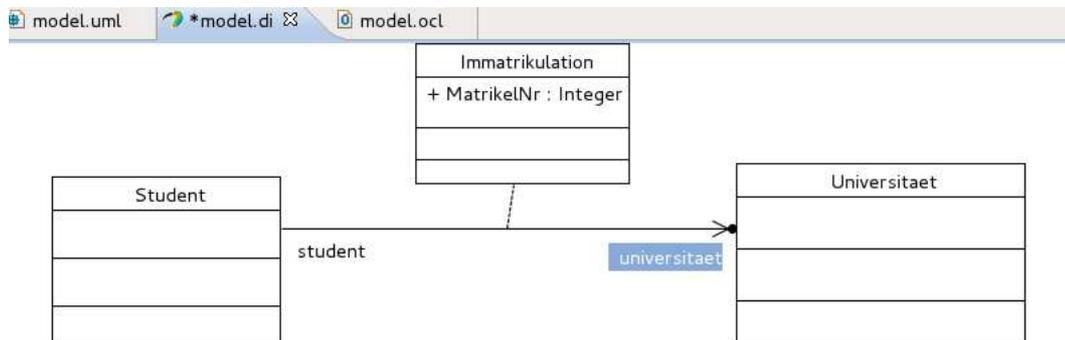
```
context Firma  
def: employee : Set(Person) = job.employee->asSet()  
  
context Person  
def: employer : Firma = job.employer  
einführen.
```

Aufgabe: Wie unterscheiden sich diese beiden Workaround-Vorgehensweisen in Bezug auf die Benutzbarkeit?

Weitere OCL-Ausdrücke:

```
context Person :: income (d:Date) : Integer  
body: self.job → select(d >= startDate).salary → sum()
```

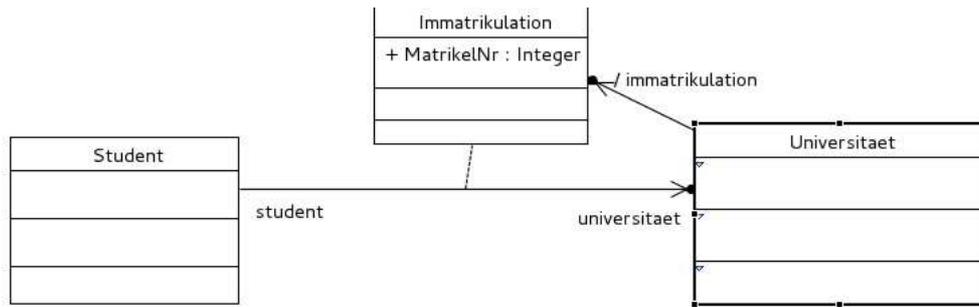
In Papyrus 0.8.2 werden Assoziationsklassen leider noch unsymmetrisch bezüglich beider Assoziationsenden behandelt:



```
interactive OCL
Evaluating:
immatikulation = self
Results:
Unrecognized variable: (immatikulation)

Evaluating:
student
= self
Results:
'Successfully parsed.'
```

Die Pfeilspitze deutet das nicht owned Assoziationsende an. Die dort befindliche Klasse sieht die Assoziationsklasse zur Zeit noch nicht (nicht im Sichtbarkeitsbereich): Um im Kontext Universitaet mittels des Rollennamens immattribution auf die Assoziationsklasse zugreifen zu können benötigt man einen Workaround:



2.5 Qualifizierte Assoziationen



Abbildung 2.7: qualifizierte Aggregation

context Bank

inv: self.customer[145].name = 'Maier'

Workarounds ohne qualifizierten Zugriff:

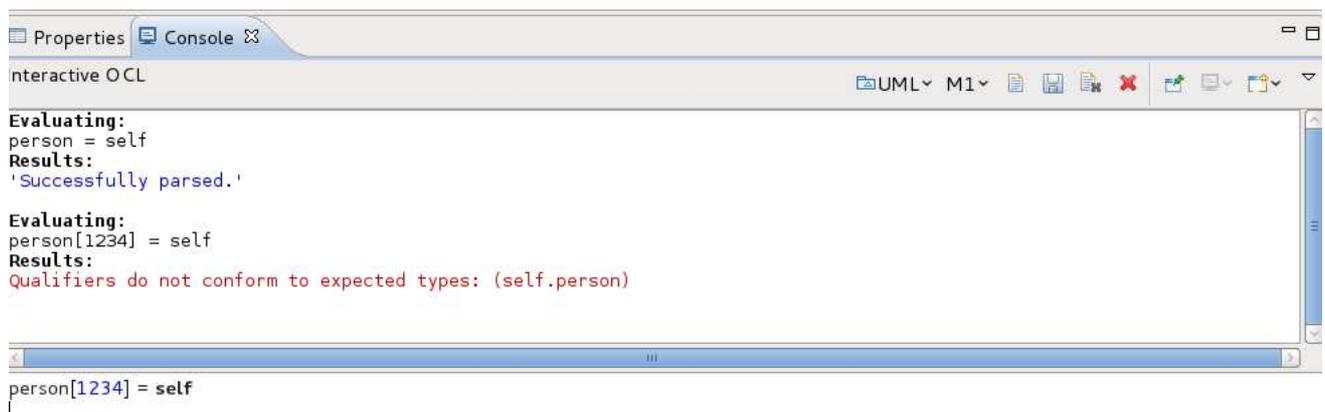
context Bank

inv: self.customer->exists(name='Otto')

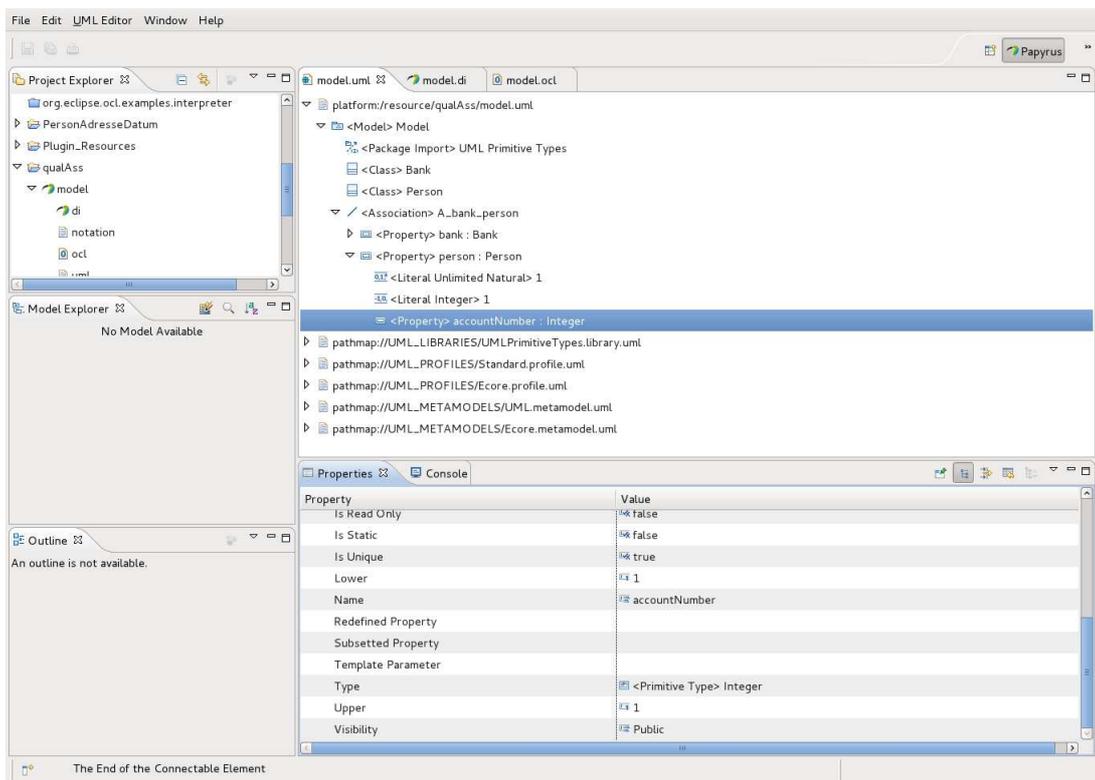
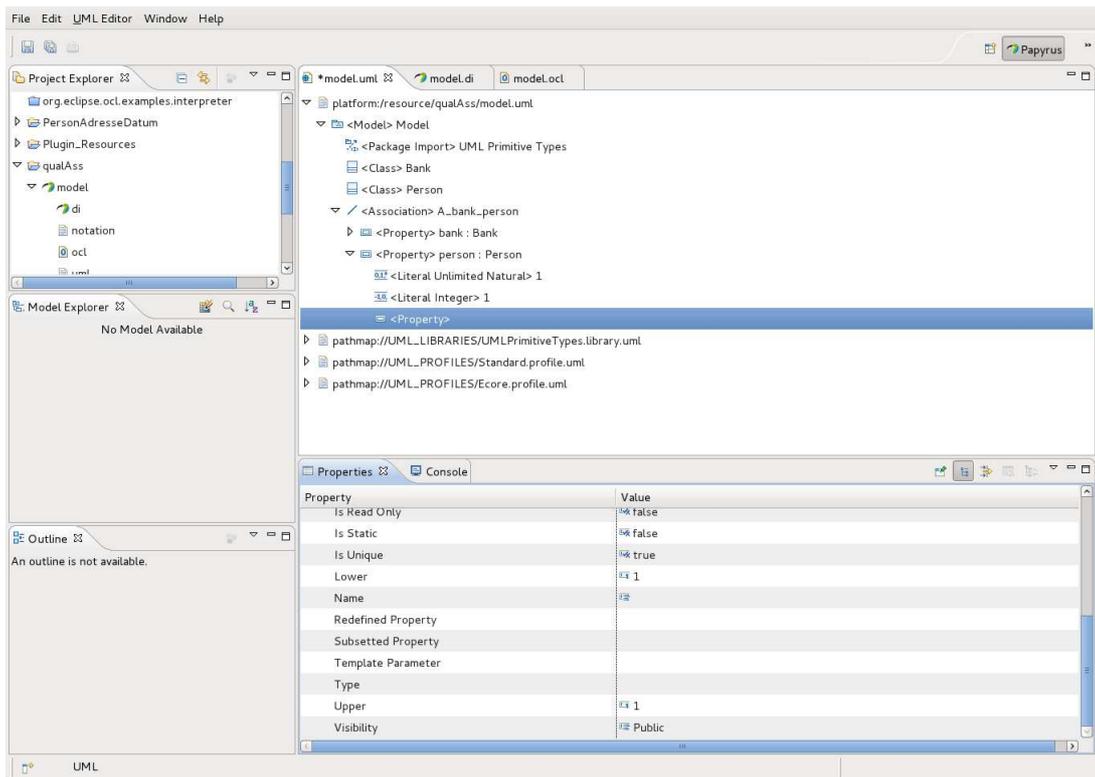
context Bank

inv: customer->select(accountNumber = 145).name = **Bag**{ 'Maier' }

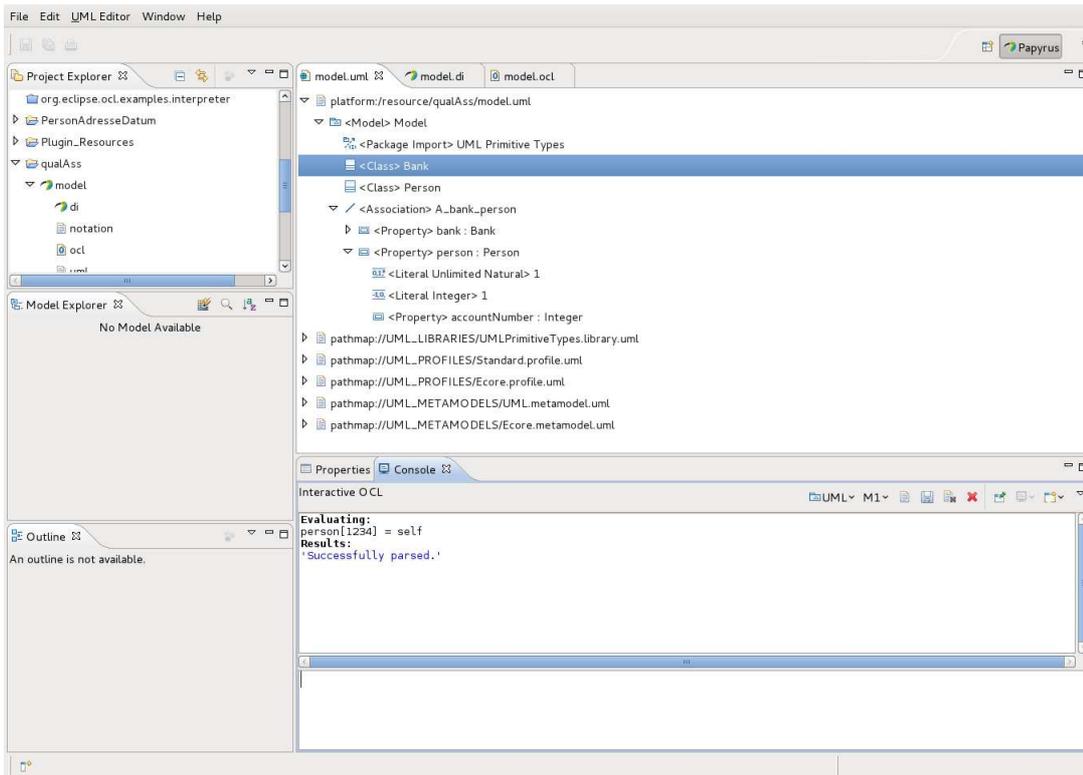
Der Papyrus-Diagrammeditor unterstützt zur Zeit qualifizierte Assoziationen noch nicht: Sie werden weder im Diagramm model.di angezeigt noch kann man Assoziationsenden als qualifiziert navigierbar spezifizieren! Ocl-Ausdrücke mit Qualifiern führen zu Fehlermeldungen:



Der uml-Editor erlaubt es jedoch, Assoziationsrollen mit Qualifier-Namen und -Typ qualifizierbar zu machen, wenn Sie dem Rollennamen mittels **New Child**, **Qualifier**, **Property** die Qualifizierung über eine Property spezifizieren, dann Qualifier-Namen und -Typ angeben



um dann schließlich entsprechende qualifizierte OCL-Ausdrücke benutzen zu können:



2.6 Andere Methoden für die Collections Bag/Set

(im context Flugzeug)

Beispiele für Flugnummern:

$\underbrace{fluege \rightarrow asSet()}_{Set(Flug)} \rightarrow one(flugnr = 123)$

Boolean

true \iff

Es existiert genau ein Flug mit der flugnr 123

$\underbrace{fluege \rightarrow asSet()}_{Set(Flug)} \rightarrow isUnique(flugnr)$

Boolean

true \iff

Jeder Flug hat eine andere flugnr als jeder andere Flug

fluege→ size()	UnlimitedNatural
fluege→ isEmpty()	Boolean
fluege→ notEmpty()	Boolean
fluege→ includes(Berlin_NewYork)	Boolean
↑ von Typ Flug	
fluege→ includesAll(Berlin_NN)	Boolean
↑ von Typ set(Flug)	
fluege→ excludes (Berlin_NewYork)	Boolean
fluege→ excludesAll (Berlin_NN)	Boolean
fluege.verfuegbarePlaetze()→ sum()	Integer, Real, ...
=	Boolean
<>	Boolean
-	Differenzmenge
a → intersection (b)	Durchschnitt
a → union(b)	Vereinigungsmenge
a → symmetricDifference(b)	Menge aller Elemente in A oder B, aber nicht in beiden mathematisch: $(A \cup B) \setminus (A \cap B)$
flatten()	rekursives Entpacken von verschachtelten Mengen

Tabelle 2.2: Methoden für die Collection Set

2.7 Schleifen und Iteratoren

(im **context** Flugzeug)

fluege→ exists(flugnr=12)	Boolean
fluege→asSet().passagiere→ one(name='Meier')	Boolean
fluege→ forAll (verfuegbarePlaetze() <= 3)	Boolean
fluege→ select(verfuegbarePlaetze() > 0)	Collection
fluege→ reject (verfuegbarePlaetze() < 10)	Collection
fluege→ any(verfuegbarePlaetze() > 0)	ein Element
fluege.verfuegbarePlaetze()	Bag{1,3,3,10, ...}
fluege→ collect (verfuegbarePlaetze()) (collectNested (expr))	Bag{1,3,3,10, ...}
fluege→asSet()→isUnique (flugnr)	Boolean
fluege→asSet()→sortedBy (flugnr)	liefert: orderedSet oder Sequences
Set{1,2,3}→ iterate(i;sum:Integer=0 sum+i)	
Set{1,2,3}→ iterate(i:Integer; sum:Integer = 0 sum+i)	

Tabelle 2.3: Schleifen und Iteratoren

2.8 Andere Collections

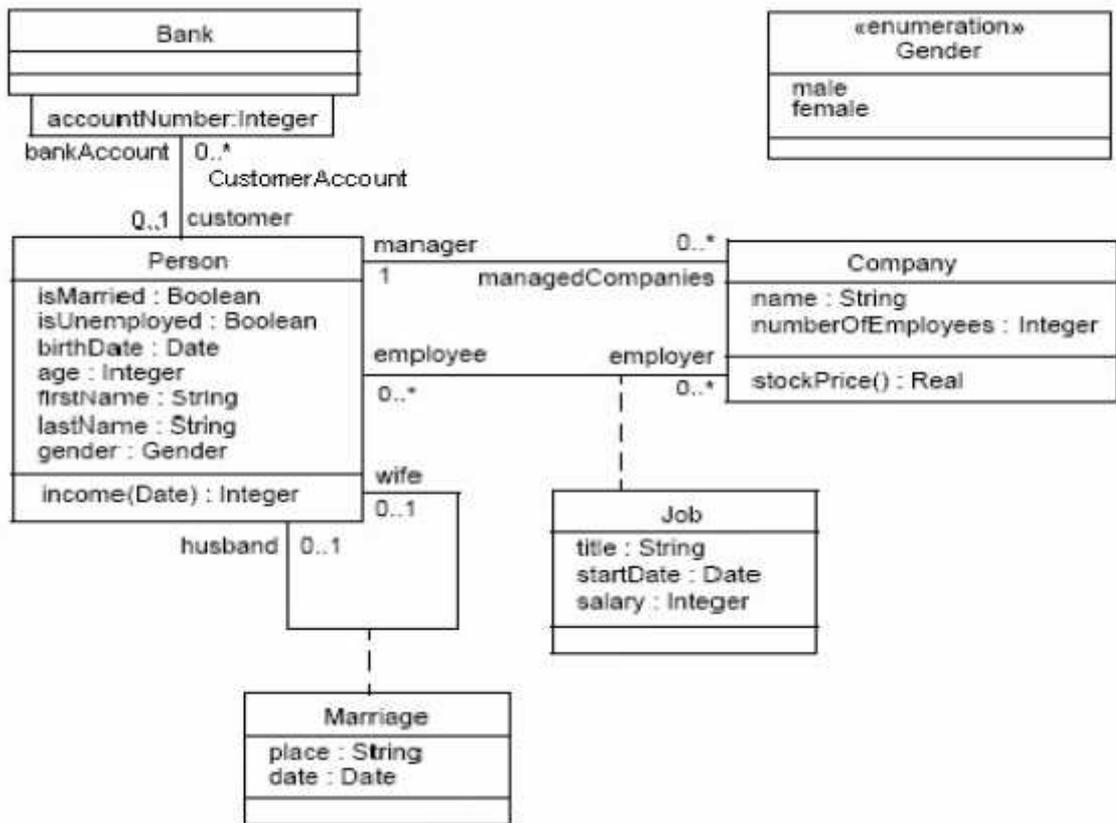
	non ordered	ordered
Element kann nur einfach vorhanden sein	Set	OrderedSet
Element kann mehrfach vorhanden sein	Bag	Sequence

OPERATION	SET	ORDEREDSET	BAG	SEQUENCE
=	X	X	X	X
<>	X	X	X	X
-	X	X	-	-
append(object)	-	X	-	X
as Bag()	X	X	X	X
asOrderedSet()	X	X	X	X
asSequence()	X	X	X	X
asSet()	X	X	X	X
at(index)	-	X	-	X
excluding(object)	X	X	X	X
first()	-	X	-	X
flatten()	X	X	X	X
including(object)	X	X	X	X
indexOf(object)	-	X	-	X
insertAt(index, object)	-	X	-	X
intersection(coll)	X	-	X	-
last()	-	X	-	X
prepend(object)	-	X	-	X
subOrderedSet(lower, upper)	-	X	-	-
subSequence(lower, upper)	-	-	-	X
symetricDifference(coll)	X	-	-	-
union(coll)	X	X	X	X

Tabelle 2.4: Collection Operationen mit verschiedenen Bedeutungen

Die genaue Bedeutung von zum Beispiel `excluding(object)`, `indexOf(object)`, `count(object)`, `intersection(coll)`, `prepend()` und `symmetricDifference(coll)` lesen Sie bitte im Handbuch <http://www.omg.org/spec/OCL/2.3.1/PDF> (Nachbedingungen der Collection-Operationen) nach!

2.9 Together und automatische Code-Erzeugung



(Aus: <http://www.omg.org/spec/OCL/2.3.1/PDF>)

Listing 2.1: Klasse Bank mit qualifizierter Assoziation

```

/*  Generated by Together  */

# ifndef BANK_H
# define BANK_H

class Bank {
private:

    /**
     * @associates Person
     * @supplierQualifier customer
     * @clientCardinality 0..1
     * @supplierCardinality 0..*
     * @undirected
     * @bidirectional Person#bank
  
```

```

    * @clientQualifier bank
    */
    integer accountNumber;
    map < integer , Person * > customer;
};
#endif //BANK_H

```

Listing 2.2: Enumeration Gender

```

/*    Generated by Together    */

# ifndef GENDER_H
# define GENDER_H

/** @stereotype enumeration */
enum Gender {
    male, female
};
#endif //GENDER_H

```

Listing 2.3: Klasse Person

```

/*    Generated by Together    */

# ifndef PERSON_H
# define PERSON_H
class Bank;
class Company;

class Person {
public:

    Integer income(Date);

private:
    Boolean isUnemployed;
    Date birthDate;
    Integer age;
    String firstName;
    String lastName;

    /**
     * @clientCardinality 1
     * @link association
     */

```

```

Gender sex;
Boolean isMarried;
    /**
     * @supplierCardinality 0..*
     * @supplierQualifier managedCompanies
     * @clientCardinality 1
     * @clientQualifier manager
     * @undirected
     * @bidirectional Company#manager
     */
//Company * linkCompany;
vector < Company * > managedCompanies;

    /**
     * @supplierCardinality 0..*
     * @supplierQualifier employer
     * @clientCardinality 0..*
     * @supplierQualifier employee
     * @associationAsClass Job
     * @undirected
     * @bidirectional Company#employee
     */
vector < Company * > employer;

    /**
     * @bidirectional Person#wife
     * @clientCardinality 0..1
     * @supplierQualifier husband
     * @supplierQualifier wife
     * @associationAsClass Marriage
     */
Person * husband;
/** bidirectional */
Person * wife;

Person * husband;
/** bidirectional */
Bank * bank;
};
#endif //PERSON_H

```

2.10 Fallstudie: Person/Haus/Hypothek

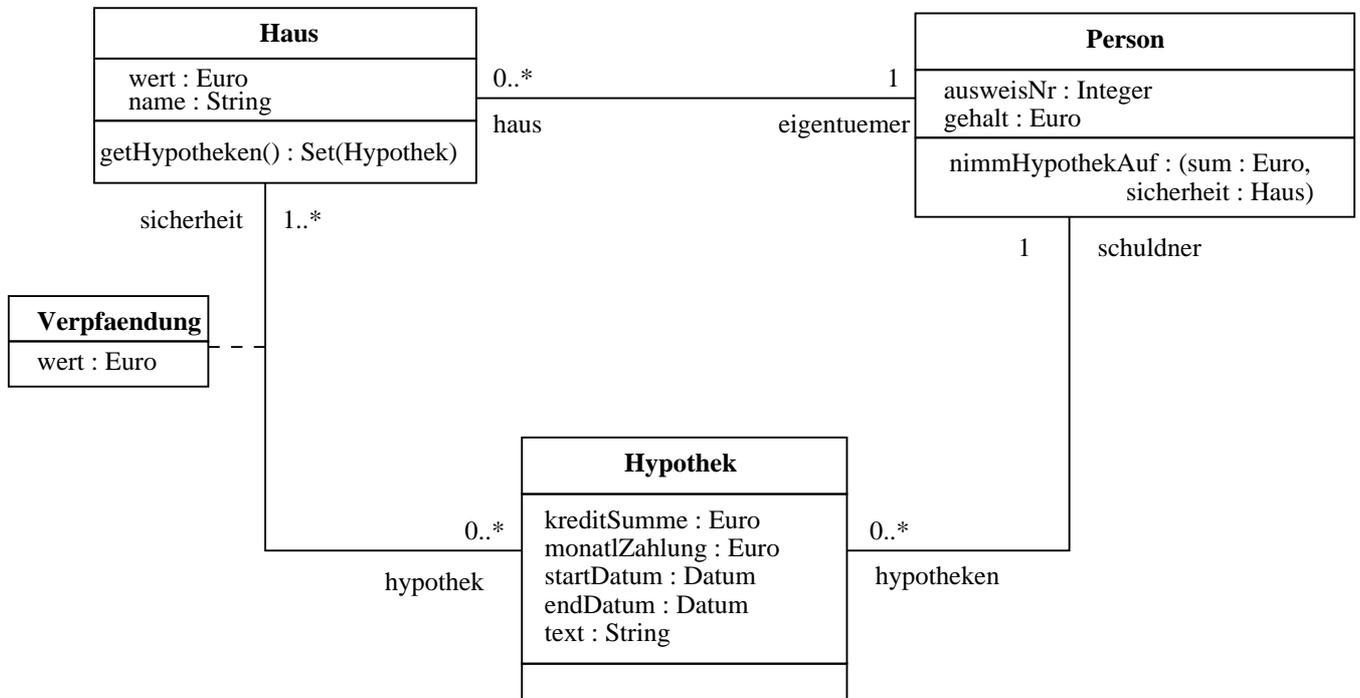


Abbildung 2.8: Klassendiagramm Hypothek

```

context Hypothek
inv: sicherheit.eigentuemer=Bag{schuldner}

context Hypothek
inv: startDatum < endDatum

context Person
def: anzahlHypotheke : Integer =  $\underbrace{\underbrace{\text{haus} . \text{hypotheke}}_{\text{Bag}(\text{Haus})}}_{\text{Bag}(\text{Hypothek})} \rightarrow \text{asSet}() \rightarrow \text{size}()$ 
  
```

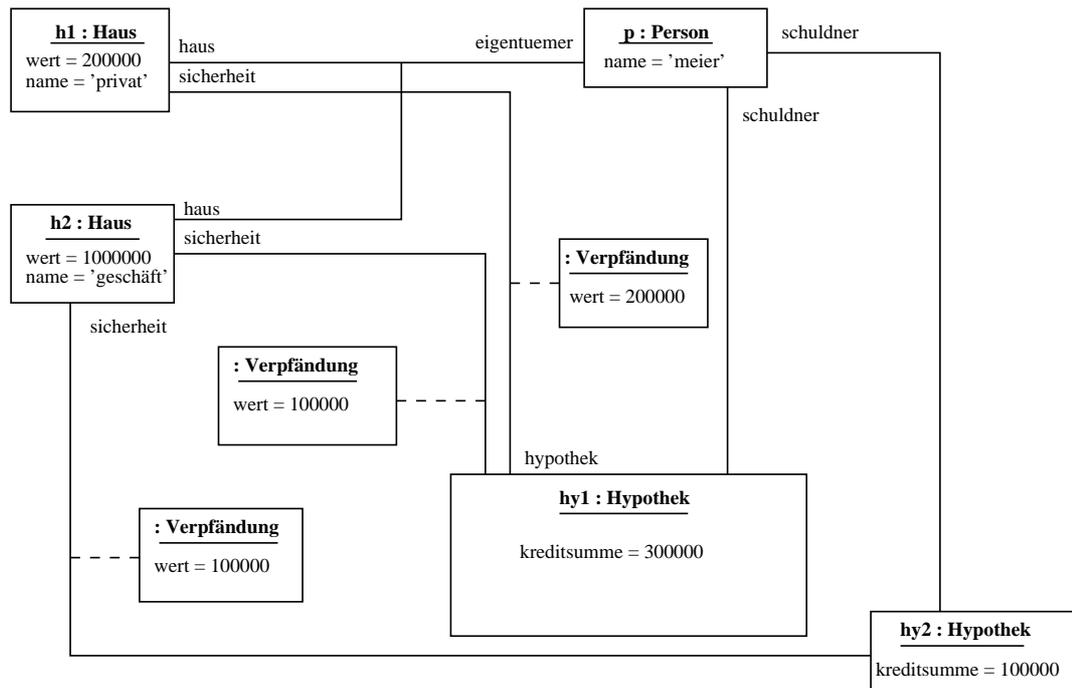
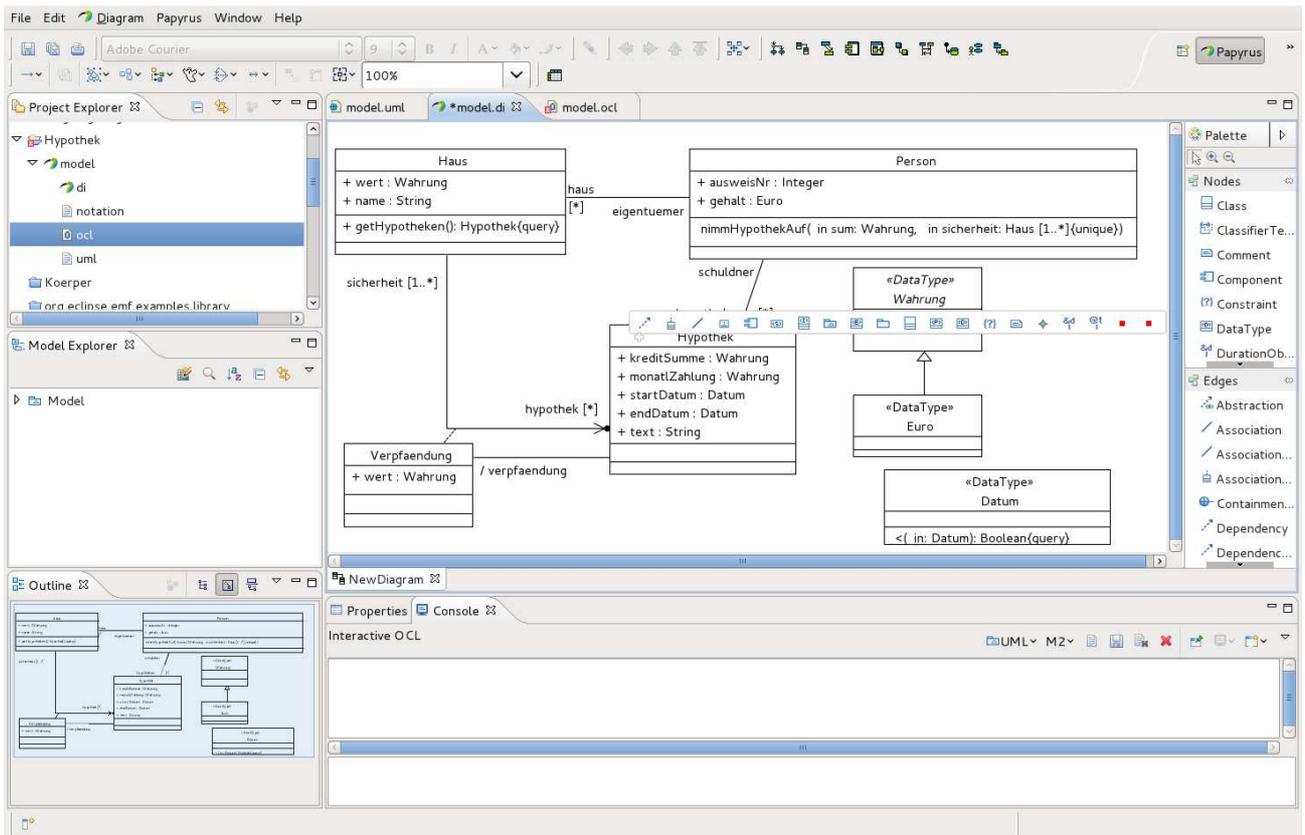
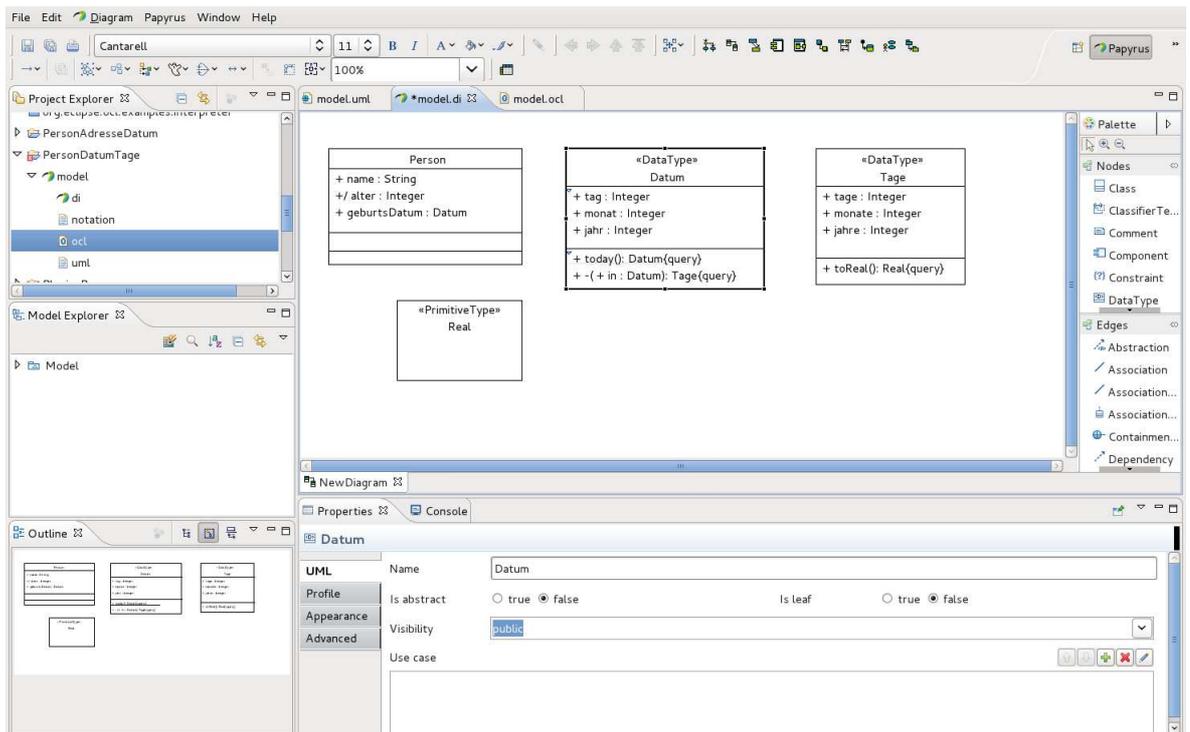


Abbildung 2.9: Hypothek mit zwei Häusern

$p.haus = \text{Bag}\{h1, h2\}$ mit $h1.hypothek = \text{Bag}\{hy1\}$
 $h2.hypothek = \text{Bag}\{hy1, hy2\}$
 $p.haus.hypothek = \text{Bag}\{hy1, hy1, hy2\}$
 $p.haus.hypothek \rightarrow \text{as Set}() = \{hy1, hy2\}$



2.11 Einige erste Hilfskomponenten



The screenshot shows the Papyrus OCL editor interface. The main workspace displays the following OCL constraints:

```
import 'model.uml'  
  
context Person  
inv alterGultig: alter >= 0  
  
context Person  
inv nameNichtleer: name->notEmpty()  
  
context Person  
inv alterKonsistent: let aktAlter: Tage = (Datum::today() - geburtsDatum) in  
aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1
```

The Properties view at the bottom shows a table with the following columns:

Property	Value

Papyrus - DatumTagePerson/DatumTagePerson.d12 - Papyrus

File Edit View Navigate Search Project Run Window Help

Navigator

- DatumTagePerson
 - FlugPersonKlugzeug
 - Meeting
 - OCIL2
 - PersHausHypo
 - .project
 - PersHausHypo.d12
 - PersHausHypo.uml
 - PersonFirma
 - PersonFirma2
 - Ue2

DefaultDiagram

Properties

DatumTagePerson::Person

General

Profile

Comments

Constraints

Appearance

Advanced

Constraints:

- (?) alterGueltig -> <Class> Person
- (?) nameNichtleer -> <Class> Person
- (?) alterKonsistent -> <Class> Person

Level M1 Language OCL Save

```
let aktAlter : Tage = (Datum::today() - geburtsDatum) In
aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1
```

Evaluate

Successfully parsed.

Palette

- Select
- Marquee
- UML Links
- Dependency
- Generalization
- Containment
- Connection
- Association
- UML Elements
- Packages
- Class
- Component
- Property
- Data Type
- Templates

«dataType»

Person

```
name: String [1]
/ alter: Integer [1]
geburtsDatum: Datum...
```

«dataType»

Datum

```
tag: Integer [1]
monat: Integer [1]
jahr: Integer [1]
today(): Datum
-(in Datum): Tage
```

«dataType»

Tage

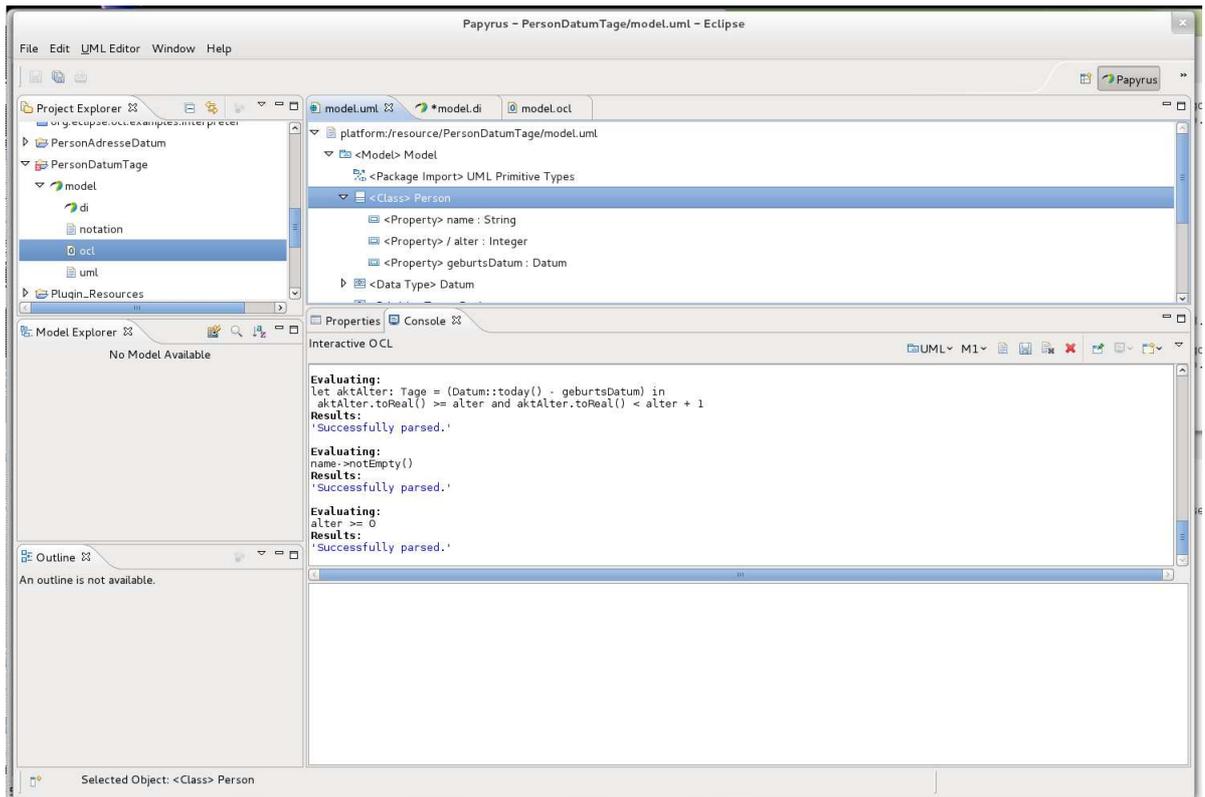
```
tage: Integer [1]
monate: Integer [1]
jahre: Integer [1]
toReal(): Real
```

«dataType»

Real

```
>=(in Integer): Bo...
<(in Integer): Bool...
```

Birdview



Listing 2.4: OCL-Constraints Datum

```

context Datum

inv tagGueltig: tag >= 1 and tag <= 31
inv: monat >=1 and monat <= 12
inv: jahr >= 1600 and jahr <= 2500

```

Listing 2.5: OCL-Constraints Person

```

context Person

inv: alter >= 0
inv: alter < 200
inv: name <> ''
inv: let aktAlter : Tage = (Datum::today() - geburtsDatum) in
    aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1

```

und ihre Benutzung in (alten) Papyrus:

DatumTagePerson::Datum

General | Profile | Comments | **Constraints**

Constraints: [down] [up] [plus] [cross]

- tagGueltig -> <Data Type> Datum
- monatGueltig -> <Data Type> Datum
- jahrGueltig -> <Data Type> Datum

Level: M1 | Language: OCL

```
tag >= 1 and tag <= 31
```

DatumTagePerson::Datum

General | Profile | Comments | **Constraints**

Constraints: [down] [up] [plus] [cross]

- tagGueltig -> <Data Type> Datum
- monatGueltig -> <Data Type> Datum
- jahrGueltig -> <Data Type> Datum

Level: M1 | Language: OCL

```
monat >= 1 and monat <= 12
```

DatumTagePerson::Datum

General | Profile | Comments | **Constraints**

Constraints: [down] [up] [plus] [cross]

- tagGueltig -> <Data Type> Datum
- monatGueltig -> <Data Type> Datum
- jahrGueltig -> <Data Type> Datum

Level: M1 | Language: OCL

```
jahr >= 1600 and jahr <= 2500
```

DatumTagePerson::Person

General | Profile | Comments | **Constraints**

Constraints: [down] [up] [plus] [cross]

- alterGueltig -> <Class> Person
- nameNichtleer -> <Class> Person
- alterKonsistent -> <Class> Person

Level: M1 | Language: OCL

```
alter >= 0 and alter < 200
```

DatumTagePerson::Person

General | Profile | Comments | **Constraints**

Constraints: [down] [up] [plus] [cross]

- alterGueltig -> <Class> Person
- nameNichtleer -> <Class> Person
- alterKonsistent -> <Class> Person

Level: M1 | Language: OCL | Save

```
let aktAlter : Tage = (Datum::today() - geburtsDatum) in  
aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1
```

2.12 Alle Instanzen einer Klasse: allInstances()

Die Methode

```
allInstances() : Set(T)
```

liefert alle (endlich viele) Instanzen einer User-definierten Klasse. Sie darf nicht benutzt werden für String, Integer und Real.

Ein Beispiel:

```
context Person
inv uniqueNames: Person.allInstances()->forall(p1, p2 |
    p1 <> p2 implies p1.name <> p2.name)
```

Alternativ:

```
context Person
inv uniqueNames: Person.allInstances()->isUnique(name)
```

Siehe auch::

<http://www.springerlink.com/content/j8g72037gn63t1h6/>

Damit ist unsere mydictionary-Spezifikation vollständig realisierbar:



The screenshot shows a software tool interface with a sidebar on the left containing tabs for 'General', 'Profile', 'Comments', and 'Constraints'. The 'Constraints' tab is selected, displaying a list of constraints. The main area shows a constraint definition: 'KEY.allInstances()->forall(kl | has@pre(kl) implies (kl=k or value_for@pre(kl)=value_for(kl)))'. Below the constraint, there is a status bar that says 'Successfully parsed.' and an 'Evaluate' button.

allInstances() macht aus einer Klasse eine Collection von (existierenden) Objekten dieser Klasse.

OCL Query Examples

These query examples are pure OCL only, they need to be embedded into an OCL query!

Find all instances of Patient

```
Patient.allInstances()
```

Get the number of instances of Patient

```
Patient.allInstances()->size()
```

Get all instances of Patient where nachname is exactly Descher

```
Patient.allInstances()->select(p:Patient|p.nachname='Descher')
```

Get all instances of Patient where nachname starts with G

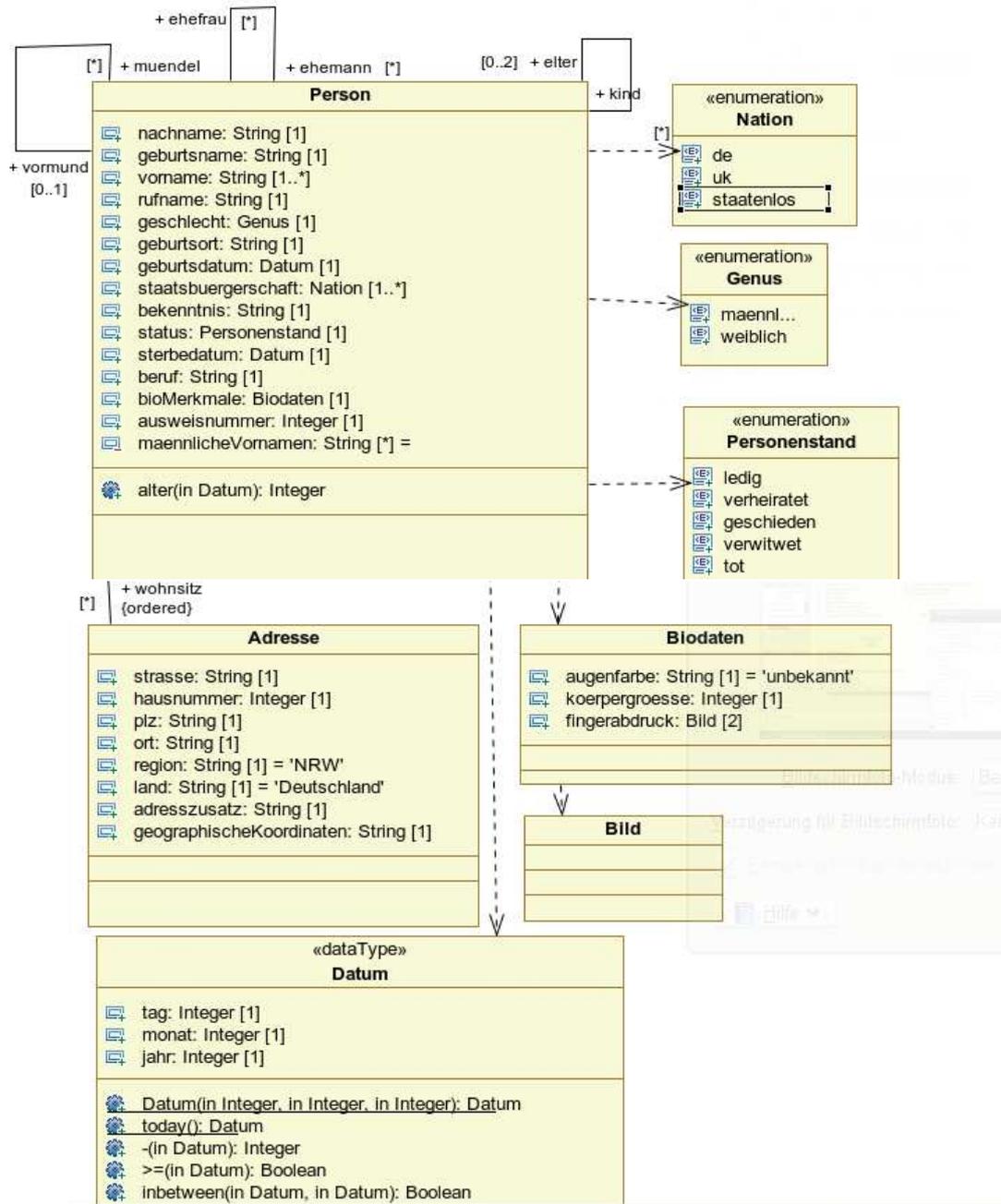
```
Patient.allInstances()->select(p:Patient|p.nachname.substring(1,1)='G')
```

Get the dates of all Konsultations

```
Konsultation.allInstances().datum
```

aus: [allInstances-Query](#)

2.13 Fallstudie Personenstandsdaten, Hilfsklassen: Adresse, BioDaten, Datum, Personenstand, Nation, Genus, Bekenntnis



Listing 2.6: Constraints Adresse

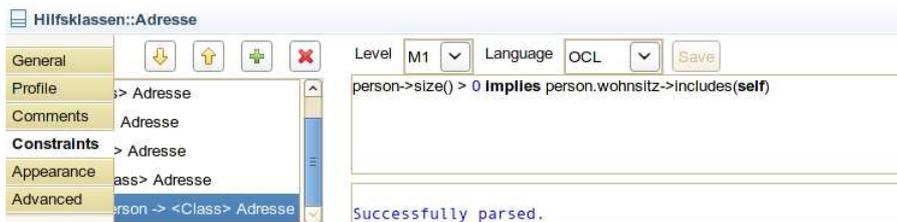
```

context Adresse
inv : hausnummer > 0
inv : land <> ''
inv : ort <> ''
inv : Land = 'Deutschland' implies Set{'Berlin', 'Wuppertal',
...}->includes(ort)
inv : plz.toInteger() > 0 and plz.toInteger() <100000
    and plz.size()=5
inv : strasse <> ''
inv : person->size() > 0 implies person.wohnsitz->includes(self
)

context Adresse::land
init : 'Deutschland'

context Adresse::region
init : 'NRW'

context Adresse::person
init : Bag{}
```



Listing 2.7: Constraints Biodaten

```

context BioDaten
inv: augenfarbe <> ''
inv: Set{'bernsteinfarben', 'braun', 'gruen', 'grau', '
    graugruen', 'blaugruen', 'blau', 'blaugrau', 'unbekannt'}->
    includes(augenfarbe)
context BioDaten::augenfarbe
init : 'unbekannt'
```

Listing 2.8: Constraints Datum

```

context Datum — virtuelle Methode = Hilfsmethode / OclHelper
static def : gueltigesDatum( t : Integer, m : Integer, j :
  Integer ) : Boolean =
  1920 <= j and j <= 2500 and
  1 <= m and m <= 12 and
  1 <= t and
  t <=
    if      Set{4, 6, 9, 11}->includes(m)
    then 30
    else if Set{1, 3, 5, 7, 8, 10, 12}->includes(m)
    then 31
    else if ((j.mod(4)=0) and
              not(j.mod(100)=0)) or
              (j.mod(400)= 0)
    then 29 — Schaltjahr
    else 28

    endif
    endif
    endif

context Datum
inv : gueltigesDatum(tag, monat, jahr)

context Datum::Datum( t : Integer, m : Integer, j : Integer ) :
  Datum
pre : gueltigesDatum(t, m, j)
post : result.oclIsNew()
post : result.tag = t
post : result.monat = m
post : result.jahr = j

/* Problem context Datum::-(d:Datum):Integer wird nicht
  akzeptiert , deshalb Workaround: */
context Datum::minus( d : Datum ) : Integer — in vollen Jahren
  (Alter, ...)
pre : self >= d
post : if monat > d.monat then result = jahr - d.jahr
    else if monat < d.monat then result = jahr -d.jahr - 1
    else — monat = d.monat
      if tag >= d.tag then result = jahr - d.jahr
      else result = jahr - d.jahr - 1
    endif

```

```

    endif
    endif

/* analoges Workaround; */
context Datum::groesserGleich(d:Datum) : Boolean
post : if jahr > d.jahr then result = true
      else if jahr < d.jahr then result = false
      else — jahr = d.jahr
          if monat > d.monat then result = true
          else if monat < d.monat then result = false
          else — monat = d.monat
              if tag >= d.tag then result = true
              else result = false
              endif
          endif
      endif
  endif
endif

context Datum::inbetween( from : Datum, to : Datum ) : Boolean
pre : to >= from
body : (to >= self) and (self >= from)

context Datum — virtuelles Attribut = Hilfsattribut
static def : invalidDatum : Datum =
    Datum::Datum(31, 12, 2500) — 1. Versuch eines opt. DTs

— und jetzt ein tagesgenaues minus()
— mit Hilfe von de.wikipedia.org/wiki/Julianisches_Datum
class Datum
static def: toChronoJD(t : Integer,
                      m : Integer,
                      j : Integer) : Integer =
    let y : Real = j + (m - 2.85) / 12 in
    let A : Real = (367 * y).floor() - 1.75 * y.floor() + t
    in
    let B : Real = A.floor() - 0.75 * (y / 100).floor() in
    B.floor() + 1721115

context Datum::-(d : Datum): Integer
body: toChronoJD(tag, monat, jahr) - toChronoJD(d.tag, d.monat,
d.jahr)

— und fuer geschaeftliche Anwendungen:

```

```

context Datum
static def: minusZinstage(d : Datum): Integer = — nach der E30
  /360–Methode
    (jahr – d.jahr) * 360 +
    (monat – d.monat) * 30 +
    (tag.min(30) – d.tag.min(30))

```

Für eine analoge Methode `fromChronoJD`(in `jd : Integer`) benötigt man als Ergebnistyp Tripel (vgl. OCL-Tuple in Abschnitt 2.19 beziehungsweise im OCL-Manual Seite 17 und 26f.).

Listing 2.9: Constraints Person

```

context Person
inv : elter->size() <=2
inv : ausweisnummer >= 0 /* 0 bei fehlendem Ausweis */
inv : wohnsitz->size() > 0 implies wohnsitz.person->includes(
  self)
inv : muendel->notEmpty() implies muendel.vormund->asSet() =
  Set{self}
inv : vormund->notEmpty() implies vormund.muendel->includes(
  self)
inv : elter->notEmpty() implies elter.kind->includes(self)
inv : kind->notEmpty() implies kind.elter->includes(self)
inv : ehemann->notEmpty() implies ehemann.ehefrau->includes(
  self)
inv : ehefrau->notEmpty() implies ehefrau.ehemann->includes(
  self)
inv : geschlecht=Genus::maennlich implies ehemann->isEmpty()
inv : geschlecht=Genus::weiblich implies ehefrau->isEmpty()
inv : elter->size() = 0 implies not vormund.oclIsUndefined()
context Person
def: maennlicheVornamen : Set(String) =
  Set{'Hans', 'Georg' /* bitte ergaenzen */}
def: weiblicheVornamen : Set(String) =
  Set{'Heidrun', 'Cornelia' /* bitte ergaenzen */}
context Person
static def: istMaennlicherVorname(s : String) : Boolean =
  maennlicheVornamen->includes(s)
def: istWeiblicherVorname() : Boolean =
  weiblicheVornamen->includes(self.vorname)
context Person
inv : elter->size() > 0 implies elter->forall( e | (
  geburtsdatum – e.geburtsdatum) >= 8)

```

```

inv : elter->size() = 1 implies elter->forall( e | nachname = e
    .nachname)
inv : elter->size() = 2 implies
    elter->forall( e | if e.geschlecht = Genus::maennlich
        then
            e.hochzeit[ehemann]->size() >
                0 implies
            e.hochzeit[ehemann]->forall(
                h |
                    geburtsdatum.inbetween(
                        h.hochzeitsdatum, h.
                        scheidungsdatum)
                    implies nachname = h.
                        familienname
                )
            else
                e.hochzeit[ehfrau]->size() >
                    0 implies
                e.hochzeit[ehfrau]->forall(
                    h |
                        geburtsdatum.inbetween(
                            h.hochzeitsdatum, h.
                            scheidungsdatum)
                        implies nachname = h.
                            familienname
                    )
            endif
        )
    )

```

context Person

```

def: keineZeitweiseBigamie( s : Set(Hochzeit) ) : Boolean =
    s->forall( h1, h2 | h1 <> h2 implies
        (/* h1.scheidungsdatum >= */ h1.hochzeitsdatum >=
            h2.scheidungsdatum /* >= h2.hochzeitsdatum */
            or
            /* h2.scheidungsdatum >= */ h2.hochzeitsdatum >=
                h1.scheidungsdatum /* >= h1.hochzeitsdatum */
        )
    )

```

context Person

```

inv : geschlecht = Genus::maennlich implies
    istMaennlicherVorname(vorname)

```

```

inv : geschlecht = Genus::maennlich implies hochzeit [ehemann]->
    size() > 0 implies keineZeitweiseBigamie (hochzeit [ehemann
    ]->asSet())
inv : geschlecht = Genus::weiblich implies istWeiblicherVorname
    ()
inv : geschlecht = Genus::weiblich implies hochzeit [ehefrau]->
    size() > 0 implies keineZeitweiseBigamie (hochzeit [ehefrau]->
    asSet())
inv : status = Personenstand::tot implies sterbedatum <> Datum
    ::invalidDatum
inv : status <> Personenstand::tot implies sterbedatum = Datum
    ::invalidDatum

context Person::alter ( amDatum : Datum ) : Integer
pre : amDatum >= geburtsdatum
post : result = amDatum - geburtsdatum

context Person::status
init : Personenstand::ledig

context Person::staatsbuergerschaft
init : Nation::de

context Person::ausweisnummer
init : 0

context Person
inv : geschlecht=Genus::maennlich implies
    hochzeit [ehemann]->select (scheidungsdatum=Datum::
    invalidDatum)->size() <= 1

context Person
inv : geschlecht=Genus::weiblich implies
    hochzeit [ehefrau]->select (scheidungsdatum=Datum::
    invalidDatum)->size() <= 1

context Person::nachname
init : if elter ->size() = 1 then
    elter ->any(true).nachname
else if elter ->size() = 2 then
    let aktuelleEheDerMutter : hochzeit =
    elter ->any (geschlecht=Genus:: weiblich) . hochzeit [
    ehefrau]->select (scheidungsdatum=Datum::

```

```

        invalidDatum)->any(true) in
    let aktuellerEhemannDerMutter : Person =
        aktuelleEheDerMutter .ehemann in
    if elter ->includes(aktuellerEhemannDerMutter) then
        aktuelleEheDerMutter .familienname
    else
        elter ->any(geschlecht=Genus::weiblich) .nachname
    endif
else — keine Eltern
    'NameVonGericht'
endif
endif

```

```

/* Verwandschaftsbeziehungen: – allgemeine implizite
   Voraussetzungen: Existenz der jeweiligen Verwandten: */

```

```

context Person

```

```

def : istNeffe( n : Person ) : Boolean =
    elter .kind->asSet()->excluding(self) .kind->includes(n) and n
    .geschlecht=Genus::maennlich

```

```

def : Cousins() : Set(Person) =
    (elter .elter .kind->asSet() - elter) .kind->asSet()->select(
        geschlecht=Genus::maennlich)

```

```

def : istStiefvater( s : Person ) : Boolean =
    let mutter : Person = elter ->any(geschlecht=Genus::weiblich)
    in
        (s .geschlecht=Genus::maennlich) and
        (mutter .ehemann->asSet() - (elter ->select(geschlecht=Genus::
            maennlich))) ->includes(s) and
        (s .hochzeit [ehemann]->select(ehefrau=mutter)->select(
            scheidungsdatum >= Datum::today()) ->size() > 0)

```

```

context Person::sammleVorfahren( level : Integer ) : Sequence(
    Person)

```

```

pre : level >= 0

```

```

post :

```

```

    if (level = 0) or (elter ->size() = 0) then
        result = Sequence{ }

```

```

    else if elter ->size() = 1 then

```

```

        result = elter ->asSequence()->union(elter ->asSequence()->
            at(1) .sammleVorfahren( level - 1 ))

```

```

    else — elter ->size() = 2

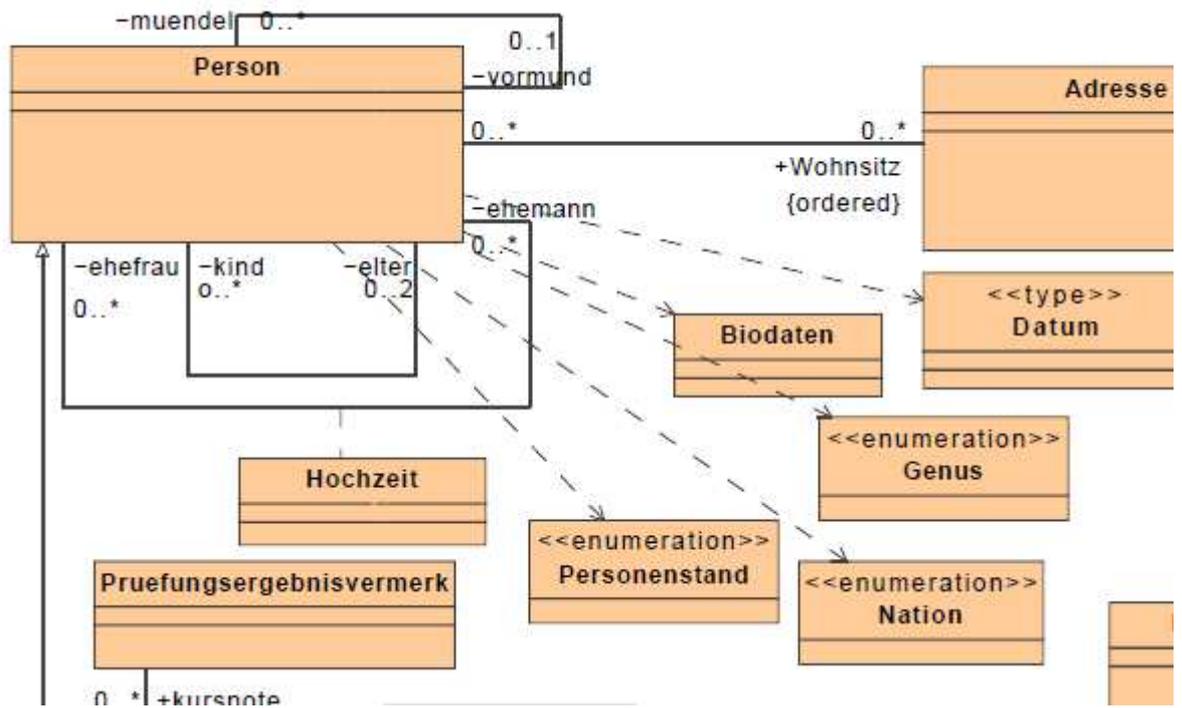
```

```

result = elter ->asSequence() ->union( elter ->asSequence() ->
                                     at(1).sammleVorfahren(
                                         level-1 ) ->
                                     union( elter ->asSequence() ->at
                                         (2).
                                     sammleVorfahren( level-1 ) )
endif
endif

```

Weitere Assoziationen von Person zu Person



und zugehörige Constraints:

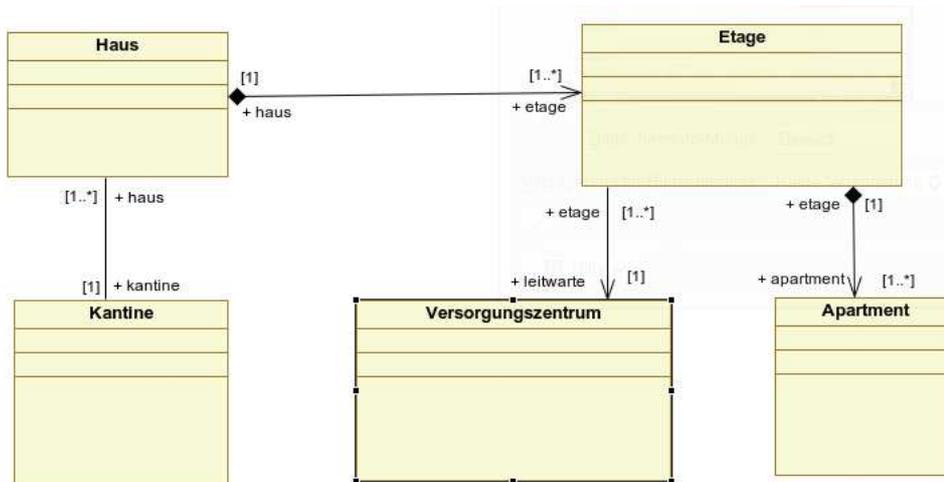
Listing 2.10: Constraints Hochzeit

```
context Hochzeit
inv : familienname <> ''
inv : scheidungsdatum >= hochzeitsdatum
inv : hochzeitsort <> ''
inv : Person[ehemann].alter(hochzeitsdatum) >= 14
inv : Person[ehefrau].alter(hochzeitsdatum) >= 14
inv : Person[ehemann].geschlecht = Genus::maennlich
inv : Person[ehefrau].geschlecht = Genus::weiblich

context Hochzeit::familienname
init : Person[ehemann].nachname

context Hochzeit::scheidungsdatum
init : Datum::invalidDatum
```

2.14 Fallstudie Modell Wohnanlage



Im Modell *Wohnanlage* werden mehrere Apartmenthäuser, Versorgungszentren (Leitwarten) und Kantinen modelliert. Die Assoziation *etage* der Klasse *Haus* sei *{ordered}*. Jedes mit dem Konstruktor *Haus()* erzeugte Exemplar muß gültig sein, also die Invarianten (und hier insbesondere die konzipierten Vielfachheiten des UML-Modells) richtig aufbauen:

Listing 2.11: Constraints *Haus()*

```

context Haus(in k : Kantine,
             in s : Etage[*] {ordered}) : Haus
pre: not k.ocIsUndefined() and
      s->size() > 0
post: result.ocIsNew() and
      result.kantine = k and
      result.etage = s
  
```

ist deshalb der einzige sinnvolle Konstruktor für die Klasse *Haus*. Will man auch den parameterlosen Default-Konstruktor zulassen, sind die Vielfachheiten im UML-Diagramm anders zu spezifizieren (wie?). Welche Nachteile würde aber eine solche Modellierungsalternative mit sich bringen?

Bemerkung: Die Assoziation *apartment* der Klasse *Etage* sollte qualifiziert (Qualifizierer: *Raumnummer: Integer*) sein. Warum? Wie ist das UML-Diagramm dann zu modifizieren?

Haus::addEtage():

- Nach dem Hinzufügen einer Etage **e** zu einem Haus mittels `Haus::addEtage(e : Etage)` enthält dieses Haus **e** und eine Etage mehr als vorher.

```
context Haus::addEtage(e : Etage)
post: etage->includes(e)
post: etage->size() - etage@pre->size() = 1
```

- Formal vollständiger inklusive einer expliziten Framebedingung:

```
context Haus::addEtage(e : Etage)
post: etage = etage@pre->union(e)
```

- Alle vor Anwendung von `addEtage()` existierenden Etagen sind auch danach noch vorhanden.

```
context Haus::addEtage(e : Etage)
post: etage->includesAll(etage@pre)
```

- Eine bereits einem Haus zugeordnete Etage darf keinem (anderen) Haus mehr zugeordnet werden und **e** muß definiert sein.

```
context Haus::addEtage(e : Etage)
pre: not e.ocIsUndefined()
pre: Haus.allInstances().etage->excludes(e)
```

Klassen-Invarianten:

- Kein Haus darf mehr als 10 Etagen besitzen.

```
context Haus
inv: etage->size() <= 10
```

- Jede Etage hat 1..20 Apartments.

```
context Etage
inv: apartment->size() <= 20
```

- Jedes Apartment hat 0..4 Bewohner.

```
context Apartment
inv: bewohner->size() <= 4
      — oder
inv: 0 <= anzBewohner and anzBewohner <= 4
```

- Jede Kantine hat ein redundantes Attribut `anzahlHaeuser`, das die Anzahl der assoziierten Häuser beinhaltet.

```
context Kantine::anzahlHaeuser
derive: haus->size()
```

- Jede Kantine kann höchstens 6 Häuser bedienen.

```
context Kantine
inv: anzahlHaeuser <= 6
```

Systeminvarianten:

- Jedes Apartment besitzt eine Identifikationsnummer `apartmentID`, die in der gesamten Wohnanlage eindeutig ist.

```
context Apartment
inv: Apartment.allInstances()->isUnique(apartmentID)
```

- Jedes Apartment enthält als redundantes Attribut die zugehörige Leitwarte (schnellerer Zugriff bei technischen Problemen).

```
context Apartment::zugLeitwarte
derive: self.etape.leitwarte
```

- Mehrere (verschiedene) Häuser können derselben Kantine zugeordnet sein.

```
context Kantine
inv: haus->size() >= 1
```

- Die Apartments mit Raumnummern kleiner als 20 können höchstens 2 Bewohner aufnehmen.

```
context Apartment
inv: raumnummer < 20 implies bewohner->size() <= 2
```

- Die Anzahl der Bewohner aller einer Kantine zugeordneten Apartments darf 1000 nicht überschreiten.

```
context Kantine
inv: haus.etape.apartment->collect(anzBewohner)->sum() <= 1000
```

Destruktor-Spezifikation:

- Der Destruktor `reisseHausAb()` der Klasse `Haus` darf nur aufgerufen werden, wenn alle zugeordneten Apartments keine Bewohner mehr haben.

```
context Haus::reisseHausAb()
pre: etape.apartment->forAll(anzBewohner = 0)
```

- Nach Aufruf von `reisseHausAb()` existieren die zugehörigen Etagen und Apartments nicht mehr.

```

context Haus :: reisseHausAb ()
post: Etage.allInstances ()->excludesAll (etage@pre)
post: Apartment.allInstances ()->excludesAll (etage@pre .
    apartment@pre)

```

- Nach Aufruf von `reisseHausAb()` existieren alle diejenigen zugehörigen Versorgungszentren nicht mehr, die lediglich für Etagen des abgerissenen Hauses zuständig waren.

```

context Haus :: reisseHausAb ()
post: let allVZs : Set(Versorgungszentrum) =
    etage@pre.leitwarte@pre->asSet ()
    in
    allVZs->forall (vz | vz.etage@pre.haus@pre->size () = 1
    implies Versorgungszentrum.allInstances ()->
    excludes (vz))

```

2.15 Fortsetzung Fallstudie Person/Haus/Hypothek

context Verpfaendung

inv: wert <= hypothek.kreditSumme

inv: wert <= sicherheit.wert

inv: wert >= Euro(0)

context Haus

inv valid_security: hypothek → size() > 0

implies wert >= verpfaendung.wert → sum()

context Hypothek

inv valid_security: verpfaendung.wert → sum() = kreditSumme

context Person

inv: Person.allInstances() → isUnique(ausweisNr)

def: anzHypotheke : Integer = hypotheke → size()

context Person:: nimmHypothekAuf(sum:Euro, sec:Haus)

pre: self.hypotheke.monatZahlung → sum() + sum*0.01 <= self.gehalt*0.30

pre: sec.wert - sec.verpfaendung.wert → sum() >= sum

post: ...

context Haus

.... hypothek → select(monatZahlung > Euro(500))

Bag der Hypotheke auf Haus für die die monatlich Zahlung > 500 Euro ist

2.16 Startwerte von Attributen und Ergebnisse von Observatoren

```
context Hypothek::kreditSumme
init: Euro(0)
```

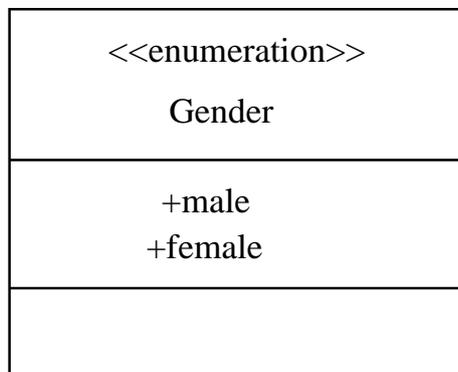
```
context Hypothek::text
init: schuldner.name.concat(':').concat('- - - -')
```

```
context Haus::getHypotheke(n): Set(Hypotheke)
body: hypotheke->asSet()
```

2.17 Virtuelle OCL Variablen / Operationen/OclHelper

```
context Person
def: income :Integer = self.job.salary → sum()
def: nickname :String = 'little Joe'
def: hasTitle (t:String):Boolean = self.job → exists(title=t)
```

2.18 Enumeration



2.19 Tuple (structures)

... als Ergänzung zur **Sequence/Collection**:

Tupel bieten benannte Komponenten:

(1) Tuple {name:String = 'Hans', age:Integer = 20}

(2) Tuple {a: Collection(Integer) = Set{1,3,4},
 b: String = 'foo',
 c: String = 'bar'}

(Tuple-Typangaben sind optional und die Reihenfolge der Tuple-Komponenten ist nicht signifikant!)

(3) Tuple {age = 20, name = 'Hans'}

(3) ist also identisch zu (1)

Neben **Tuple-Literalen** kann man auch Tuple-Typen spezifizieren:
TupleType(name:String, age:Integer)

Benutze etwa den `fromChronoJD()`-Rückgabebetyp
TupleType(t: Integer, m: Integer, j: Integer).

Vergleiche Abschnitt 7.6.15 des OCL-Standards.

2.20 Typ-Konformität

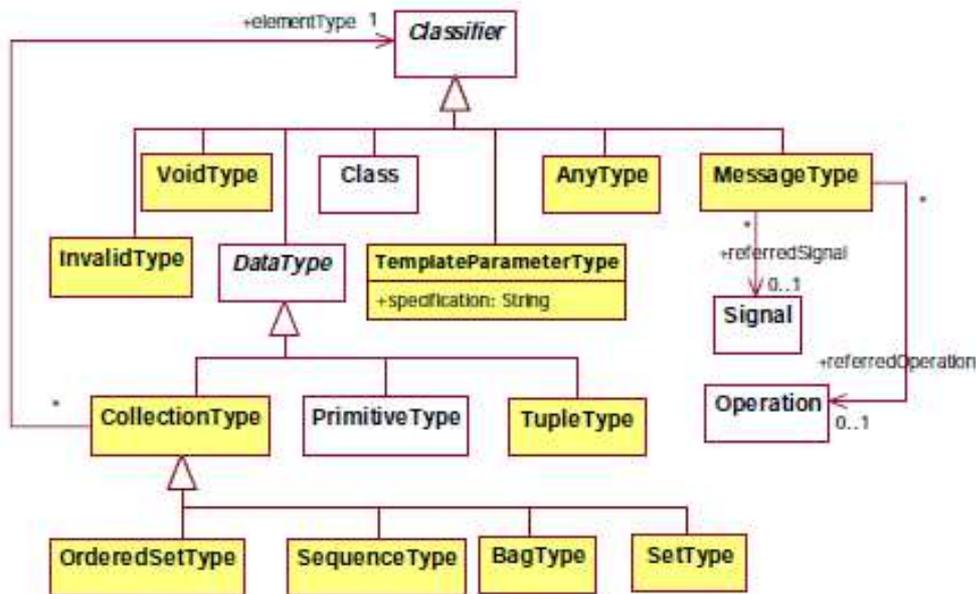
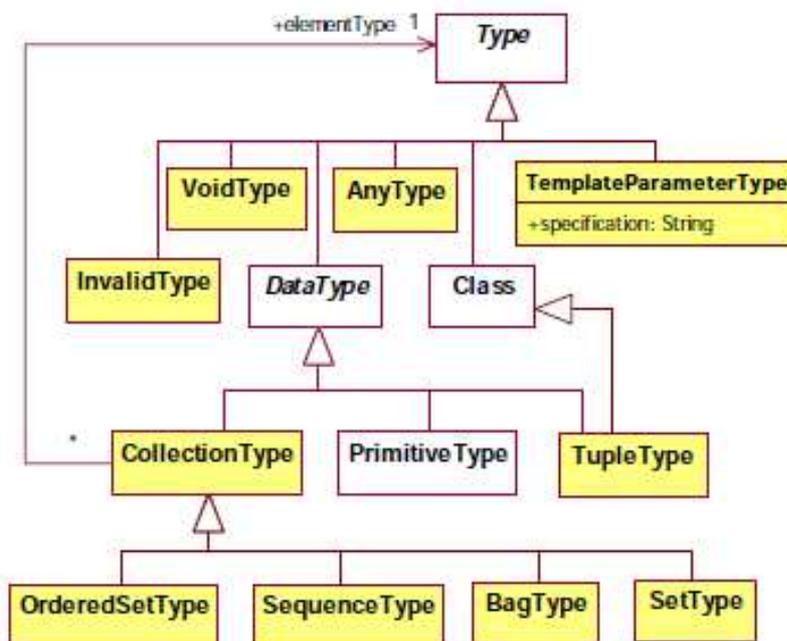


Abbildung 2.10: Die Typen der OCL-Standard-Bibliothek



- Jeder Typ ist konform zu seinen Elter-Typen.
- Die Typenkonformität ist transitiv:

Beispiele:

Integer ist konform zu Real

oclVoid ist konform zu oclAny

Set(T1) ist konform zu Collection(T1)

Set(Integer) ist konform zu Collection(Real)

Sei **obj1** vom Typ OCLAny und es enthalte einen Integer-Wert. Dann erzwingt

`obj1.oclAsType(Integer)`

die Nutzung von `obj1` als Integer.

`oclAsType()` kann nur für Subtypen wandeln.

2.21 Operator-Vorrangsregeln

höchste	()
	@pre
	. →
:	not - (unär)
	* /
↑	+ - (binär)
	if - then - else - endif
:	< > <= >=
	= <>
	and
	or
	xor
	implies
niedrigste	let - in

2.22 oclIsUndefined()

Auf Objekten des Typs OclAny kann mittels

`obj.oclIsUndefined()`

die Existenz abgefragt werden (in Konkurrenz zu `... ->notEmpty()`).

2.23 Vordefinierte Operationen auf OclAny

OclAny = (object : OclAny) : Boolean

True, falls *self* is dasselbe Objekt wie *object* ist.

OclAny <> (object : OclAny) : Boolean

post: result = not (self = object)

Weitere Operationen auf **OclAny**:

```
oclIsTypeOf (t : Classifier) :Boolean
oclIsKindOf (t : Classifier) :Boolean
oclIsNew    ()              :Boolean
oclType     ()              :Classifier
```

2.24 Statusdiagramme in UML

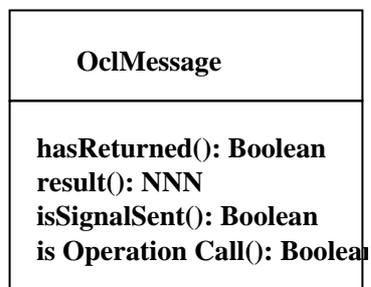
Hinweis:

Strukturdiagramme und wechselnde Stati werden in OCL unterstützt durch (siehe Literatur):

```
state
oclInState(xx)
OclMessage
```

Bemerkung:

OclMessage besitzt folgende Methoden:



2.25 Grundlegende Observatoren bei Existenz von Assoziationen

Zur vollständigen Statusbeschreibung eines Objekts eines Modells *mit Assoziationen* benötigt man grundlegende Observatoren (get-Methoden) auch für die Assoziationen. Am Beispiel des Wohnanlagenmodells (Abschnitt 2.14):

```
Kantine::getHaeuser(): Haus [1...*] {unique}
Haus::getKantine(): Kantine
Haus::getEtagen(): Etage [1..*] {unique, ordered}
Etage::getApartment(apartment_nummer: Integer): Apartment
...
```

Diese können in C++ bei Benutzung der STL mittels

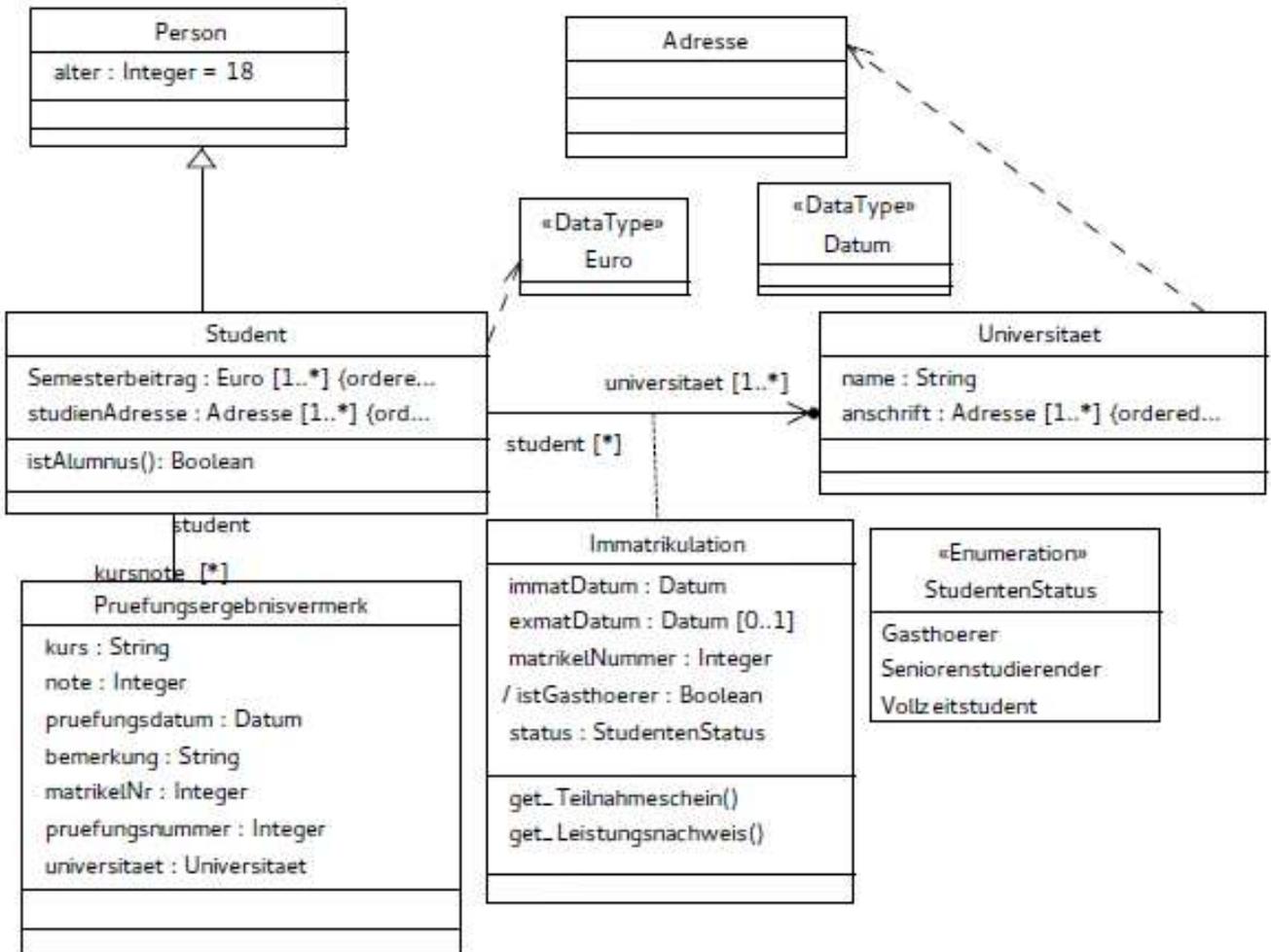
set im Falle **unique**,
multiset im Falle **non unique**,
vector im Falle **ordered** beziehungsweise
vector im Falle **ordered, unique** (mit geeigneter spezieller Invariante)

als Rückgabewert modelliert werden. Für qualifizierte Assoziationen steht die STL **map** bereit.

2.26 Modell Student/Universitaet/Pruefungsergebnisvermerk

Das UML-Modell mit Assoziationsklassen und qualifizierten Assoziationen:

und seine Papyrus-Form:



2.27 pre-Zustand in Nachbedingungen

Aus dem OCL-Manual:

7.5.14 Previous Values in Postconditions

As stated in 7.3.4, 'Pre- and Postconditions' OCL can be used to specify pre- and postconditions on operations and behaviors in UML. In a postcondition, the expression can refer to values of any feature of an object at two moments in time:

- the value of a feature at the start of the operation or behavior
- the value of a feature upon completion of the operation or behavior

The value of an operation call or a property navigation in a postcondition is the value upon completion of the operation. To refer to the value of a feature at the start of the operation, one has to postfix the property name with the keyword '@pre':

```
context Person::birthdayHappens()
  post: age = age@pre + 1
```

The property *age* refers to the property of the instance of Person that executes the operation. The property *age@pre* refers to the value of the property *age* of the Person that executes the operation, at the start of the operation.

In the case of an operation call, the '@pre' is postfixed to the operation name, before the parameters.

```
context Company::hireEmployee(p : Person)
  post: employees = employees@pre->including(p) and
       stockprice() = stockprice@pre() + 10
```

When the pre-value of a feature evaluates to an object, all further properties that are accessed of this object are the new values (upon completion of the operation) of this object. So:

```
a.b@pre.c      -- takes the old value of property b of a, say x
                -- and then the new value of c of x.
a.b@pre.c@pre  -- takes the old value of property b of a, say x
                -- and then the old value of c of x.
```

The '@pre' postfix is allowed only in OCL expressions that are part of a Postcondition, and only on invocations of the features of model classifiers. Asking for a current property of an object that has been destroyed during execution of the operation results in null. Also, referring to the previous value of an object that has been created during execution of the operation results in null.

Auch

```
self@pre
```

ist möglich, wie Abschnitt 11.3.1 des OCL-Manuals (Nachbedingung von `oclIsNew()`) zeigt.

2.28 Contracts zum Modell Student/Universitaet/Pruefungsergebnisvermerk

```
context Universitaet
inv : name <> ''
inv : student->size() >= 0
inv : immatrikulation->isUnique(matrikelNummer)
inv : student.universitaet->includes(self)
/* nach Einfuehrung einer qualifizierten Assoziation mit
   Assoziationsklasse auch moeglich:
   inv : student[0312345].familienname = 'Bauer'
*/

context Student
inv : immatrikulation->size() >= 1
inv : immatrikulation->select(not istGasthoerer)->size() >= 1
inv : universitaet.student->includes(self)
inv : pruefungsergebnisvermerk->size() > 0 implies
    pruefungsergebnisvermerk.student->asSet() = Set{self}

context Student::istAlumnus() : Boolean
body : not immatrikulation->exists(exmatDatum.oclIsUndefined())

context Student::getOpernKarte( einzahlung : Euro ) : Boolean
pre : einzahlung >= Euro(10) — allg. Vorbedingung als in Person
post : true

context Immatrikulation
inv : matrikelNummer > 0
inv : not exmatDatum.oclIsUndefined() implies exmatDatum >=
    immatDatum

inv : immatDatum - student.geburtsdatum >= 12
inv : not exmatDatum.oclIsUndefined() implies Datum::today() >=
    exmatDatum

context Immatrikulation::immatDatum
init : Datum::today()

context Immatrikulation::exmatDatum
init : OclUndefined — 2. Versuch eines optionalen Datentyps:
    — hier: OclUndefined als Ungueltig-Marke

context Immatrikulation::istGasthoerer
```

```

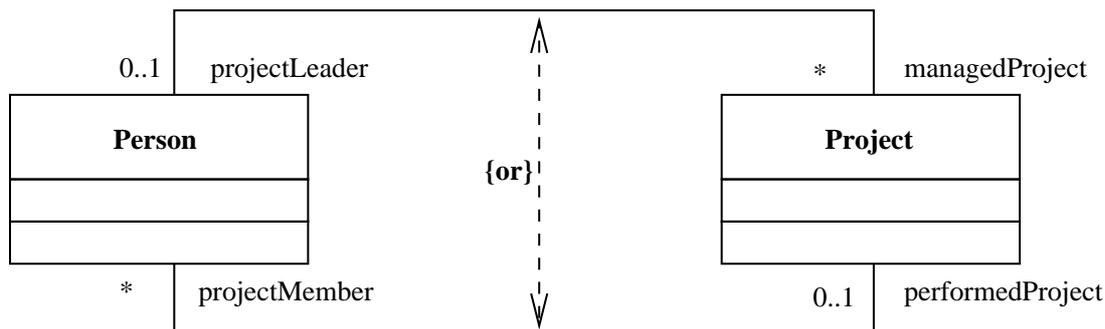
init : false

context Pruefungsergebnisvermerk
inv : kurs <> ''
inv : 1 <= note and note <= 5
inv : matrikelNr > 0
inv : student.immatrikulation.matrikelNummer->includes(
    matrikelNr)
inv : Pruefungsdatum.inbetween(student.immatrikulation->any(
    matrikelNummer=matrikelNr).immatDatum,
    student.immatrikulation->any(matrikelNummer=matrikelNr).
    exmatDatum)
inv : pruefungsnummer > 0
inv : Pruefungsergebnisvermerk.allInstances()->isUnique(
    pruefungsnummer)
inv : Student.kursnote->includes(self)
— spaeter besser: inv : student.kursnote[pruefungsnummer] =
    self

```

2.29 UML Constraints in UML 1.x

2.29.1 or / xor



ist **mehrdeutig**:

Bezieht sich das **or** auf die linke oder die rechte Assoziationsendseite?

OCL kann die betreffende Seite eindeutig beschreiben:

context Person

inv: not managedProject → isEmpty() or
 not performedProject → isEmpty()

Jede Person leitet entweder das Projekt oder arbeitet an ihm mit.

(oder falls die linken Assoziationsenden von Bedeutung:)

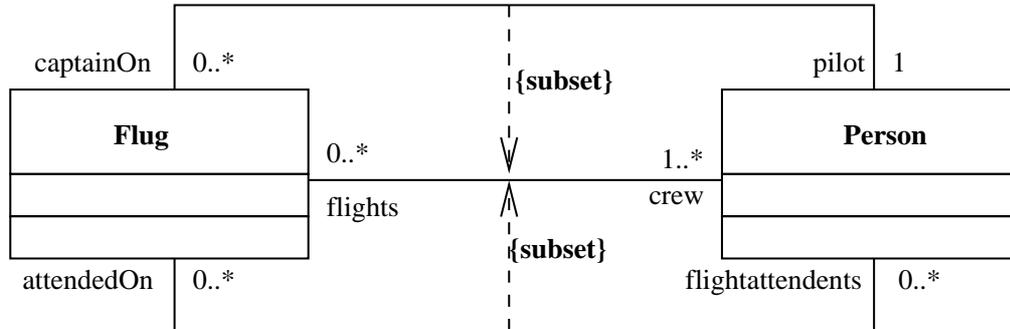
```

context Project
inv: not projectLeader → isEmpty() or
      not projectMember → isEmpty()

```

Jedes Projekt hat entweder ein Projektleiter oder ein Projektmitglied.

2.29.2 subset



Subset-Constructs machen UML-Diagramme häufig schlecht lesbar.

Besser ist:

```

context Flug
inv: self.crew → includes(self.pilot)
inv: self.crew → includesAll(self.flightattendants)

```

oder:

```

context Person
inv: self.flights → includesAll(self.captainOn)
inv: self.flights → includesAll(self.attendedOn)

```

Vergleiche auch `{subsets rollenname}` in UML 2.x.

2.30 OCL primitive type Real

Leider werden die gültigen Real-Literale im OCL-Manual nur natürlichsprachig spezifiziert:

9.4.19 RealLiteralExpCS

This rule represents real literal expressions. A real literal consists of an integer part, a fractional part and an exponent part. The exponent part consists of either the letter 'e' or 'E', followed optionally by a '+' or '-' letter followed by an exponent integer part. Each integer part consists of a sequence of at least one of the decimal digit characters. The fractional part consists of the letter '.' followed by a sequence of at least one of the decimal digit characters. Either the fraction part or the exponent part may be missing but not both.

`RealLiteralExpCS ::= <Real Lexical Representation>`

Das ist bedauerlich, zumal im OCL-Manual andere Literale formal exakt spezifiziert werden:

```
[A] StringLiteralExpCS ::= #x27 StringChar* #x27
[B] StringLiteralExpCS [1] ::= StringLiteralExpCS [2] WhiteSpaceChar* #x27
StringChar* #x27
...
StringChar ::= Char | EscapeSequence
WhiteSpaceChar ::= #x09 | #x0a | #x0c | #x0d | #x20
Char ::= [#x20-#x26] | [#x28-#x5B] | [#x5D-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
EscapeSequence ::= '\ ' 'b' — #x08: backspace BS
                  | '\ ' 't' — #x09: horizontal tab HT
                  | '\ ' 'n' — #x0a: linefeed LF
                  | '\ ' 'f' — #x0c: form feed FF
                  | '\ ' 'r' — #x0d: carriage return CR
                  | '\ ' '"' — #x22: double quote "
                  | '\ ' ''' — #x27: single quote '
                  | '\ ' '\' — #x5c: backslash \
                  | '\ ' 'x' Hex Hex — #x00 to #xFF
                  | '\ ' 'u' Hex Hex Hex Hex — #x0000 to #xFFFF
Hex ::= [0-9] | [A-F] | [a-f]
```

definiert formal mit regulären Ausdrücken, was String-Literale sind (Abschnitt 9.4.20).

Selbst das im Manual zitierte XML-Schema *double*

3.2.5 *double*

[Definition:] The **double** datatype is patterned after the IEEE double-precision 64-bit floating point type [IEEE 754-1985]. The basic *value space* of **double** consists of the values $m \times 2^e$, where m is an integer whose absolute value is less than 2^{53} , and e is an integer between -1075 and 970, inclusive. In addition to the basic *value space* described above, the *value space* of **double** also contains the following three *special values*: positive and negative infinity and not-a-number (NaN). The *order-relation* on **double** is: $x < y$ iff $y - x$ is positive for x and y in the value space. Positive infinity is greater than all other non-NaN values. NaN equals itself but is *incomparable* with (neither greater than nor less than) any other value in the *value space*.

Note: "Equality" in this Recommendation is defined to be "identity" (i.e., values that are identical in the *value space* are equal and vice versa). Identity must be used for the few operations that are defined in this Recommendation. Applications using any of the datatypes defined in this Recommendation may use different definitions of equality for computational purposes; [IEEE 754-1985]-based computation systems are examples. Nothing in this Recommendation should be construed as requiring that such applications use identity as their equality relationship when computing.

Any value *incomparable* with the value used for the four bounding facets (*minInclusive*, *maxInclusive*, *minExclusive*, and *maxExclusive*) will be excluded from the resulting restricted *value space*. In particular, when "NaN" is used as a facet value for a bounding facet, since no other **double** values are *comparable* with it, the result is a *value space* either having NaN as its only member (the inclusive cases) or that is empty (the exclusive cases). If any other value is used for a bounding facet, NaN will be excluded from the resulting restricted *value space*; to add NaN back in requires union with the NaN-only space.

This datatype differs from that of [IEEE 754-1985] in that there is only one NaN and only one zero. This makes the equality and ordering of values in the data space differ from that of [IEEE 754-1985] only in that for schema purposes NaN = NaN.

A literal in the *lexical space* representing a decimal number d maps to the normalized value in the *value space* of **double** that is closest to d ; if d is exactly halfway between two such values then the even value is chosen. This is the *best approximation* of d ([Clinger, WD (1990)], [Gay, DM (1990)]), which is more accurate than the mapping required by [IEEE 754-1985].

3.2.5.1 *Lexical representation*

double values have a lexical representation consisting of a mantissa followed, optionally, by the character "E" or "e", followed by an exponent. The exponent *must* be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for *integer* and *decimal*. If the "E" or "e" and the following exponent are omitted, an exponent value of 0 is assumed.

The *special values* positive and negative infinity and not-a-number have lexical representations INF, -INF and NaN, respectively. Lexical representations for zero may take a positive or negative sign.

For example, -1E4, 1267.43233E12, 12.78e-2, 12, -0, 0 and INF are all legal literals for **double**.

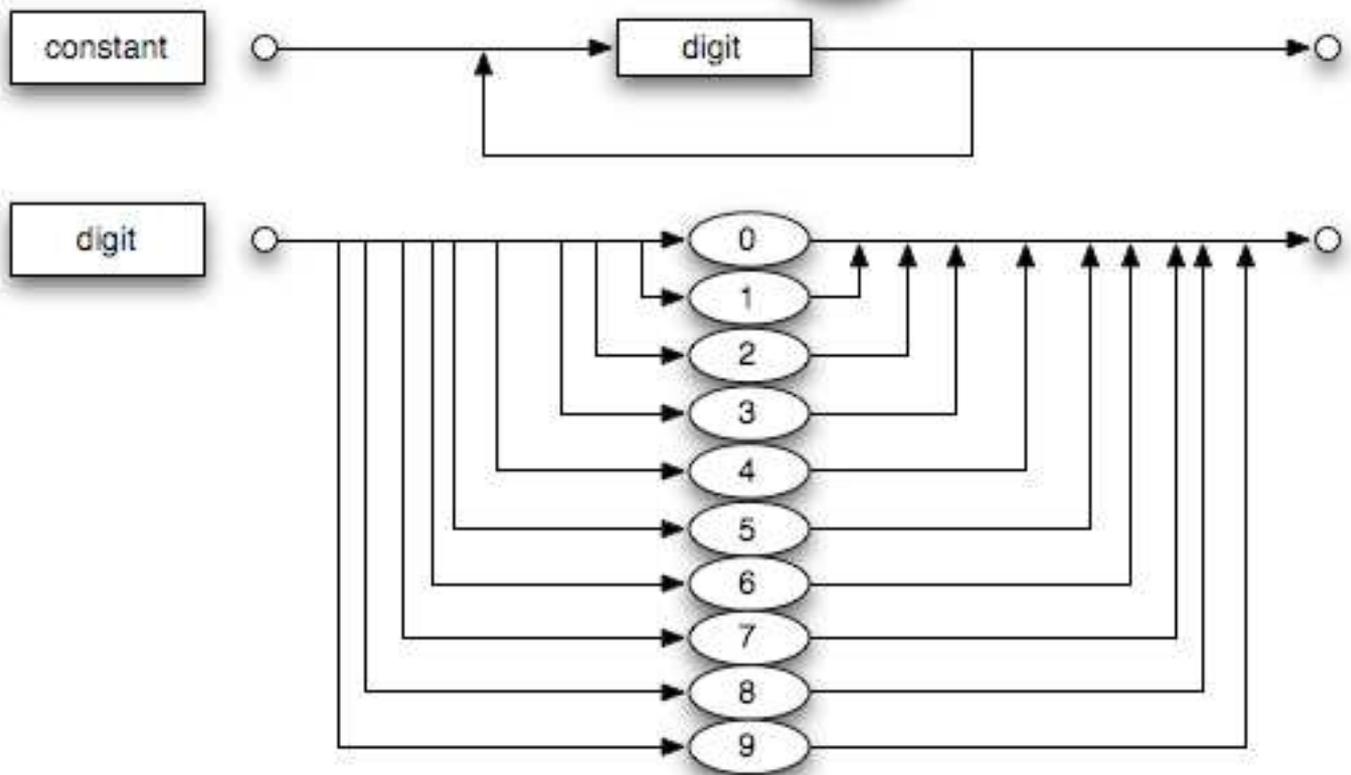
3.2.5.2 *Canonical representation*

The canonical representation for **double** is defined by prohibiting certain options from the *Lexical representation* (§3.2.5.1). Specifically, the exponent must be indicated by "E". Leading zeroes and the preceding optional "+" sign are prohibited in the exponent. If the exponent is zero, it must be indicated by "E0". For the mantissa, the preceding optional "+" sign is prohibited and the decimal point is required. Leading and trailing zeroes are prohibited subject to the following: number representations must be normalized such that there is a single digit which is non-zero to the left of the decimal point and at least a single digit to the right of the decimal point unless the value being represented is zero. The canonical representation for zero is 0.0E0.

überläßt die genaue nicht-natürlichsprachige Spezifikation der *XML Schema Language*-Sprachimplementierung.

Wegen der vielen Vorteile formaler Spezifikationen, wäre eine solche formale viel angebrachter. Dazu bieten sich an:

- **Railroad-Syntaxdiagramme**



- **EBNF-Spezifikationen**

```
constant = digit , {digit};
digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
```

Also:

```
RealLiteralExpCS = integerpart , fractionalpart , [exponentpart] |
integerpart , exponentpart;
integerpart      = ["+" | "-"] , digit , {digit};
fractionalpart  = "." , digit , {digit};
exponentpart    = ("E" | "e") , ["+" | "-"] , digit , {digit};
digit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
```

Aufgabe: Wie unterscheidet sich die hier (nach **EBNF**) benutzte EBNF-Symbolik von der in Abschnitt 9.4.20 des OCL-Manuals benutzten?

2.31 OCLs dreiwertige Logik

partielle Funktionen = Funktionen mit undefinierten Stellen, zum Beispiel:

$$Inv(x) = \begin{cases} \frac{1.0}{x} & \text{falls } x \in \mathbb{R} \setminus \{0.0\} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

schreibt man heute (im Zeitalter des funktionalen Programmierstils) als:

$$Inv(x) = \begin{cases} \frac{1.0}{x} & \text{falls } x \in \mathbb{R} \setminus \{0.0\} \\ \perp & \text{sonst} \end{cases}$$

und macht dadurch $Inv()$ zur totalen Funktion auf \mathbb{R} , indem man den Wertebereich auf $\mathbb{R} \cup \{\perp\}$ vergrößert. \perp steht für undefiniert oder ungültig.

Table A.2 - Semantics of boolean operations

b_1	b_2	b_1 and b_2	b_1 or b_2	b_1 xor b_2	b_1 implies b_2	not b_1
false	false	false	false	false	true	true
false	true	false	true	true	true	true
true	false	false	true	true	false	false
true	true	true	true	false	true	false
false	\perp or ϵ	false	\perp	\perp	true	true
true	\perp or ϵ	\perp	true	\perp	\perp	false
\perp or ϵ	false	false	\perp	\perp	\perp	\perp
\perp or ϵ	true	\perp	true	\perp	true	\perp
\perp or ϵ	\perp or ϵ	\perp	\perp	\perp	\perp	\perp

Table A.3 - Additional semantics of unlimited natural comparisons

v_1	v_2	$v_1 = v_2$	$v_1 \lt \gt v_2$	$v_1 < v_2$	$v_1 \leq v_2$	$v_1 \geq v_2$	$v_1 > v_2$
a	b	$a = b$	$a \lt \gt b$	$a < b$	$a \leq b$	$a \geq b$	$a > b$
a	∞	false	true	true	true	false	false
∞	b	false	true	false	false	true	true
∞	∞	true	false	false	true	true	false

Table A.4 - Additional semantics of unlimited natural monadic operations

v	$abs(v)$	$toInteger(v)$
a	a	a
∞	∞	\perp

the set of real numbers, *Boolean* the truth values true and false, and *String* all finite strings over a given alphabet. Each domain also contains two special values ϵ and \perp . ϵ corresponds to the *null* value, and \perp , pronounced bottom, corresponds to the invalid value. These are motivated in the next sub clause. The *UnlimitedNatural* domain also includes a special

A.4.1.1 Error Handling

Each domain of a basic type t contains two special values ε and \perp . ε represents a *null* or undefined value and \perp an *invalid* value. These are useful for the following purposes:

1. An undefined or *null* value may, for example, be assigned to an attribute of an object. In this case the undefined value helps to model the situation where the attribute value is not yet known (for example, the email address of a customer is unknown at the time of the first contact, but will be added later) or does not apply to this specific object instance (e.g., the customer does not have an email address). This usage of undefined values is well-known in database modeling and querying with SQL [Dat90, EN94]), in the Extended ER-Model [Gog94], and in the object specification language TROLL *light* [Her95].
2. An *invalid* value can signal an error in the evaluation of an expression. An example for an expression that is defined by a partial function is the division of integers. The result of a division by zero is undefined. The problems with partial functions can be eliminated by including an invalid value \perp into the domains of types. For all operations we can then extend their interpretation to total functions.

The interpretation of operations is considered strict unless there is an explicit statement in the following. Hence, an *invalid* or *null* argument value causes an *invalid* operation result. This ensures the propagation of error conditions.

dreiwertige Logik

2.32 Stil-Hinweise für OCL-Constraints

- Schreibe lieber für viele Klassen kurze OCL-Ausdrücke, die nur wenige Assoziationen tief „navigieren“, als lange Navigationsketten, die das ganze Modell durchlaufen.
- Vermeide `allInstances()` wenn immer möglich:
Zum Beispiel ist

```
context Person
inv: parents → size() <= 2
```

und

```
context Person
inv: Person.allInstances() → forAll(p | p.parents → size() <= 2)
```

äquivalent, aber unterschiedlich effizient.
- Nutze Invarianten in Klassen, um die möglichen Werte der Attribute einer Klasse von den unmöglichen zu unterscheiden.
- Schreibe Invarianten in die Klasse, zu der sie gehören:
 - Attributwerteinschränkungen gehören in die Klasse, die das Attribut definieren.
 - Falls eine Invariante die Attribute mehr als einer Klasse einschränkt, kann jede dieser Klassen als Kontext gewählt werden. Eventuell kann man einer

dieser Klassen die Verantwortung über die andere zuteilen.

- Jede Invariante sollte durch möglichst wenig Assoziation navigieren.
- Versuche bei Bedarf, eine Invariante testweise im Kontext verschiedener Klassen zu formulieren. Wähle dann die einfachste Version.

Zum Beispiel ist

Context Company

```
inv: employees.wife -> intersection(
    self.employees) -> isEmpty()
```

einfacher als

Context Person

```
inv: wife -> notEmpty() implies
    wife.employers -> intersection(self.employers)
    -> isEmpty()
```

- Nutze viele einfache,

inv: ...

inv: ...

statt einer einzelnen inv-Zeile komplizierterer Bauart.

- Vermeide der Lesbarkeit halber collect():

context Person

```
inv: self.parents.brothers.children → notEmpty()
```

ist äquivalent zu:

context Person

```
inv: self.parents →
```

```
    collect(brothers) →
```

```
    collect(children) → notEmpty()
```

- Gib allen Assoziationsenden einen Namen.

2.33 Einfache Beispielverträge und die geeignete Kontextwahl

Siehe: [Introduction to OCL](#)

2.34 OCL String als ADT

ADT

„The standard type String represents string. A string is a sequence of characters in some suitable character set used to display information about the model. Character sets may include non-Roman alphabets and characters. String is itself an instance of the metatype PrimitiveType (from UML).“ (OCL 11.4.3)

Mögliche Werte für Stringobjekte sind auch ϵ und \perp (A.4.1).

In 9.4.20 wird StringLiteralExpCS sehr schön formal als

```
[A] StringLiteralExpCS ::= #x27 StringChar* #x27
[B] StringLiteralExpCS [1] ::= StringLiteralExpCS [2]
    WhiteSpaceChar* #x27 StringChar* #x27
```

```
StringChar ::= Char | EscapeSequence
WhiteSpaceChar ::= #x09 | #x0a | #x0c | #x0d | #x20
Char ::= [#x20–#x26] | [#x28–#x5B] | [#x5D–#xD7FF] | [#xE000–#
    xFFFD] | [#x10000–#x10FFFF]
EscapeSequence ::= '\ ' 'b' — #x08: backspace BS
    | '\ ' 't' — #x09: horizontal tab HT
    | '\ ' 'n' — #x0a: linefeed LF
    | '\ ' 'f' — #x0c: form feed FF
    | '\ ' 'r' — #x0d: carriage return CR
    | '\ ' '"' — #x22: double quote "
    | '\ ' ''' — #x27: single quote '
    | '\ ' '\' — #x5c: backslash \
    | '\ ' 'x' Hex Hex — #x00 to #xFF
    | '\ ' 'u' Hex Hex Hex Hex — #x0000 to #xFFFF
Hex ::= [0–9] | [A–F] | [a–f]
```

spezifiziert.

toString(): **String**

für die Datentypen Real, Integer, Boolean, UnlimitedInteger.

Lediglich für UnlimitedInteger wird mehr als „Converts self to a string value.“ spezifiziert:

„Converts self to a string value, using the canonical form as defined by <http://www.w3.org/TR/xmlschema-2/> nonNegativeInteger. If self is unlimited the result is '*’.“

A.4.1.2 nennt die String-Operationen <, >, <=, >=, size(), concat(String), toUpperCase(), toLowerCase(), substring(Integer, Integer).

11.5.3 erläutert die Operationen +(String), size(), toInteger(), toReal(), toUpperCase(), toLowerCase() ohne genaue formale Spezifikation.

substring(Integer, Integer) spezifiziert seine Vorbedingungen, +(String), concat(String), indexOf(String), equalsIgnoreCase(String), at(Integer), characters(), toBoolean() wird mit formalen Nachbedingungen spezifiziert. >, <=, >= werden auf <, = zurückgeführt.

Vergleiche und toUpperCase(), toLowerCase() werden durch den Wert von

`oclLocale: String`

beeinflusst, der etwa auf 'de_DE' gesetzt werden kann (11.1).

2.35 OCL in Together-Tools

mit OCL-Constraints, Code-Erzeugung für diese, ...

OCL Basic Types

... and their language binding:

OCL Type	Java	C#	Delphi	EMF*
Boolean	boolean	bool	Boolean	EBoolean
Integer	int	int	Integer	EInt
Real	float	float	Double	EFloat
String	String	string	WideString	EString
OclAny	Object	object	TObject	EObject
OclType	Class	Type	TTypeInfo	EClassifier
OclVoid	null	null	Nil or Null	null

OCL Collection Types

Tic-Tac-Toe Example

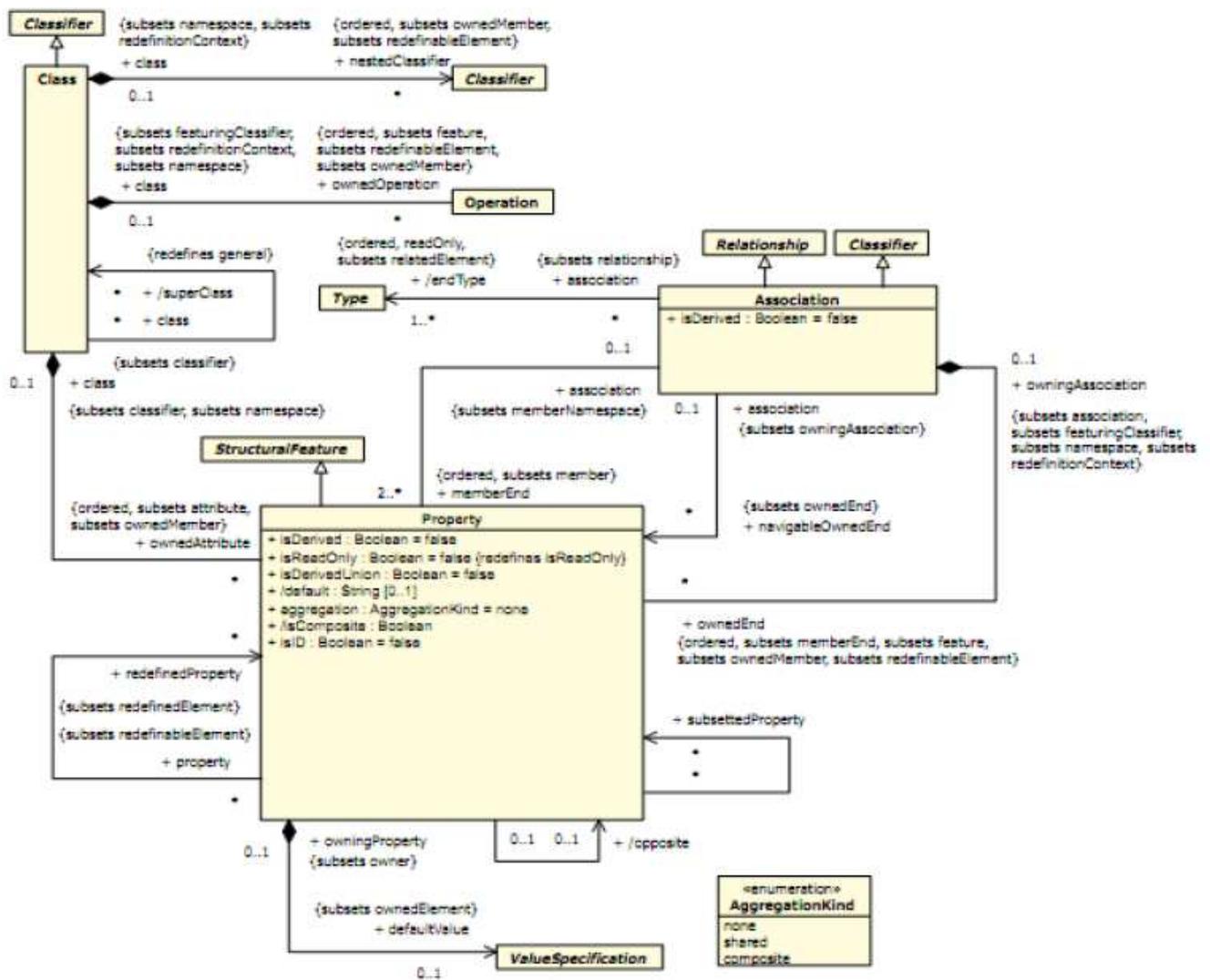
Generating Code from OCL

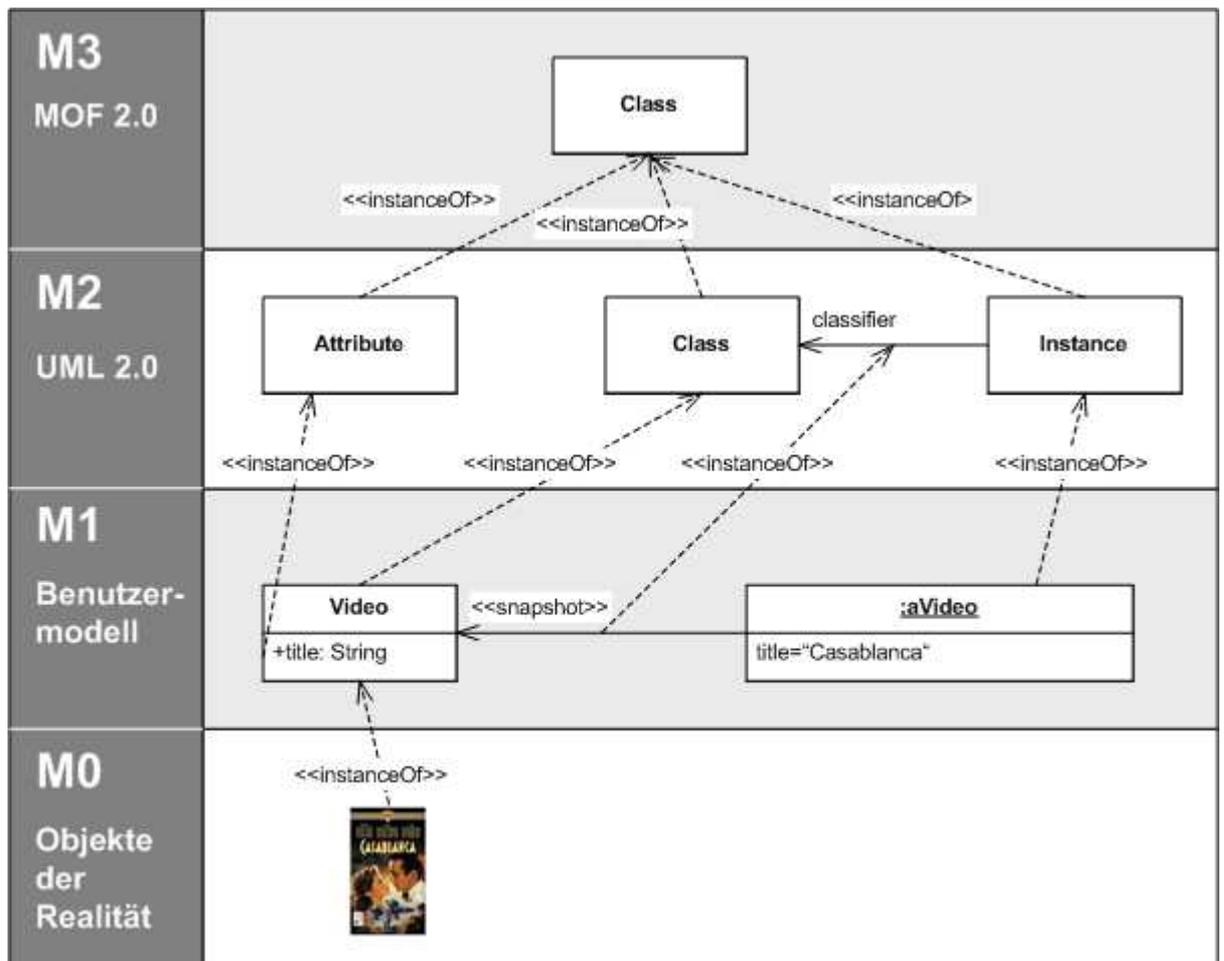
2.36 OMGs C++ Language-Mapping für CORBA

C++ language mapping
CORBA
Currency Service

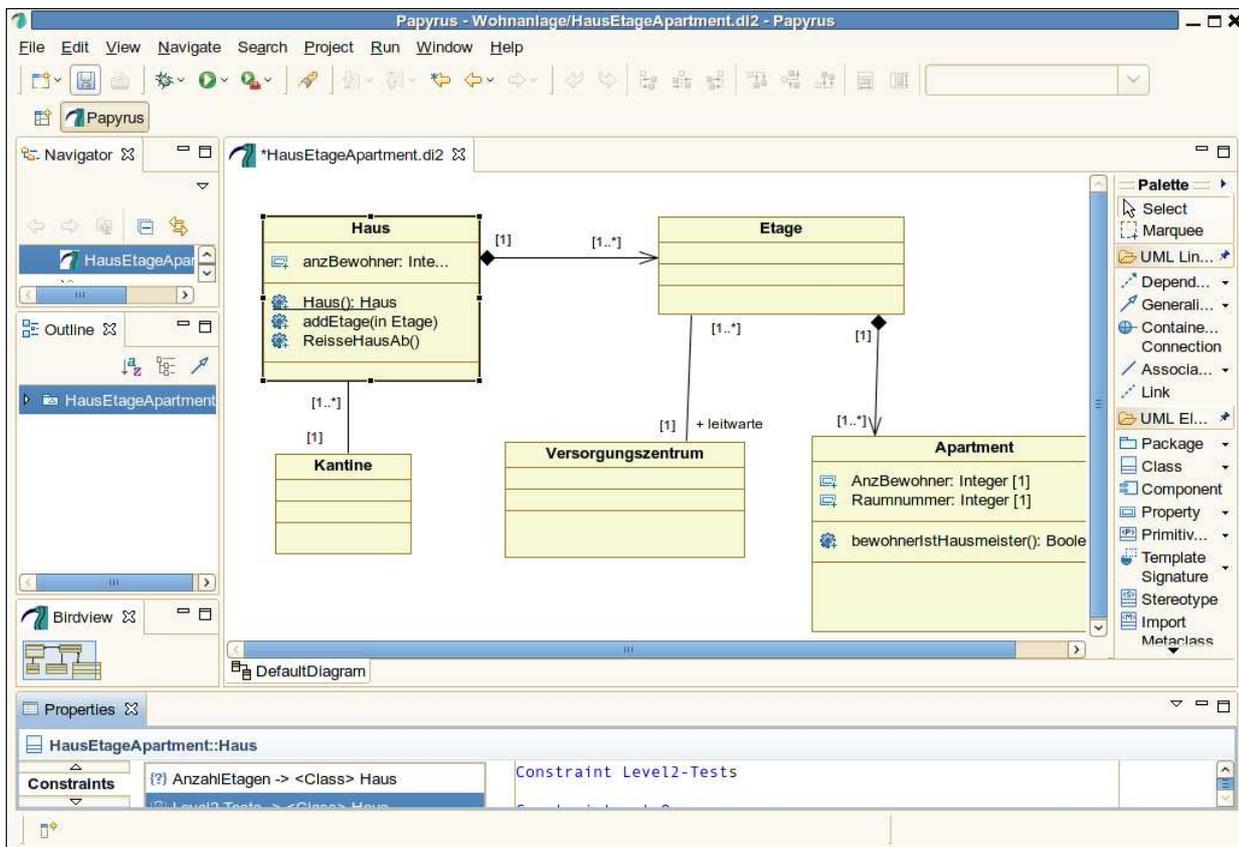
2.37 Was ist ein UML-Modell: MOF

Meta Object Facility
MOF
<http://www.omg.org/mof/>
EMOF OCL Constraints





2.38 Metalevel2-Constraints = Wohldefinierte Regeln für Modelle



Im **M2-Level** des Constraints-Checkers von Papyrus kann man ein Modell algorithmisch mit OCL-Ausdrücken abfragen (Query-Sprache) und zum Beispiel Regeln des Programmiererteams als Invarianten definieren. Einige Queries:

```

context Haus
self                                     — Class Haus

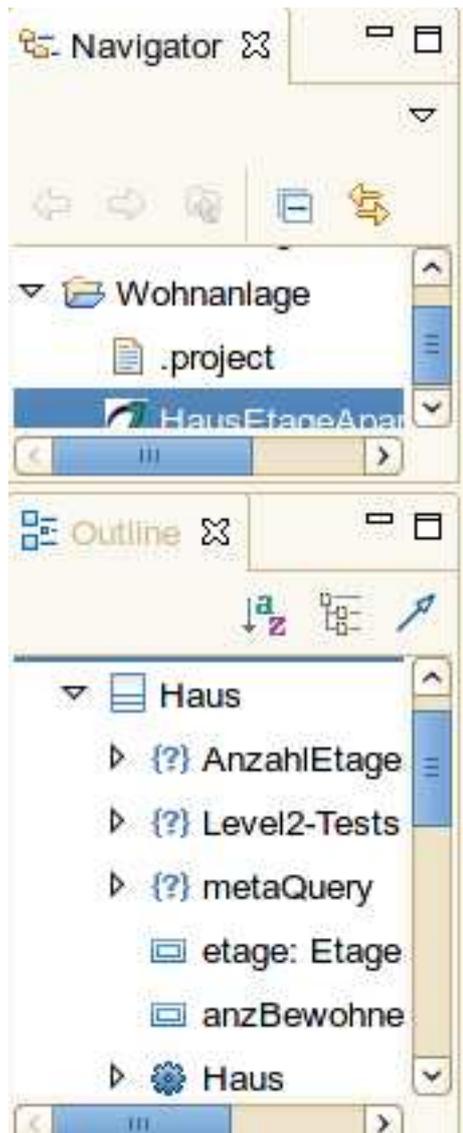
self.name                               — Haus

self.feature->size()                    — 5
self.feature->asSequence()->at(1)       — operation ReiseHausAb

namespace.ownedElement->size()         — 11

```

Die Modellelemente können Sie natürlich auch im Papyrus-Outline-Fenster betrachten:



Einige WFR-Regeln:

```
context Haus
```

```
inv: namespace.ownedElement->collect(name)->count(self.name)=1
```

```
context Class
```

```
inv: not self.name.oclIsUndefined() and self.name <> ''
```

```
context ModelElement
```

```
inv: NamedElement.allInstances()->forAll(c1,c2|c1<>c2 implies  
c1.name<>c2.name)
```

```
context Model
```

```
inv: Class.allInstances()->collect(c:Class| not c.name.
```

```
oclIsUndefined()->size ()=1
```

modeling levels: AST, UML-model/SdV, Object-diagram

”The name of an opposite AssociationEnd may **not** be the same as the name of an Attribute **or** a ModelElement contained in the Classifier.”

context Classifier

inv WFR₅:

```
self.oppositeAssociationEnds->forAll(o |  
not self.allAttributes->union(self.allContents)->collect(  
q | q.name)->includes(o.name))
```

”The name of an Attribute may **not** be the same as the name of an opposite AssociationEnd **or** a ModelElement contained in the Classifier.”

context Classifier

inv WFR₄:

```
self.feature->select(a | a.oclIsKindOf (Attribute))->forAll(a |  
not self.allOppositeAssociationEnds->union(self.allContents)  
->collect(q | q.name)->includes(a.name))
```

M2 Modellierungsrichtlinien, WFRs

M1 Geschäftsregeln, SdV, DbC, Spezifikation von Testfällen

M0 Ausführung von Testfällen

Aussagen über das aktuelle UML-Modell kann man so etwa über den folgenden M2 OCL-Ausdruck erhalten:

```

self.allOwnedElements ()
  ->select (e | e.ocIsKindOf (NamedElement) and e.ocAsType (
    NamedElement).name <> null)
  ->collect (n | n.ocAsType (NamedElement).name.concat (' : ').
    concat (n.eClass ().name))

```

Results :

```

'Writer : Class '
'Mystery : EnumerationLiteral '
'category : Property '
'Biography : EnumerationLiteral '
'books : Property '
'books : Property '
'Book2 : Class '
'author : Property '
'title : Property '
'ScienceFiction : EnumerationLiteral '
'pages : Property '
'BookCategory : Enumeration '
'writers : Property '
'Library : Class '
'name : Property '
'name : Property '

```

(aus: <http://wiki.eclipse.org/OCLSnippets>)

2.39 OCL-Beispiele

<http://www.empowertec.de/ocl/example1.htm>

context Person def:

```

statistics : Bag(TupleType( company: Company,
                             numEmployees: Integer,
                             wellpaidEmployees: Set(Person),
                             totalSalary: Integer)) =
  managedCompanies->collect (c |
    Tuple { company: Company = c,
            numEmployees: Integer = c.employee->size (),
            wellpaidEmployees: Set(Person) =
              c.job->select (salary >10000).employee->asSet (),
            totalSalary: Integer = c.job.salary->sum ()
          }
  )

```

(vgl. [OCL demonstration](#))

[The university model](#)


```

import 'model.uml'

context TreueKonto::punkte: Integer
init: 0

context KundenKarte::gueltig: Boolean
init: true

context KundenKarte::nameInGrossbuchstaben: String
derive: besitzer.titel.toUpper().concat(' ').concat(besitzer.name.toUpper())

context TreueProgramm::holeAngebote(): Set(Angebot)
body: partner.bereitgestelltesAngebot->asSet()

context TreueProgramm
def: holePartnerAngebote(pp: ProgrammPartner): Set(Angebot) =
  if partner->includes(pp) then
    pp.bereitgestelltesAngebot
  else
    Set{}
  endif

context TreueKonto
def: umsatz : Real = transaktion.betrag->sum()

context TreueProgramm
def: holeAngeboteEinesLevels(levelName: String): Set(Angebot) =
  level->select(name=levelName).verfuegbaresAngebot->asSet()

context Kunde
inv geschaeftsfaehigesAlter: alter >= 18

context KundenKarte
inv: gueltigAb.istVorher(gueltigBis)

context KundenKarte
inv altersBeschraenkung: besitzer.alter >= 18

context TreueProgramm
inv nurBekannteAngebotsLevels: levels->includesAll(Mitgliedschaft.aktuellesLevel)

context Mitgliedschaft
inv KarteKorrekt: teilnehmer.karte->includes(karte)

context Mitgliedschaft
def: holeAktuellenLevelNamen():String =
  aktuellerLevel.name

context Mitgliedschaft
inv konsistenteLevelNamen: aktuellesLevel.name='silber' implies
                           karte.farbe=Color::silber
                           and
                           aktuellesLevel.name='gold' implies
                           karte.farbe=Color::gold

context TreueProgramm
inv minimalesAngebot: partner.bereitgestelltesAngebot->size() >= 1

context TreueProgramm
inv konsistenteAngebote: partner.bereitgestelltesAngebot->
  includesAll(level.verfuegbaresAngebot)

context Kunde
inv genugendKarten: treueprogramm->size() = karte->select(gueltig=true)
  ->size()

```

```

context TreueProgramm
inv keineKontenWennKeinPunkterwerbMoeglich:
    partner.bereitgestelltesAngebot ->forall(
        erworbenePunkte = 0 and eingeloestePunkte = 0
    ) implies Mitgliedschaft.konto->isEmpty()

context ProgrammPartner
inv TeilnehmerAnzahl: anzahlKunden = programm.teilnehmer->asSet()->size()

context TreueProgramm
inv: level->first().name='silber'

context TreueProgramm
inv: level->at(2).name='gold'

context TreuKonto::istLeer(): Boolean
pre: true
post: result = (punkte = 0)

context TreueKonto::holeKundenName(): String
body: Mitgliedschaft.karte.besitzer.name

context Kunde::feiereGeburtstag(): Boolean
post: alter = alter@pre + 1

context Angebot::aktualisiereErworbenePunkte(betrag: Integer): OclVoid
pre: betrag >= 0
post: berechnePunkte() = berechnePunkte@pre() + betrag

context ProgrammPartner
inv maxTotalPunkte: bereitgestelltesAngebot.transaktion
    ->select( OclIsTypeOf(Erwerb)).punkte->sum() < 10000

context KundenKarte
def: KarteIstGueltig: Boolean=
    let gueltigesDatum : Boolean =
        gueltigAb.istVorher(Datum::heute) and
        gueltigBis.istNachher(Datum::heute)
    in
        valid and gueltigesDatum

context TreueKonto
inv: transaktion.karte.besitzer->size()=1

context TreueKonto::erwerbe(i: Integer): OclVoid
pre: i > 0

context TreueKonto::loeseEin(i : Integer): OclVoid
pre: i > 0

context TreueKonto::istLeer():Boolean
body: punkte=0

context TreueKonto::transaktion: Set(Transaktion)
init: Set{ }

context TreueKonto
def: benutzteAngebote(): Set(Angebot)=
    transaktion.erzeugtDurch->asSet()

context TreueKonto
inv EinBesitzer: transaktion.karte.besitzer->asSet()=size()=1

context TreueProgramm::meldeAn(c: Kunde): OclVoid
pre: c.name <> ''

```

```

pre: c->notEmpty()
pre: not (teilnehmer->includes(c))
post: teilnehmer = teilnehmer@pre->including(c)

-- derived Class TransaktionsBerichtZeile
context TransaktionsBerichtZeile::partnerName: String
derive: transaktion.erzeugtDurch.partner.name

context TransaktionsBerichtZeile::angebotsBeschreibung: String
derive: transaktion.erzeugtDurch.beschreibung

context TransaktionsBerichtZeile::betrag: String
derive: transaktion.betrag

context TransaktionsBerichtZeile::punkte: String
derive: transaktion.punkte

context TransaktionsBerichtZeile::datum: Datum
derive: transaktion.datum

```

```

context TransaktionsBericht::name: String
derive: karte.besitzer.name

context TransaktionsBericht::kontostand: Integer
derive: karte.mitgliedschaft.konto.punkte

context TransaktionsBericht::kontoNummer: Integer
derive: karte.mitgliedschaft.konto.kontoNummer

context TransaktionsBericht::gesamtzahlErworbenerPunkte: Integer
derive: zeilen.transaktion->select(OclIsTypeOf(Erwerb)).punkte->sum()

context TransaktionsBericht::gesamtzahlEingeloesterPunkte: Integer
derive: zeilen.transaktion->select(OclIsTypeOf(Einloesung)).punkte->sum()

context TransaktionsBericht
inv berichtszeitraumOK: zeilen.datum->forAll(d| d.istVorher(bis) and
                                                    d.istNachher(von))

context TransaktionsBericht
inv berichtOk: karte.transaktionen->includesAll(zeilen.transaktion)

```

```

context TreueProgramm::fuegeTransaktionHinzu( kontoNr: Integer ,
                                             partnerName: String ,
                                             angebnr: Integer ,
                                             betr: Real ,
                                             d : Datum ): OclVoid
pre: mitgliedschaft.konto.kontoNummer->includes(kontoNr)
pre: partner.name->includes(partnerName)
pre: angebnr > 0
pre: betr >= 0.0
pre: not d.istVorher(Datum::heute())
post: let kto: TreueKonto =
        mitgliedschaft.konto->select(k| k.kontoNummer=kontoNr) ,
        neueT: Transaktion =
        partner->select(p| p.name=partnername).
        bereitgestelltesAngebot
        ->select(a| a.angebotsNr=angebnr).transaktion
        ->select(datum=d and betrag=betr) ,
        karte: KundenKarte =
        mitgliedschaft->select(m| m.konto.kontoNummer=kontoNr)
in
kto.punkte = kto.punkte@pre + neueT.punkte and

```

```

neueT.ocIsNew()
betr = 0 implies neueT.OcIsTypeOf(Einloesung)
betr > 0 implies neueT.OcIsTypeOf(Erwerb)
kto.transaktion - kto.transaktion@pre = Set{newT}
karte.transaktion - karte.transaktion@pre = Set{newT}

```

and
and
and
and

```

context TreueProgramm :: fuegeAngebotHinzu( p : ProgrammPartner ,
                                           l : AngebotsLevel ,
                                           a : Angebot
                                           ): OclVoid

```

```

pre: partner->includes(p)
pre: level->includes(l)
pre: a->notEmpty()
post: partner.bereitgestelltesAngebot ->includes(a)
post: level.verfuegbaresAngebot->includes(a)

```

```

context Kunde
def: extensivGenutzteKarten : Set(KundenKarte)=
    karte->select(transaktion.punkte > 10000)

def: firmentrueZu : Bag(ProgrammPartner)=
    treueprogramm.partner

def: kartenFuerTreueProgramm(p : TreueProgramm): Set(KundenKarte) =
    p.mitgliedschaft.karte

```

```

context TreueProgramm
def: ohnePunktErwerbungsMoeglichkeiten : Boolean =
    partner.bereitgestelltesAngebot ->forAll(erworbenePunkte=0)

```

```

context Mitgliedschaft
inv keinRabatt : treueprogramm.ohnePunktErwerbungsMoeglichkeiten implies
    konto->isEmpty()

```

```

context KundenKarte
def: holeGesamtpunktzahlAm(d: Datum): Integer =
    transaktion->select(not datum.istNachher(d)).punkte->sum()

```

Vertiefende Literatur:

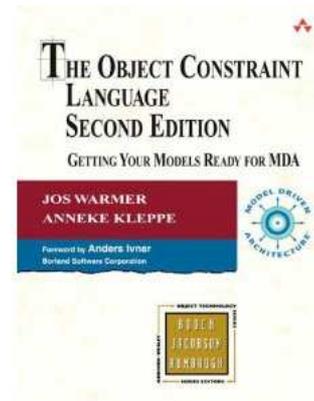
Jos Warmer

The Object Constraint Language Second Edition

Addison-Wesley

Erscheinungsdatum: 2003

ISBN: 0321179366



2.1. The “Royal and Loyal” System Example (Safari Books Online)