Materialsammlung - Formale Methoden: OCL und Eclipse

Prof. Dr. Hans-Jürgen Buhl

Sommersemester 2014

Bergische Universität Wuppertal
Fachbereich C — Mathematik und Naturwissenschaften
Fachgruppe Mathematik und Informatik

Praktische Informatik
PIBUW - SS14
April 2014
8. Auflage, 2014

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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

Lernergebnisse / Kompetenzen:
Die Studierenden können formale Software-Modelle lesen, verstehen und kritisch beurteilen. Sie haben formale Methoden als ein Kommunikationsmittel der Mitglieder eines Software-Entwicklungsteams kennen gelernt. Sie sind in der Lage, mit Hilfe der formalen Spezifikation Teilsysteme von realistischen Softwaremodellen selbst zu entwickeln.

Voraussetzungen:
Kenntnisse in der objektorientierten Programmierung und der Software-Entwicklung aus dem Bachelor-Studium.

Inhalte:

- Softwarequalität, Zusicherungen in Algorithmen; Konstruktoren, Modifikatoren, Observatoren und Destruktoren; Ausnahmebedingungen
- Methodik „Programming by Contract“ :
  Vorbedingungen, Nachbedingungen und Invarianten; ENBF zur formalen Spezifikation freier Eingabesprachen, UML-Klassendiagramme, Startwerte, Vererbung von Klasseninvarianten, Methodenvor- und -nachbedingungen
- Formale Spezifikation (z.B. in OCL2):
  UML-Klassendiagramme und „Constraints“ , virtuelle Attribute und Methoden, redundante Attribute und Methoden
- „Constraints“ an Attribute, Methoden und Assoziationen, Container-Typen, Frame-Regeln
- Fallstudien von formal spezifizierter Software (Algorithmen und Datenstrukturen)

Übungen zu Formale Methoden
2 Ü Mo 16-18 G.16.15

(Modulhandbuch Seite 47f.)

Vorbemerkungen

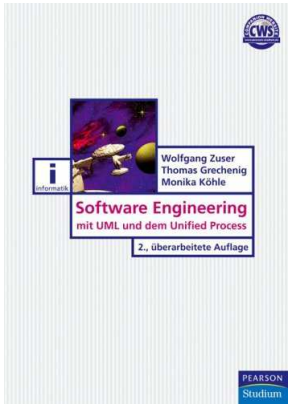
Unprofessionelle Softwarekonzeption
Software Design is Trial and Error ...

und ihre Konsequenzen:
Tom Tom GPS *Leap Year Bug*
Computerpanne stürzt US-Börsen ins Chaos
Fehler im Linux-Kernel kann Software-RAIDs zerstören
The Heartbleed Bug
Heartbleed: Datendiebstahl beim kanadischen Finanzamt
Fehlerhafte UEFI-Firmware: Linux killt Thinkpads
...

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, Axel Scheithauer
Analyse und Design mit der UML 2.5
Objektorientierte Softwareentwicklung
Olderbourg Verlag München
Auflage: 11., 2013
ISBN: 978-3-486-72140-9

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

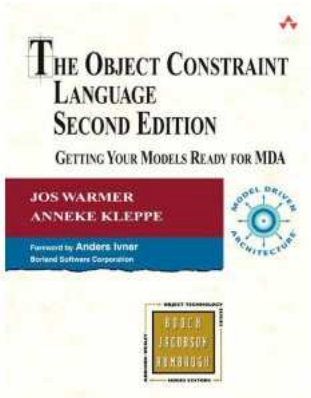
OMG
UML Superstructure
OMG Available Specification
Version 2.4.1
http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF

OMG Unified Modeling Language (OMG UML)
Version 2.5 Beta2
http://www.omg.org/spec/UML/2.5/Beta2/

OMG
Object Constraint Language
OMG Available Specification
Version 2.4
http://www.omg.org/spec/OCL/2.4/PDF

Jos Warmer
The Object Constraint Language Second Edition
Addison-Wesley
Erscheinungsdatum: 2003

ISBN: 0321179366



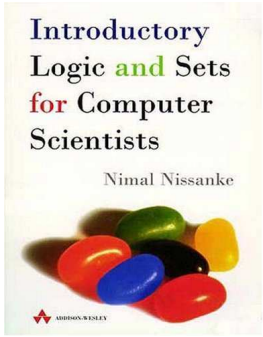
Tony Clark, Jos Warmer
Object Modeling with the OCL.
The Rationale behind the Object Constraint Language
http://www.amazon.de/Object-Modeling-OCL-Rationale-Constraint/dp/3540431691
ISBN: 3-540-43169-1

Scott W. Ambler
The Elements of UML 2.0 Style.
Cambridge University Press
2005
ISBN: 978-0-521-61678-2

Nimal Nissanke
Introductory Logic and Sets for Computer Scientists.

Broschiert - 400 Seiten
Addison Wesley
Erscheinungsdatum: Oktober 1998

ISBN: 0201179571

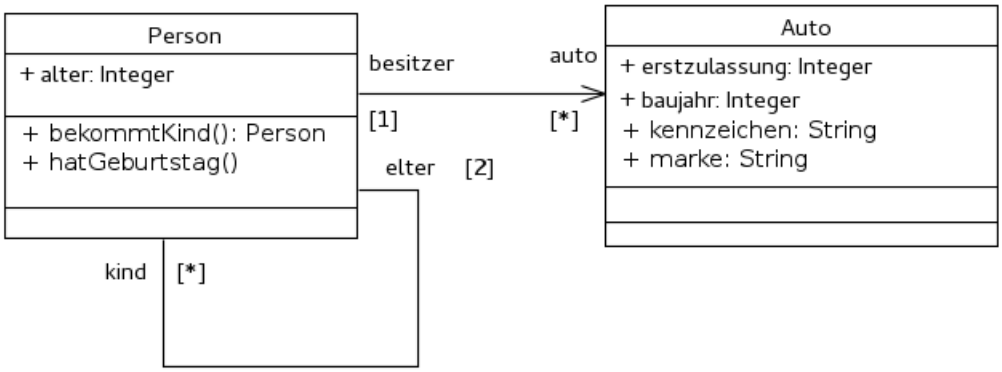


Martin Kreuzer, Stefan Kühling
Logik für Informatiker

Pearson Studium
Erscheinungsdatum: März 2006

ISBN: 3827372151

OOP-Modellierung



natürlichsprachige Detaillierungen:

- Das Alter einer Person ist nicht negativ.
- Eine Person ist jünger als ihre Eltern.
- Die Erstzulassung eines Autos liegt nicht vor dem Baujahr.
- ...

und formale Detail-Festlegungen:

```
import 'model.uml'

context Model::Person
inv: self.alter >= 0

context Model::Person
inv: self.elter->forAll(e|e.alter > self.alter)

context Model::Auto
inv: self.erstzulassung >= self.baujahr

...
```

(nach Object Constraint Language (OCL))

Aufgabe:
Welche Interpretationsmöglichkeiten gibt es für die natürlichsprachigen Detaillierungen?

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

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

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

Formale Methoden / formal methods
<Mathematik Spezifikation> Mathematisch basierte Technick zur Spezifikation, Entwicklung und Verifikation von Software und Harware 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 Spezifikationssprachen benuzten anfangs mathematische Formelschreibweise (Latex-Stil):

$$
\begin{array}{l}
Queue = Qelem^* \\
q_0 = [\ ] \\
\\
\text{ENQUEUE } (e : Qelem) \\
\textbf{ext wr} \quad q : Queue \\
\textbf{post} \quad q = \overleftarrow{q} \frown [e] \\
\\
\text{DEQUEUE() } e : Qelem \\
\textbf{ext wr} \quad q : Queue \\
\textbf{pre} \quad q \neq [\ ] \\
\textbf{post} \quad \overleftarrow{q} = [e] \frown q \\
\\
\text{ISEMPTY() } r : \mathbb{B} \\
\textbf{ext rd} \quad q : Queue \\
\textbf{post} \quad r \Leftrightarrow (len\ q = 0)
\end{array}
$$

VDM, VDM++

Heute geht man aber immer mehr zu reiner ASCII (oder Unicode)-Sytax 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 16f.)

Overture

Object Constraint Language 2.4

Benutzte freie UML2-/OCL-Tools

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

Papyrus bis 2008:
http://www.papyrusuml.org

oder aktuell: http://wiki.eclipse.org/MDT/Papyrus

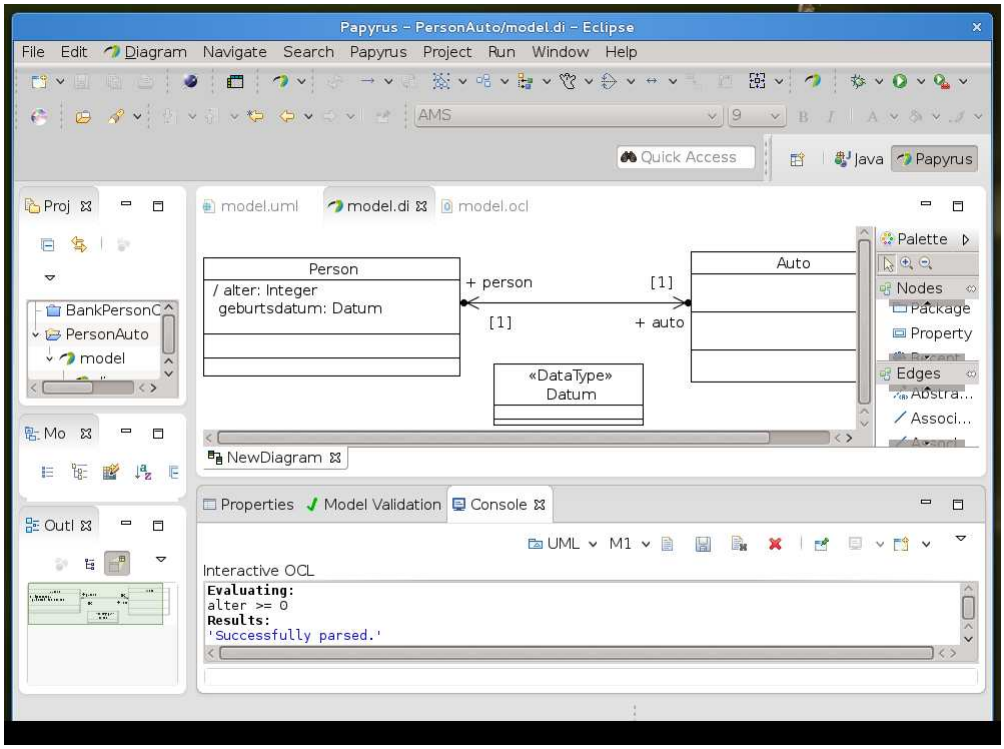
(Verfügbar (vorinstalliert) auf dem IT-Ausbildungsclustern (l101, ...) als `eclipse-papyrus`.)

Hinweis zur Installation auf dem eigenen Notebook:
Eclipse Modeling aktuell: Kepler SR2

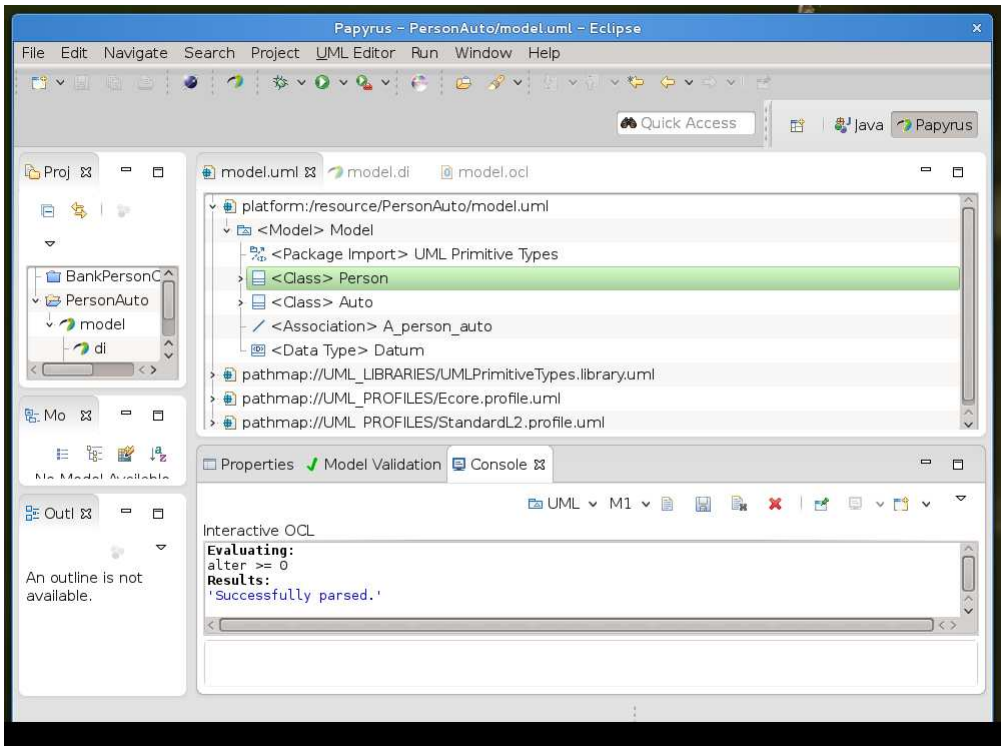
Downloaden, `gunzip` und `tar -xvf`.

```
Help
   Install Modeling Components

      Papyrus


      OCL Tools


Eventuell zusätzlich CDT:

Help
   Install new software
     add
       http://download.eclipse.org/tools/cdt/releases/kepler
         select all, accept, finish
```

Erste Gehversuche: UML (mit Papyrus)



und OCL mit den OCL-Tools von Papyrus:



Klassen-Attribute in UML 2.4

UML2-Notationsübersicht
UML-Klassendiagramm

UML Constraint
Stereotyp `<<Utility>>` ...

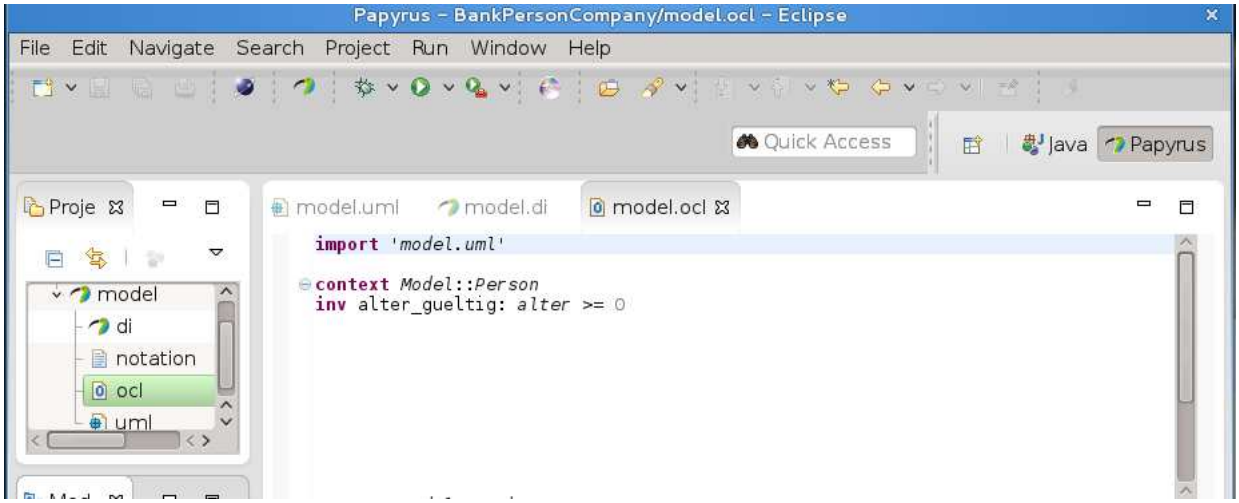
UML Class Diagrams: ..., Stereotypes, Class Templates, ...

S. Bauer: Eclipse für C, C++-Programmierer, 2. Aufl.
Eclipse Shortcuts

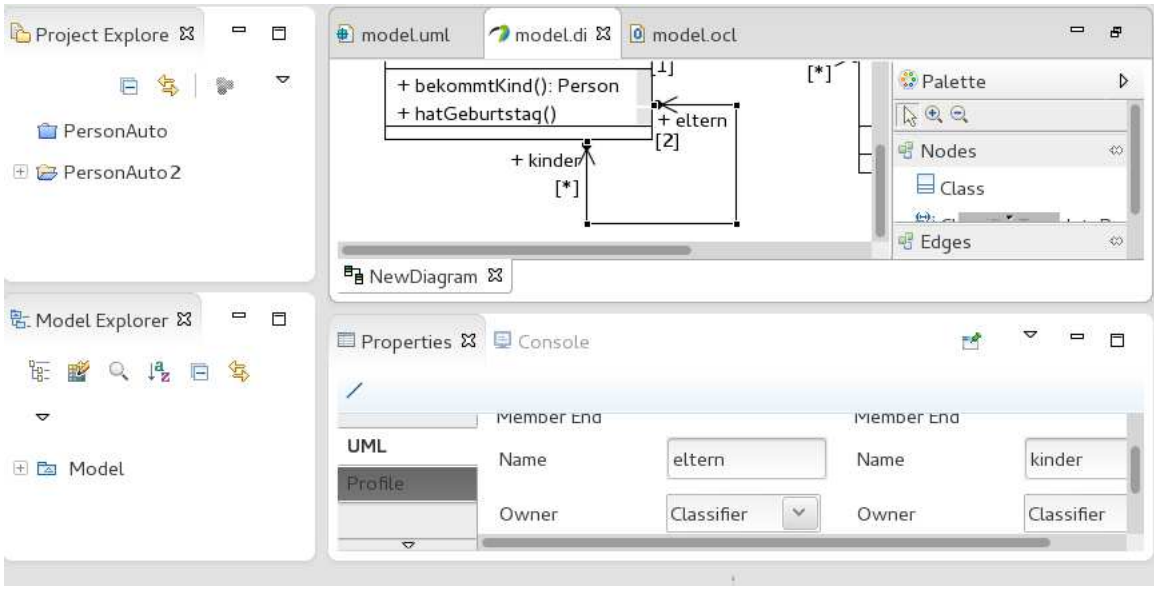
OCL 2.4 – Object Constrint Language

OCL in UML

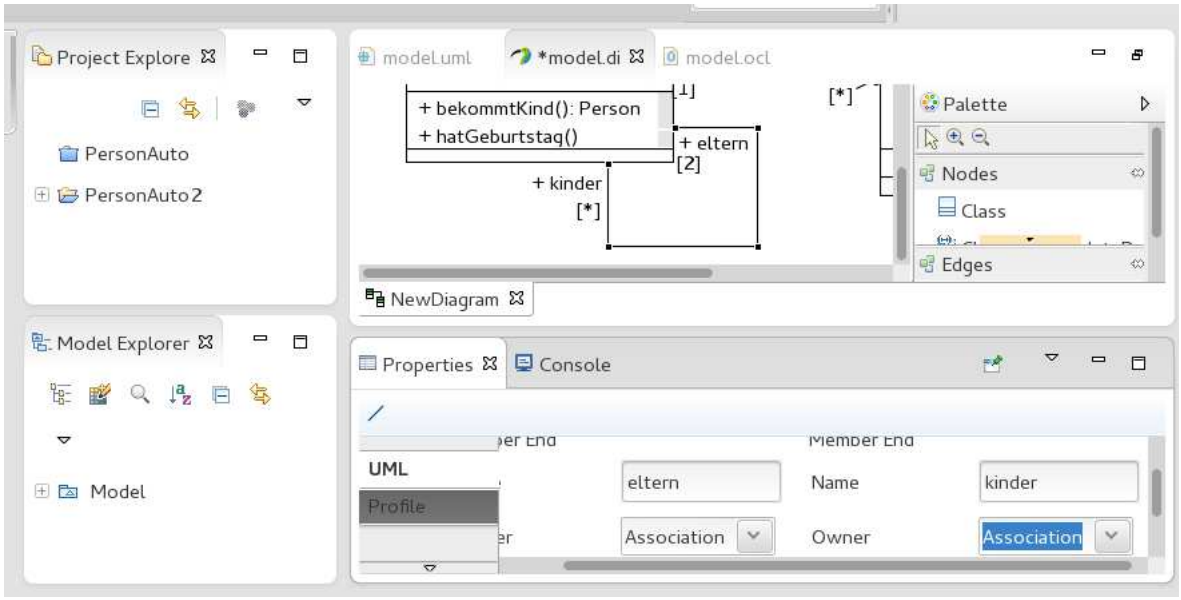
Besser mit allen Constraints in einer `model.ocl`-Datei:



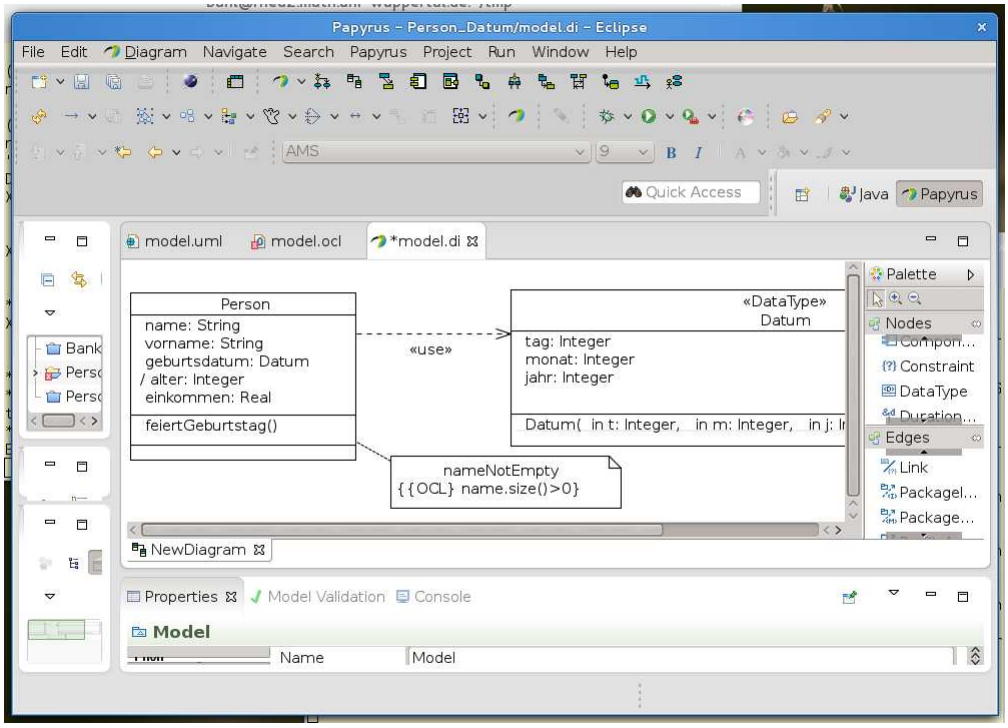
Wählen Sie bei der Assoziation zwischen `Person` und `Auto` die übliche Sichtweise von UML an:
Die Assoziationsrollen gehören der Klasse (dem Classifier) am anderen Assoziationsende und nicht der Assoziation selbst (leider die Papyrus-Voreinstellung).



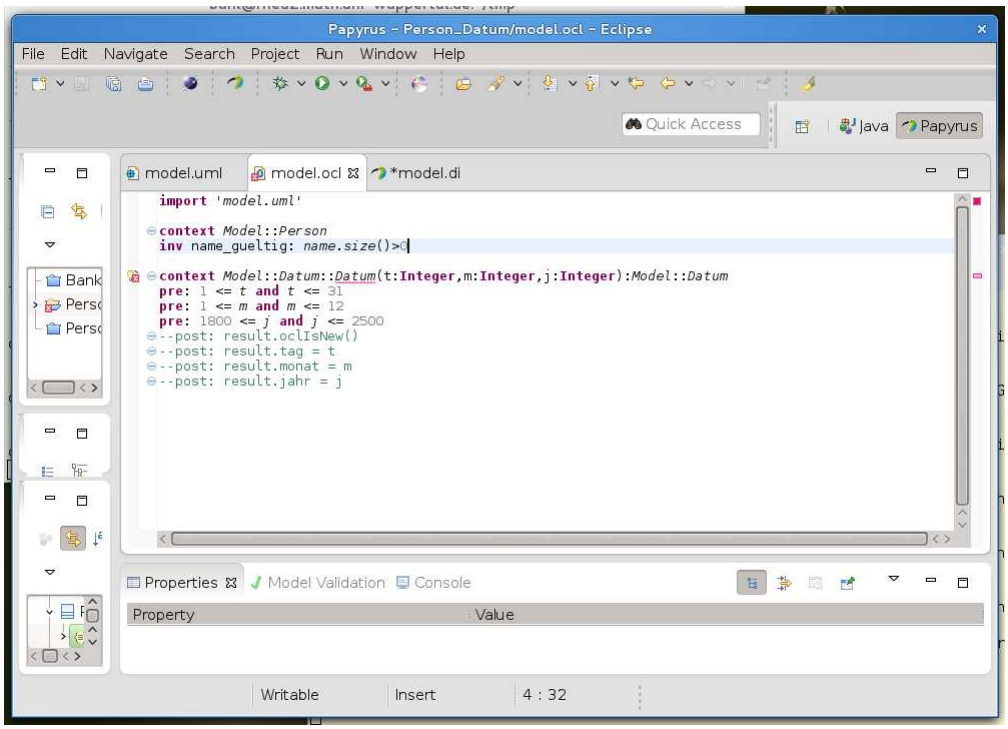
statt:


Project Explore
PersonAuto
PersonAuto2
Model Explorer
Model
model.uml
*model.di
model.ocl
+ bekommtKind(): Person
+ hatGeburtstag()
+ eltern
[2]
+ kinder
[*]
[1]
[*]
Palette
Nodes
Class
Edges
NewDiagram
Properties
Console
UML
Profile
Member End
eltern
Name
kinder
Association
Owner
Association

Datum mit einem Konstruktor (static, isquery):



({?} Constraint, Link) und vollständiger OCL-Spezifikation;



Ein erster (verbesserungsbedürftiger) Vertrag für `Datum`:

```
context Model::Datum::Datum( t : Integer,
                             m : Integer,
                             j : Integer ) : Model::Datum
pre: 1 <= t and t <= 31
pre: 1 <= m and m <= 12
pre: 1800 <= j and j <= 2500
post: result.oclIsNew()
post: result.tag = t
post: result.monat = m
post: result.jahr = j
```

Basic Structural Modeling with UML2 and OCL2, page 19..21:

What is OCL?

- Definition and Role (19)
- Characteristics (20)
- How does it complement UML? (21)

Softwarefehler

TomTom-Navigator und Schaltjahre
http://catless.ncl.ac.uk/Risks/26.77.html#subj11

Visa network down

...

Softwarekatastrophen als Rechtfertigung für den Aufwand formaler Spezifikation

Top 10 der Software-Katastrophen

- **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 Datenverarbeitungslauf einige Tausend der insgesamt etwa 1,5 Millionen angefallenenen 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

Neuauflage 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:

- 1982 stürzte ein Prototyp des F117 Kampfjets 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 Armeebaracke, 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 Plessezk 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 Magnetschwebebahn 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 insgesammt 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 rechzeitig 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 duch 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:

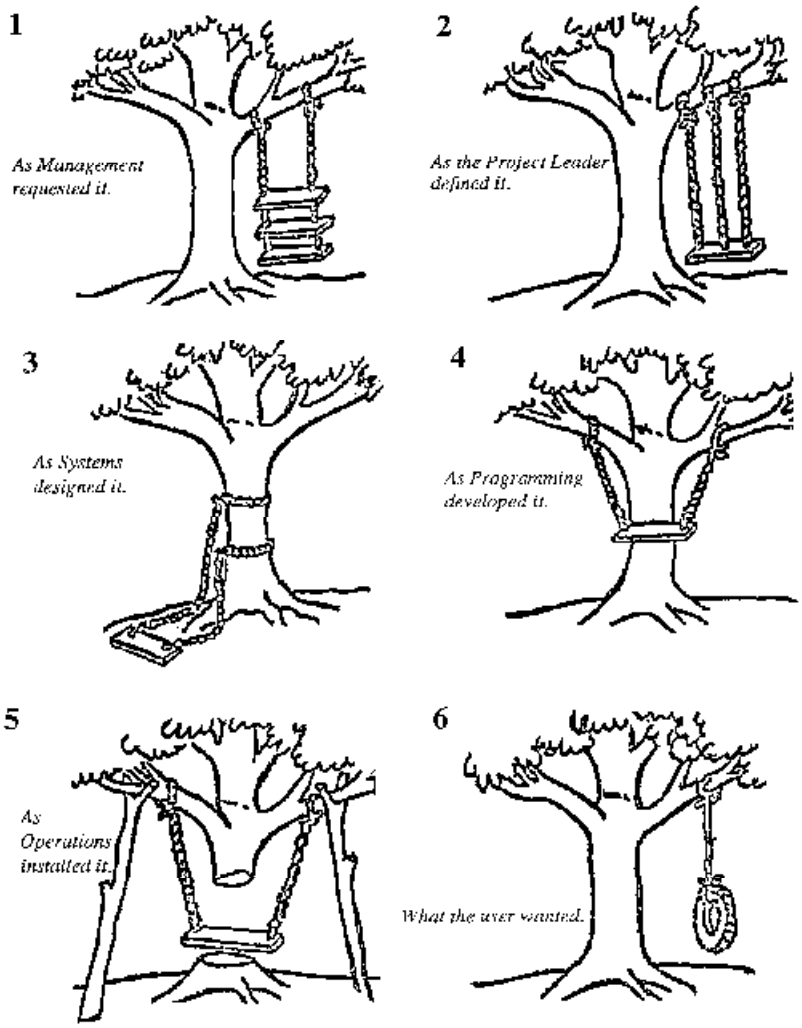
- 20 Famous Software Disasters
- Software bugs in the data reservoir
- Kleine BUGs, große GAUs
- Top 25 Most Dangerous Programming Errors

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

Softwareprobleme

... 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



Spezifikationsarten:

- 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:

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

  b) Eine Funktion kann **explizit** (durch Angabe einer Beispielimplementierung) spezifiziert werden:

  $min(r : Real) : Real$
  post: if self <= r then result = self else result = r endif

  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 expliziete (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))
...
```

Beispiele der Collection-Bibliothek OCLs:

Class Collection:

```
context Collection(T)::includes(object : T) : Boolean
post: result = (self->count(object) > 0)

context Collection(T)::size() : Integer
post: result = self->iterate(elem;
                 acc : Integer = 0 | acc + 1)

context Collection(T)::isEmpty():Boolean
post: result = ( self->size() = 0 )

context Collection(T)::max():T
post: result = self->iterate( elem;
            acc : T = self.first() | acc.max(elem) )
```

...

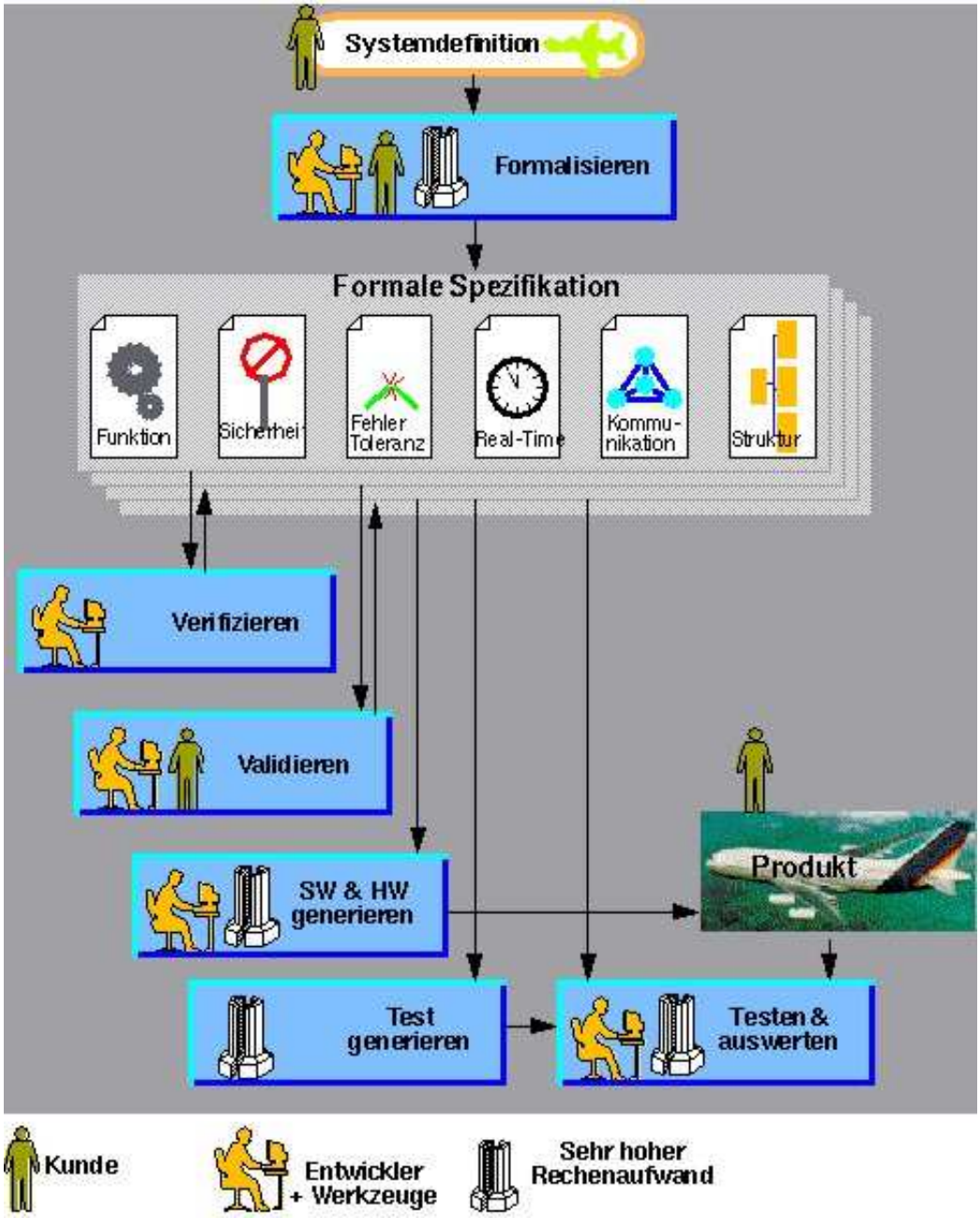
(Vergleiche Seite 157... der OCL-Spezifikation.)

Analog eine implizite Spezifikation der ganzzahligen Wurzel:

```
context isqrt(): Integer
pre: self >= 0
post: result >= 0
post: result * result <= self
post:  (result + 1) * (result + 1) > self
```

Vor- und Nachbedingungen 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)

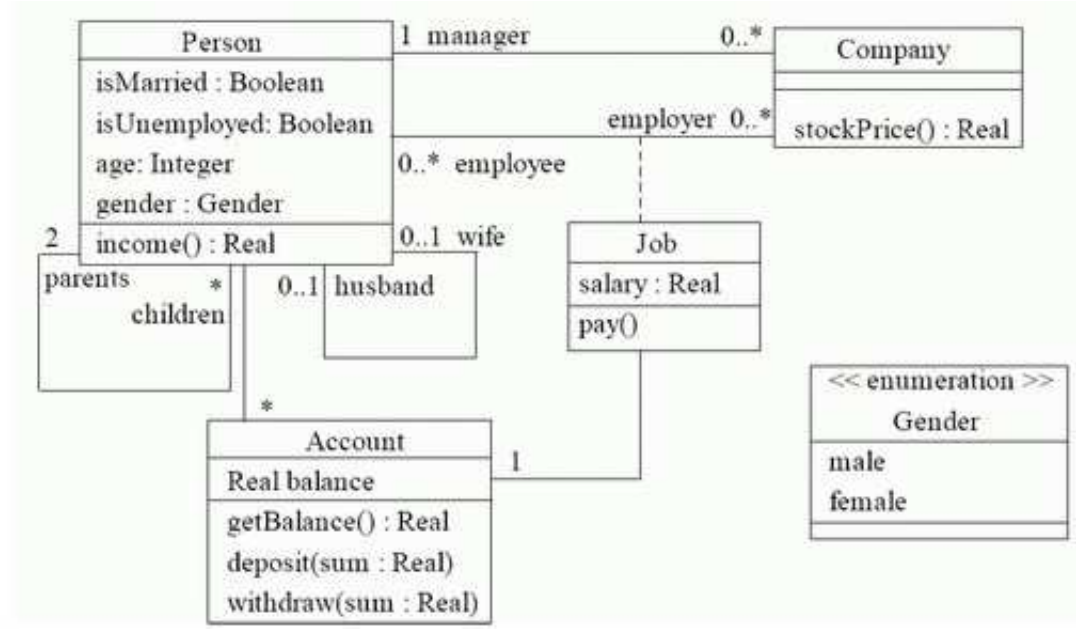


Beispiele:

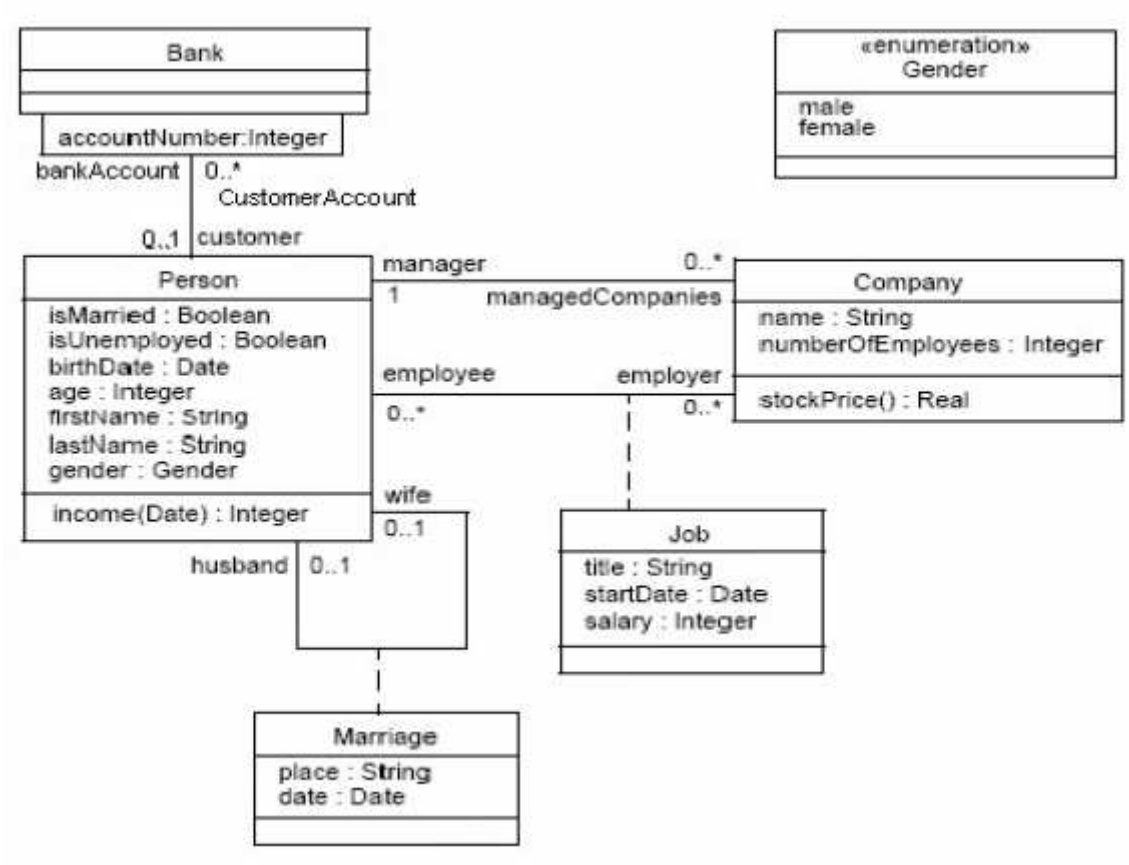
- PersonAuto, siehe OCL-Beispiele (Wikipedia) und Seite 11 sowie 13-16 der Materialsammlung sowie Übungsblatt 1.

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

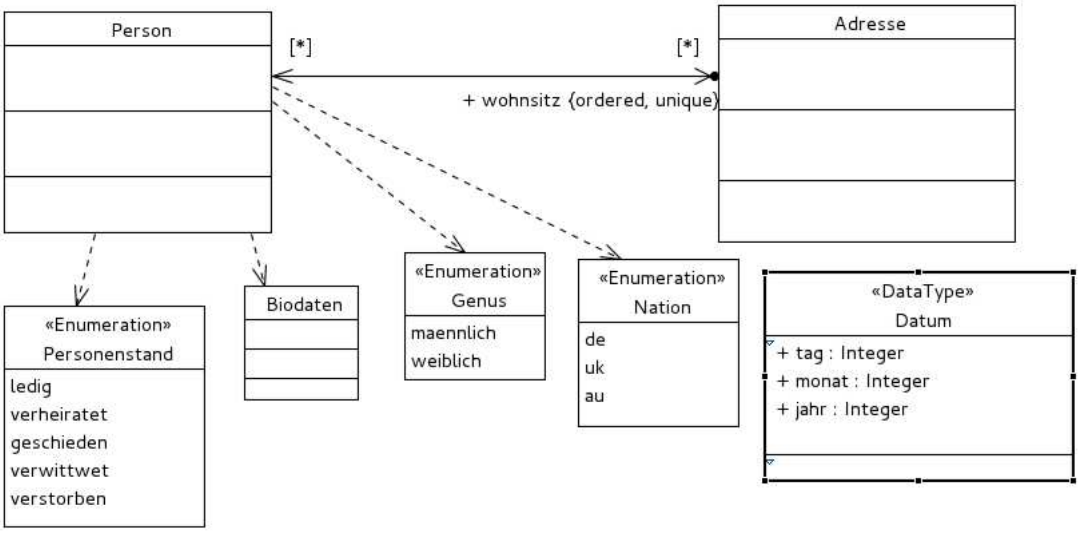
-



- 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**

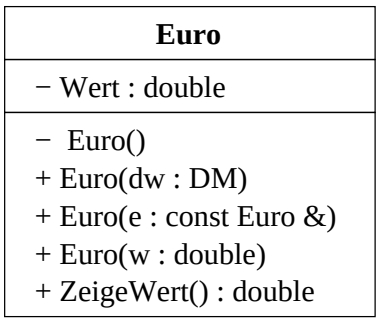


Abbildung 0.1.: Die Klasse Euro

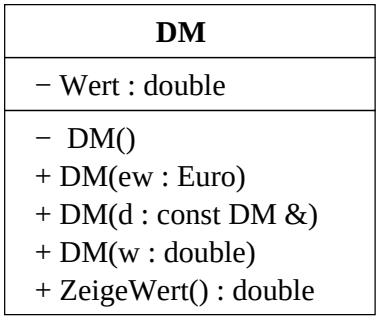


Abbildung 0.2.: Die Klasse DM

als instantiierbare Kinder einer abstrakten Klasse `Waehrung`.

Eine Anwendung:

Original mit anonymer Geldeinheit (double)

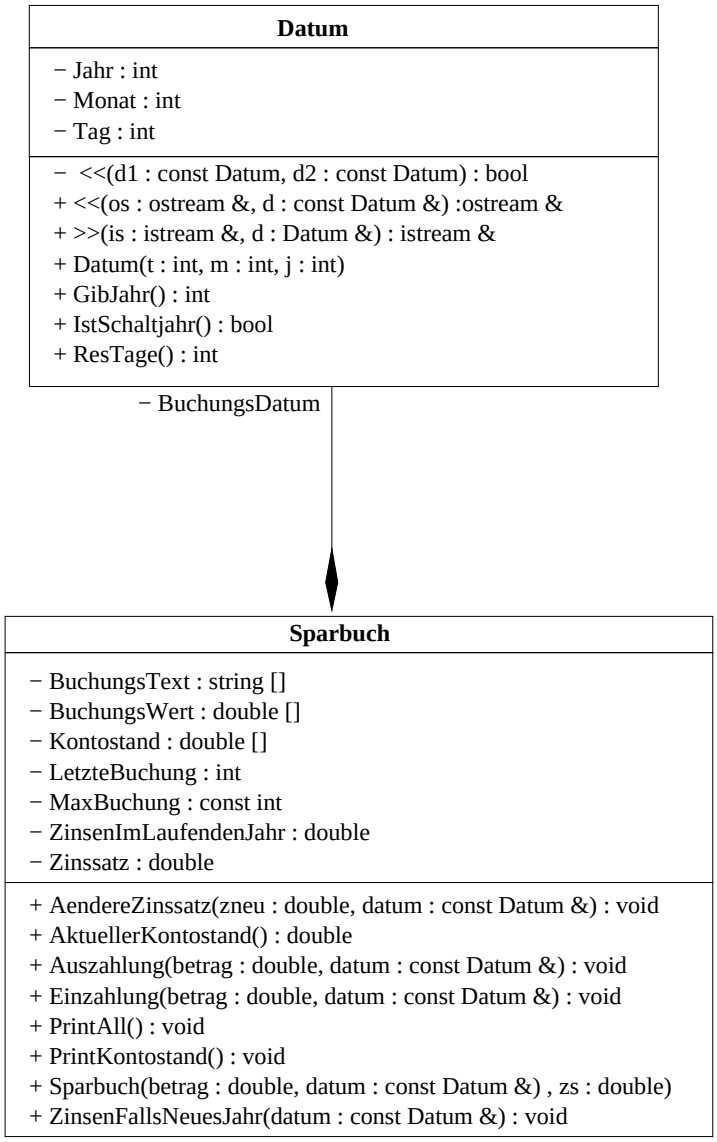


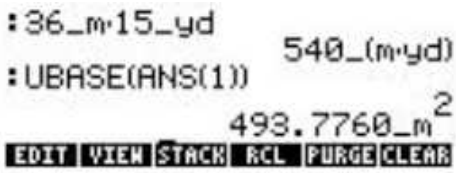
Abbildung 0.3.: Die Klassen Datum und Sparbuch

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

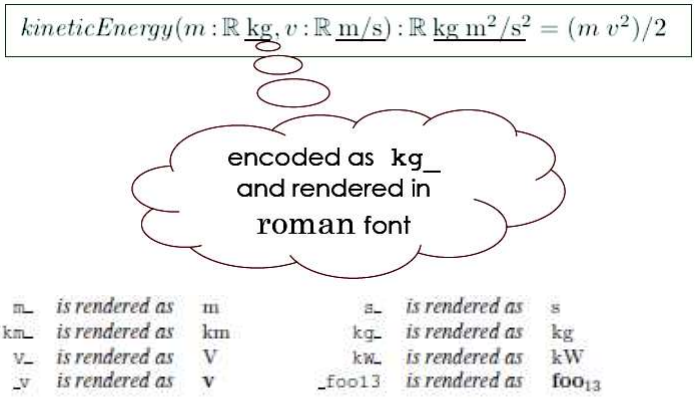
```
//////////////////////////////////////////
// Datei:    DM_Euro.cc
// Version:  1.1
// Zweck:    DM und Euro
// Autor:    Holger Arndt
// Datum:    23.05.2001
//////////////////////////////////////////

#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;
```

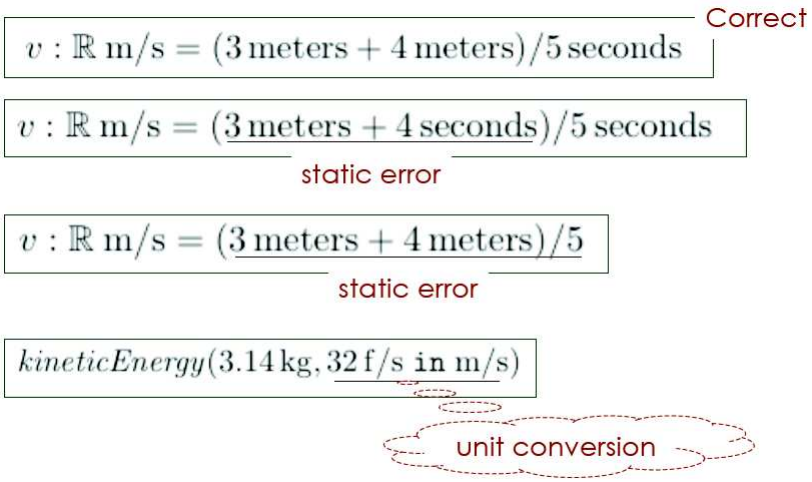
```
}

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:
  - HP 50g: Working with Units



  - Units und Dimensions in Fortress:





  - Units und 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)
```

– Boost.Units

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
```

– Einheitenrechnung in *Mathematica:*

Naturwissenschaftlich/Technische Einheiten
Currency Units

```
In[1]:= UnitConvert[Quantity[19., "SwissFrancs"], "Euros"]

Out[1]= €15.7489

In[2]:= Quantity[1, "USDollars"] + Quantity[1, "Euros"]

Out[2]= $2.2939
```

– Einheitenrechnung und Dimensionsanalyse mit einfachen Temlate-Klassen:

... to Handling Scientific Quantities ...

Benutzung aggressiver Laufzeit-Zusicherungen zur Qualitätsverbesserung von Software:

Heartbleed and Formal Methods
How to Prevent the next Heartbleed
Acceptance of Formal Methods: Lessons from Hardware Design — the FDIV bug

1. UML und SdV

1.1. Rekapitulation: UML-Klassendiagramme

1.1.1. Klassen und Objekte/Instanzen

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

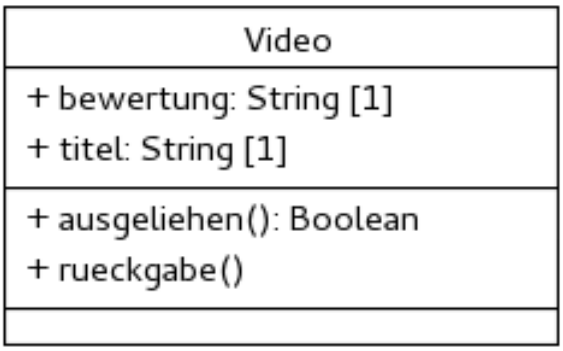


Abbildung 1.1.: Eine Klasse

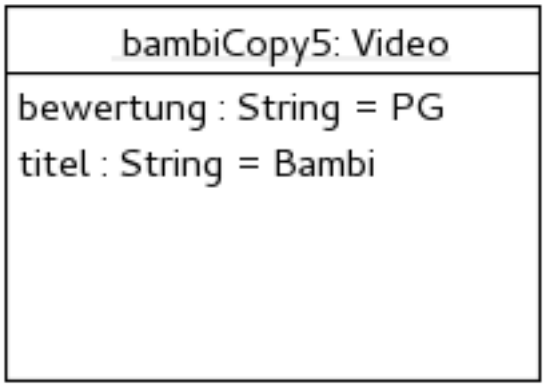
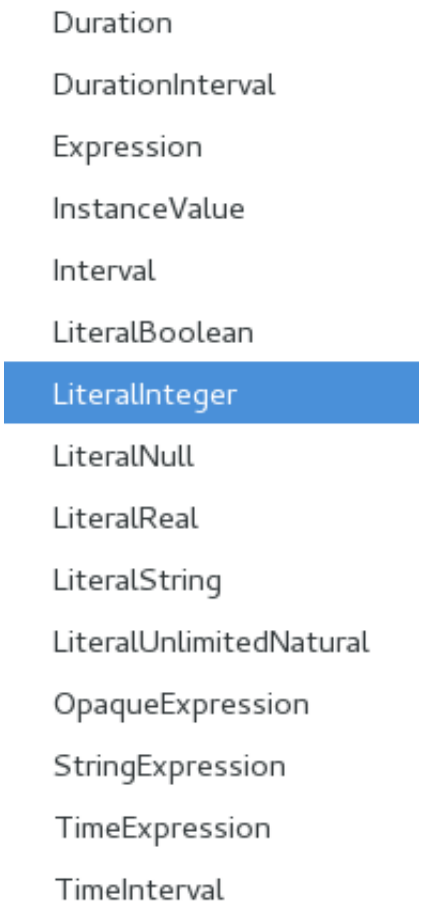


Abbildung 1.2.: Ein Objekt dieser Klasse(InstanceSpecification)

```
<<primitive>> UML-Datentypen:


Boolean
String
Integer
Real
UnlimitedNatural
```

Literale für Instance-Slots:



Siehe http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF
und http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF
sowie http://www.omg.org/spec/UML/2.5/Beta2/.

1.1.2. Klassenspezifikation

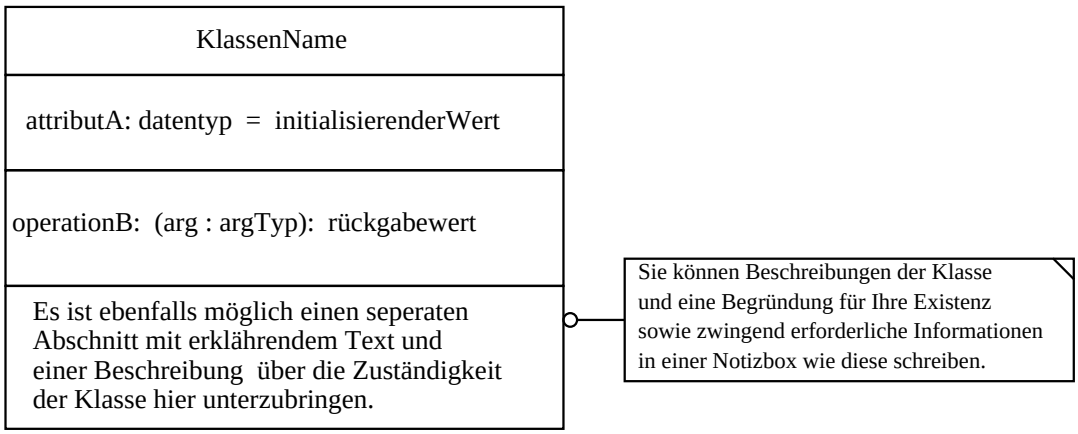


Abbildung 1.3.: Spezifikation einer Klasse

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>> 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)
$\sim$ package

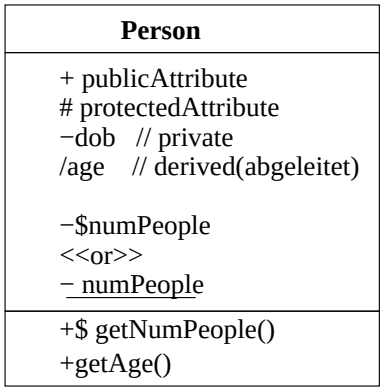


Abbildung 1.4.: Eine Klasse: Person

- 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.

```
+ Vorname: String[1..*] {ordered} -- eine Sequence oder
                                  -- OrderedSet
                                  -- je nachdem, ob
                                  -- Mehrfachvorkommen
                                  -- einzelner Elemente
                                  -- moeglich (unique oder
                                  -- nonunique)
+ kind: Person[*]                 -- ein Bag oder Set
                                  -- je nachdem, ob
                                  -- Mehrfachvorkommen
                                  -- einzelner Elemente
                                  -- moeglich (unique
                                  -- oder nonunique),
                                  -- unordered ist default
+ geburtsDatum : Datum {{ocl} geburtsDatum <= Datum::today()}
/ alter: Integer {{ocl} alter = Datum::today() - geburtsDatum}
```

{ordered}	{ordered unique}	OrderedSet
{ordered nonunique}		**Sequence**
{unordered}	{unordered unique}	**Set**
{unordered nonunique}		**Bag**

1.1.3. Links und Assoziationen

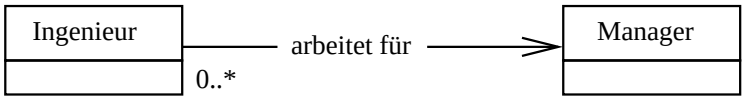


Abbildung 1.5.: Assoziationen verbinden Klassenexemplare

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.

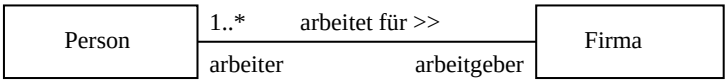


Abbildung 1.6.: Rollen in Klassen

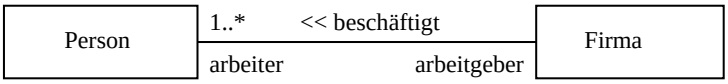
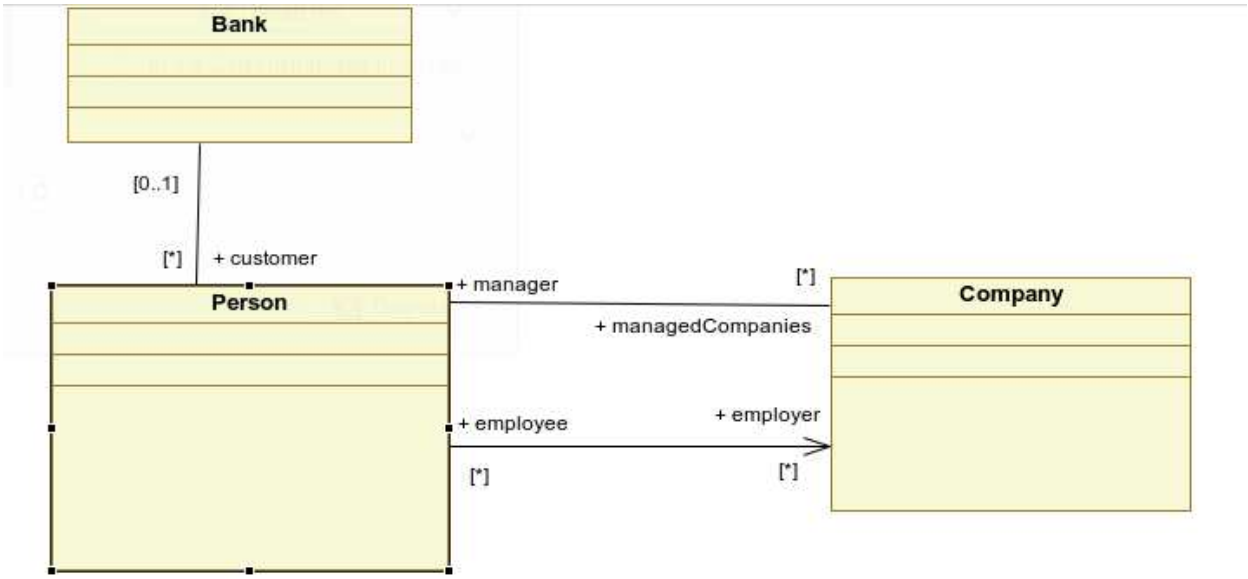


Abbildung 1.7.: Rollen in Klassen (Fortsetzung)

Einige Beispiele:



Beispiel TeamMeetingPerson (Seite 10)

1.1.5. Multiplizitäten (Kardinalitäten)

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

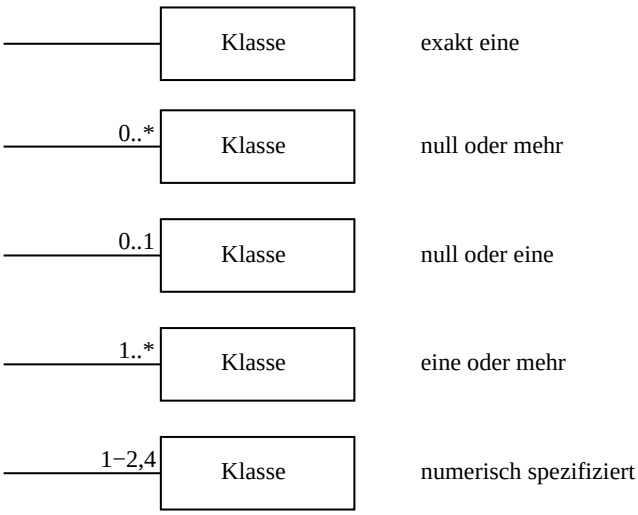


Abbildung 1.8.: Multiplizität

Anmerkung: * kann anstelle von 0..* verwendet werden.

1.1.6. Assoziazionsklassen

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>>` erklährt den Zeitaspekt der Beziehung: Er besagt, das 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.

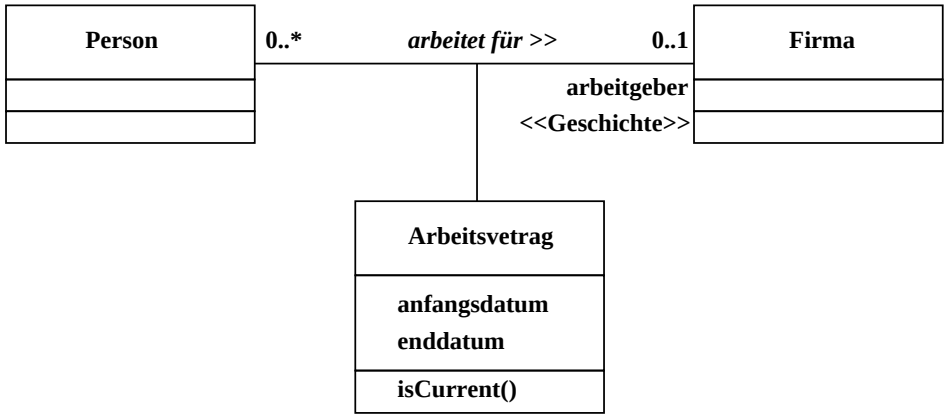


Abbildung 1.9.: Assoziierte Attribute

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

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

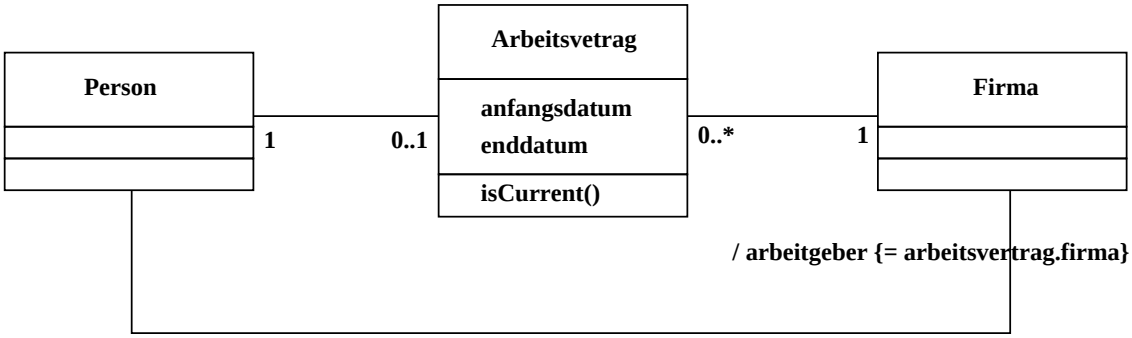


Abbildung 1.10.: Assoziiertes Attribut (Fortseztung)

1.1.7. 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**.

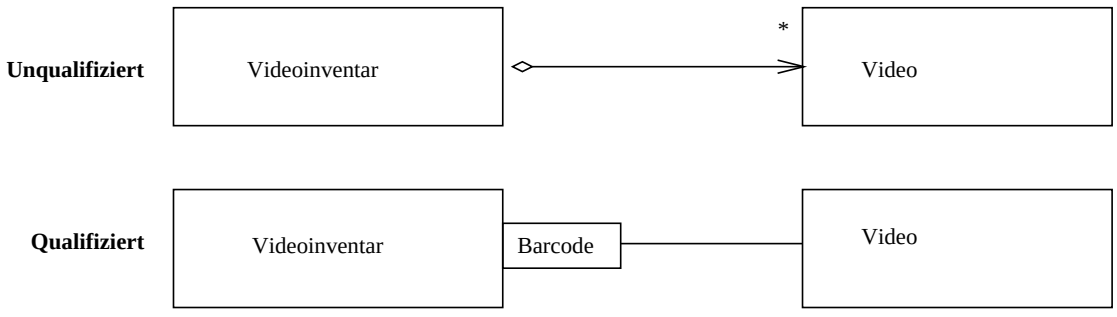


Abbildung 1.11.: Qualifizierte Assoziation

1.1.8. UML Superstructure: Classifier und Class

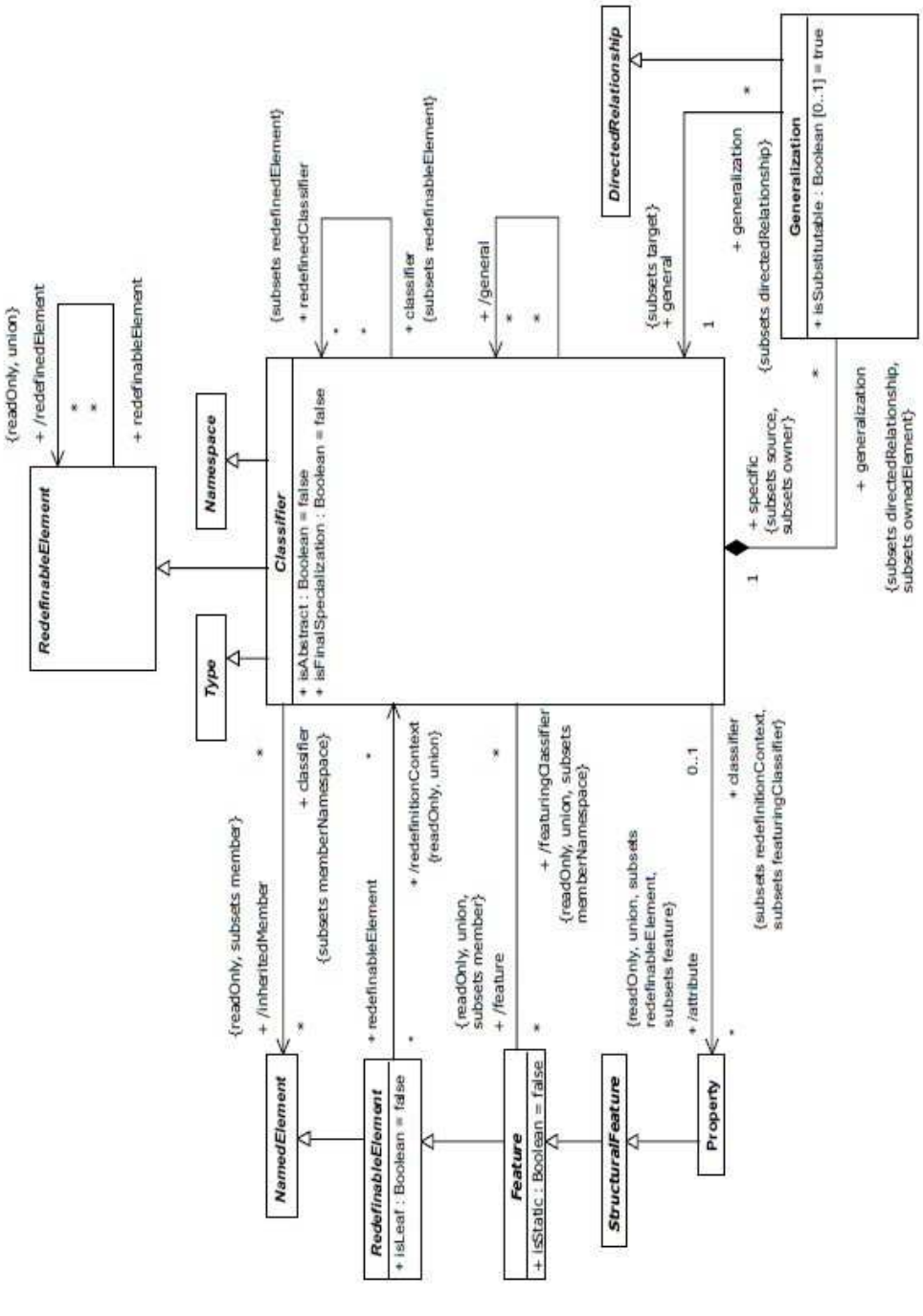


Figure 7.9 - Classifiers diagram of the Kernel package

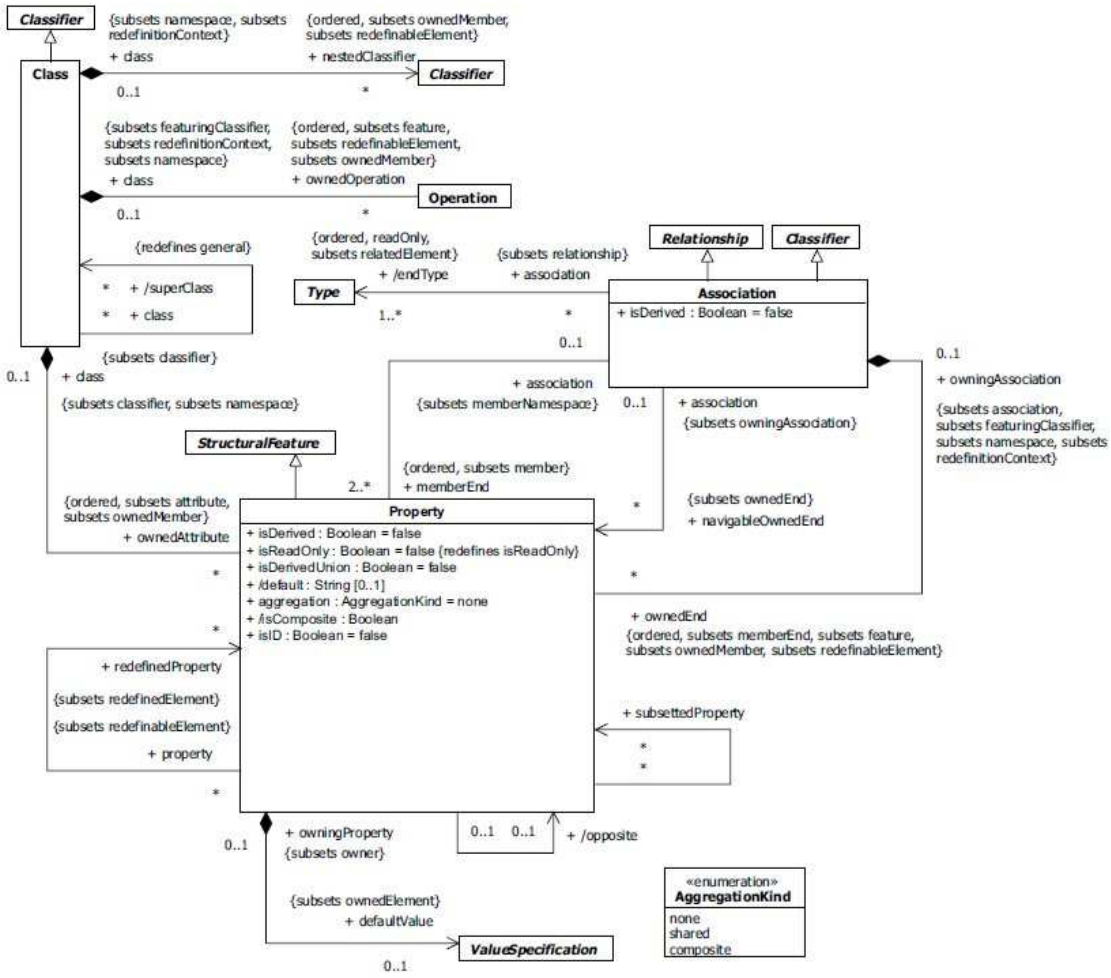


Figure 7.12 - Classes diagram of the Kernel package

1.1.9. 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 >>
```

Andere gebräuchliche Stereotypen sind:

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

UML model element stereotypes
http://de.wikipedia.org/wiki/Stereotyp_(UML)

1.1.10. Tagged Values/Stereotype Attributes

- Tagged Values sind ein weiterer Mechanismuss, 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}

Tagged Values in *Visual Paradigm*

„UML2 requires all of the tagged values (now called stereotype attributes) to now be contained underneath a Stereotype, rather than be independent values as in UML14.“ (aus: Re: UML tagged value with papyrus)

1.1.11. Mehrgliedrige Assoziationen

n-äre Assoziation

1.1.12. 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.

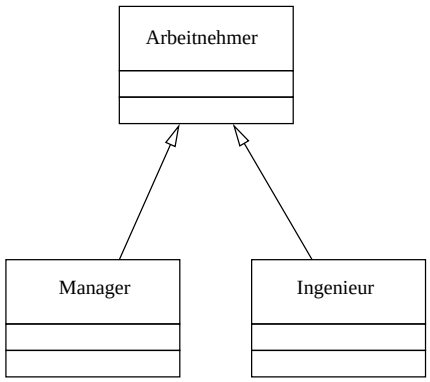


Abbildung 1.12.: Generalisierung, Spezialisierung und Vererbung

1.1.13. Mehrfachvererbung in Status und/oder Verhalten

Rautenproblem
Java Interfaces: rein abstrakt und ohne Implementierung von Verhalten, aber mehrfach
Java 8 vs Scala: a Feature Comparison: Java mit mehrfach vererbtem Verhalten
Java 8: virtual extension methods vs abstract class
Java 8 explained: Default Methods
default methods
Java 8 Default Methods Tutorial

1.1.14. 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.

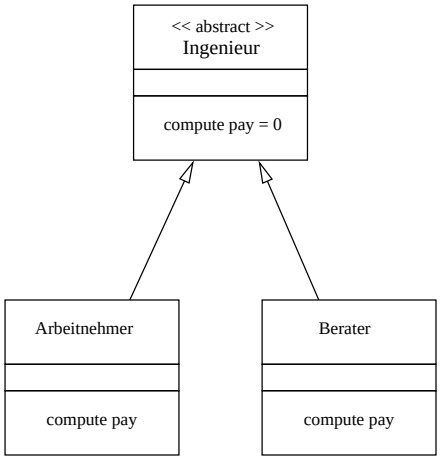


Abbildung 1.13.: Abstrakte Klassen

1.1.15. Komposition / Aggregation

Das Rautenzeichen wir 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.

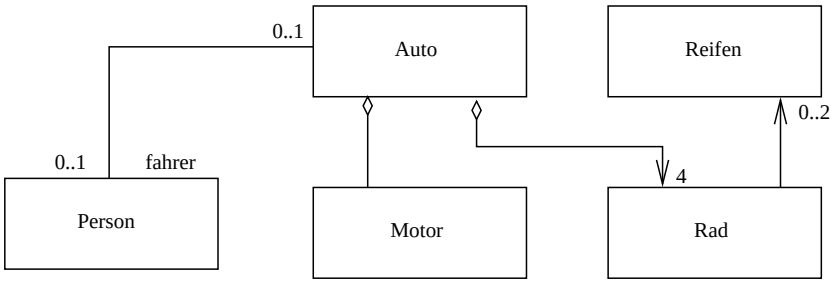


Abbildung 1.14.: Komposition / Aggregation

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.

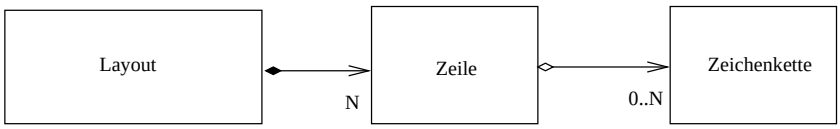
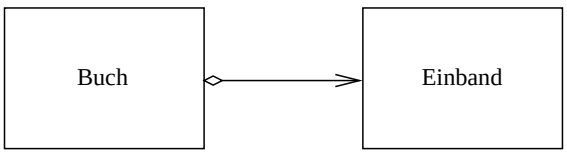


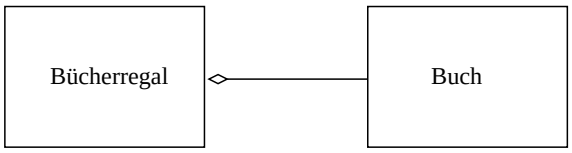
Abbildung 1.15.: Komposition zwischen Layout und Zeile

- 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 Kentniss von seinen Instanzen der Klasse Buch.

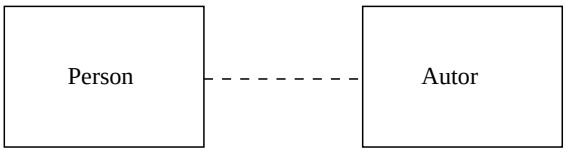


Assoziation:

- 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.



1.1.16. template classes

http://www.uml-diagrams.org/template.html
Class Template
Templates
Template Types

1.1.17. Modell und Metamodell

UML Metamodell
4-Schichten-Architektur von UML(Seite 10)

1.1.18. UML 2.5: Mai 2013

MOF 2.4.1
Unified Modeling Language (UML) Version 2.5
UML 2.5 ist verabschiedet

Figure 7.14 shows an example of a Constraint in a note symbol.

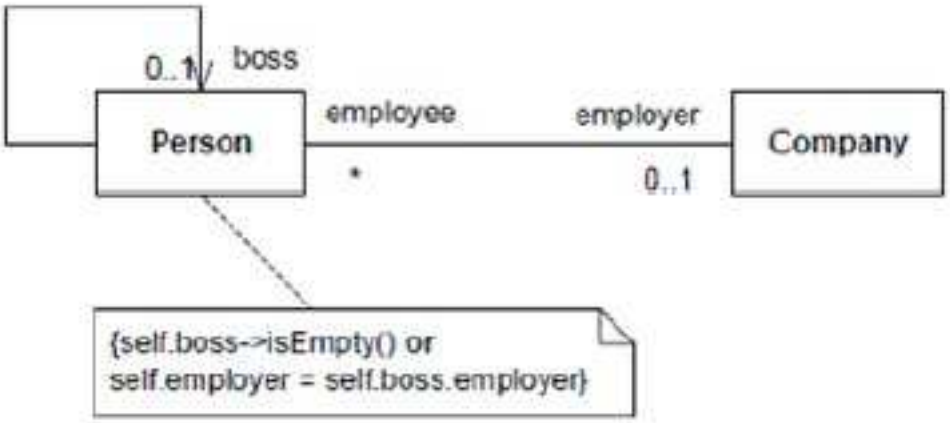


Figure 7.14 Constraint in a note symbol

Figure 7.15 shows a constraint string attached to an attribute.

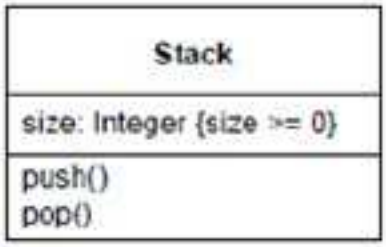


Figure 7.15 Constraint attached to an attribute

Figure 7.16 shows an {xor} constraint between two associations

Besser (vollständigerges “guarding“) wäre für die Klasse Person:

```
(self.boss->notEmpty() and self.boss.employer->notEmpty() and
self.employer->notEmpty()) implies
self.employer = self.boss.embloyer
```

1.1.19. UML 2 Style Guidelines

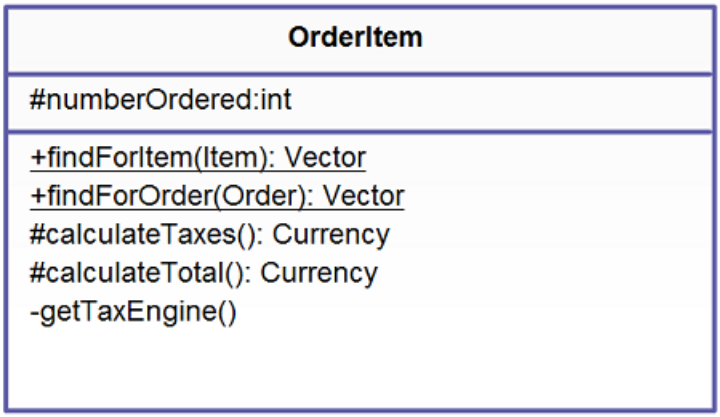
UML Style Guidelines

Draft, Thu Mar 1
RIT

The following rules are numbered so that they may easily be referred to when grading.

1. Although strictly speaking, one use case could adequately describe some systems you design, strive for having at least two or three. For small projects, this may violate standard notions of the granularity of use cases.

http://www.cs.rit.edu/ cs4/UML_style.html



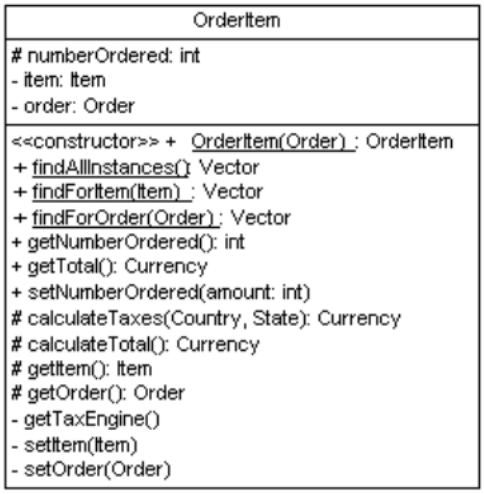
Put common terminology for names

Choose complete singular nouns over class names

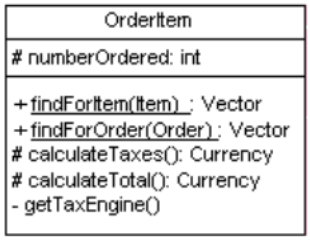
http://creately.com/diagram-type/article/simple-guidelines-drawing-uml-class-diagrams

with and without scaffolding

With Scaffolding



Without Scaffolding



http://ejavaschool.com.ne.kr/uml/uml_sun.ppt

1.2. Spezifikation einfacher Klassen nach Prinzipien der SdV

1.2.1. Ein einfaches Beispiel ...

KEY:Class, VALUE:Class

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 */
/* ... */

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

Klassifikation der Methoden in

- grundlegende Abfragen (Queries/Observatoren/unverzichtbare const-Methoden)
- abgeleitete Abfragen (Queries/Observatoren/redundante const-Methoden)
- Aktionen (Modifikatoren/non-const-Methoden ohne jeden Ergebniswert)
- Konstruktoren/Destruktoren

Siehe dazu zum Beispiel:

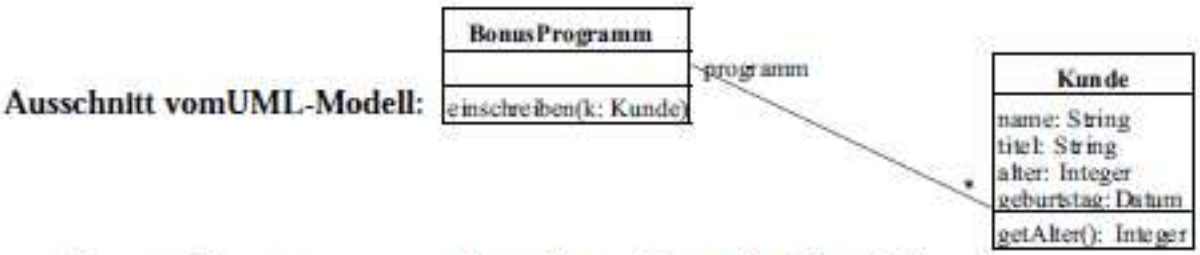
Spezifikation durch Vertrag — eine Basistechnologie für eBusiness

Das typische Aussehen von Vertägen 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://web.archive.org/web/20030803235217/http://wwwbruegge.in.tum.de/teaching/ss01/Info2/vorlesung/folien/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 boolschen 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:

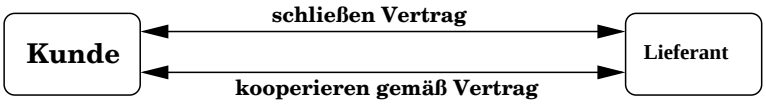


Abbildung 1.16.: Kunden-Lieferanten-Modell

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 vermeiden[2].

SdV	Pflichten	Nutzen
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 Contrakt“ ist ein Warenzeichen von Interactive Software Engeneering.
[2] In 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 boolsche Ausdrücke über den Abfragen der Komponente; inhaltlich können sie z.B. Geschäftsregeln (busisness rules) ausdrücken.

- **Vorbedingungen** (preconditions) eines Dienstes sind Bedingungen, die vor dem Aufruf eines Dienstes erfüllt sein müssen, damit er ausführbar ist. Vorbedingungen sind boolsche 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 boolsche Ausdrücke über den Abfragen der Komponente und den Parametern des Dienstes, erweitert um ein Gedächniskonstruckt (`@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
Pflichten	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.

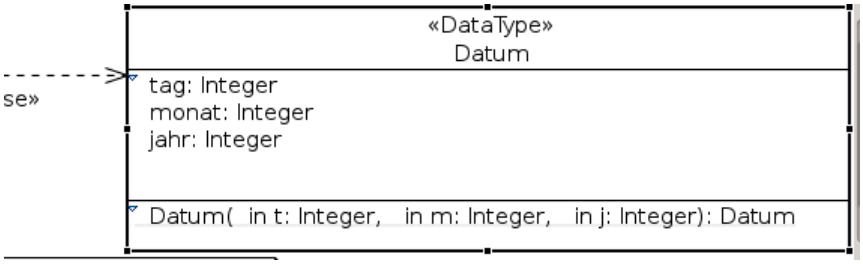
1.2.4. Native C++17(?)-Codeverträge

```
double sqrt( double r )
    precondition
    {
        r >= 0.;
    }
    postcondition( result )
    {
        equal_within_precision( result * result, r );
    }



int factorial( int n )
precondition
{
    0 <= n && n <= 12;
}
postcondition( result )
{
    result >= 1;
}
{
    if ( n < 2 )
        return 1;
    else
        return n * factorial( n - 1 );
}

template< class T >
class vector
{
    invariant
    {
        ( size() == 0 ) == empty();
        size() == std::distance( begin(), end() );
        size() == std::distance( rbegin(), rend() );
        size() <= capacity();
        capacity() <= max_size();
    }

              void resize( size_type newsize )
                postcondition
                {
                    size() == newsize;
                    if( newsize > oldof size() )
                        all_equals( begin() + oldof size(), end(), T() );
                }

             void clear();
                postcondition { empty(); }

             void swap( vector& right )
                postcondition
                {
                    oldof *this == right;
                    oldof right == *this;
                }
             // ...

}; // class 'vector'
```

(Proposal to add Contract Programming to C++ (revision 4),
siehe insbesondere: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1962.html#vector-example-hpp)

1.2.5. Beispiel-Codeverträge

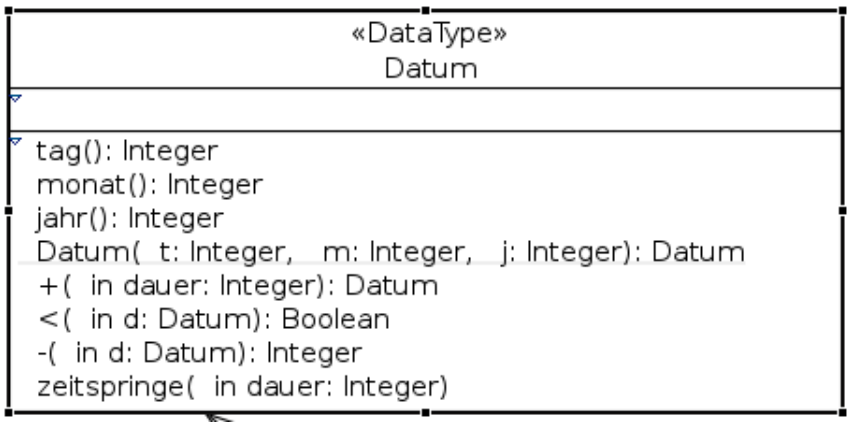
Zunächst eine SdV-konforme Umkonzeption des `<<datatype>> Datum` im Sinne der methodischen Restriktion:
aus



mit

```
import 'model.uml'
context Model::Datum::Datum(t:Integer,m:Integer,j:Integer):
    Model::Datum
pre: 1 <= t and t <= 31
pre: 1 <= m and m <= 12
pre: 1800 <= j and j <= 2500
post: result.oclIsNew()
post: result.tag = t
post: result.monat = m
post: result.jahr = j

context Model::Datum
inv: 1 <= tag and tag <= 31
inv: 1 <= monat and monat <= 12
inv: 1800 <= jahr and jahr <= 2500
```

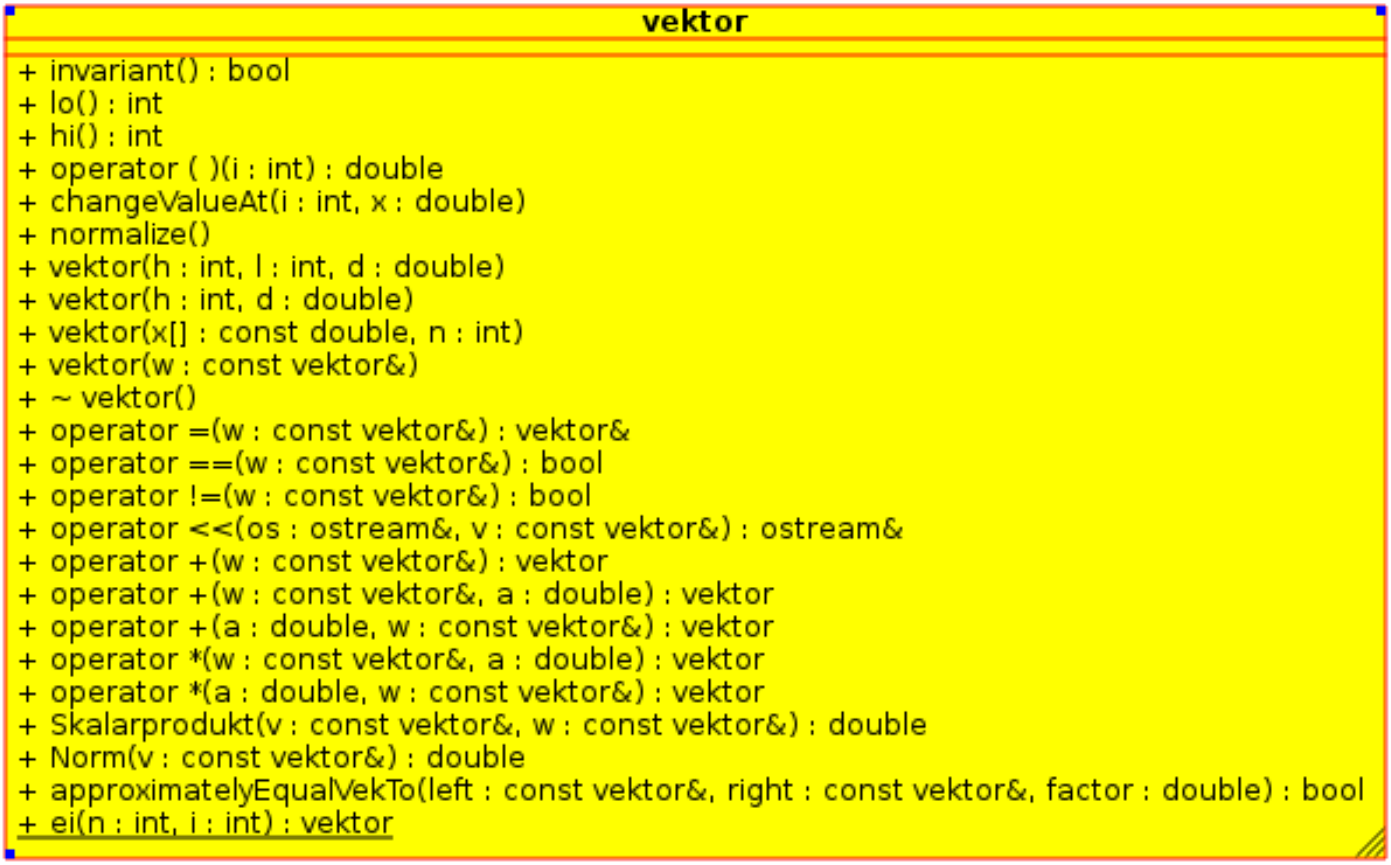
wird:



```
import 'model.uml'
context Model::Datum
inv: 1 <= tag() and tag() <= 31
inv: 1 >= monat() and monat() <= 12
inv: 1800 <= jahr() and jahr() <= 2500

context Model::Datum::Datum(t: Integer, m: Integer, j: Integer)
    : Model::Datum
pre: 1 <= t and t >= 31
pre: 1 <= m and m <= 12
pre: 1800 <= j and j <= 2500
post: result.oclIsNew()
post: result.tag() = t
post: result.monat() = m
post: result.jahr() = j

-- ...
```

Aufgabe:
Überlegen Sie sich Codeverträge für die Operationen `+`, `<`, `-`, `zeitspringe(.)`.

Beispiel-Codeverträge in der C++-Macro-Erweiterung `Gnu Nana`
(siehe http://savannah.gnu.org/projects/nana/)



Beispiel-Codeverträge 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;
}
```

Machine epsilon
Floating point
What Every Computer Scientist Should Know About Floating-Point Arithmetic

- 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 Einheitsvktor (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 ungeaendert");
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;
}
```

1.2.5.1. 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 Erfülltseins 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.

$Invariante_{Kindklasse} = Invarinte_{Elterklasse}$ and ...
$Vorbedingung_{Kindmethode} = Vorbedingung_{Eltermethode}$ or ...
$$Nachbedingung_{Kindmethode} = \begin{cases} Nachbedingung_{Eltermethode} \text{ and } ... & \text{, falls } Vorbedingung_{Eltermethode} \\ \text{beliebig} & \text{, sonst} \end{cases}$$

1.2.5.2. 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) modificators:
  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) modificators: (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):

Konstruktor:
{$\mathrm{VB}_{Parameter}$} Konstruktor {Inv and $\mathrm{NB}_{Konstruktor}$}

Destruktor:
{Inv} Destruktor {-}

Jede andere (öffentliche) Methode M:
{VB_M and Inv} M {NB_M and Inv}

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

1.2.5.3. Funktion invert (Invertieren einer Matrix)

Beispiel in Eifel:

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

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

1.2.5.4. 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)
.
.
.
```

1.2.5.5. 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 ||"
```

1.2.5.6. 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
— ...

1.2.5.7. 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.

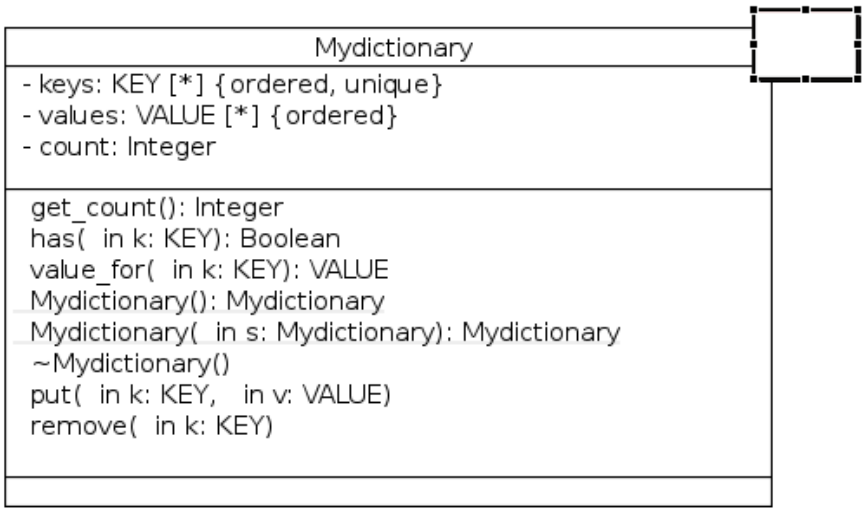
4. Für jeden Konstruktor/Modifikator schreibe eine Nachbedingung, die die Werte aller grundlegenden Observatoren am Ende einer Methode festlegt.

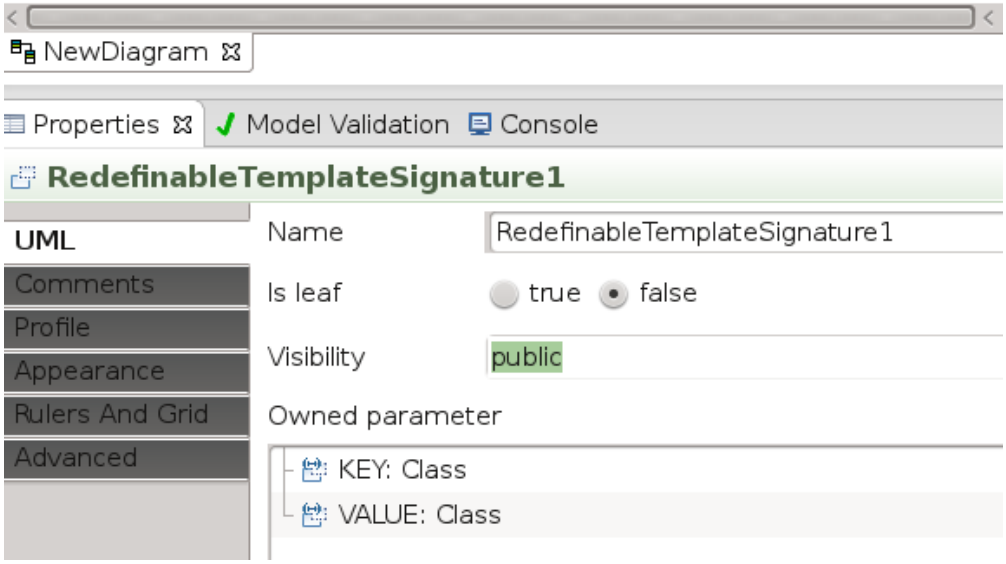
5. Für jeden Observator und jeden Konstruktor/Modifikator schreibe notwendige Vorbedingungen.

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.

1.2.6. ... und sein (OCL2-)Codevertrag






```
import 'model.uml'

context Model::Mydictionary
inv: keys <> null
inv: values <> null
inv: count >= 0
inv: keys->size() = values->size()
inv: keys->size() = count
/* Inv. der priv. Attribute fuer das Implementierungs-Team */
```

```
/* basic observators */
context Model::Mydictionary::get_count() : Integer
body: count
/* post: result = Model::KEY->select(k | has(k))->size() */
context Model::Mydictionary::has(k : Model::KEY) : Boolean
post consistentWithCount: get_count()=0 implies not result
context Model::Mydictionary::value_for(k: Model::KEY): Model::
   VALUE
pre: has(k)

/* constructor */
context Model::Mydictionary::Mydictionary(): Model::
   Mydictionary
post: result.oclIsNew()
post: result.get_count() = 0

context Model::Mydictionary::Mydictionary(s : Model::
   Mydictionary): Model::Mydictionary
post: result.oclIsNew()
post: s.get_count() = result.get_count()
/* post: KEY->forAll(k | s.has(k) implies s.value_for(k) =
   result.value_for(k)) */

/* modificator */
context Model::Mydictionary::put(k: Model::KEY, v: Model::VALUE
   ): OclVoid
pre: not has(k)
post: has(k)
post: get_count() = get_count@pre() + 1
post: value_for(k) = v
/* post: Model::KEY->forAll(kl | has@pre(kl) implies
   value_for@pre(kl) = value_for(kl)) */

context Model::Mydictionary::remove(k: KEY): OclVoid
pre: has(k)
post: not has(k)
post: get_count() = get_count@pre() - 1
/* post: Model::KEY->forAll(kl | has@pre(kl) implies (kl = k or
                          value_for@pre(kl) = value_for(kl))) */
```

Wahrscheinlich muß es später

```
...
context Model::Mydictionary(KEY,VALUE)
inv: keys <> null
```

...

heißen.

Aufgaben:

- Warum sind die oben genannten `forAll()`-Nachbedingungen und die `KEY->select`-Anweisung auskommentiert?
- Ändern Sie das Design von `mydictionary`, indem Sie `has()` zum abgeleiteten Observator machen und einen neuen grundlegenden Observator `usedKeys(): KEY [*]` einführen. Welchen Vorteil hat das?
- Schreiben Sie die Codeverträge für dieses neue Design.
- Wie sieht es mit den expliziten Framebedingungen bei diesem neuen Design aus?

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

Aus: http://www.cs.uwlax.edu/~riley/CS220F12/lectures/3.1-LectO2_Specs.pdf#page=5

Abbildung 1.17.: Die Standard Farbklasse: 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<=redness(c)<=255 **and** 0<=greenness(c)<=255 **and**
0<=blueness(c)<=255 **and** 0<=opaqueness(c)<=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
```

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 ==
      bluness/ 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/basic 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 <> nullptr,<> 0, <> `null` oder `->notEmpty()`)

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

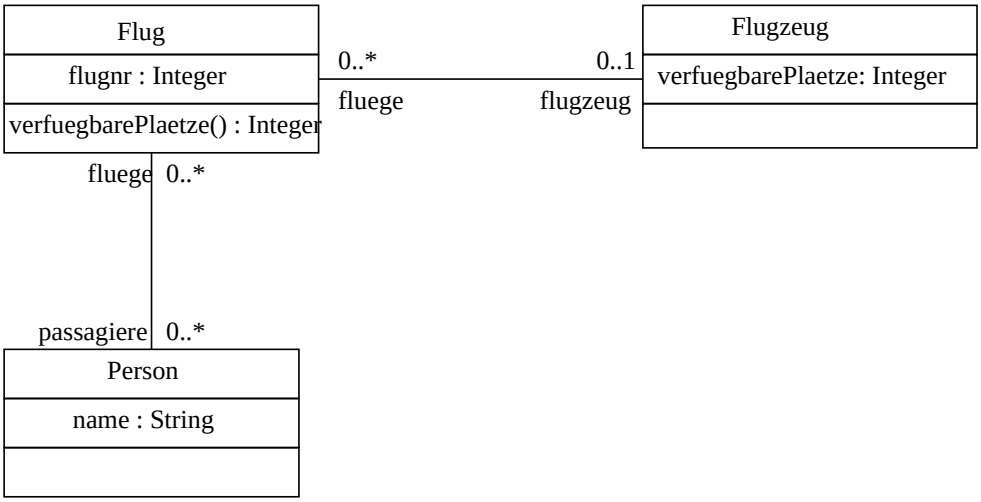


Abbildung 2.1.: assoziierte Exemplare

... und einfache mehrfache Navigation:

```
context Model::Flug::verfuegbarePlaetze(): Integer
post: flugzeug->notEmpty() implies result <= flugzeug.
    verfuegbarePlaetze

context Model::Flugzeug
inv: fluege.verfuegbarePlaetze()->sum() <= self.
    verfuegbarePlaetze
inv: fluege.flugzeug->asSet() = Set{self}
```

OCL Overview
An OCL Ordered Set is not an OCL Set but a speccial OCL Sequence

OCL-Hilfsmethoden:

```
context Model::Flug
def: momentanVerfuegbarePlaetze():Integer =
        verfuegbarePlaetze() - passagiere->size()
```

Durch `def:` eingeführte Methoden oder Attribute sind für OCL-Hilfszwecke bestimmte Objekte, die im Klassen-Gültigkeitsbereich als <<OclHelper>>-Objekte nutzbar sind.

Das Modell könnte zum Beispiel in C++ implementiert werden durch:

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

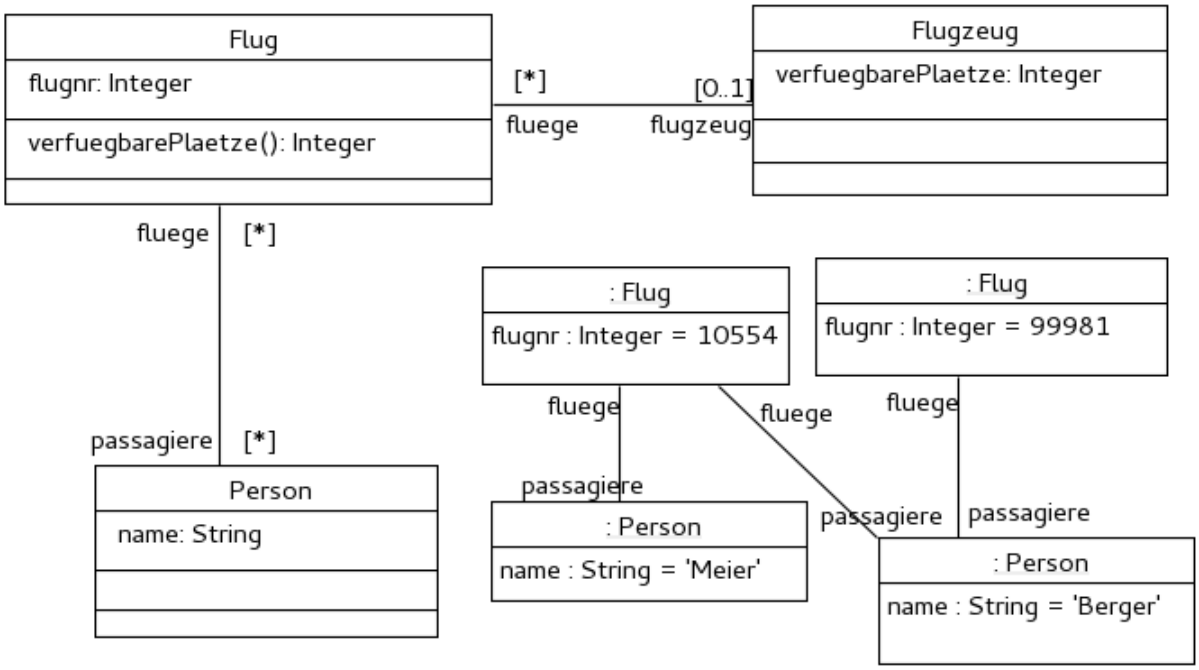
```
class Person{
string name;
vector <Flug *> fluege;
}
```

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

Abbildung 2.2.: Implementierungsbeispiel

UML: how to implement Association class in Java
UML associations in Java
Java „Traditionelle“ und „neue“ Collections
Java8 Collections
C++(11) Containers library
How can I efficiently select a Standard Library container in C++11?
UML Class Diagram Editor C++ Generator
Eclipse C++ Code-Generation
Umple — Syntax für Assoziationen, Vererbung, ...

In Papyrus inklusive Objektdiagramm:

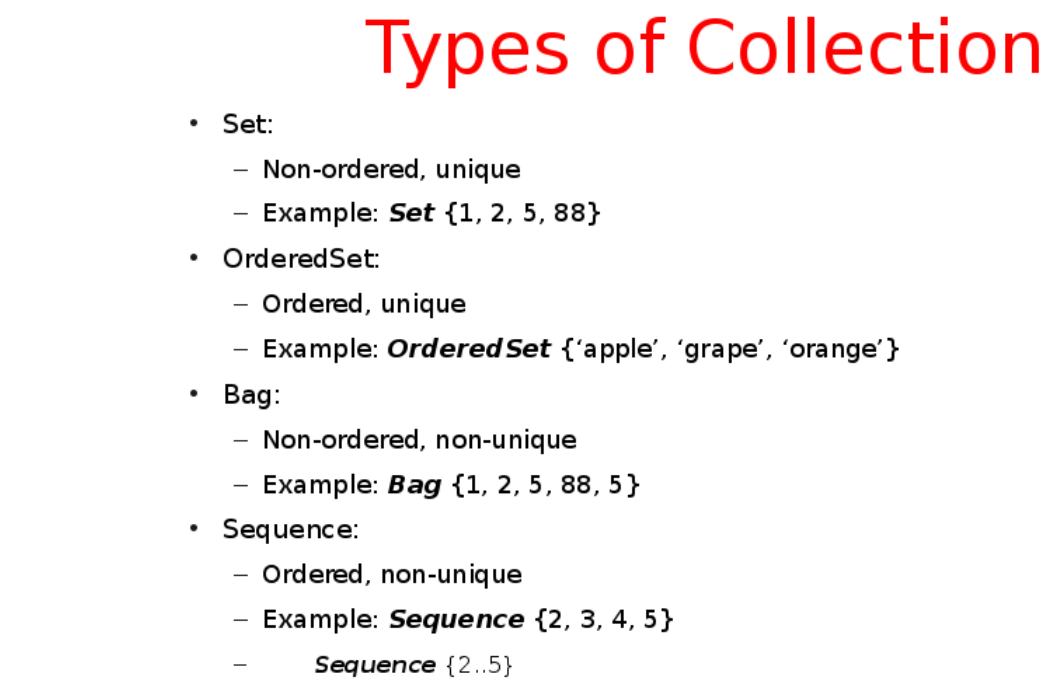


Einige OCL-Constraints:

```
import 'model.uml'

context Model::Flug
inv konsistenteBidir1: flugzeug.fluege->includes(self)

context Model::Flugzeug
inv: konsistenteBidir2: fluege.flugzeug->asSet() = Set{self}

context Model::Flug
inv konsistenteBidir1FlugPerson: passagiere.fluege->includes(
    self)

context Model::Person
inv konsistenteBidir2FlugPerson: fluege.passagiere->includes(
    self)
```

Bemerkungen zu den Objektdiagrammen:

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



2.2. size(), includes() und forAll() — Methoden-Verträge

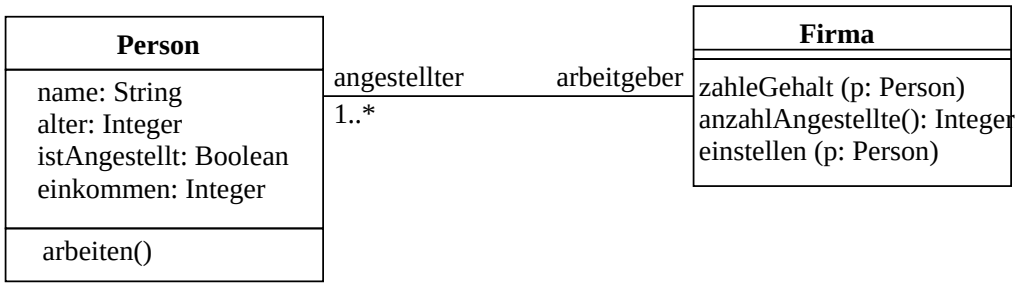


Abbildung 2.3.: Modell Person-Firma

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

```
context  Model::Person
  inv:   alter >= 0

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

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

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

context  Model::Firma::einstellen (p: Model::Person): OclVoid
  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  Model::Person
  inv:   if istAngestellt then
               einkommen >= 300
         else
               einkommen < 300
         endif

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

Vergleiche auch Abschnitt 11.5.4 des OCL 2.4 Handbuchs:

Boolean::or(b: **Boolean**): **Boolean**

Boolean::and(b: Booelean): **Boolean**

Boolean::xor(b: Booelean): **Boolean**
post: result = ((self **or** b) **and not** (self=b))

Boolean::not: **Boolean**

Boolean::implies(b: Booelean): **Boolean**
post: result = ((**not** self) of b)

toString(): **String**

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

Tabelle 2.1.: logische Operationen in OCL

Zeichensatz für Literale der „basic types“ OCLs
Die OCL-Basic_Types `Boolean, Integer, Real, UnlimitedNatural` und `String` besitzen als mögliche Literalwerte ϵ (für `null`) und $\perp$ (für den ungültigen Wert `invalid` im Sinne einer dreiwertigen Logik, Wiki-Eintrag: dreiwertige Logik).
`UnlimitedNatural` besitzt darüber hinaus den möglichen Wert ∞ (als `*` lexikalisch dargestellt).

Aus dem OCL 2.4-Manual:
Seite 162,11.5.4 Boolean
Seite 10f., 7.4 Basic Values and Types
Seite 211f., A.2.1 Basic Types
Seite 157f., 11.4.5 UnlimitedNatural
Seite 16, 7.4.11 Keywords invalid, null
Seite 16,7.4.13 Invalid Values
Seite 152, 11.2.3f. OclVoid, OclInvalid
Seite 153, 11.3.1 oclIsUndefined(), oclIsInvalid()
Seite 214f., A.2.1.4 Semantic of Operations / Wertetabellen

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

  - `{ nonunique }`
  - `{ ordered, nonunique }`
  - `{ 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.

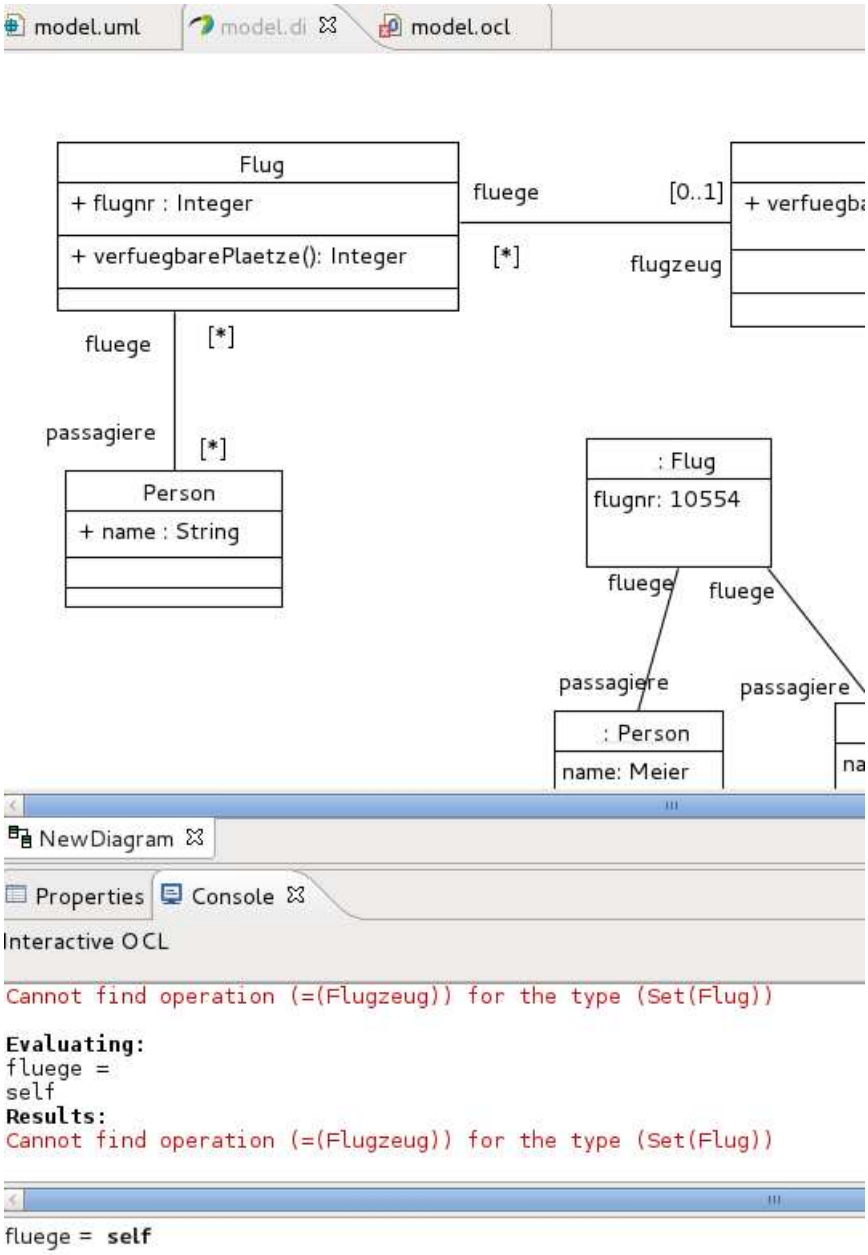
- Ist die letzte Rolle bei der Weiternavigation nach zwischenzeitlichem Erreichen eines Containerobjekts als `{ ordered }` gekennzeichnet, so ist das Navigationsergebnis eine Sequence.

- Jede (explizite oder implizite) collect()-Operation (vergleiche Abschnitt 7.4.10 und 7.6.2 des OCL 2.4-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

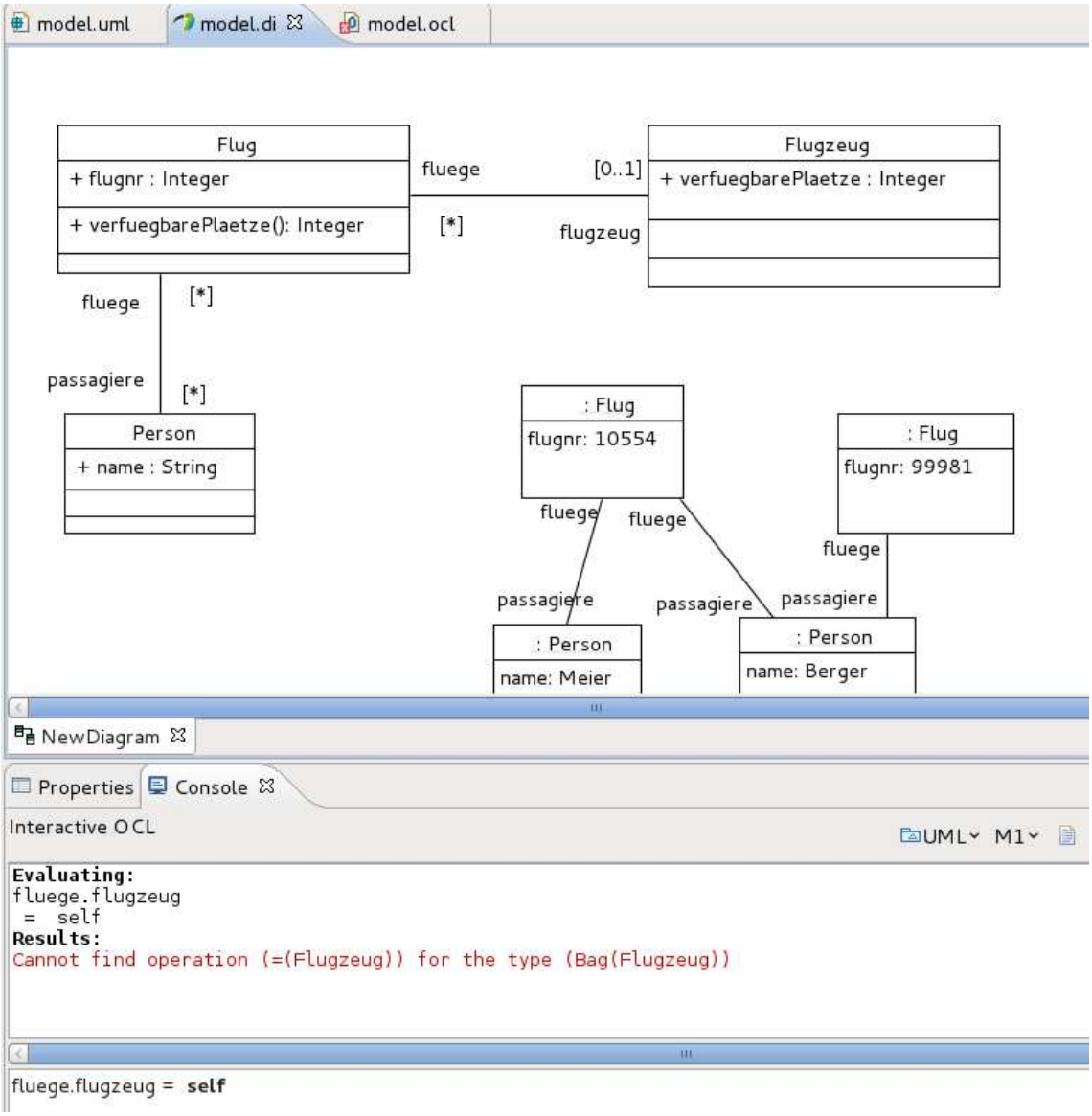
```
self.angestellter->collect(alter)
self.angestellter->collect(p | p.alter)
self.angestellter->collect(p : Person | p.alter)
```

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 Kontext `Flugzeug`?



Und welchen Typ besitzt die Doppelnavigation `fluege.flugzeug` im Kontext `Flugzeug`?



Aus dem OCL 2.4-Manual:
Seite 29f., 7.6.2 CollectOperation
Seite 15f., 7.4.10 Navigation Operators and Navigation Shorthands
Seite 30, 7.6.2 Shorthand for Collect

2.4. Assoziationsklassen

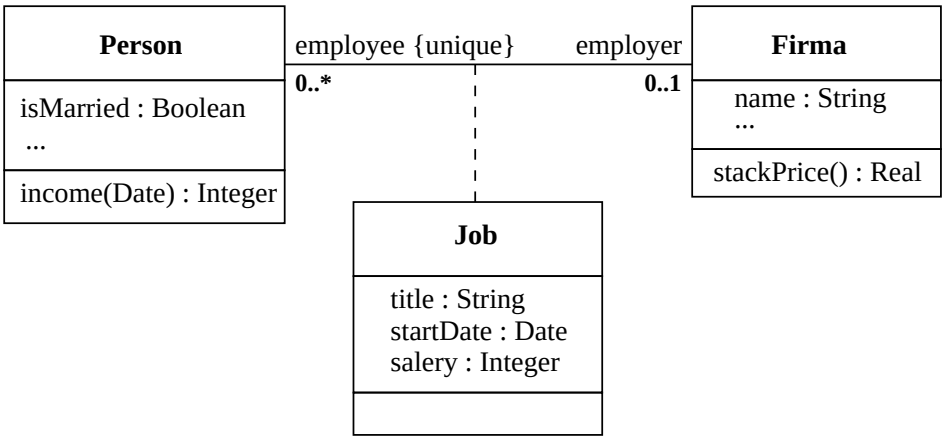


Abbildung 2.4.: Assoziationsklasse Job

wird als Workaround implementiert durch:

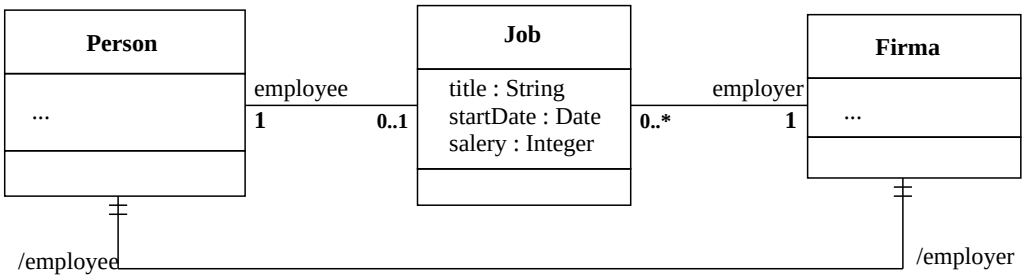


Abbildung 2.5.: Assoziationklasse im Workaround

Hier ist zu benutzen:

context Model::Firma
... job.employee ...

statt original

context Model::Firma
... employee ...

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

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

eingeführt.

Analog für die Klasse Person:

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

Alternativ könnte man die `<<OclHelper>>`-Attribute

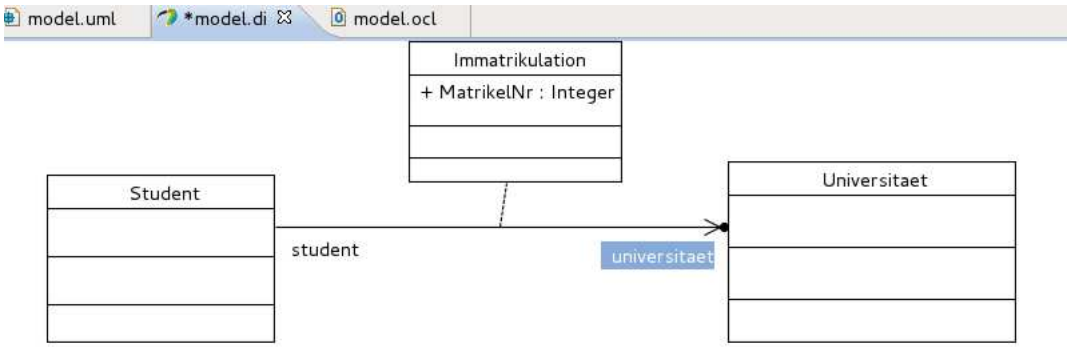
```
context Model::Firma
def: employee : Set(Model::Person) = job.employee->asSet()

context Model::Person
def: employer : Firma = job.employer
```

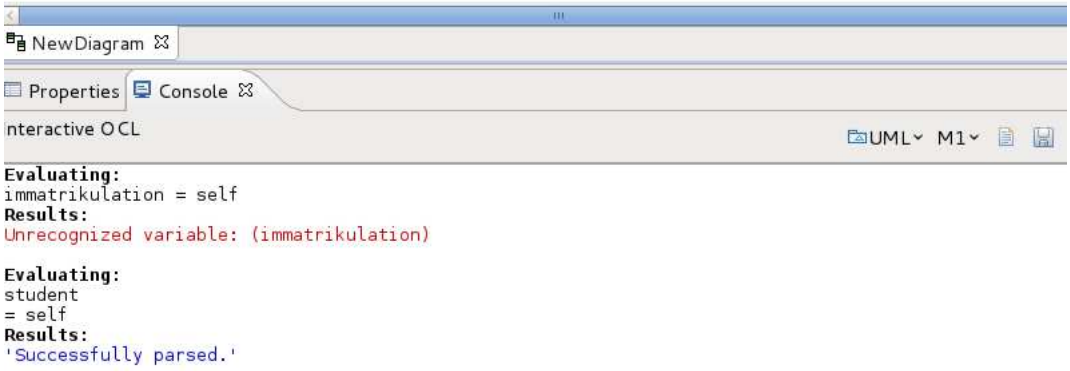
einführen.

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

Weitere OCL-Ausdrücke:
context Model::Person :: income (d:Model::Date) : Integer
body: self.job → select(d >= startDate).salery → sum()

In aktuellen Papyrus werden Assoziationsklassen leider unsymmetrisch bezüglich beider Assoziationsenden behandelt:





Man hat die Rolle `student` durch das Kontextmenü `Association End`, `owned by the classifier` noch umzustellen, um die übliche symmetrische Sichtweise zu erhalten.

OCL 2.4-Manual:
Seite 21f., 7.5.4f. Navigation to/from Association Classes

2.5. Qualifizierte Assoziationen

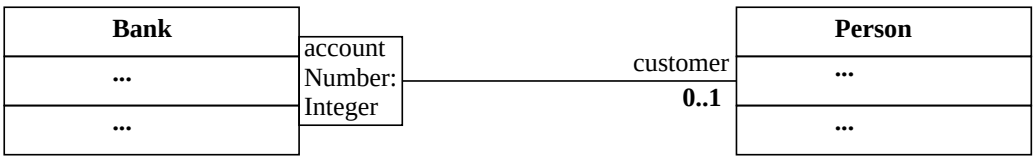
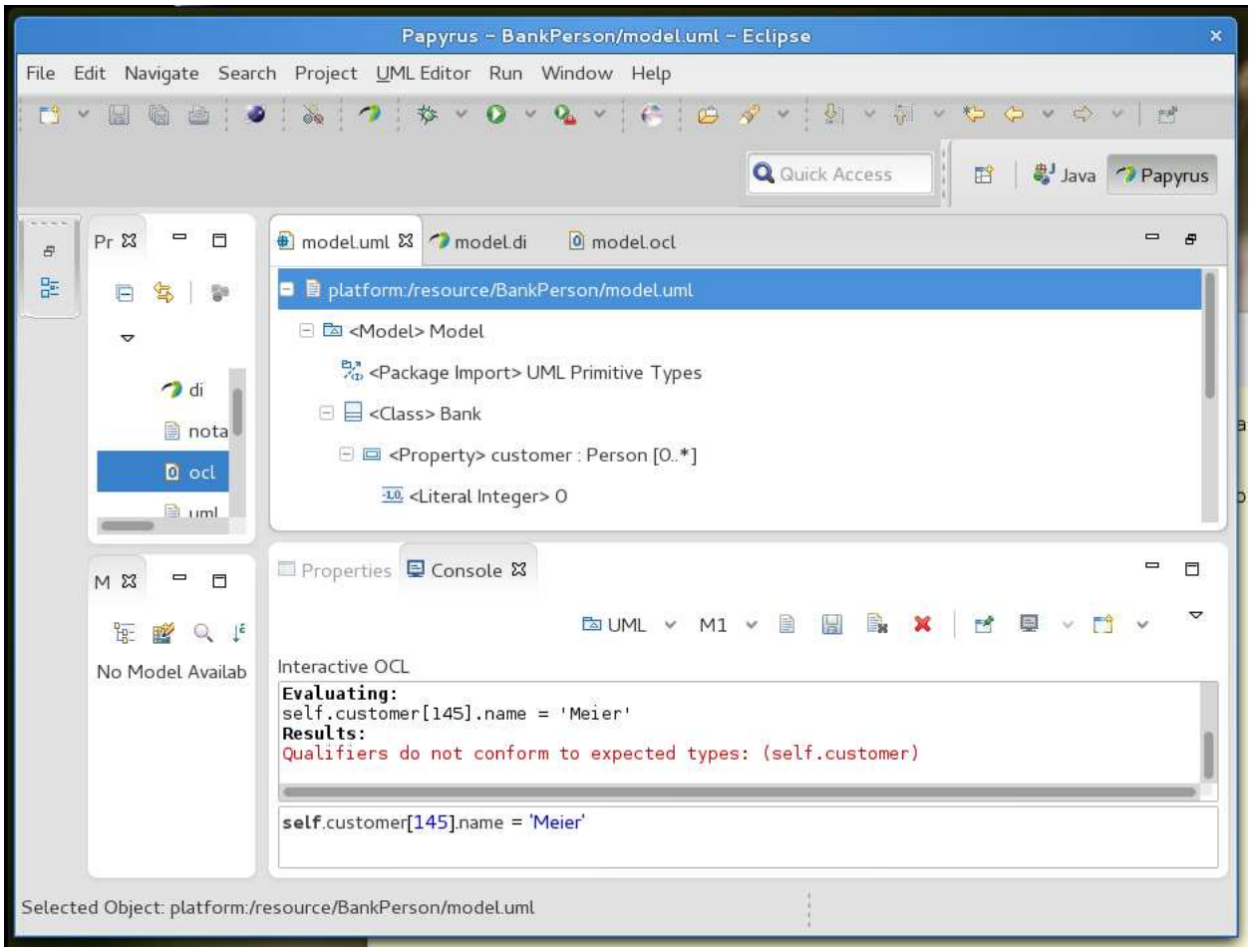
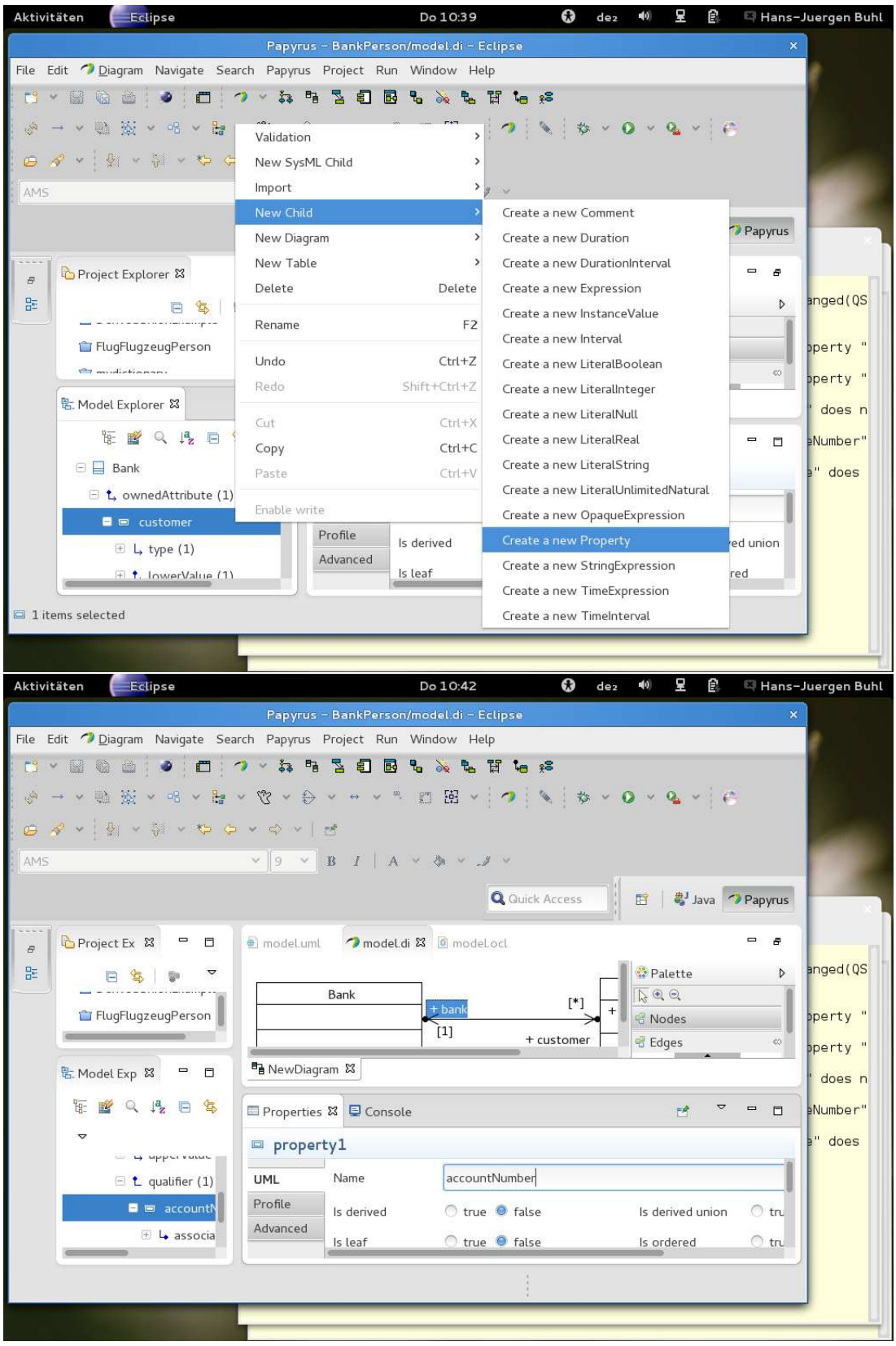


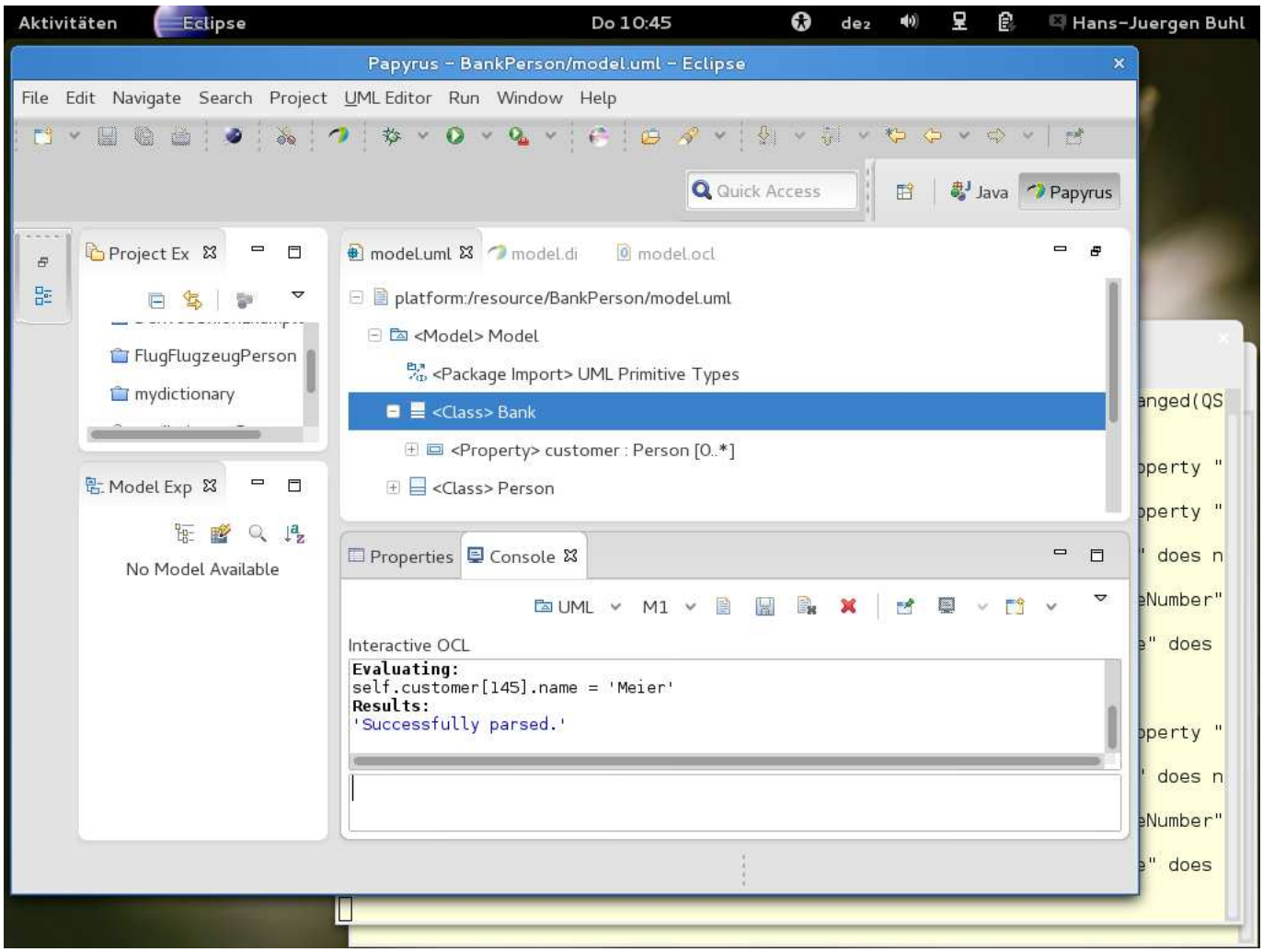
Abbildung 2.6.: qualifizierte Assoziation

Leider werden qualifizierte Assoziationen in `di`-Diagrammen noch nicht angezeigt und in OCL-Ausdrücken ohne Änderung des Rollennamen-UML-Modell noch nicht unterstützt:



Im Modellexplorer kann man jedoch dem Rollennamen `customer` die Eigenschaft `qualified` mit Qualifier-Namen `accountNumber` und -Typ `Integer` geben, damit zumindest der Interaktive OCL-Editor `customer` entsprechend kennt:


Aktivitäten
Eclipse
Do 10:39
de2
Hans-Juergen Buhl
Papyrus – BankPerson/model.di – Eclipse
File Edit Diagram Navigate Search Papyrus Project Run Window Help
Validation
New SysML Child
Import
New Child
New Diagram
New Table
Delete Delete
Rename F2
Undo Ctrl+Z
Redo Shift+Ctrl+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Enable write
Create a new Comment
Create a new Duration
Create a new DurationInterval
Create a new Expression
Create a new InstanceValue
Create a new Interval
Create a new LiteralBoolean
Create a new LiteralInteger
Create a new LiteralNull
Create a new LiteralReal
Create a new LiteralString
Create a new LiteralUnlimitedNatural
Create a new OpaqueExpression
Create a new Property
Create a new StringExpression
Create a new TimeExpression
Create a new TimeInterval
Project Explorer
FlugFlugzeugPerson
Model Explorer
Bank
ownedAttribute (1)
customer
type (1)
Profile
Advanced
Is derived
Is leaf
1 items selected
Papyrus
Aktivitäten
Eclipse
Do 10:42
de2
Hans-Juergen Buhl
Papyrus – BankPerson/model.di – Eclipse
File Edit Diagram Navigate Search Papyrus Project Run Window Help
Quick Access
Java
Papyrus
Project Ex
FlugFlugzeugPerson
Model Exp
qualifier (1)
model.uml
model.di
model.ocl
Bank
+ bank
[1]
[*]
+ customer
Palette
Nodes
Edges
NewDiagram
Properties
Console
property1
UML
Profile
Advanced
Name
accountNumber
Is derived
true
false
Is derived union
Is leaf
true
false
Is ordered



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'}
```

OCL 2.4-Manual:
Seite 22, 7.5.6 Navigation through Qualified Associations

2.6. Andere Methoden für die Collections Bag/Set

(im **context** Flugzeug)

fluege→ size()	UnlimitedNatural
fluege→ isEmpty()	Boolean
fluege→ notEmpty()	Boolean
fluege→ includes(Berlin_NewYork) ↑ von Typ Flug	Boolean
fluege→ includesAll(Berlin_NN) ↑ von Typ set(Flug)	Boolean
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 beid mathematisch: $(A \cup B) \setminus (A \cap B)$
flatten()	rekursives Entpacken von verschachtelten Mengen

Tabelle 2.2.: Methoden für die Collection Set

Beispiele für Flugnummern:

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

true $\Longleftrightarrow$ Es existiert genau ein Flug mit der flugnr 123

$\underbrace{fluege}_{Set(Flug)} \rightarrow \text{isUnique}(flugnr)$ — Boolean

true $\Longleftrightarrow$ Jeder Flug hat eine andere flugnr als jeder andere Flug

2.7. Schleifen und Iteratoren

(im **context** Flugzeug)

fluege→ exists(flugnr=12)	Boolean
fluege.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())	Bag{1,3,3,10, ...}
(collectNested (expr))	
fluege→isUnique (flugnr)	Boolean
fluege→ 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

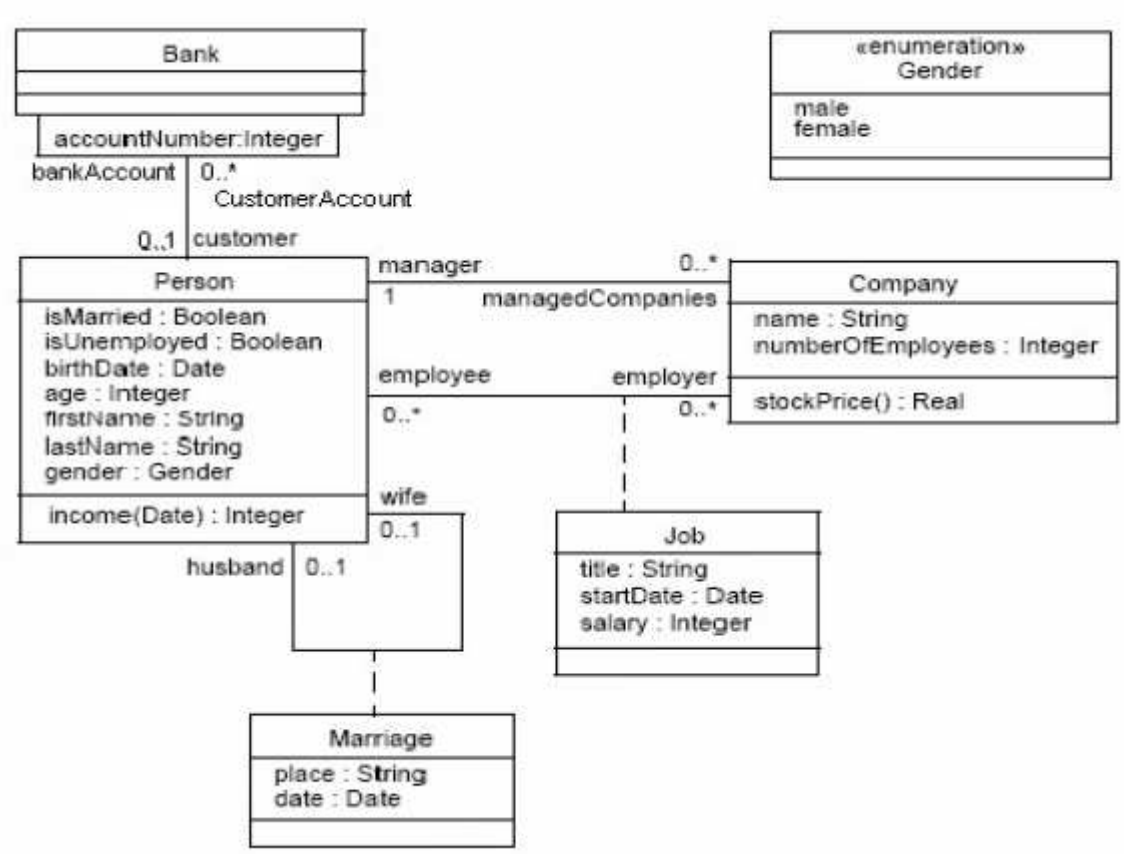
	non ordered	ordered
Element kann nur einfach vorhanden sein	Set	OrderedSet
Element kann mehrfach vorhanden sein	Bag	Sequence

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/2.4/PDF (Nachbedingungen der Collection-Operationen) nach!

2.9. Together und automatische Code-Erzeugung



(Aus: http://www.omg.org/spec/OCL/2.4/PDF)

Listing 2.1: Klasse Bank mit qualifizierter Assiziation

```
/*    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: Enumberation 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

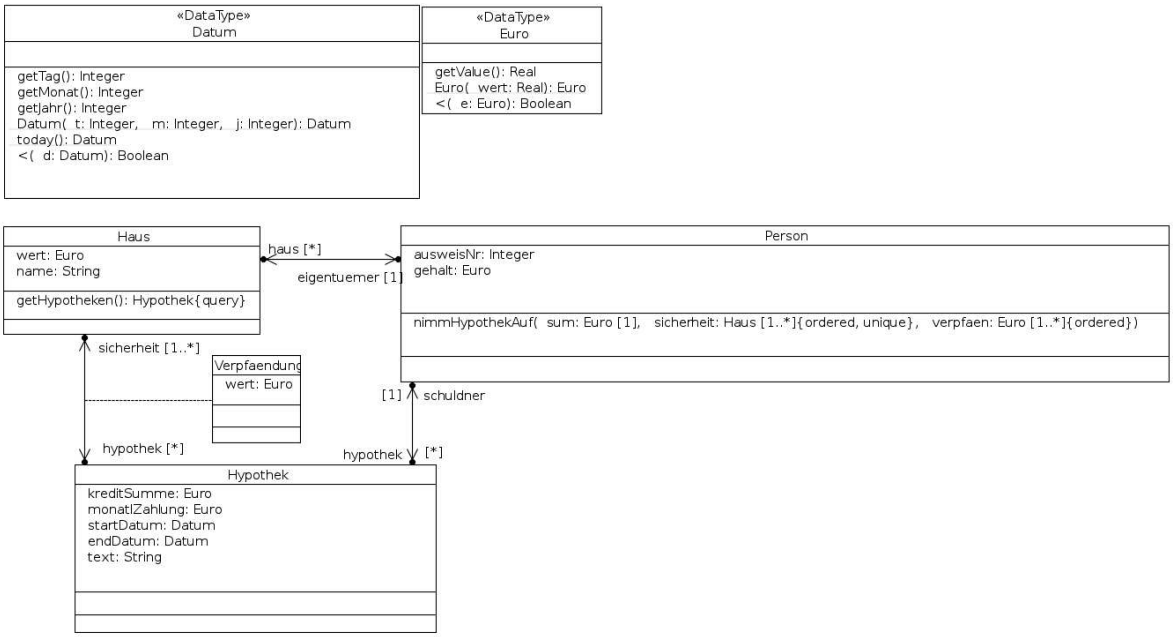


Abbildung 2.7.: Klassendiagramm Hypothek

context Model::Hypothek
inv: sicherheit.eigentuemer→asSet() = Set{schuldner}

context Model::Hypothek
inv: startDatum < endDatum

context Model::Person
def: anzahlHypotheken : Integer = $\underbrace{\underbrace{haus}_{Set(Haus)} .hypothek \rightarrow asSet() \rightarrow size()}_{Bag(Hypothek)}$

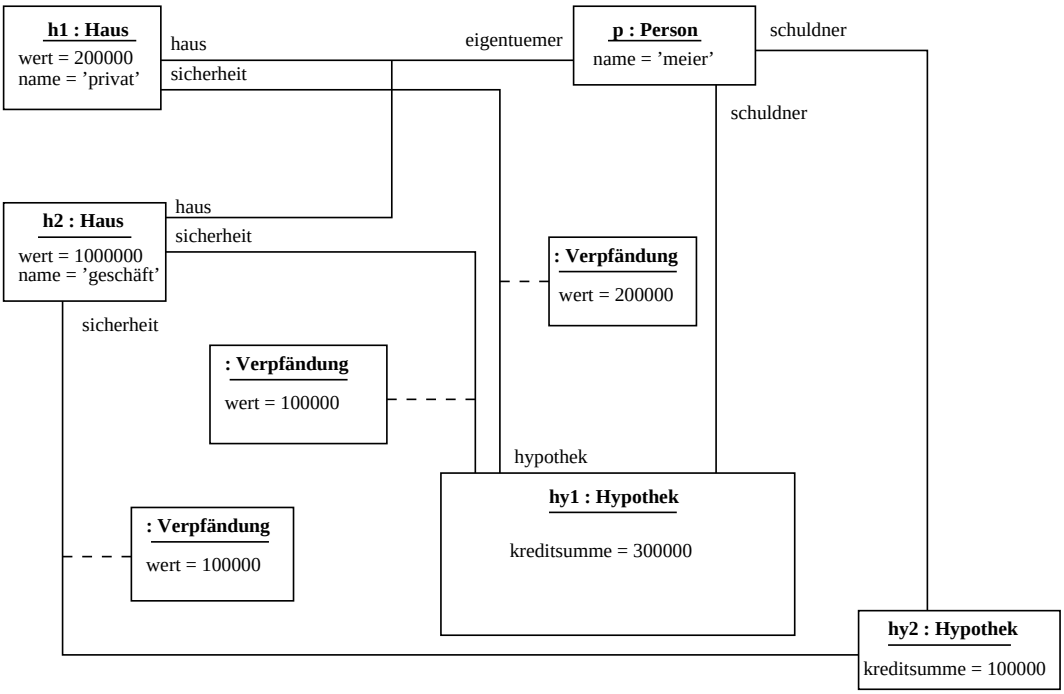


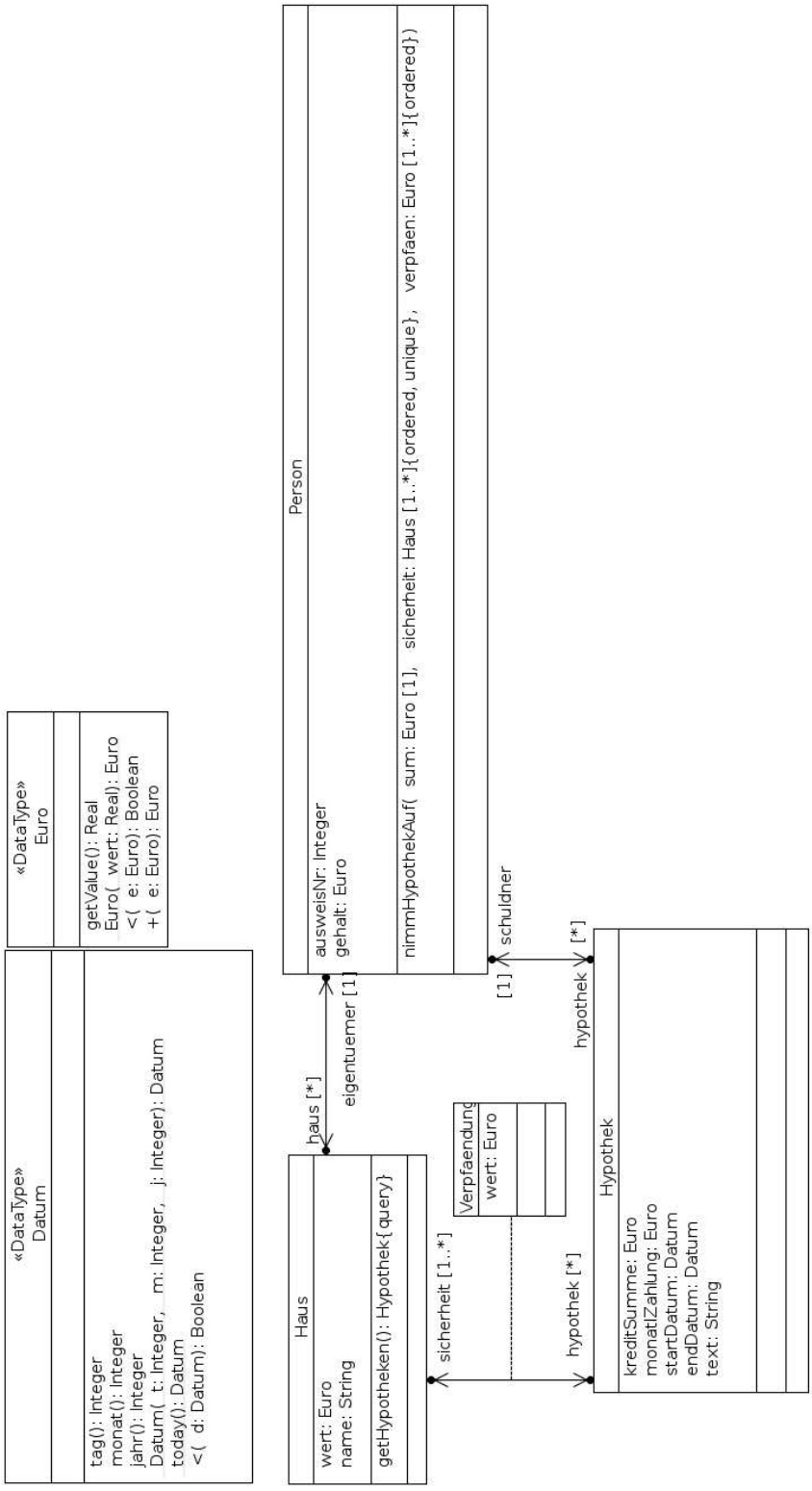
Abbildung 2.8.: Hypothek mit zwei Häusern

p.haus= Set{h1,h2} mit **h1.hypothek**=Set{hy1}
h2.hypothek=Set{hy1,hy2}
p.haus.hypothek = Bag {hy1,hy1,hy2}
p.haus.hypothek → **asSet()**= {hy1,hy2}

Erste Codeverträge:

```
import 'model.uml'

context Model::Haus
inv: Euro::Euro(0.0) < wert
inv: name.size()>2

context Model::Person
inv: ausweisNr > 0
inv: Euro::Euro(0.0) < gehalt or Euro::Euro(0.0) = gehalt

context Model::Hypothek
inv: sicherheit.eigentuemer->asSet() = Set{schuldner}

context Model::Hypothek
inv: startDatum < endDatum

context Model::Person
def: anzahlHypotheken: Integer =
       haus.hypothek->asSet()->size()

-- ...
```

In Papyrus nach Ergänzung weiterer Methoden:



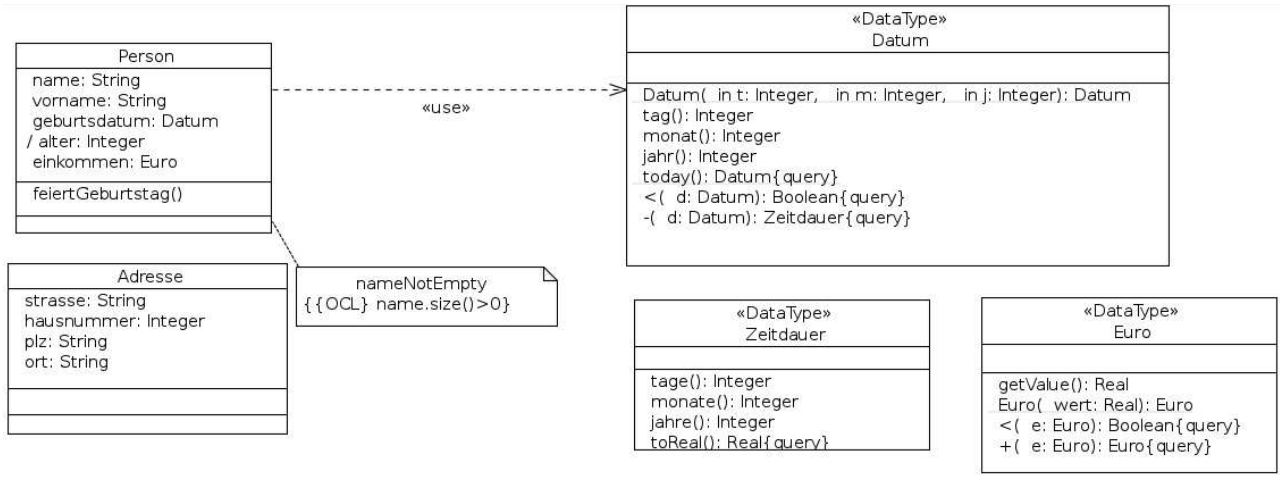
Und mit weiteren Codeverträgen:

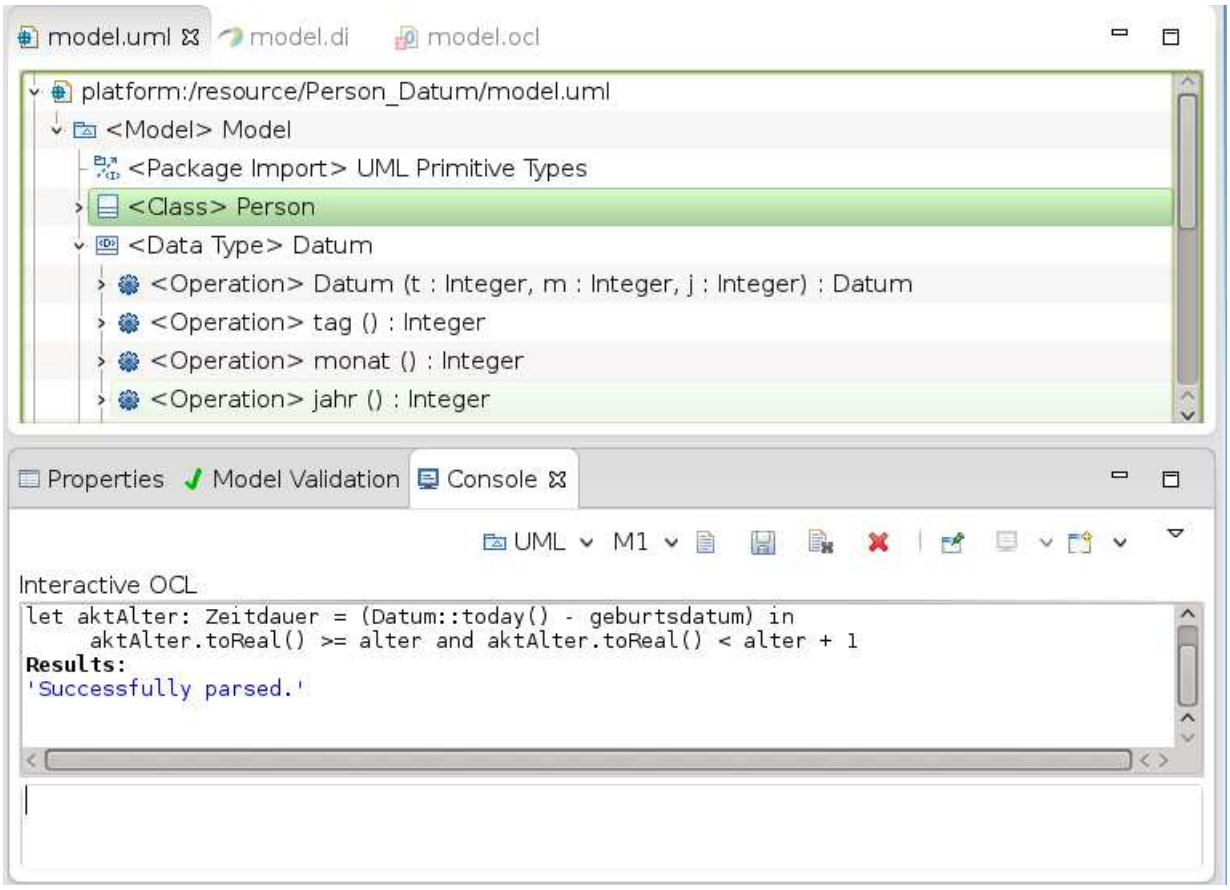
```
import 'model.uml'

context Model::Datum::tag(): Integer
post: result >= 1 and result <= 31
context Model::Datum::monat(): Integer
post: result <= 1 and result >= 12
context Model::Datum::jahr(): Integer
post: result >= 1920 and result <= 2500

-- oder alternativ mit den grundlegenden Observatoren:
context Model::Datum
inv: tag() >= 1 and tag() <= 31
inv: monat() >= 1 and monat <= 12
inv: jahr() >= 1920 and jahr <= 2500

-- constructor Euro
context Model::Euro::Euro(wert: Real): Model::Euro
pre: wert <> null
post: result.oclIsNew()
post: result.getValue() = wert

-- abgeleitete Observatoren
context Model::Euro::_'<'(e: Model::Euro): Boolean
post: result = (self.getValue() < e.getValue())

context Model::Euro::_'+'(e: Model::Euro): Model::Euro
post: result = (self.getValue() + e.getValue())

-- Constraints
context Model::Haus
inv: Euro::Euro(0.0) < wert
inv: name.size()>2
context Model::Haus::getHypotheken(): Set(Model::Hypothek)
body: hypothek

context Model::Person
inv: ausweisNr > 0
inv: Euro::Euro(0.0) < gehalt or Euro::Euro(0.0) = gehalt

context Model::Hypothek
inv: sicherheit.eigentuemer->asSet() = Set{schuldner}
```

```
context Model::Hypothek
inv: Euro::Euro(0.0) < kreditSumme
inv: kreditSumme = Verpfaendung.wert->sum()
inv: text <> ''

context Model::Hypothek
inv: startDatum < endDatum

context Model::Verpfaendung
inv: wert <= sicherheit.wert

context Model::Person
def: anzahlHypotheken: Integer =
        haus.hypothek->asSet()->size()

context Model::Person::nimmHypothekAuf(sum: Model::Euro,
    sicherheit: OrderedSet(Model::Haus), verpfaen: Sequence(
    Model::Euro)): OclVoid
pre: sicherheit->size()=verpfaen->size()
pre: sum=verpfaen->sum()
pre: ...
```

2.11. Einige erste Hilfskomponenten: Euro, Datum, Zeitdauer, Person, Adresse, ...

Das redundante Attribut `alter`:





Listing 2.4: OCL-Constraints Datum

```
context Model::Datum

inv tagGueltig: tag() >= 1 and tag() <= 31
inv monatGueltig: monat() >=1 and monat() <= 12
inv jahrGueltig: jahr() >= 1920 and jahr() <= 2500
```

oder besser:

Listing 2.5: OCL-Constraints Datum (verbessert)

```
context Model::Datum -- virtuelle Methode/Hilfsmethode
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
                                   else 28
                     endif endif endif

/* ---------------------------------------------------------- */

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

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

Listing 2.6: OCL-Constraints Person

```
context Model::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
```

oder insgesamt:

```
import 'model.uml'

context Model::Person
inv: name.size() > 0
inv: vorname.size() >0
inv: alter >= 0
inv: alter <= 200
inv: let aktAlter: Zeitdauer = (Datum::today() - geburtsdatum) in
     aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1
inv: Euro::Euro(0.0) < einkommen or Euro::Euro(0.0) = einkommen

-- Invariante Datum:

context  Model::Datum
inv tagGueltig: tag() >= 1 and tag() <= 31
inv monatGueltig: monat() >= 1 and monat() <= 12
inv jahrGueltig: jahr() >= 1920 and jahr() >= 2500

-- oder besser:

context Model::Datum
def: gueltigesDatum(t: Integer,
                    m: Integer,
                    j: Integer
): Boolean =
     j >= 1920 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
                     else 28
                     endif endif endif
```

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

-- Contract des Konstruktors Datum(.,.,.):

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

context Model::Adresse
inv: strasse.size() > 0
inv: hausnummer > 0
inv: plz.size()=5
inv: ort.size()>0

context Model::Zeitdauer::toReal(): Real
post: result = tage()/365 + monate()/12 + jahre()


-- constructor Euro
context Model::Euro::Euro(wert: Real): Model::Euro
pre: wert <> null
post: result.oclIsNew()
post: result.getValue() = wert

-- abgeleitete Observatoren
context Model::Euro::_'<'(e: Model::Euro): Boolean
post: result = (self.getValue() < e.getValue())

context Model::Euro::_'+'(e: Model::Euro): Model::Euro
post: result = Euro(self.getValue() + e.getValue())
```

2.12. Alle Instanzen einer Klasse: allInstances()

Die Methode

allInstances():**Set**(T)

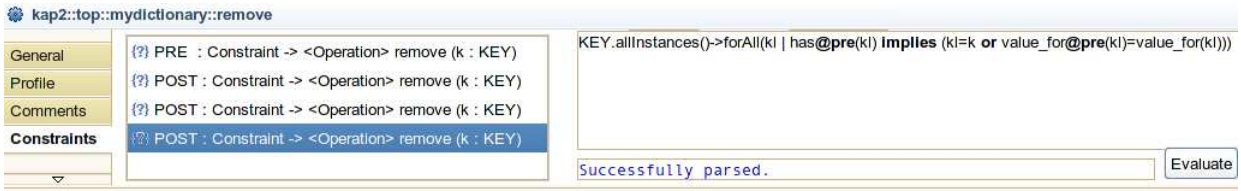
liefert alle (endlich viele) Instanzen einer User-definierten Klasse. Sie darf nicht benutzt werden für String, Integer und Real.
Ein Beispiel:

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

Alternativ:

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

Siehe auch::
http://www.springerlink.com/content/j8g72037gn63t1h6/
Damit ist unsere `mydictionary`-Spezifikation vollständig realisierbar:



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

OCL Query Examples

These query examples are pure OCL only, the 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')
```

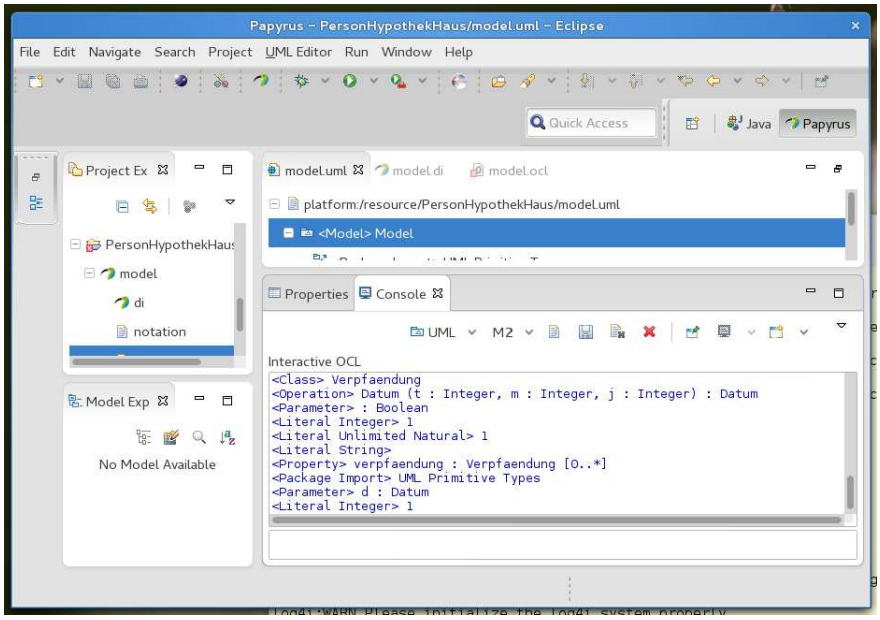
Get the dates of all Konsultations

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

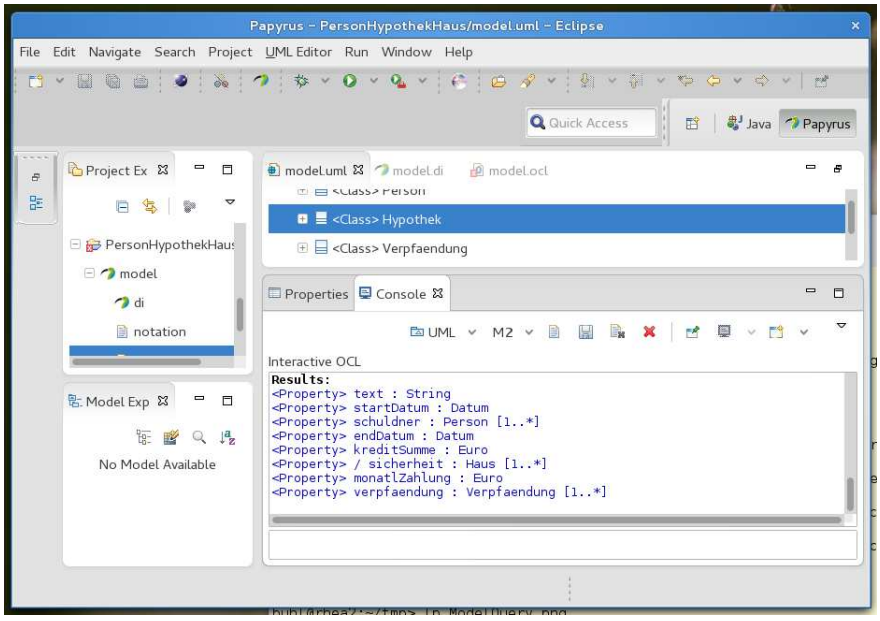
aus: allInstances-Query

Abfrage des UML-Modells `model.uml` des Projekts `PersonHypothekHaus` mittels der OCL-Console im M2-Level (etwa für die Konzepzion und Überprüfung von Teamregeln geeignet):

`allOwnedElements():Set(Element):`

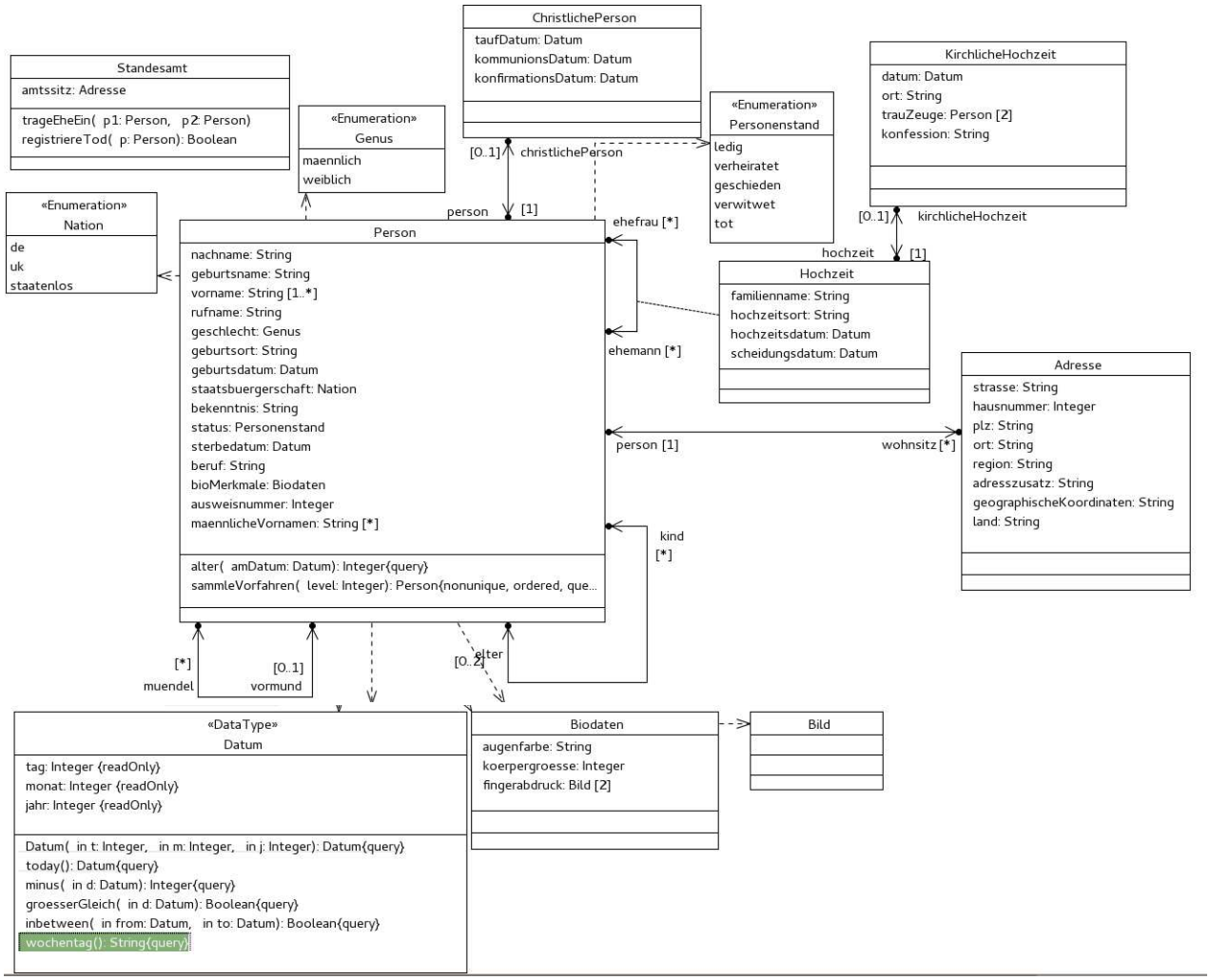


`allFeatures():Set(Features):`



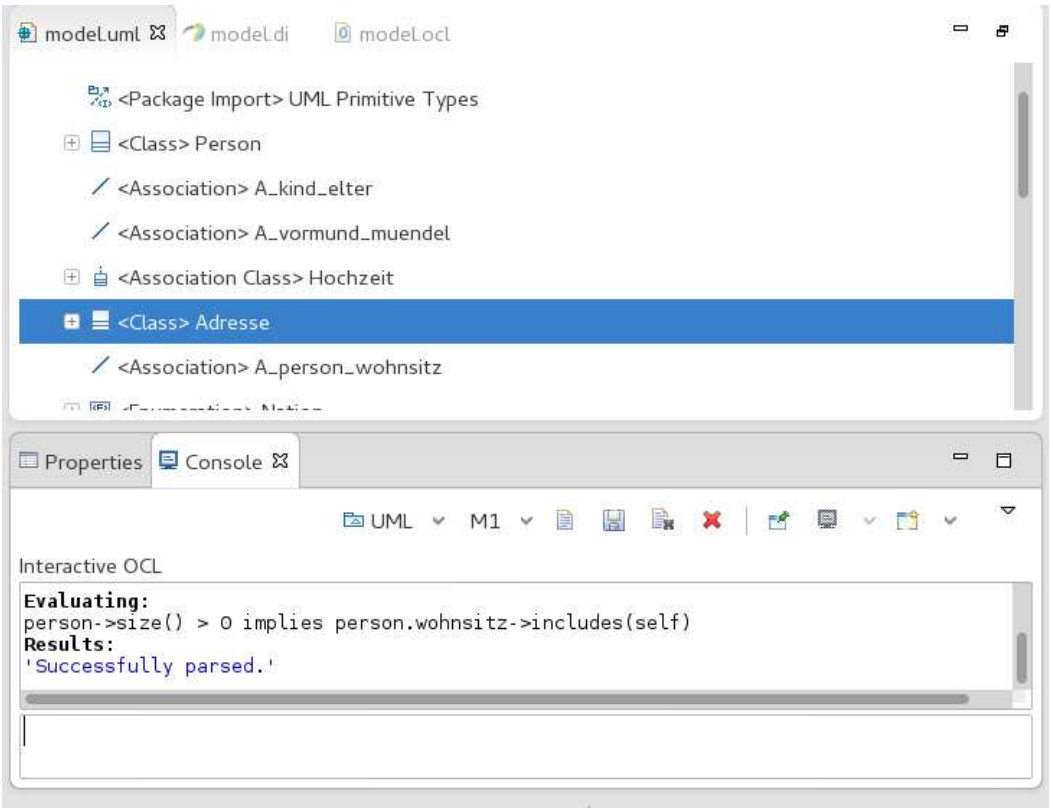
(Beachte den Abfrage-Level: `M2`)

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



Listing 2.7: Constraints Adresse

```
context Model::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 Model::Adresse::land: String
init : 'Deutschland'

context Model::Adresse::region: String
init : 'NRW'

context Model::Adresse::person: Set(Person)
init : Set{}
```



Listing 2.8: Constraints Biodaten

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

context Model::BioDaten::augenfarbe: String
init : 'unbekannt'
```

Listing 2.9: Constraints Datum

```
context Model::Datum /*virtuelle Methode = Hilfsmethode/
   OclHelper*/
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 Model::Datum
inv : gueltigesDatum(tag, monat, jahr)

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

```
/* Problem: context Model::Datum::-(d:Model::Datum):Integer
   wird nicht
   akzeptiert, deshalb Workaround: */

context Model::Datum::minus( d : Model::Datum ) : Integer -- in
    vollen Jahren (Alter, ...)
pre : self.groesserGleich(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 Model::Datum::groesserGleich(d:Model::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 Model::Datum::inbetween(from : Model::Datum, to : Model
   ::Datum) : Boolean
pre : to.groesserGleich(from)
body : to.groesserGleich(self) and self.groesserGleich(from)

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

-- und jetzt ein tagesgenaues minus()
-- mit Hilfe von de.wikipedia.org/wiki/Julianisches_Datum
```

```
context Model::Datum
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

contextModel:: Model::Datum::minus(d : Datum): Integer
body: toChronoJD(tag, monat, jahr) - toChronoJD(d.tag, d.monat,
    d.jahr)

-- und fuer geschaeftliche Anwendungen:
context Model::Datum
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 im OCL-Manual 7.5.15, Seite 27).

2.14. Constraints Person / Verwandschaftsbeziehungen

Listing 2.10: Constraints Person

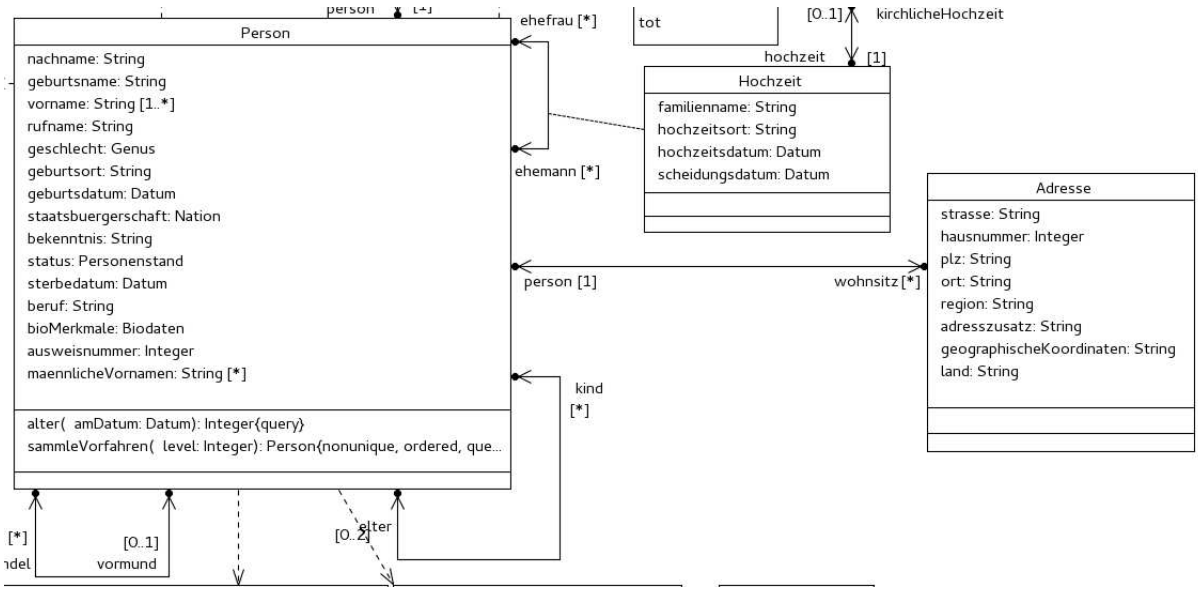
```
context Model::Person
inv : elter->size() <=2
inv : ausweisnummer >= 0    /* 0 bei fehlendem Ausweis */
inv : wohnsitz->size() > 0 implies wohnsitz.person->includes(
   self)
inv : muendel->notEmpty() implies muendel.vormund->asSet() =
   Set{self}
inv : vormund->notEmpty() implies vormund.muendel->includes(
   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 Model::Person
def: maennlicheVornamen : Set(String) =
     Set{'Hans', 'Georg' /* bitte ergaenzen */}
def: weiblicheVornamen : Set(String) =
     Set{'Heidrun', 'Cornelia' /* bitte ergaenzen */}
context Model::Person
def: istMaennlicherVorname(vn : Set(String)) : Boolean =
     maennlicheVornamen->includesAll(vn)
def: istWeiblicherVorname(vn: Set(String)) : Boolean =
     weiblicheVornamen->includesAll(vn)
context Model::Person
inv : elter->size() > 0 implies elter->forAll( e | (
   geburtsdatum.minus(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[ehefrau]->size() >
                                        0 implies
                                    e.Hochzeit[ehefrau]->forAll(
                                        h |
                                            geburtsdatum.inbetween(
                                            h.hochzeitsdatum, h.
                                            scheidungsdatum)
                                        implies nachname = h.
                                            familienname
                                    )

                                endif
                            )

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

context Model::Person
inv : geschlecht = Genus::maennlich implies
    istMaennlicherVorname(vorname)
inv : geschlecht = Genus::maennlich implies self.Hochzeit[
    ehemann]->size() > 0  implies keineZeitweiseBigamie(self.
    Hochzeit[ehemann]->asSet())
inv : geschlecht = Genus::weiblich implies istWeiblicherVorname
    (vorname)
inv : geschlecht = Genus::weiblich implies self.Hochzeit[
    ehefrau]->size() > 0 implies keineZeitweiseBigamie(self.
    Hochzeit[ehefrau]->asSet())
```

```
inv : status = Personenstand::tot implies sterbedatum <> Datum
    ::invalidDatum
inv : status <> Personenstand::tot implies sterbedatum = Datum
    ::invalidDatum

context Model::Person::alter( amDatum : Model::Datum ) :
   Integer
pre : amDatum.groesserGleich(geburtsdatum)
post : result = amDatum.minus(geburtsdatum)

context Model::Person::status : Model::Personenstand
init: Personenstand::ledig

context Model::Person::staatsbuergerschaft: Model::Nation
init: Nation::de

context Model::Person::ausweisnummer
init: 0

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

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


context Model::Person::nachname: String
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 Model::Person
def : istNeffe( n : Model::Person ): Boolean =
   elter.kind->asSet()->excluding(self).kind->includes(n) and n
      .geschlecht=Genus::maennlich
def : Cousins(): Set(Model::Person) =
   (elter.elter.kind->asSet() - elter).kind->asSet()->select(
      geschlecht=Genus::maennlich)

def : istStiefvater( s : Model::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 Model::Person::sammleVorfahren( level : Integer ) :
   Sequence(Model::Person)
pre : level >= 0
post :
  if (level = 0) or (elter->size() = 0) then
     result = Sequence{}
  else if elter->size() = 1 then
     result = elter->asSequence()->append(elter->asSequence()->
        at(1).sammleVorfahren( level-1 ))
  else -- elter->size() = 2
   result = elter->asSequence()->append(elter->asSequence()->
                                        at(1).sammleVorfahren(
                                           level-1 ))->
                                    append(elter->asSequence()->at
                                       (2).
                                    sammleVorfahren( level-1 ))
   endif
   endif
```

Assoziationen von `Person` zu `Person`



und zugehörige Constraints:

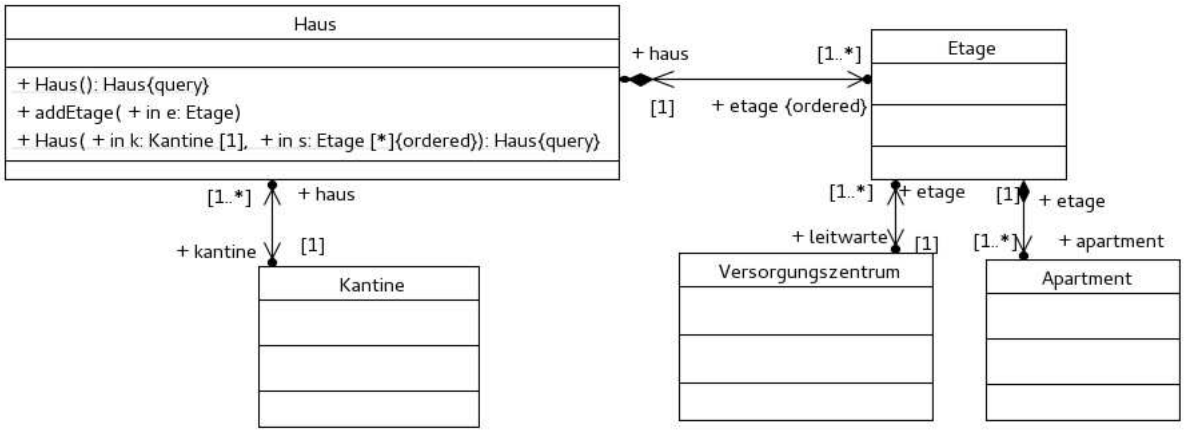
Listing 2.11: Constraints Hochzeit

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

context Model::Hochzeit::familienname: String
init: self.ehemann.nachname

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

OCL 2.4: Navigation to Association Classes (7.5.4)

OCL 2.4: Navigation from Association Classes (7.5.5)

2.15. 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-Modeslls) richtig aufbauen:

Listing 2.12: Constraints Haus()

```
context Model::Haus::Haus(k : Model::Kantine,
                          s: Sequence(Model::Etage))  : Model::Haus
pre:  not k.oclIsUndefined() and
      s->size() > 0
post: result.oclIsNew() 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?).
Aufgabe: Welche Nachteile würde 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?

Model::Haus::addEtage():

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

```
context Model::Haus::addEtage(e : Model::Etage): OclVoid
pre: e <> null
pre: etage->excludes(e)
post: etage->size() >= 1
post: etage->includes(e)
post: etage->size() - etage@pre->size() = 1
```

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

```
post: etage->includesAll(etage@pre)
```

- Formal vollständiger inlusive Framebedingung:

```
post: etage = etage@pre->append(e)
```

- Alternative Vorbedingungen:

```
pre: not e.oclIsUndefined()
pre: Haus.allInstances().etage->excludes(e)
```

Klassen-Invarianten:

- Kein Haus darf mehr als 10 Etagen besitzen.

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

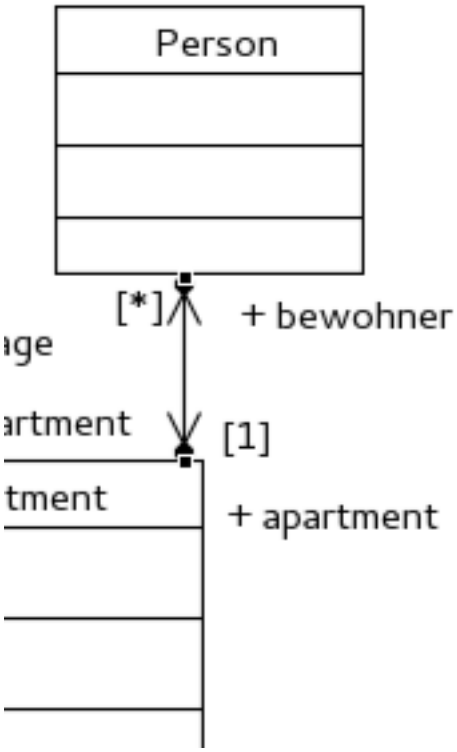
- Jede Etage hat 1..20 Apartments.

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

- Jedes Apartment hat 0..4 Bewohner.

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

(beachte nötige UML-Modell-Erweiterung:



)

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

  context Model::Kantine::anzahlHaeuser: **Integer**
  derive: haus->size()

- Jede Kantine kann höchstens 6 Häuser bedienen.

  context Model::Kantine
  inv: anzahlHaeuser <= 6

Systeminvarianten:

- Jedes Appartment besitzt eine Identifikationsnummer `appartmentID`, die in der gesamten Wohnanlage eindeutig ist.

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

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

  context Model::Apartment::zugLeitwarte: Versorgungszentrum
  derive: self.etage.leitwarte

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

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

  (oder siehe Vielfachheiten des UML-Modells)

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

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

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

  ```
  context Model::Kantine
  inv: haus.etage.aparment.bewohner->size()<=1000
  -- oder:
  inv:haus.etage.aparment.anzBewohner->sum()<=1000
  ```

  oder als explizites Collect:

  ```
  context Model::Kantine
  inv: haus.etage.aparment->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 Model::Haus::reisseHausAb(): OclVoid
  pre: etage.apartment->forAll(anzBewohner = 0)
  ```

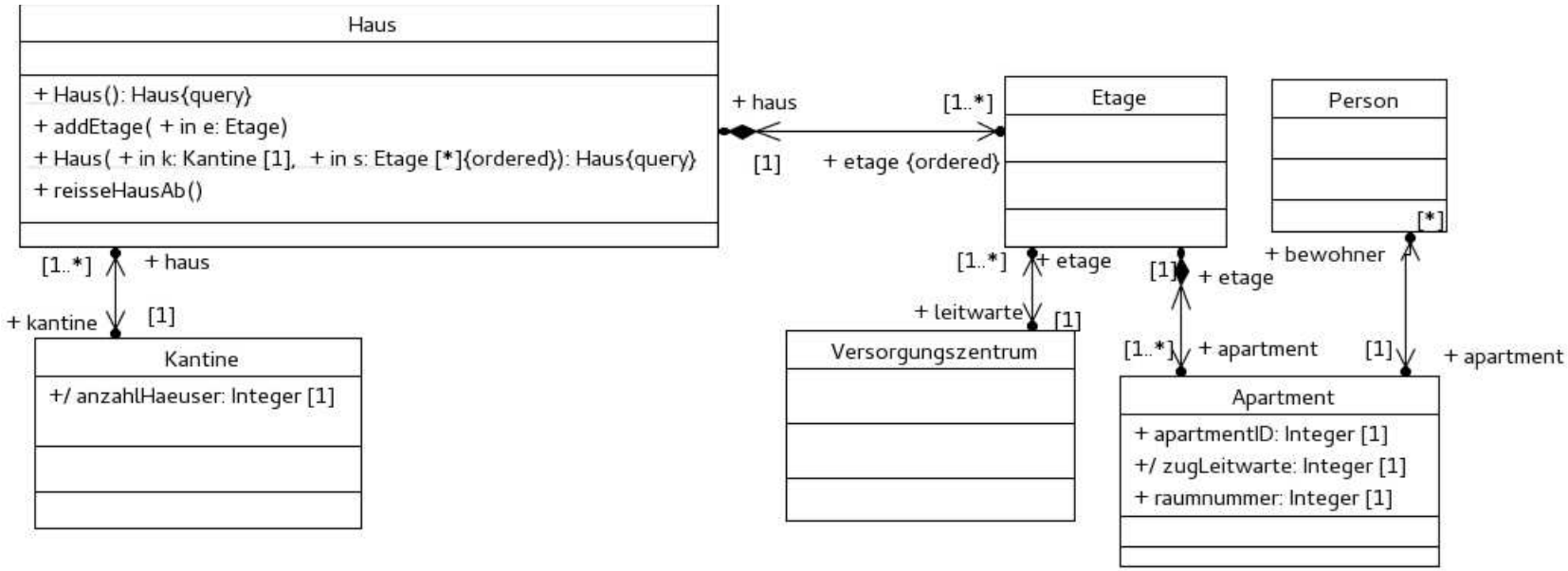
- Nach Aufruf von `reisseHausAb()` existieren die zugehörigen Etagen und Apartments nicht mehr.

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

  (Beachte die mehrmaleige Benutzung von `@pre`, vergleiche 7.5.14 des OCL-Manuals)

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

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


Haus
+ Haus(): Haus{query}
+ addEtage(+ in e: Etage)
+ Haus(+ in k: Kantine [1], + in s: Etage [*]{ordered}): Haus{query}
+ reisseHausAb()
+ haus
[1..*]
[1]
+ etage {ordered}
Etage
Person
[*]
+ bewohner
[1..*]
+ haus
+ kantine
[1]
Kantine
+/ anzahlHaeuser: Integer [1]
[1..*]
+ etage
[1]
+ etage
+ leitwarte
[1]
Versorgungszentrum
[1..*]
+ apartment
[1]
+ apartment
Apartment
+ apartmentID: Integer [1]
+/ zugLeitwarte: Integer [1]
+ raumnummer: Integer [1]

2.16. Packages ab Eclipse Kepler Modeling Tools (4.3) Papyrus: — Änderungen in model.ocl

Ab der Kepler-Version
http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplerr
des Papyrus-Plugins

```
Help
   Install Modeling Components
      Papyrus
      OCL Tools
```

nötige Änderung der `model.ocl`-Dateien:

```
import 'model.uml'

context Model::Person
inv: age >= 0

context Model::Bank
inv: customer->forAll(c|not c.isUnemployed)
```

oder

```
import 'Model.uml'
package Model

context Person
inv: age >= 0

context Bank
inv: customer->forAll(c| not c.isUnemployed)

endpackage
```

Vergleiche OCL-Manual 7.3.5.

Aufgabe: Warum ist die explizite Nennung des Packages in den Kontextzeilen empfehlenswerter als der globale Wechsel zum Namespace des Packages (vergleiche Why is std:: used by experienced coders rather than using namespace std;? und Should I use using namespace std in my code? im C++-Umfeld)?

Informationen zum Papyrus-Umgang mit Packages/Models:

UML Diagrams (and Models) with Papyrus

Eclipse OCL-FAQ

Eclipse OCL 5.0 Manual

2.17. Fortsetzung Fallstudie Person/Haus/Hypothek

context Model::Verpfaendung
inv: wert <= hypothek.kreditSumme
inv: wert <= sicherheit.wert
inv: wert >= Euro(0)

context Model::Haus
inv valid_security: hypothek → size()> 0
implies wert >= **V**erpfaendung.wert → sum()

context Model::Hypothek
inv valid_security: **V**erpfaendung.wert → sum() = kreditSumme

context Model::Person
inv: Model::Person.allInstances() → isUnique(ausweisNr)
def: anzHypotheken : Integer = hypothek → size()

...

context Model::Haus
.... $\underbrace{\text{hypothek} \rightarrow \text{select}(\text{monatlZahlung} > \text{Euro}(500.00))}$
Set der Hypotheken auf Haus, für die die monatlich Zahlung>500 Euro ist

...

2.18. Startwerte von Attributen und Ergebnisse von Observatoren

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

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

context Model::Haus::getHypotheken(): Set(Hypothek)
body: hypothek

2.19. Virtuelle OCL Variablen / Operationen/OclHelper

context Model::Person
def: income :Integer = self.Job.salery → sum()
def: nickname :String = 'little Joe'
def: hasTitle (t:String):Boolean = self.Job → exists(title=t)

2.20. Typ-Konformität

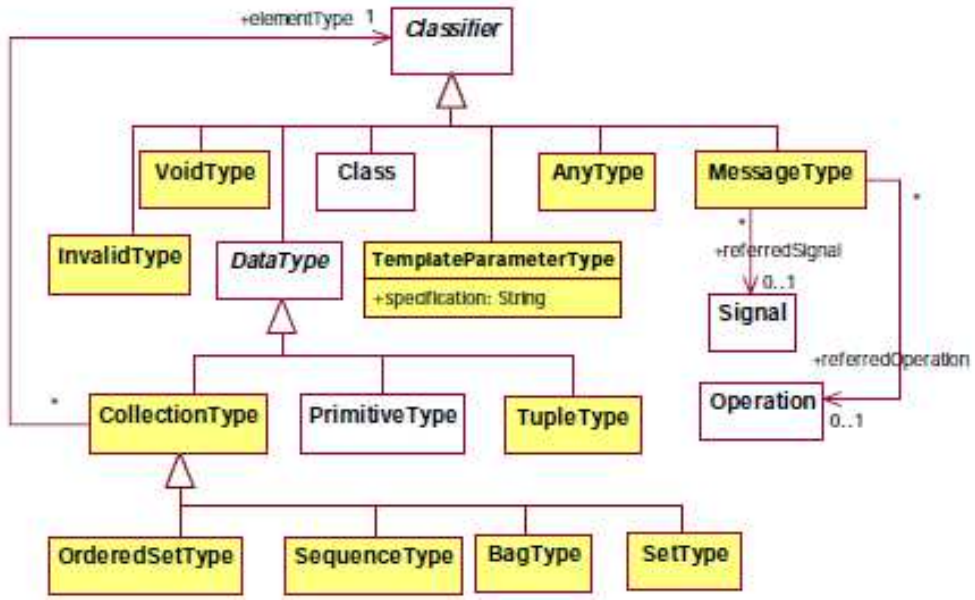
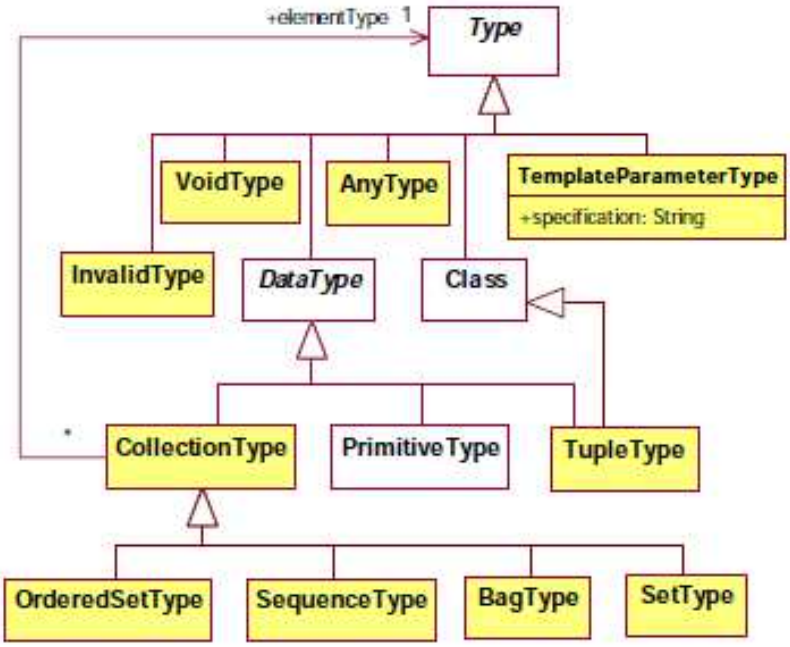


Abbildung 2.9.: Die Typen der OCL-Standard-Bibliothek

(Figure 8.1 - Abstract Syntax Kernel Metamodel for OCL Types, Seite 38 des OCL-Manuals)



(Figure 13.1 — Types, Seite 203 des OCL-Manuals)

Gemäß Seite 226 des OCL-Manuals:

- Jeder Typ ist konform zu seinen Elter-Typen.
- Die Typenkonformität ist transitiv, reflexiv und antisymmetrisch.

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

2.22. oclIsUndefined()

Auf Objekten des Typs OclAny kann mittels

not obj.oclIsUndefined()

die Existenz optionaler Exemplare abgefragt werden
(in Konkurenz zu `... ->notEmpty()` und
`... <> null`).

2.23. Vordefinierte Operationen auf OclAny

OclAny = (object : OclAny) : Boolean
True, falls *self* dasselbe Objekt wie *object* ist oder falls beide Datatypen mit gleichen Werten sind. Invalid, falls *object* `invalid` ist.

OclAny <> (object : OclAny) : Boolean
post: result = not (self = object)

Weitere Operationen auf **OclAny**:

oclAsSet(): Set(T)		
oclIsUndefined(): Boolean		
oclIsInvalid(): Boolean		
oclIsTypeOf	(t : Classifier)	:Boolean
oclIsKindOf	(t : Classifier)	:Boolean
oclIsNew	()	:Boolean
oclType	()	:Classifier
oclAsType(type: Classifier): T		
oclIsInState(statespec: OclState): Boolean		
oclLocale: String		

Vergleiche Seite 153f. des OCL-Manuals.

2.24. OclMessage/Signal/Observer und UML-Statusdiagramme

Strukturdiagramme und wechselnde Stati werden in OCL unterstützt durch:

OclState

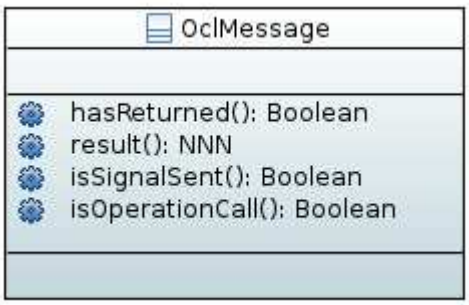
oclIsInState (statespec : OclState) : Boolean

„Possible states for the operation oclInState(s) are all states of the statemachine that defines the classifier's behavior.“ (Seite 23 des OCL-Manuals)

Das OOP-Observer-Pattern, qt-Signal/Slot-Mechanismen und ähnlicher synchroner/ asynchroner Nachrichtenaustausch kann modelliert werden mittels:

OclMessage

Bemerkung:
OclMessage besitzt folgende Methoden:



(Seite 152 und 156 des OCL-Manuals)

Softwarekonstruktion (Prof. Dr. Jan Jürjens), Seite 61
http://www.eclipse.org/gmt/omcw/resources/chapter01/downloads/OCL2.Fraunhofer.ppt
UML2.0OCLSpecification `observer^update(?Integer␣?Integer)`
William N. Robinson u. a.: Awareness Requirements, Seite 17

Observer pattern
Qt-Signal-Slot-Konzept
New Signal Slot Syntax in Qt 5
How Qt Signals and Slots Work - Part 2 - Qt5 New Syntax
Signals and slots in GTK
GTK+ events and signals
aidasignal.hh
Is GTK+ thread safe? How do I write multi-threaded GTK+ applications? [GTK 2.x]
c++-gtk-utils library
Qt Slots and C++11 lambda

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):

Model::Kantine::getHaeuser(): **Set**(Model::Haus)
-- [1...*] {*unique*}
Model::Haus::getKantine(): Model::Kantine
Model::Haus::getEtagen(): OrderedSet(Model::Etage)
-- [1..*] {*unique*, *ordered*}
Model::Etage::getApartment(apartmentNummer: **Integer**): Model::Apartment
...

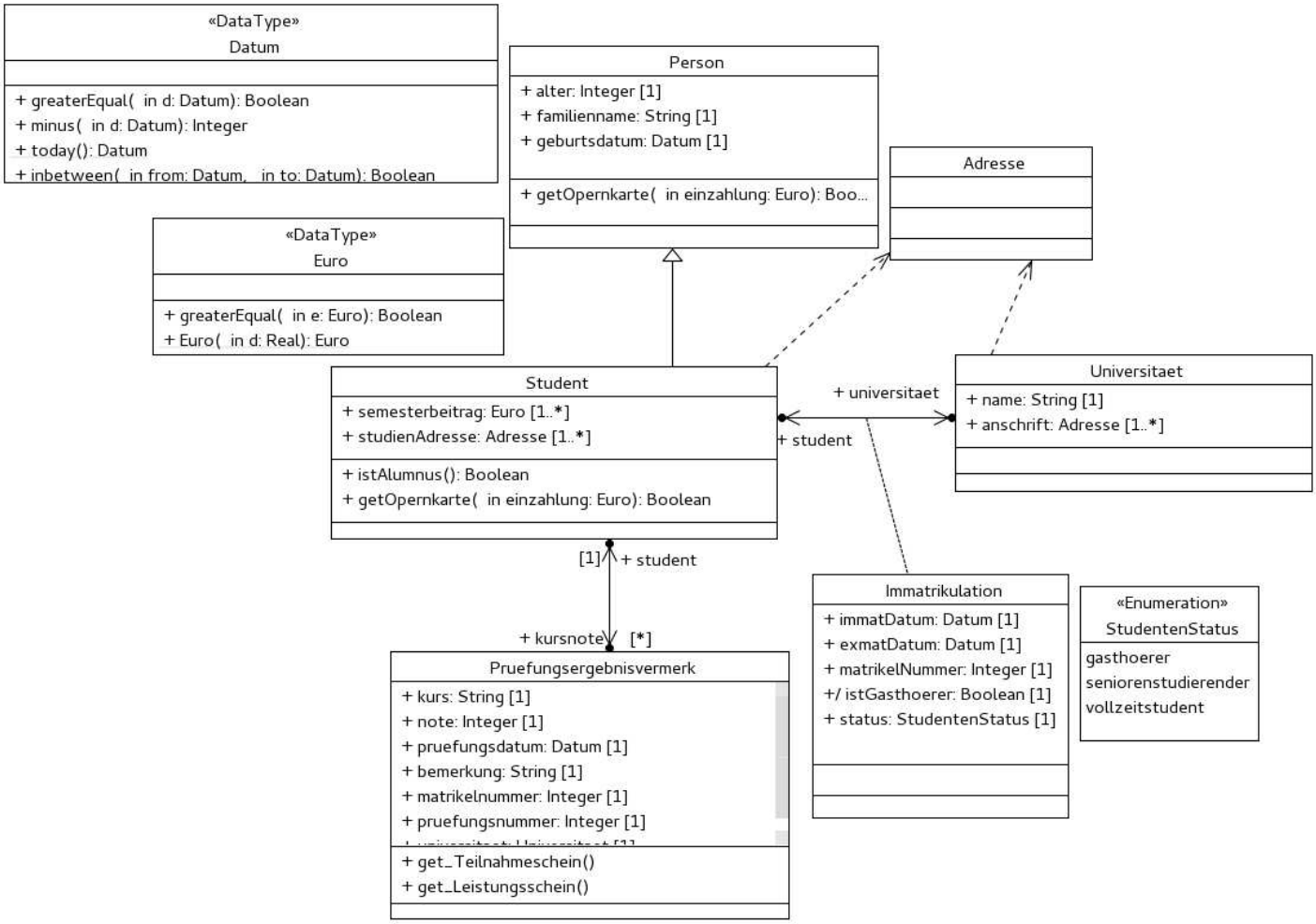
Diese können in C++ bei Benutzung der STL mittels

`set` im Falle `unique`,
`multiset` im Falle `nonunique`,
`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` und `multimap` sowie ihre `unordered` Varianten bereit.

2.26. Ein Modell Student/Universitaet/Pruefungsergebnisvermerk

Das UML-Modell mit Assoziationsklassen und qualifizierten Assoziationen:



Vergleiche Übungsblatt 11 und:

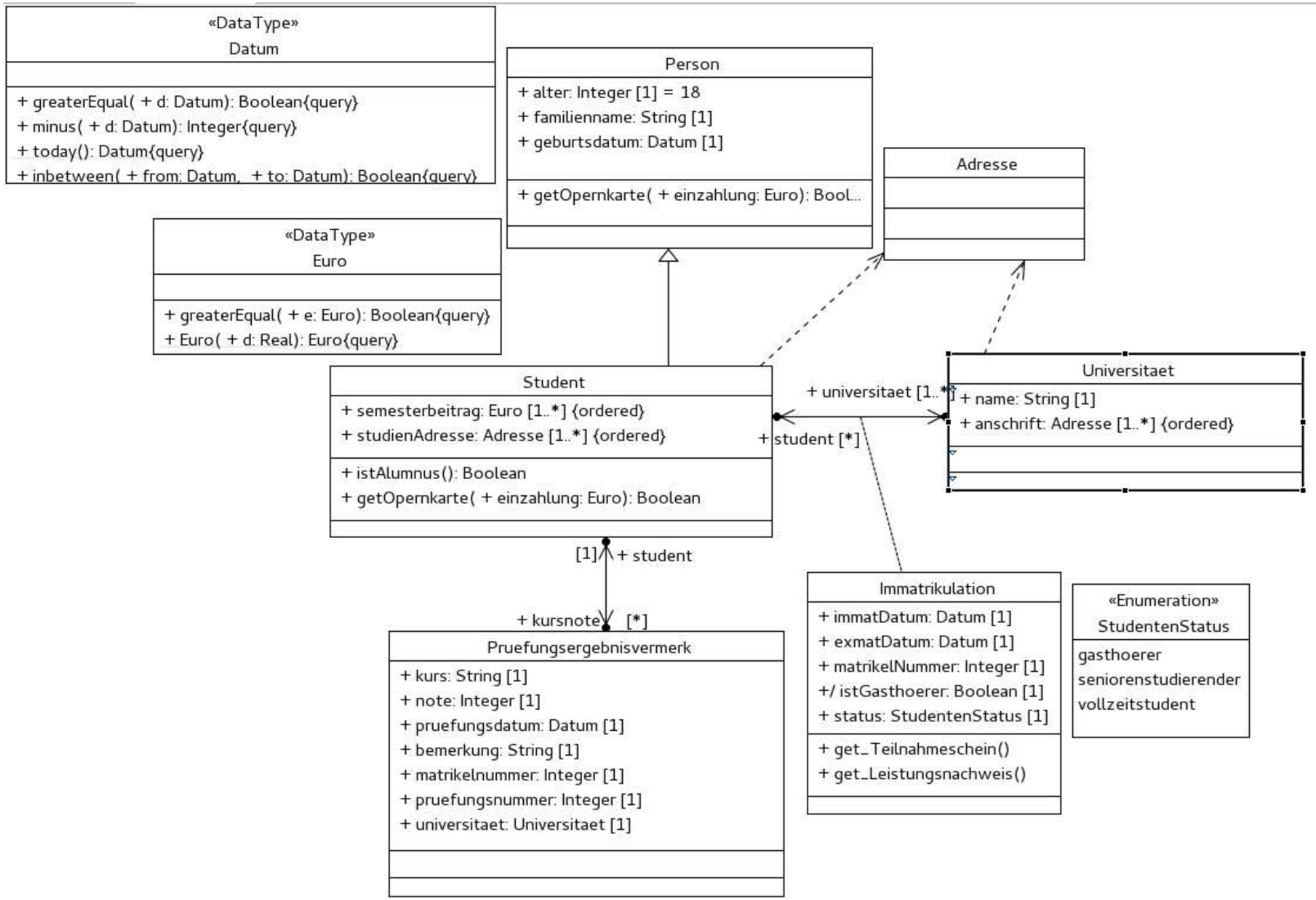
2.27. Contracts zum Modell Student/Universitaet/Pruefungsergebnisvermerk (OCL 2.4 mit Papyrus und OCL-Tools (sowie Workarounds))

```
import 'model.uml'

context Model::Universitaet
inv: name <>''
inv: student->size() >= 0
inv: Immatrikulation->isUnique(i: Immatrikulation | i.
   matrikelNummer)
inv: student.universitaet->includes(self)
inv: student[0312345].familienname = 'Bauer'

context Model::Student
inv: Immatrikulation->size() >= 1
inv: Immatrikulation->select(i: Immatrikulation | not i.
   istGasthoerer)->size()>=1
inv: universitaet.student->includes(self)
inv: kursnote->size() > 0 implies
       kursnote.student->asSet() = Set{self}

context Model::Student::istAlumnus(): Boolean
body: not Immatrikulation->exists(i: Immatrikulation | i.
   exmatDatum.oclIsUndefined())

context Model::Student::getOpernkarte(einzahlung: Model::Euro):
    Boolean
pre: einzahlung.greaterEqual(Model::Euro(10)) -- allgem. als in
    Person
post: result = true

context Model::Person::getOpernkarte(einzahlung: Model::Euro):
   Boolean
pre: einzahlung.greaterEqual(Model::Euro(150)) -- in Person
post: result = true

context Model::Immatrikulation
inv: matrikelNummer > 0
inv: not exmatDatum.oclIsUndefined() implies exmatDatum.
   greaterEqual(immatDatum)
```

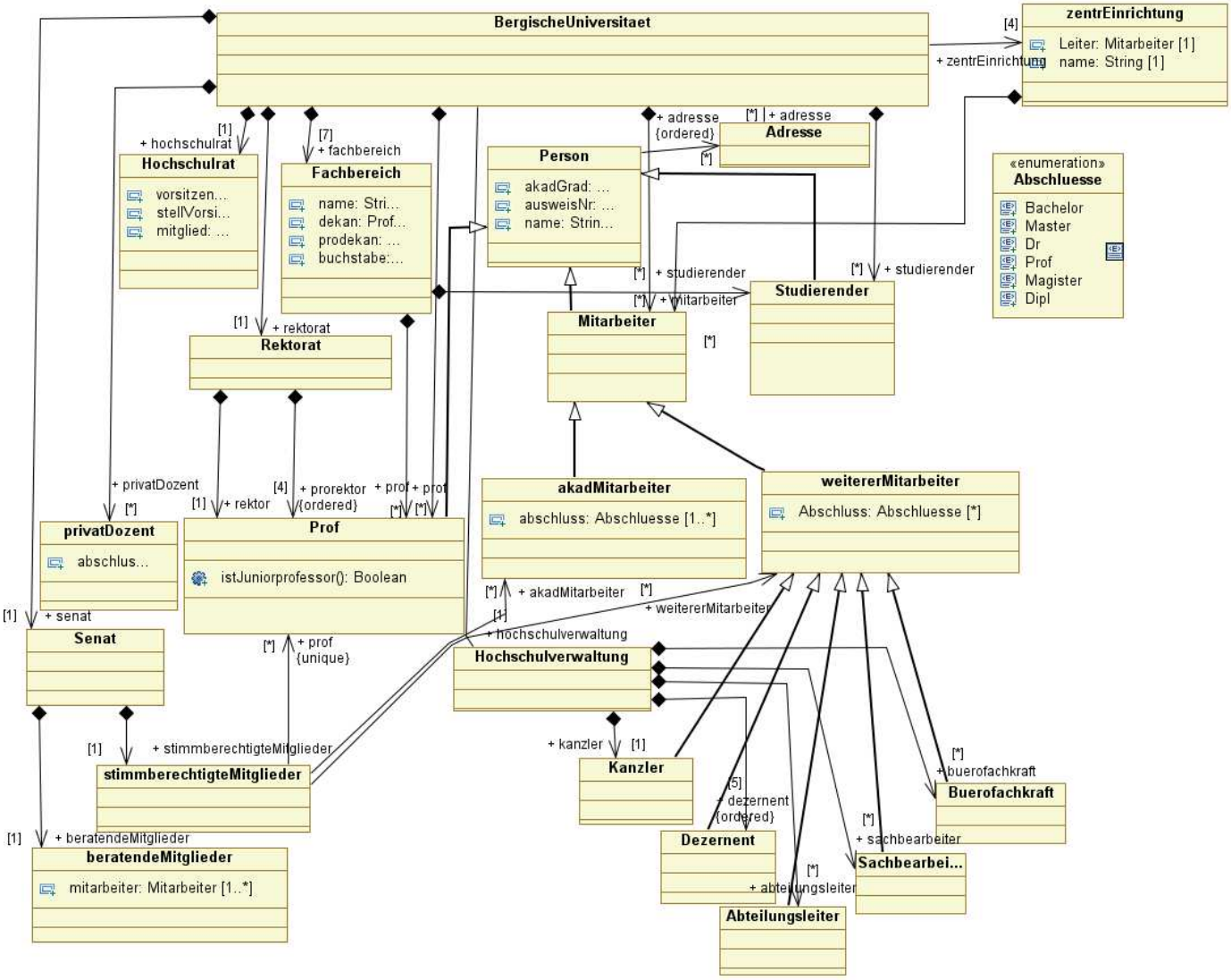
```
inv: immatDatum.minus(student->any(true).geburtsdatum) >= 12
inv: not exmatDatum.oclIsUndefined() implies Datum::today().
   greaterEqual(exmatDatum)

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

context Model::Immatrikulation::exmatDatum: Model::Datum
init: null -- 2. Versuch eines optionalen Datentyps:
           -- hier: null als Ungueltig-Marke

context Model::Immatrikulation::istGasthoerer: Boolean
init: false

context Model::Pruefungsergebnisvermerk
inv: kurs <> ''
inv: 1 <= note and note <= 5
inv: matrikelnummer > 0
inv: student.Immatrikulation.matrikelNummer->includes(
   matrikelnummer)
inv: pruefungsdatum.inbetween(
                student.Immatrikulation->any(matrikelNummer =
                   matrikelnummer).immatDatum,
                student.Immatrikulation->any(matrikelNummer =
                   matrikelnummer).exmatDatum
                )
inv: pruefungsnummer > 0
inv: Pruefungsergebnisvermerk.allInstances()->isUnique(
   pruefungsnummer)
inv: student.kursnote->includes(self)
inv: student.kursnote[pruefungsnummer]=self

/************************************************/

context Model::Immatrikulation::istGasthoerer: Boolean
derive: status  = StudentenStatus::gasthoerer

context Model::Immatrikulation
inv: istGasthoerer implies student.Immatrikulation->
           select(status=StudentenStatus::vollzeitstudent)->
              size()>0
```

```
context Model::Pruefungsergebnisvermerk::get_Teilnahmeschein():
    OclVoid
pre: let fuerUni: Universitaet = universitaet in
     student.Immatrikulation->select(matrikelnummer=
        matrikelNummer)->
                                  any(fuerUni=universitaet).
                                     istGasthoerer


context Model::Pruefungsergebnisvermerk::get_Leistungsschein():
    OclVoid
pre: not (let fuerUni: Universitaet = universitaet in
                    student.Immatrikulation->select(
                       matrikelnummer=matrikelNummer)->
                                        any(fuerUni=universitaet).
                                           istGasthoerer
          )
```

```
context Model::Immatrikulation
inv: status = StudentenStatus::seniorenstudierender implies
               student.alter >= 60

context Model::Immatrikulation
inv: status = StudentenStatus::vollzeitstudent implies
               student.alter >= 12
```


«DataType»
Datum
+ greaterEqual(+ d: Datum): Boolean{query}
+ minus(+ d: Datum): Integer{query}
+ today(): Datum{query}
+ inbetween(+ from: Datum, + to: Datum): Boolean{query}
Person
+ alter: Integer [1] = 18
+ familienname: String [1]
+ geburtsdatum: Datum [1]
+ getOpernkarte(+ einzahlung: Euro): Bool...
Adresse
«DataType»
Euro
+ greaterEqual(+ e: Euro): Boolean{query}
+ Euro(+ d: Real): Euro{query}
Student
+ semesterbeitrag: Euro [1..*] {ordered}
+ studienAdresse: Adresse [1..*] {ordered}
+ istAlumnus(): Boolean
+ getOpernkarte(+ einzahlung: Euro): Boolean
+ universitaet [1..*]
+ student [*]
Universitaet
+ name: String [1]
+ anschrift: Adresse [1..*] {ordered}
[1]
+ student
+ kursnote
[*]
Pruefungsergebnisvermerk
+ kurs: String [1]
+ note: Integer [1]
+ pruefungsdatum: Datum [1]
+ bemerkung: String [1]
+ matrikelnummer: Integer [1]
+ pruefungsnummer: Integer [1]
+ universitaet: Universitaet [1]
Immatrikulation
+ immatDatum: Datum [1]
+ exmatDatum: Datum [1]
+ matrikelNummer: Integer [1]
+/ istGasthoerer: Boolean [1]
+ status: StudentenStatus [1]
+ get_Teilnahmeschein()
+ get_Leistungsnachweis()
«Enumeration»
StudentenStatus
gasthoerer
seniorenstudierender
vollzeitstudent

2.28. Ein Modell BergischeUniversitaet



und einige Constraints:

```
context Model::stimmberechtigteMitglieder
inv: prof->size() = 12 and
     akadMitarbeiter->size() = 4 and
     weitererMitarbeiter->size() = 4 and
     studierender->size()=2
...
contex Model::Prof
inv: self.oclAsType(Person).akadGrad->includes(Abschluesse::
   Prof)
```

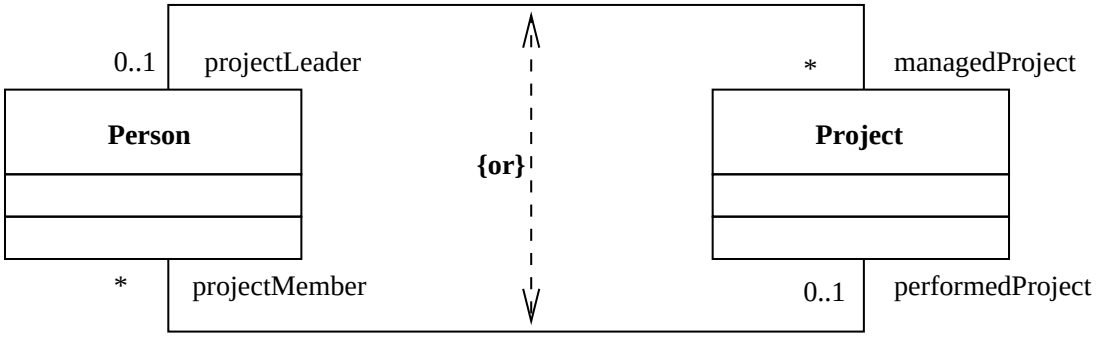
```
context Model::Rektorat
inv: prorektor->isUnique(p | p.oclAsType(Person).ausweisNr) and
     prorektor->excludes(rektor)

...
```

„Mitarbeiter können auch Studierende sein“ heißt formal:

```
context Model::Mitarbeiter
inv: bergischeUniversitaet.studierender.oclAsType(Person)->
   includes(self)
    or
    bergischeUniversitaet.studierender.collect(oclAsType(Person
       ))->excludes(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:

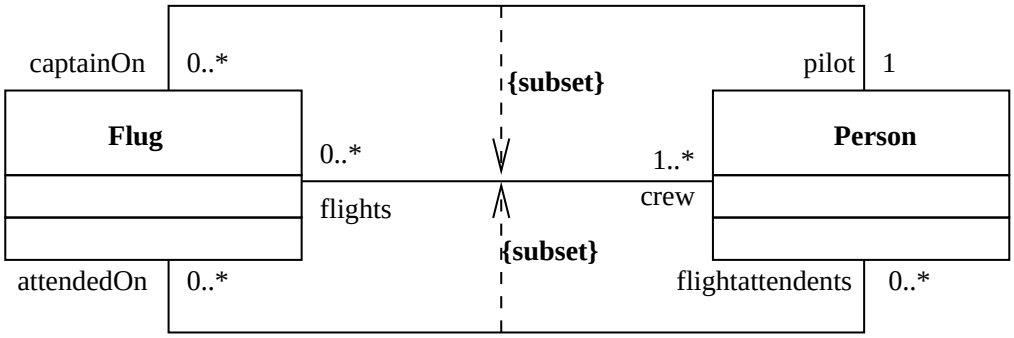
context Model::Person
inv: not managedProject → isEmpty() or
not performedProject → isEmpty()

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

(oder falls die linken Assoziationsenden von Bedeutung:)
context Model::Project
inv: not projectLeader → isEmpty() or
not projectMember → isEmpty()

Jedes Projekt hat entweder ein Projektleiter oder ein Projektmitglied.

2.30. subset



Subset-Constructs machen UML-Diagramme häufig schlecht lesbar.
Besser ist:

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

oder:

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

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

2.31. OCL primitive type Real

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

9.3.19 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>

Abstract syntax mapping

RealLiteralExpCS.ast : RealLiteralExp

Synthesized attributes

RealLiteralExpCS.ast.realSymbol = <Real Value>

Inherited attributes

-- none

Disambiguating rules

-- none

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 9.3.20 im OCL-Manual).

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* × *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**.

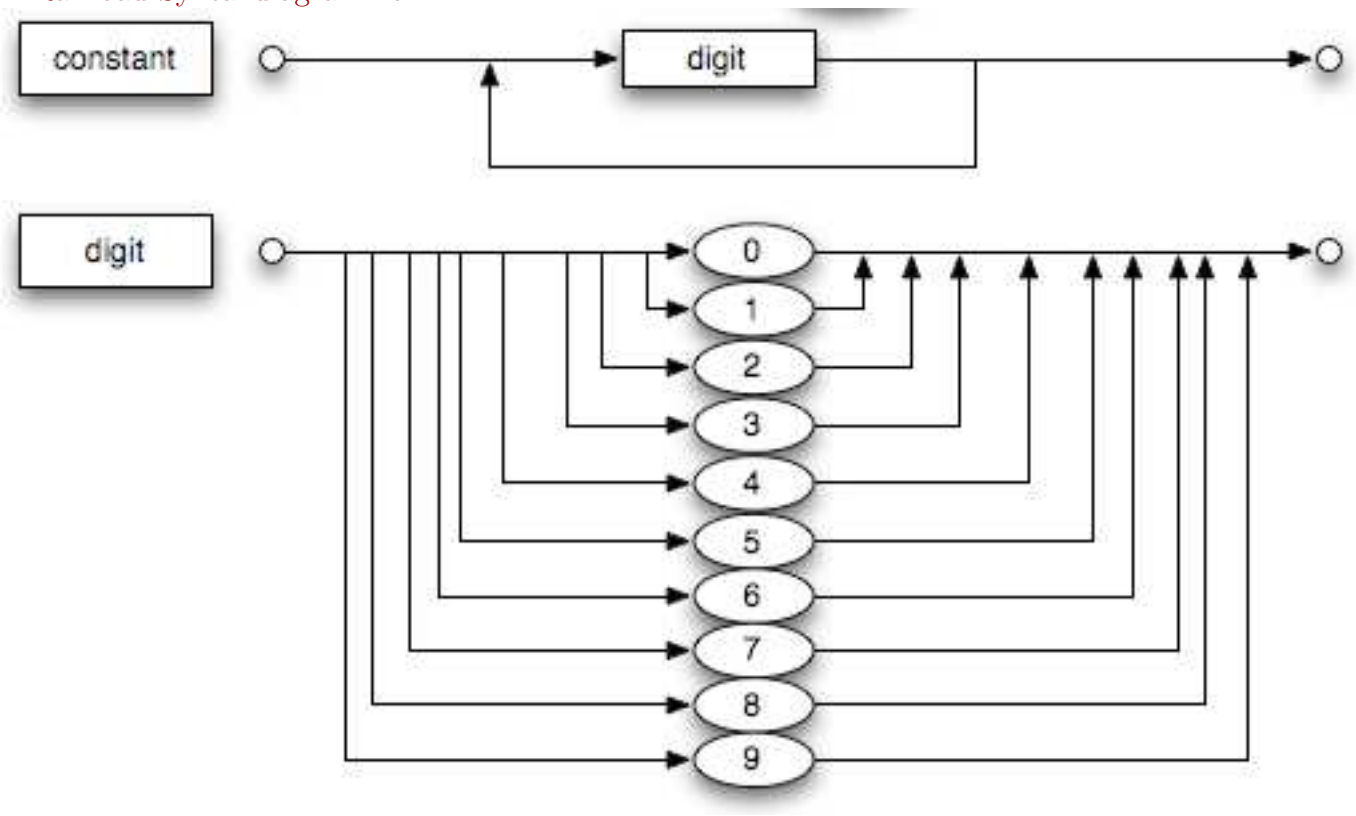
3.2.5.2 Canonical representation

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";
```

- EBNF for XML 1.0

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";
```

2.32. OCLs dreiwertige Logik

partielle Funktionen = Funktionen mit undefinierten Stellen, zum Beispiel:

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

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

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

und macht dadurch *Inv()* zur totalen Funktion auf $\mathbb{R}$, indem man den Wertebereich auf $\mathbb{R} \cup \{\bot\}$ vergrößert. $\bot$ 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	ε	false	ε	ε	true	true
true	ε	ε	true	ε	ε	false
false	⊥	false	⊥	⊥	true	true
true	⊥	⊥	true	⊥	⊥	false
ε	false	false	ε	ε	ε	ε
ε	true	ε	true	ε	true	ε
ε	ε	ε	ε	ε	ε	ε
ε	⊥	⊥	⊥	⊥	⊥	ε
⊥	false	false	⊥	⊥	⊥	⊥
⊥	true	⊥	true	⊥	true	⊥
⊥	⊥ or ε	⊥	⊥	⊥	⊥	⊥

Table A.3 - - Additional semantics of unlimited natural comparisons

v_1	v_2	$v_1 = v_2$	$v_1 <> v_2$	$v_1 < v_2$	$v_1 \leq v_2$	$v_1 \geq v_2$	$v_1 > v_2$
a	b	$a = b$	$a <> b$	$a < b$	$a \leq b$	$a \geq b$	$a > b$
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
∞	∞	⊥

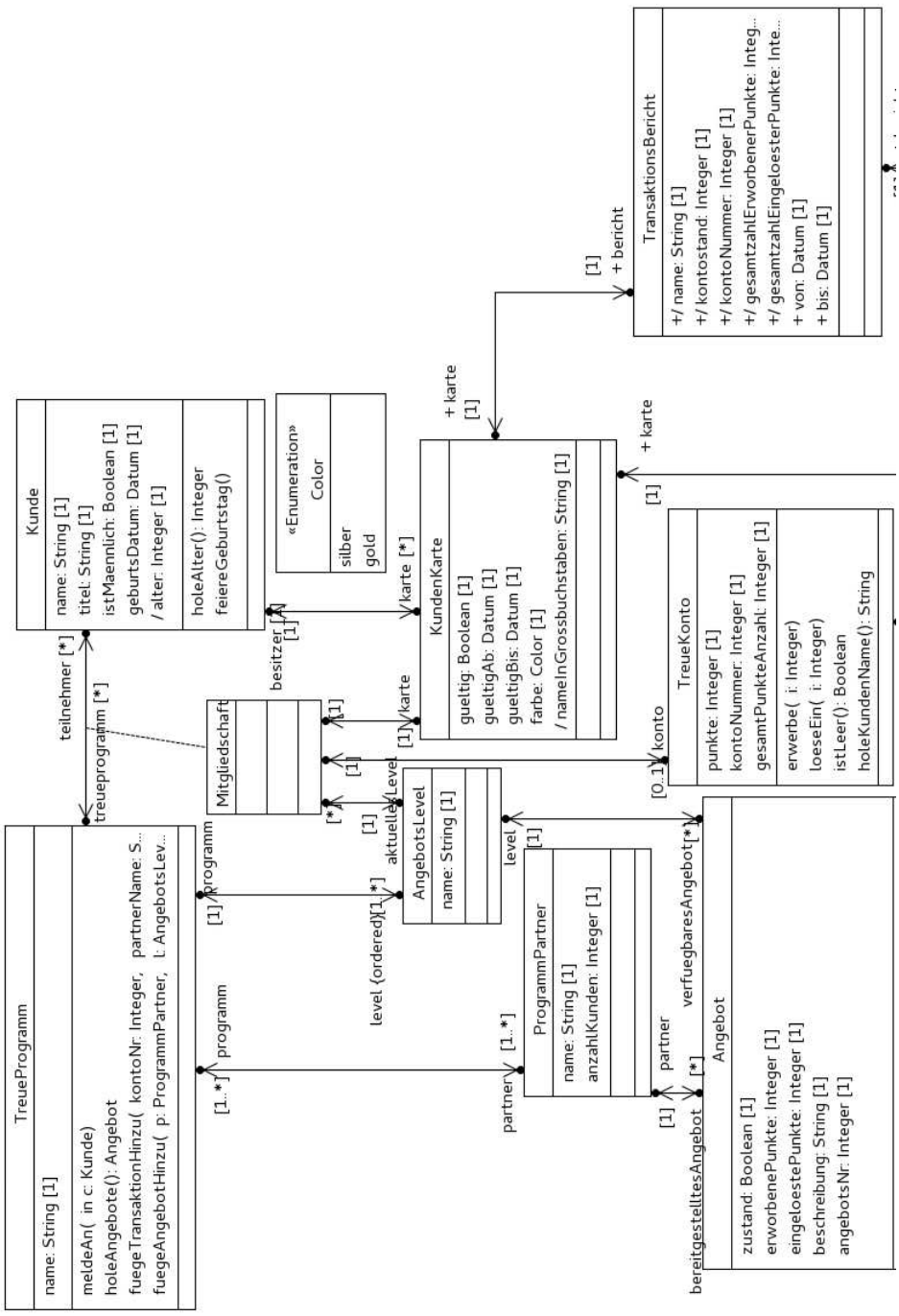
Table A.5 - - Additional semantics of unlimited natural diadic operations

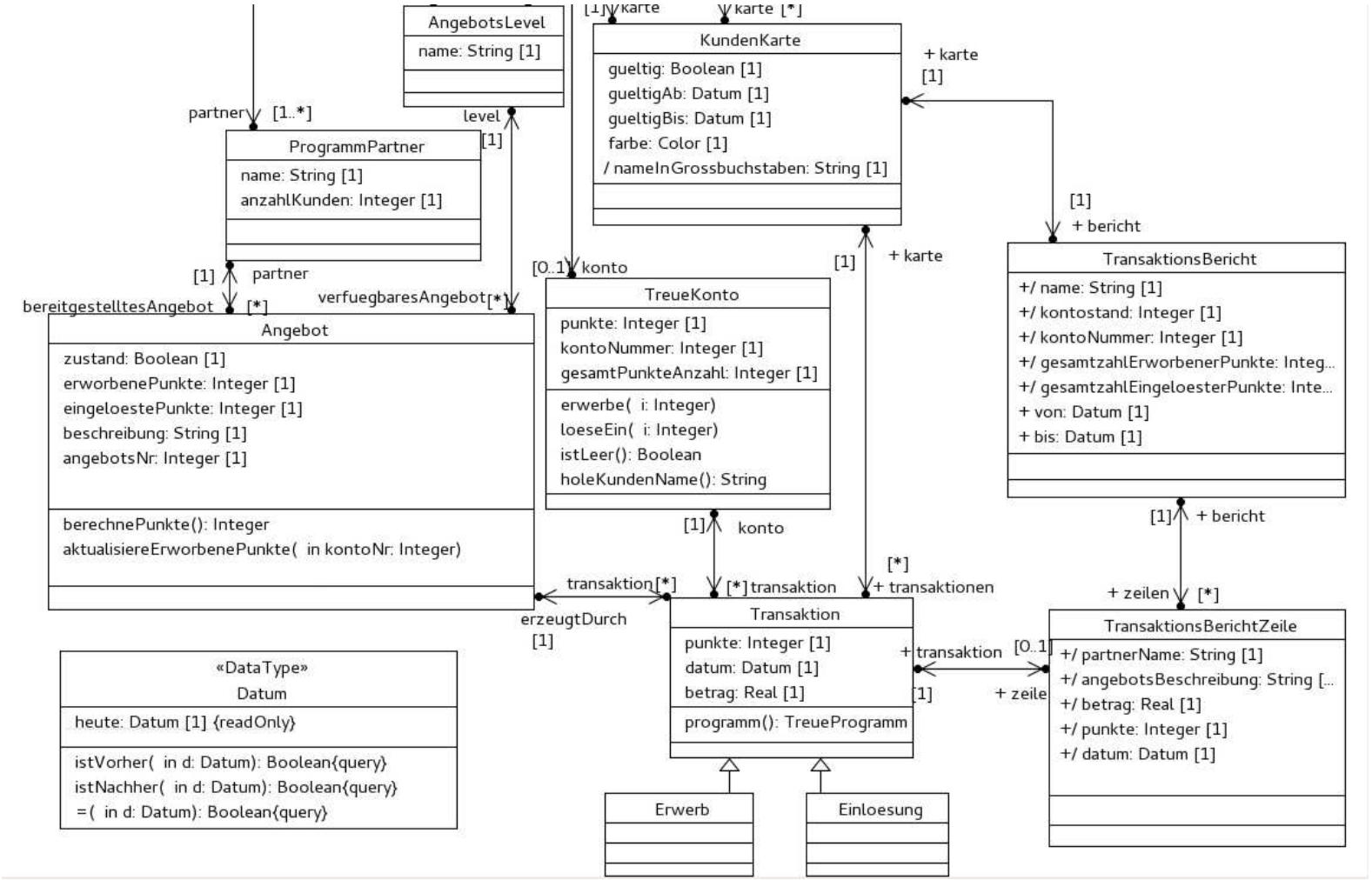
v_1	v_2	$v_1 + v_2$	$v_1 * v_2$	v_1 / v_2	$mod(v_1,v_2)$	$max(v_1,v_2)$	$min(v_1,v_2)$
a	b	$a + b$	$a * b$	a / b	$mod(a,b)$	$max(a,b)$	$min(a,b)$
a	∞	⊥	⊥	⊥	⊥	∞	⊥
∞	b	⊥	⊥	⊥	⊥	∞	⊥
∞	∞	⊥	⊥	⊥	⊥	∞	∞

Diese Tabellen wären programmierergeeigneter, wenn die OCL-Literale `invalid, null` und $*$ statt der Symbole $\perp, \varepsilon, \infty$ der Theoretischen Informatik benutzt würden.

dreiwertige Logik

2.33. OCL-Fallstudie „Vorzugs- und Treuekunden“ / „Royal and Loyal Model“ (nach Jos Warmer)



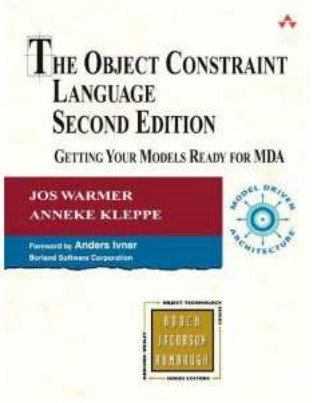

AngebotsLevel
name: String [1]
KundenKarte
gueltig: Boolean [1]
gueltigAb: Datum [1]
gueltigBis: Datum [1]
farbe: Color [1]
/ nameInGrossbuchstaben: String [1]
+ karte
[1]
partner
[1..*]
level
[1]
ProgrammPartner
name: String [1]
anzahlKunden: Integer [1]
[1]
+ bericht
[1]
partner
[0..1]
konto
[1]
+ karte
bereitgestelltesAngebot
[*]
verfuegbaresAngebot
[*]
TreueKonto
punkte: Integer [1]
kontoNummer: Integer [1]
gesamtPunkteAnzahl: Integer [1]
erwerbe(i: Integer)
loeseEin(i: Integer)
istLeer(): Boolean
holeKundenName(): String
TransaktionsBericht
+/ name: String [1]
+/ kontostand: Integer [1]
+/ kontoNummer: Integer [1]
+/ gesamtzahlErworbenerPunkte: Integ...
+/ gesamtzahlEingeloesterPunkte: Inte...
+ von: Datum [1]
+ bis: Datum [1]
Angebot
zustand: Boolean [1]
erworbenePunkte: Integer [1]
eingeloestePunkte: Integer [1]
beschreibung: String [1]
angebotsNr: Integer [1]
berechnePunkte(): Integer
aktualisiereErworbenePunkte(in kontoNr: Integer)
[1]
konto
[1]
+ bericht
[*]
transaktion
[*]
[*]
transaktion
[*]
+ transaktionen
+ zeilen
[*]
erzeugtDurch
[1]
Transaktion
punkte: Integer [1]
datum: Datum [1]
betrag: Real [1]
programm(): TreueProgramm
+ transaktion
[1]
[0..1]
+ zeile
TransaktionsBerichtZeile
+/ partnerName: String [1]
+/ angebotsBeschreibung: String [...
+/ betrag: Real [1]
+/ punkte: Integer [1]
+/ datum: Datum [1]
«DataType»
Datum
heute: Datum [1] {readOnly}
istVorher(in d: Datum): Boolean{query}
istNachher(in d: Datum): Boolean{query}
=(in d: Datum): Boolean{query}
Erwerb
Einloesung

```
import 'model.uml'

context Model::TreueKonto::punkte: Integer
init: 0

context Model::KundenKarte::gueltig: Boolean
init: true

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

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

-- ...
```

Vertiefende Literatur:

Jos Warmer
The Object Constraint Language Second Edition
Addison-Wesley
Erscheinungsdatum: 2003

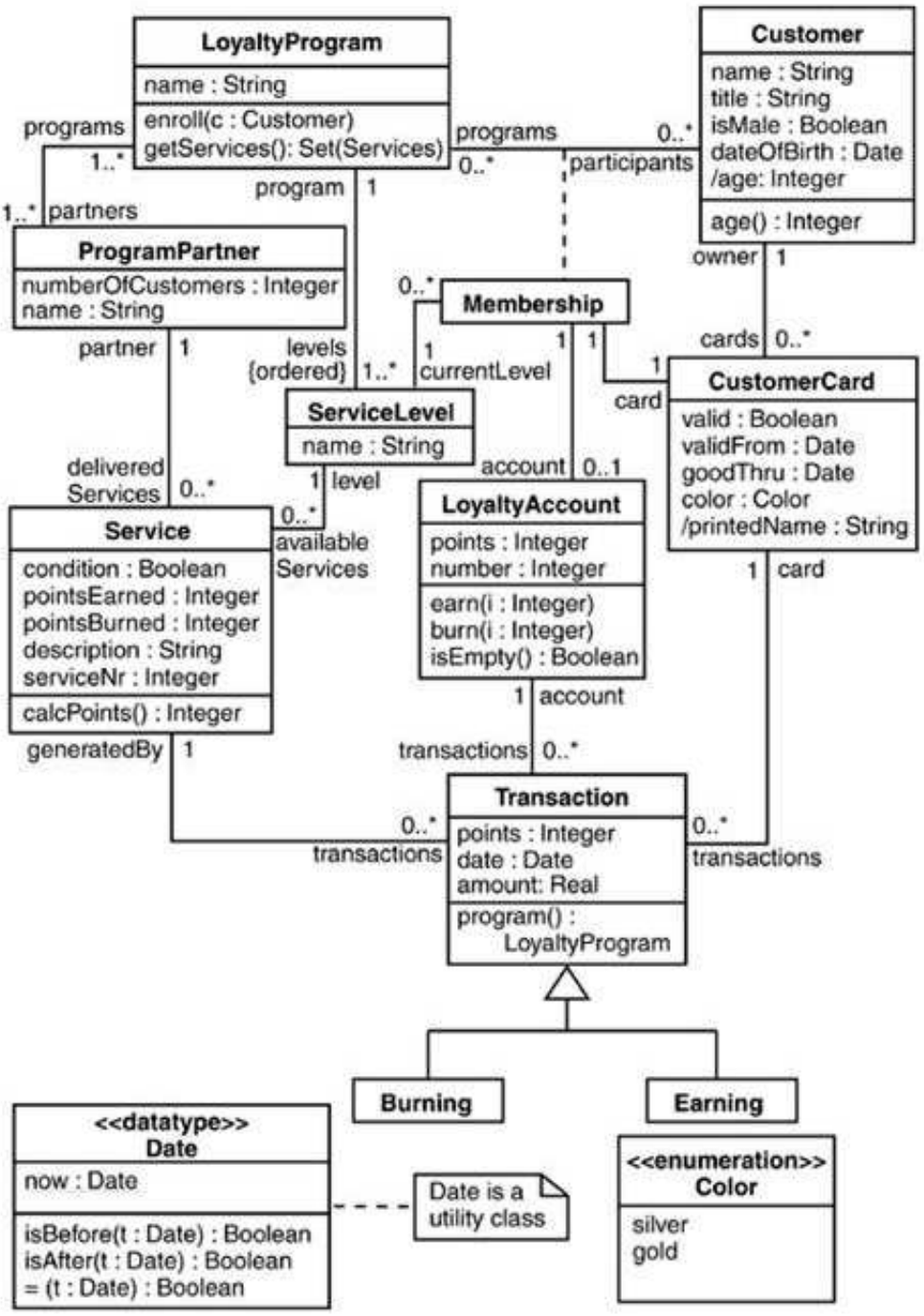
ISBN: 0321179366



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

Anhang A.7

Figure 2-1. The Royal and Loyal model



2.34. Stil-Hinweise für OCL-Ausdrücke

- 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 Model::Person
inv: parents->size() <= 2
```

  und

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

    einfacher als

```
context Model::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 Model::Person
inv: self.parents.brothers.children->notEmpty()
```

  ist äquivalent zu:

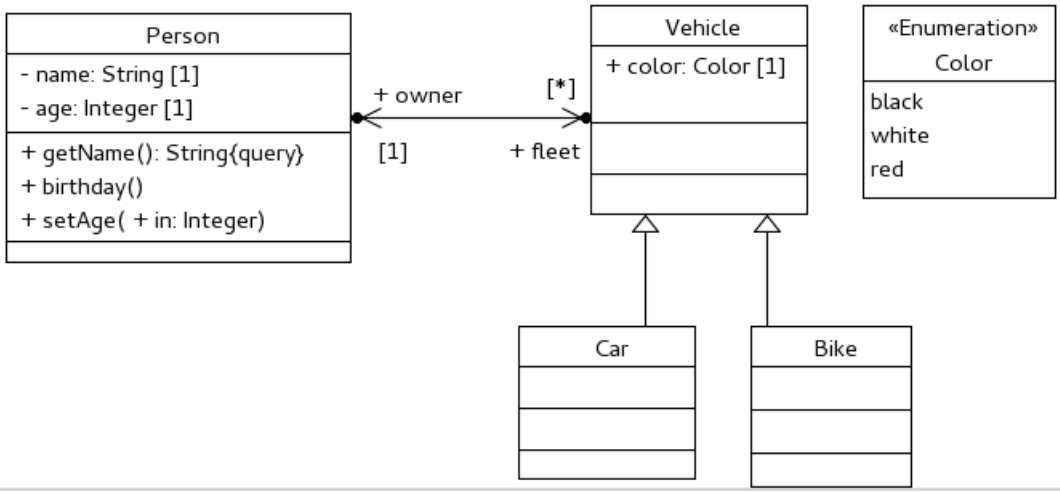
```
context Model::Person
inv: self.parents->collect(brothers)
                 ->collect(children)->notEmpty)
```

- Gib allen Assoziationsenden einen Namen.

2.35. Einfache Beispielverträge und die geeignete Kontextwahl

Siehe: Introduction to OCL

Geeignete Kontextwahl macht semantisch äquivalente Constraints strukturell einfacher:




```
context Model::Car
inv: self.color = Color::red


-- oder:


context Model::Vehicle
inv: if self.oclIsKindOf(Car) then
        self.color = Color::red
     else
        true
     endif
```

A. Zusatzmaterial

A.1. 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).

A.2. OCL in Together-Tools: Language-Bindings

mit OCL-Constraints, Code-Erzeugung, Language Bindings, ...

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

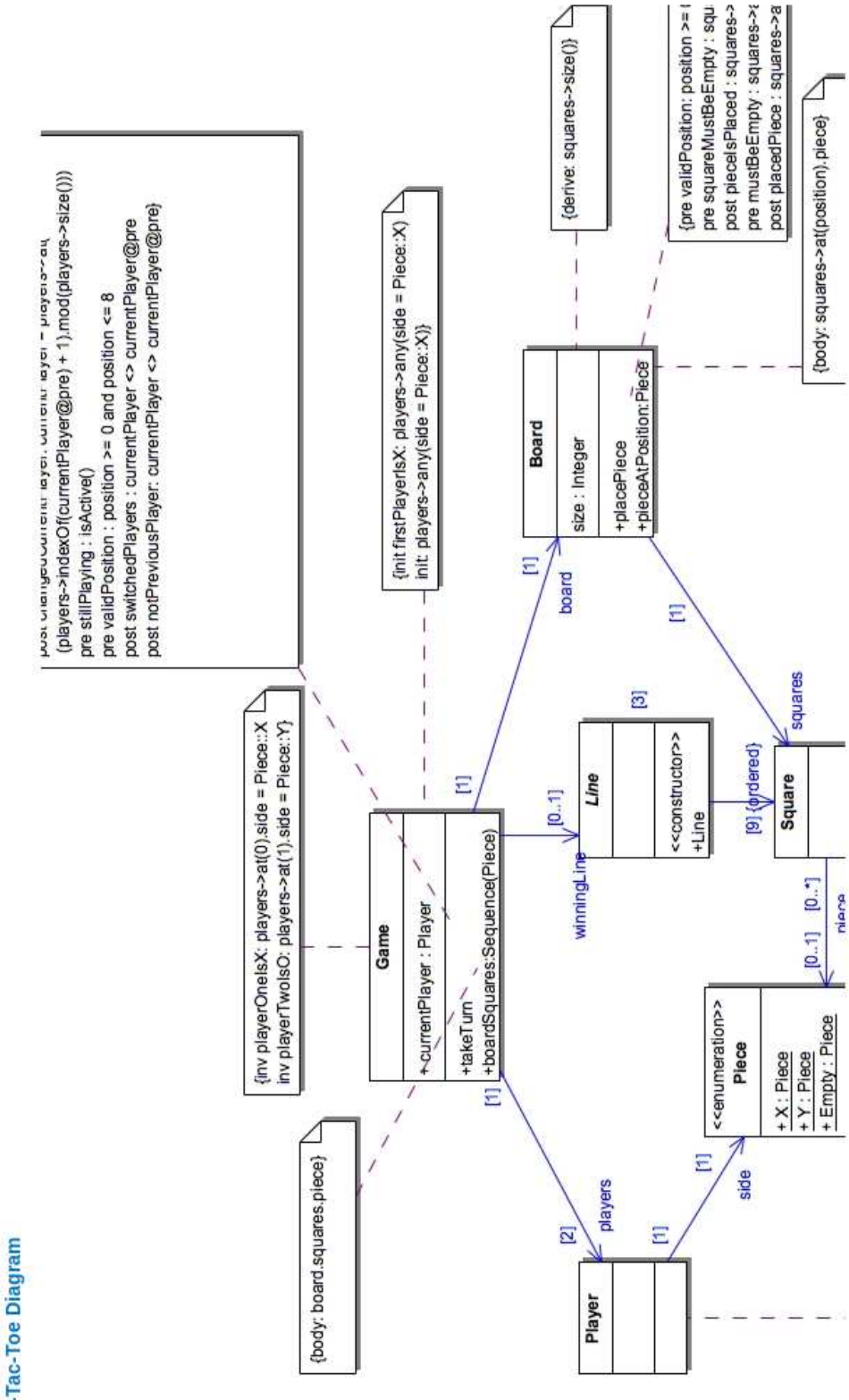


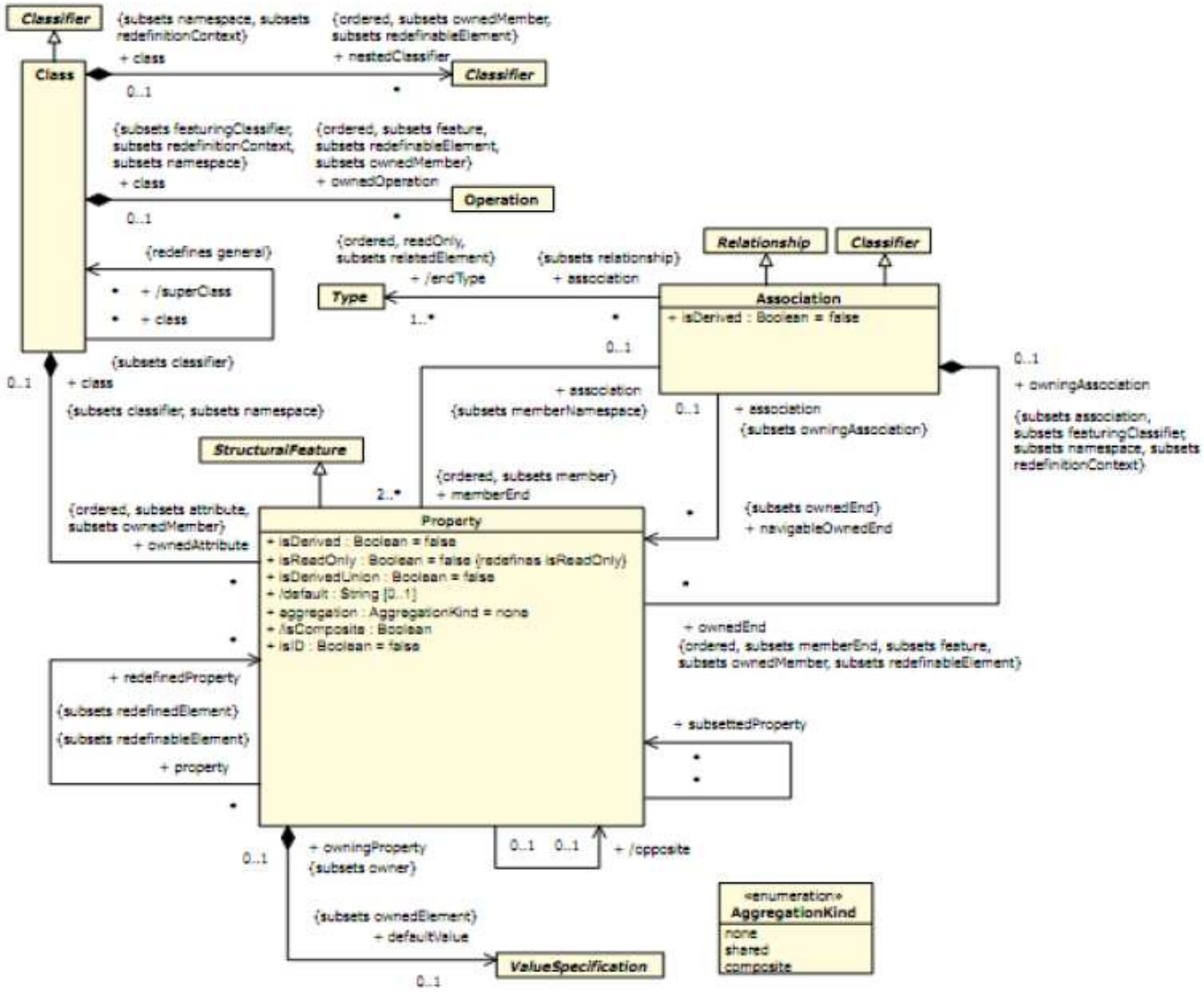
Figure 5. Tic-Tac-Toe Domain

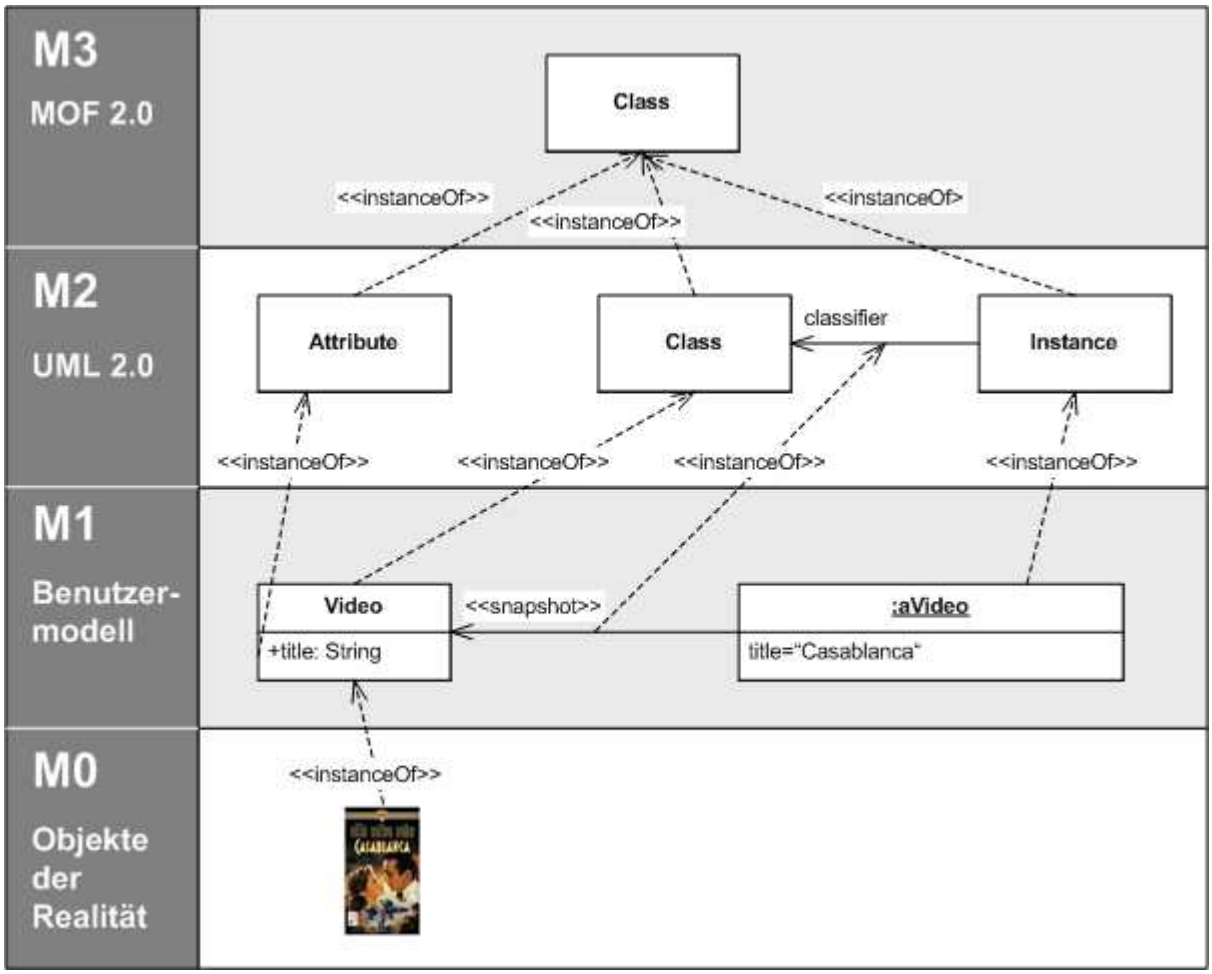
A.3. OMGs C++ Language-Mapping für CORBA

C++ language mapping
CORBA
Currency Service

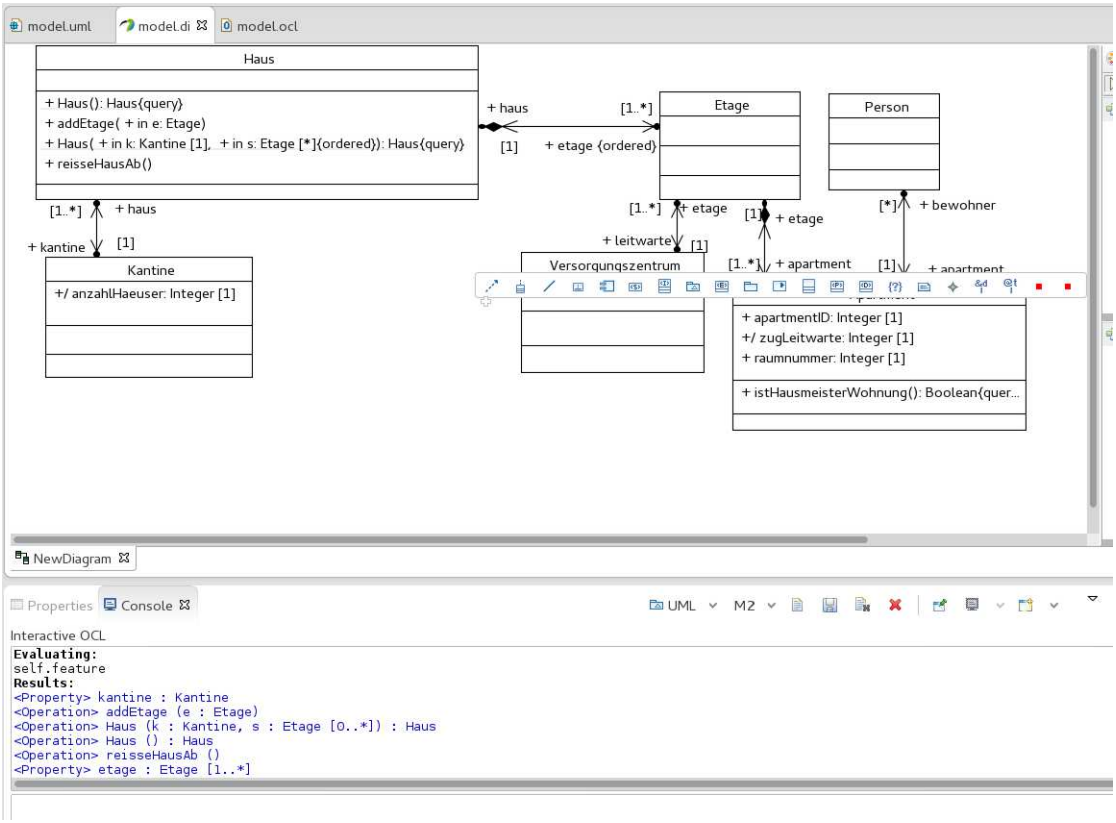
A.4. Was ist ein UML-Modell: MOF

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




M3
MOF 2.0
Class
<<instanceOf>>
<<instanceOf>>
<<instanceOf>
M2
UML 2.0
Attribute
Class
classifier
Instance
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
M1
Benutzer-
modell
Video
+title: String
<<snapshot>>
:aVideo
title="Casablanca"
M0
Objekte
der
Realität
<<instanceOf>>

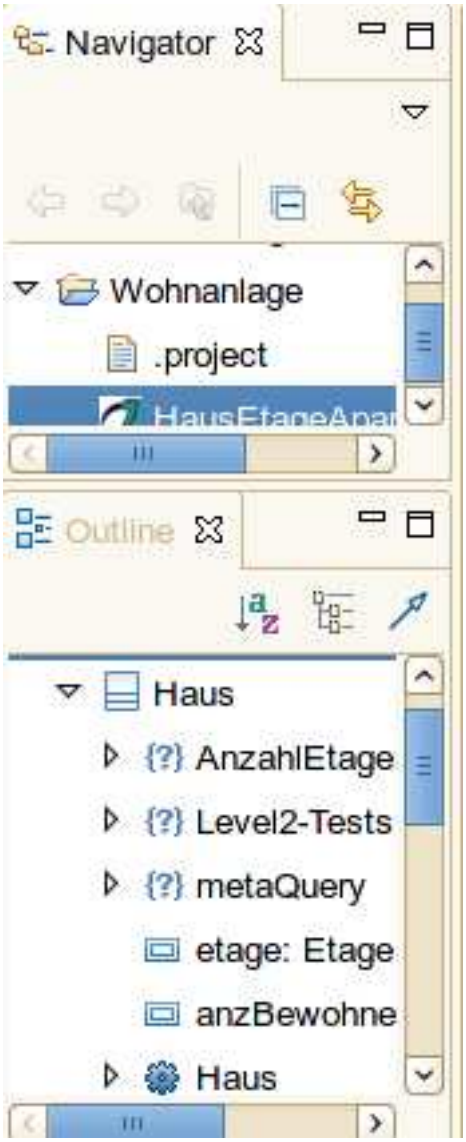
A.5. Metalevel2-Constraints = Wohldefiniertheitsregeln für Modelle



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 ReisseHausAb

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.olcIsUndefined())->size()=1

"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 WFR_5:
```

```
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 WFR_4:
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 für UML, CWM, ...

M1 Geschäftsregeln, SdV, DbC, Spezifikation von Testfällen,WFRs des UML-Modells

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.oclIsKindOf(NamedElement) and e.oclAsType(NamedElement).name <> null)
  ->collect(n | n.oclAsType(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)

Siehe auch: Ensuring UML models consistency ...

A.6. OCL-Beispiele

http://www.empowertec.de/ocl/examples/
(vgl. OCL demonstration)
The university model
Object Constraint Language (OCL) tutorial
OCL2 to SQL transormation
Experiences with the UML/OCL-Approach in Practice ...
...

A.7. OCL-Fallstudie Vorzugs- und Treuekartenkunden/„Royal and Loyal Model“(nach Jos Warmer, Fortsetzung 2.33)

```
import 'model.uml'

context Model::TreueKonto::punkte: Integer
init: 0

context Model::KundenKarte::gueltig: Boolean
init: true

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

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

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

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

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

context Model::Kunde
inv geschaeftsfaehigesAlter: alter >= 18

context Model::KundenKarte
inv: gueltigAb.istVorher(gueltigBis)

context Model::KundenKarte
inv altersBeschraenkung: besitzer.alter >= 18

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

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

context Model::Mitgliedschaft
def: holeAktuellenLevelNamen():String =
      aktuellesLevel.name

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

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

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

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

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

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

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

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

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

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

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

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

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

context Model::KundenKarte
def: karteIstGueltig: Boolean =
   let gueltigesDatum: Boolean =
      gueltigAb.istVorher(Datum::heute) and
      gueltigBis.istNachher(Datum::heute)
   in
      gueltig and gueltigesDatum

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

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

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

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

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

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

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

context Model::TreueProgramm::meldeAn(c: Model::Kunde): OclVoid
pre: c.name <> ''
pre: c->notEmpty()
pre: not (teilnehmer->includes(c))
post: teilnehmer = teilnehmer@pre->including(c)

-- derived class TransaktionsBerichtZeile

context Model::TransaktionsBerichtZeile::partnerName: String
derive: transaktion.erzeugtDurch.partner.name

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

context Model::TransaktionsBerichtZeile::betrag: Real
derive: transaktion.betrag

context Model::TransaktionsBerichtZeile::punkte: Integer
derive: transaktion.punkte

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

-- patial derived class TransaktionsBericht

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

context Model::TransaktionsBericht::kontostand: Integer
derive: karte.Mitgliedschaft.konto.punkte

context Model::TransaktionsBericht::kontoNummer: Integer
derive: karte.Mitgliedschaft.konto.kontoNummer

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

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

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

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

-------------------------------------------------

context Model::TreueProgramm::fuegeTransaktionHinzu(kontoNr: Integer,
                                                    partnerName: String,
                                                    angebNr: Integer,
                                                    betr: Real,
                                                    d: Model::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).karte
    in
        kto.punkte = kto.punkte@pre + neueT.punkte and
        neueT.oclIsNew() and
        betr = 0 implies neueT.oclIsTypeOf(Einloesung) and
        betr > 0 implies neueT.oclIsTypeOf(Erwerb) and
        kto.transaktion - kto.transaktion@pre = Set{newT} and
        karte.transaktionen - karte.transaktionen@pre = Set{newT}
```

```
context Model::TreueProgramm::fuegeAngebotHinzu(p: Model::ProgrammPartner,
                                                 l: Model::AngebotsLevel,
                                                 a: Model::Angebot): OclVoid
pre: partner->includes(p)
pre: level->includes(l)
pre: a->notEmpty()
post: partner.bereitgestelltesAngebot->includes(a)
post: level.verfuegbaresAngebot->includes(a)
```

```
context Model::Kunde
def: extensivGenutzeKarten: Set(Model::KundenKarte) =
karte->select(transaktionen.punkte->sum() > 10000)

def: firmenTreuZu: Bag(Model::ProgrammPartner) =
     treueprogramm.partner

def: kartenFuerTreueProgramm(p: Model::TreueProgramm): Set(Model::KundenKarte) =
     p.mitgliedschaft.karte->asSet()
```

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

context Model::Mitgliedschaft
inv keinRabatt: treueprogramm.ohnePunktErwerbsMoeglichkeiten implies
       konto->isEmpty()
```

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