Materialsammlung Formale Methoden: OCL, Eclipse OCL-SDK

Prof. Dr. Hans-Jürgen Buhl

Sommersemester 2018

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften
Fachgruppe Mathematik und Informatik

Praktische Informatik
PIBUW - SS18
April 2018
12. Auflage, 2018

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.

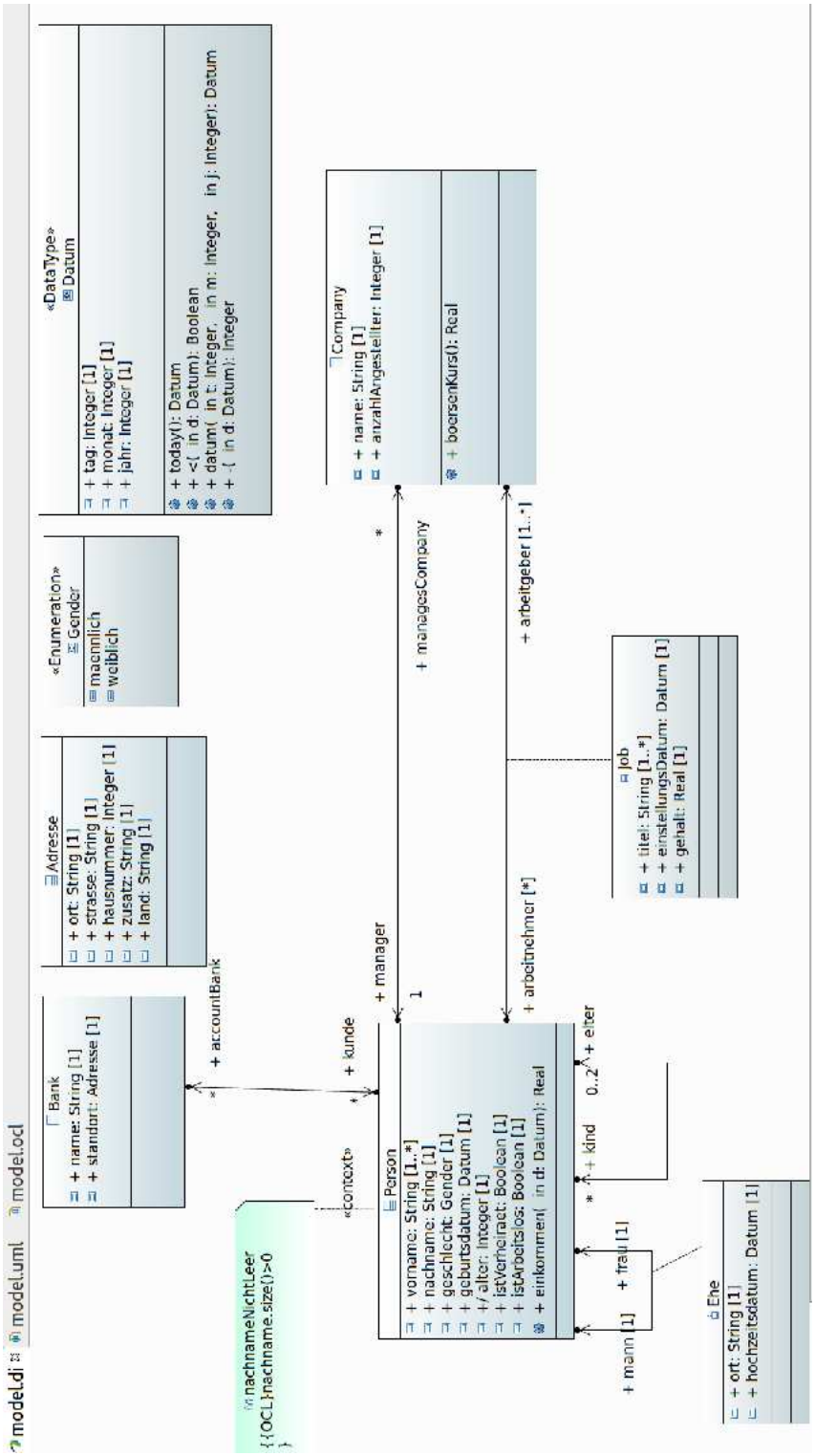
Inhalte:

- 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 Mi 16-18 G.16.15

(Modulhandbuch Seite 47f.)

Ein UML-Modell

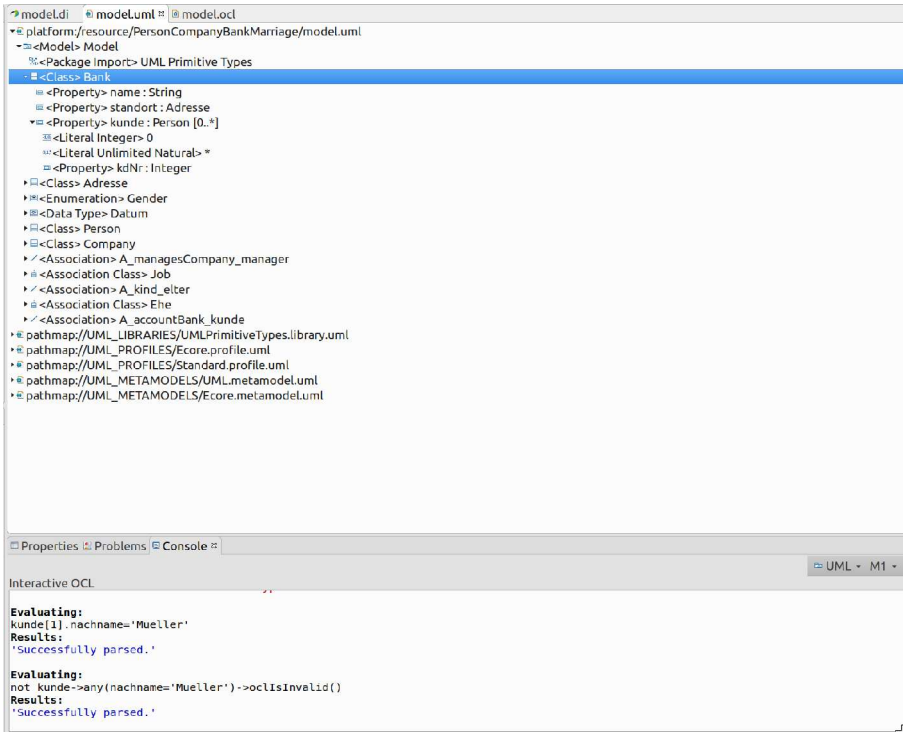


mit formalen „Constraints“ (des OCL-Editors)

model.di | model.ocl | model.uml

```
include 'model.uml'

context Model::Ehe
inv ortGueltig: ort.size()>0
inv datumVergangen: hochzeitsdatum < Datum::today()
inv mannVolljaehrig: hochzeitsdatum - mann.geburtsdatum >= 18
inv frauVolljaehrig: hochzeitsdatum - frau.geburtsdatum >= 18

context Model::Job
inv titelNotEmpty: titel->forAll(t| t.size()>0)
inv datumKonsistent: einstellungsDatum < Datum::today()
inv gehaltNichtnegativ: gehalt >=0.0

context Model::Datum::datum(t: Integer, m: Integer, j:Integer):Model::Datum
pre tagGueltig: 1 <= t and t <= 31
pre monatGueltig: 1 <= m and m <= 12
pre jahrGueltig: 1800 <= j and j <= 2300
post istNeu: result.oclIsNew()
post tagesfelderRichtigUebernommen: result.tag = t and result.monat = m and result.jahr = j

context Model::Bank
--inv vorzugskundeEins: kunde[1].nachname='Mueller'
inv muellerKundeExistiert: not kunde->any(nachname='Mueller')->oclIsInvalid()
```

oder in der „OCL-Console“ des UML-Editors, der qualifizierte Assoziationen schon beherrscht:



Erfolge formaler Methoden:

Fehler inTimSort-Standardsortieralgorithmus mit formalen Methoden aufgedeckt:
TimSort-Algorithmus mit inkorrekter Python `merge_collapse`-Function

Constraints in OOP-Modellen

Reflections on Software Engineering: Constraints
Object Constraint Language

```
context Model::Ehe
inv ortGueltig:          ort.size() > 0
inv datumVergangen:      hochzeitsdatum < Datum::today()
inv mannVolljaehrig:     hochzeitsdatum - mann.geburtsdatum >= 18
inv frauVolljaehrig:     hochzeitsdatum - frau.geburtsdatum >= 18

context Model::Datum::datum(t: Integer, m: Integer, j:Integer):
    Model::Datum
pre tagGueltig:    1 <= t and t <=31
pre monatGueltig:  1 <= m and m <= 12
pre jahrGueltig:   1800 <= j and j <= 2300
post istNeu:       result.oclIsNew()
post argsCorrectSet: result.tag = t and result.monat = m and
   reault.jahr = j
```

OCL-Semantik selbst in OCL spezifiziert

```
context Collection(T)::size() : Integer
-- The number of elements in the collection self.
post: result = self->iterate(elem; acc : Integer = 0 | acc + 1)

context Collection(T)::isEmpty() : Boolean
-- Is self the empty collection?
post: result = ( self->size() = 0 )

-- ...
```

(aus: OCL-Manual)

UML-Präzisierung durch OCL-Ausdrücke

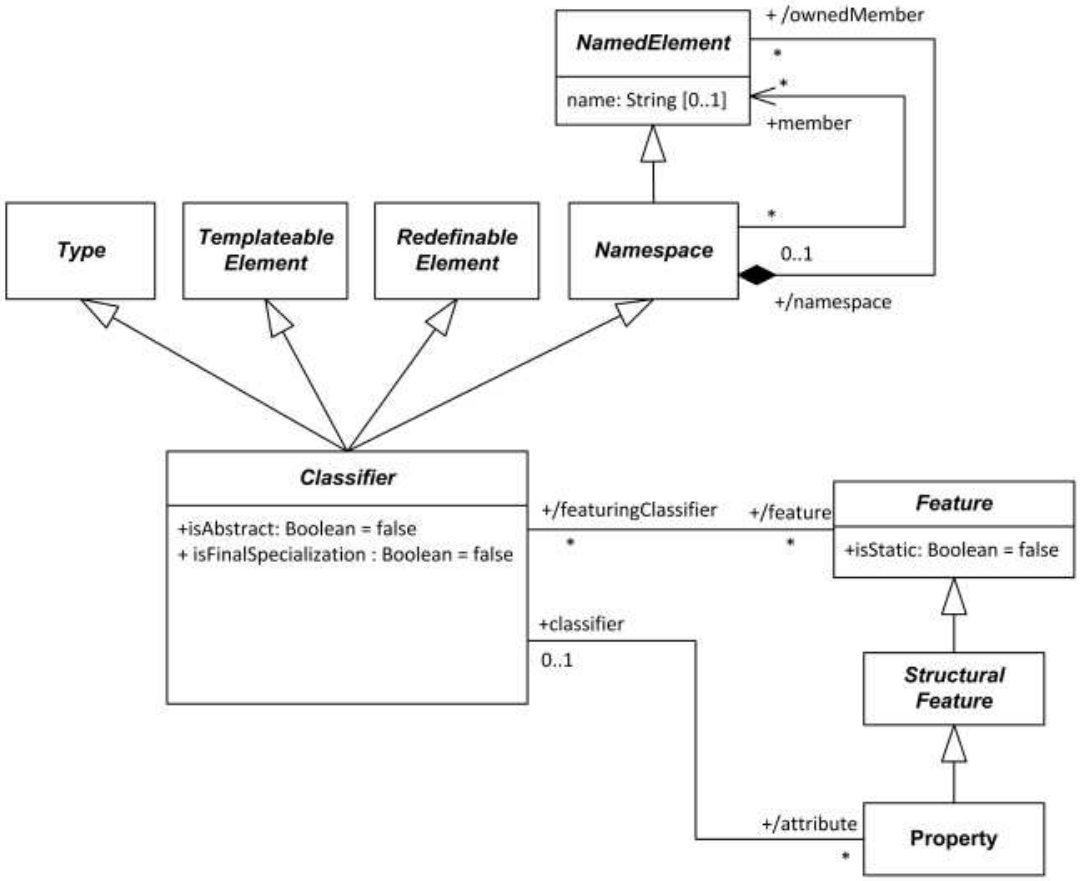
```
context Classifier
-- Classifier is an abstract metaclass which describes (classifies)
--    set of instances having common features.
conformsTo(other : Type) : Boolean {redefines Type::conformsTo()}
-- The query conformsTo() gives true for a Classifier that defines
--    a type that conforms to another. This is used,
--    for example, in the specification of signature conformance
--    for operations.
```

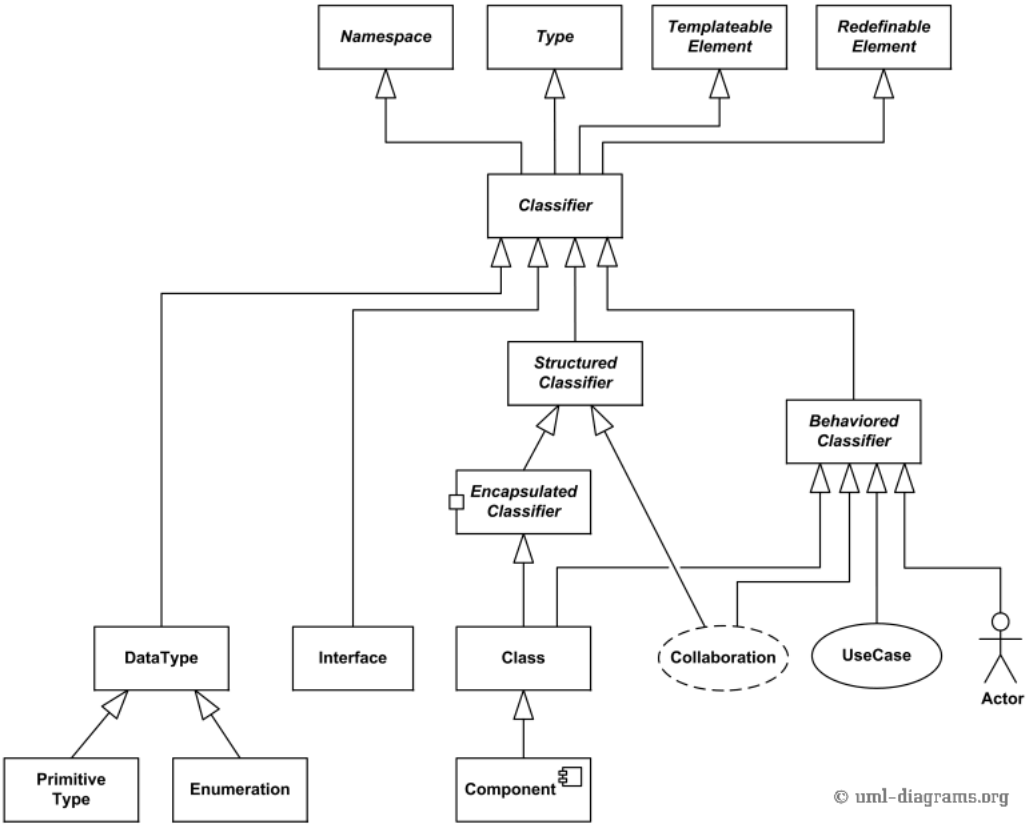
```
body: if other.oclIsKindOf(Classifier) then
    let otherClassifier : Classifier = other.oclAsType(Classifier) in
        self = otherClassifier or allParents()->includes(
            otherClassifier)
else
    false
endif
-- ...
```

(aus: 9.9.4 Classifier [Abstract Class] im UML 2.5-Handbuch)

Wikipedia: UML Classifier

https://www.uml-diagrams.org/classifier.html:




Namespace
Type
Templateable Element
Redefinable Element
Classifier
Structured Classifier
Behaviored Classifier
Encapsulated Classifier
DataType
Interface
Class
Collaboration
UseCase
Actor
Primitive Type
Enumeration
Component
© uml-diagrams.org

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

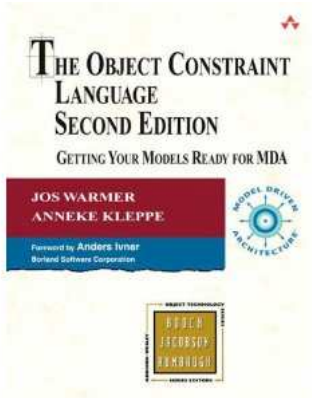
OMG
UML Superstructure
OMG Available Specification
Version 2.4.1
http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF

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

OMG
Object Constraint Language
OMG Available Specification
Version 2.4
http://www.omg.org/spec/OCL/2.4/PDF
https://de.slideshare.net/EdWillink/ocl25-plans

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

ISBN: 0321179366



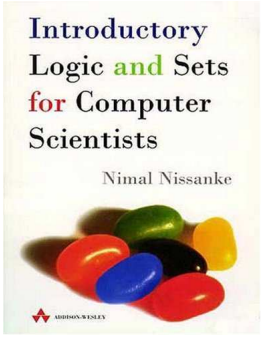
Tony Clark, Jos Warmer
Object Modeling with the OCL.
The Rationale behind the Object Constraint Language
http://www.amazon.de/Object-Modeling-OCL-Rationale-Constraint/dp/3540431691
ISBN: 3-540-43169-1

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

Eclipse für C/C++-Programmierer, 3. Auflage

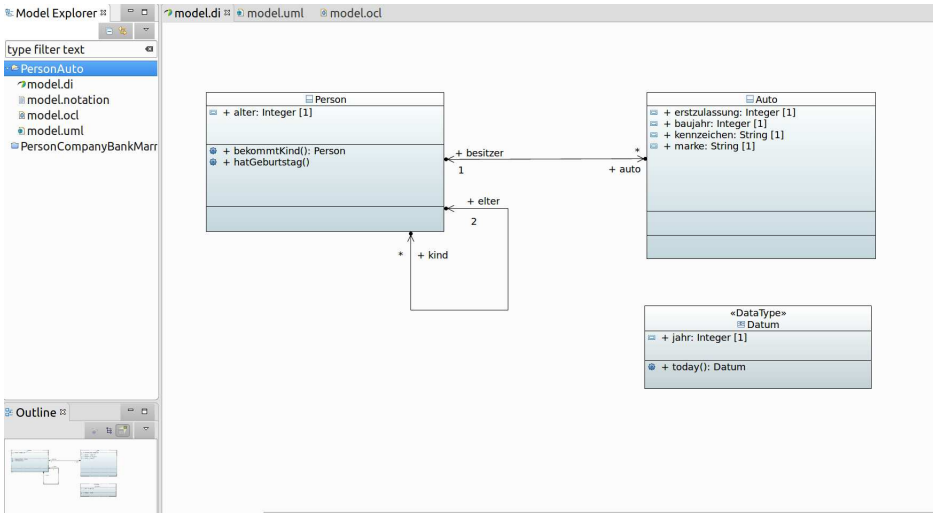
Quelloffenes Eclipse mit UML2.5/OCL2.4-Tools

Hilfsmittel (Tools) zur formalen Spezifikation von OOP-Modellen mit Hilfe von OCL2:
http://www.eclipse.org/papyrus/

und: http://wiki.eclipse.org/MDT/Papyrus

(Verfügbar (vorinstalliert) auf dem Fachgruppen-Ausbildungsclustern (l101, ...) als `eclipse-papyruso3`. Hinweis zur Installation auf dem eigenen Linux-Notebook: Siehe Anhang B, Seite xxv. Benutzungsanleitung: Erstellen eines neuen Papyrusprojekts (mit OCL-Constraints): Siehe Anhang A, Seite ix.)

OOP-Modellierung (Modells des OCL Wikipedia-Artikels):



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:

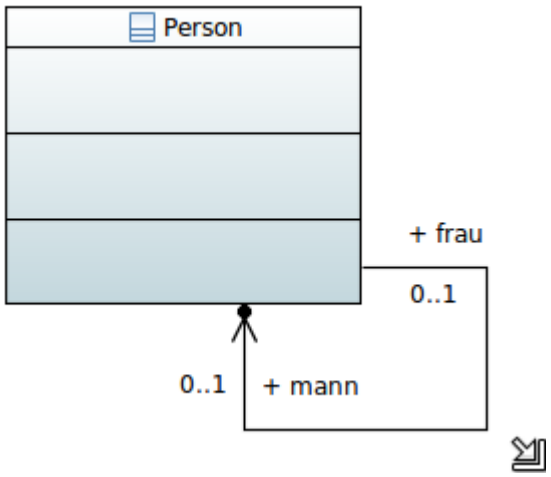
import 'model.uml'

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

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

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

...
```

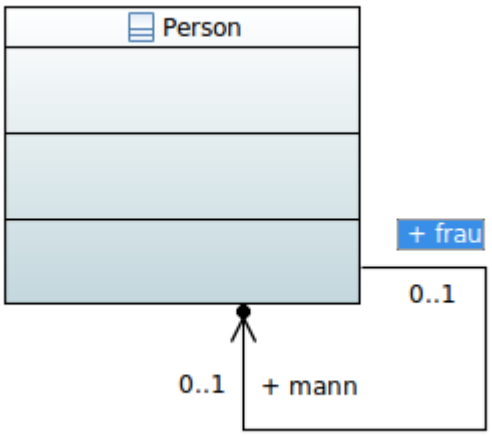
(nach Object Constraint Language (OCL))

Aufgabe:
Welche Interpretationsmöglichkeiten gibt es für die natürlichsprachigen Detaillierungen? Sind die Constraints einschränkend genug gewählt?

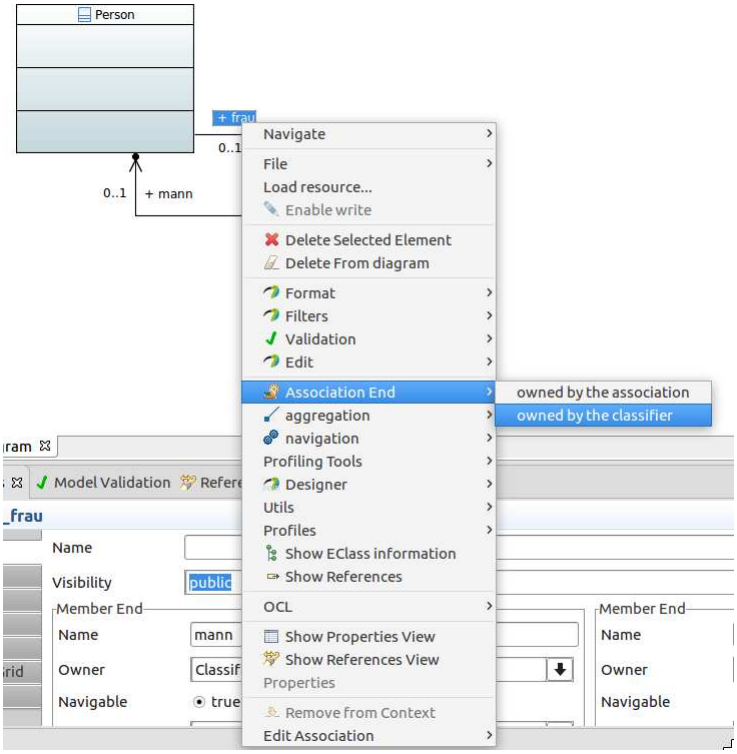
Assoziationsenden (Rollennamen) mit Dot-Eigenschaft (vgl. Ende von Abschnitt 1.1.4) ausstatten:



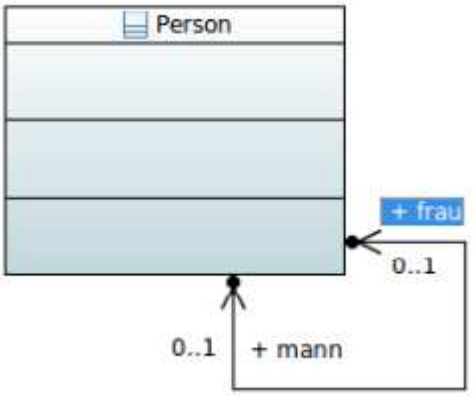
Anklicken des Rollennamens, der mit der Dot-Eigenschaft ausgestattet werden soll:



Das Kontextmenü (rechter Mausknopf) für diesen Rollennamen erscheinen lassen

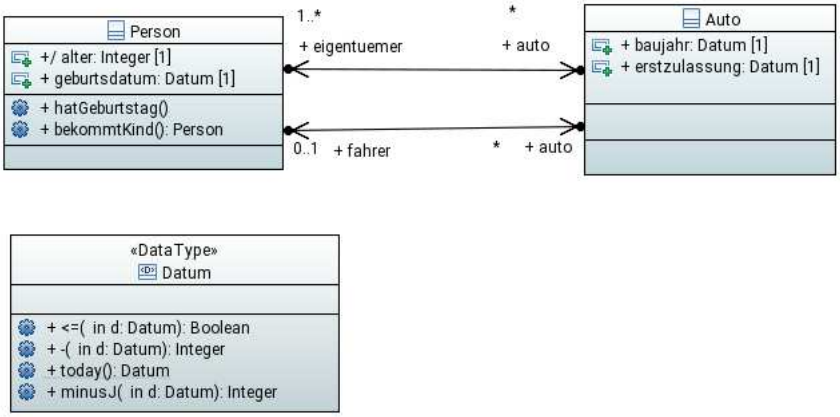


und `AssociationEnd, owned by the classifier` auswählen:



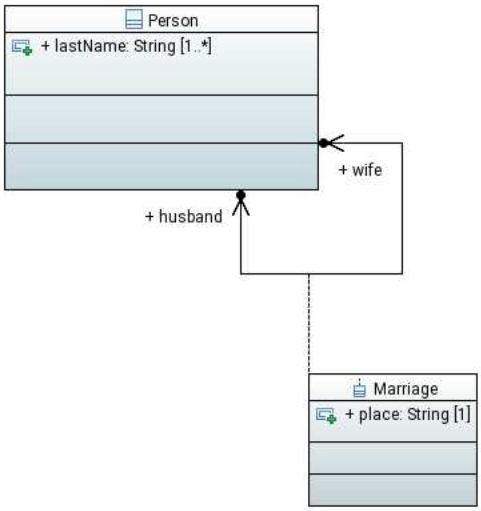
Alternativ kann man im UML-Diagramm die Assoziation anklicken und dann im Properties-Fenster für das betroffene `Member End` (z.B.) `frau` den `Owner` von `Association` auf `Classifier` ändern.

Autos mit zwei verschiedenen Arten assoziierter Personen, `Datum` mit zwei verschiedenen Abständen zweier Exemplare:




```
1  import 'model.uml'
2
3  context Model::Person
4  inv alterNonnegative: self.alter >= 0
5
6  context Model::Auto
7  inv zulassungGueltig: baujahr <= erstzulassung
8
9  context Model::Auto
10 inv fahrerAltGenug: fahrer->notEmpty() implies  fahrer.alter >= 18
11
12 context Model::Person
13 inv alterDerived: (Datum::today() - geburtsdatum) / 365 = alter
14 inv alterDerived2: Datum::today().minusJ(geburtsdatum) = alter
15
```

Reflexive Assoziationsklassen:

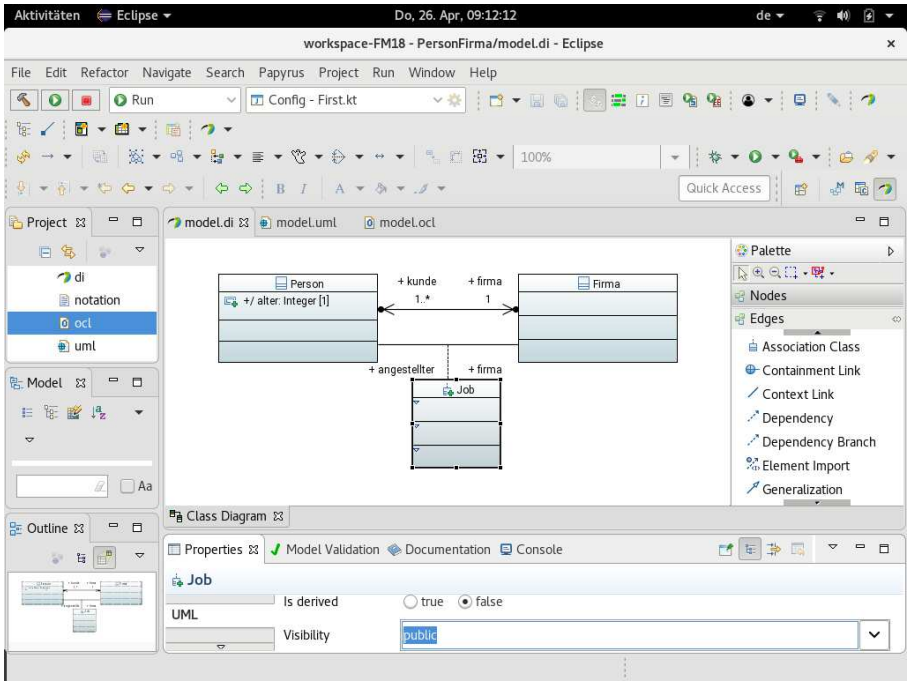


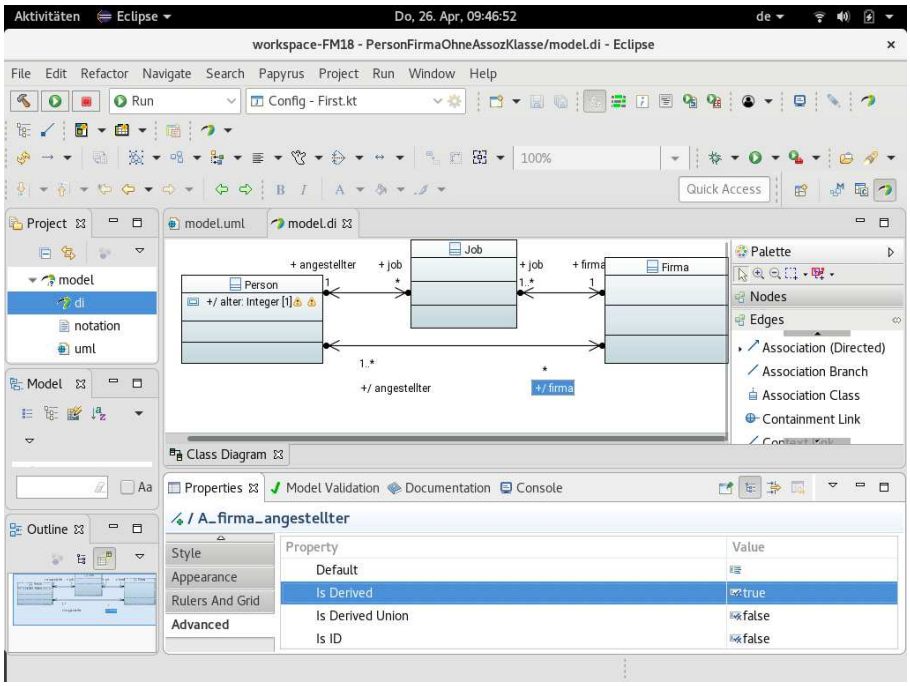
Class Diagram

Properties | Model Validation | Console | UML | M1

Interactive OCL

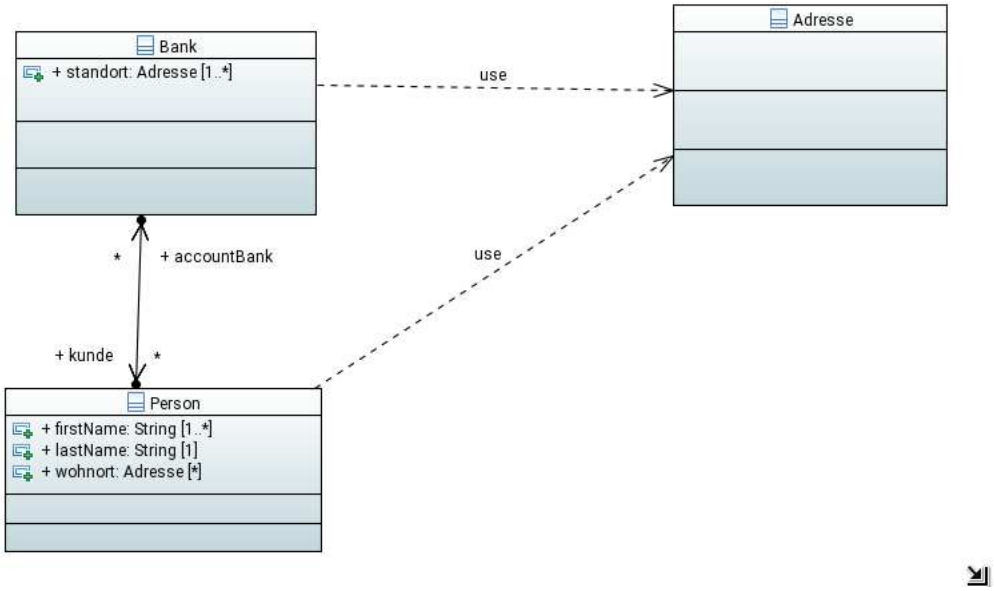
```
Evaluating:
self.marriage->notEmpty()
Results:
Qualifier needed to select navigation direction in reflexive association class: (self.marriage)

Evaluating:
self.marriage[wife]->notEmpty()
Results:
'Successfully parsed.'
```


Aktivitäten
Eclipse
Do, 26. Apr, 09:12:12
de
workspace-FM18 - PersonFirma/model.di - Eclipse
File Edit Refactor Navigate Search Papyrus Project Run Window Help
Run
Config - First.kt
100%
Quick Access
Project
di
notation
ocl
uml
Model
Outline
model.di
model.uml
model.ocl
Person
+/ alter: Integer [1]
+ kunde
1..*
+ firma
1
Firma
+ angestellter
+ firma
Job
Palette
Nodes
Edges
Association Class
Containment Link
Context Link
Dependency
Dependency Branch
Element Import
Generalization
Class Diagram
Properties
Model Validation
Documentation
Console
Job
Is derived
true
false
UML
Visibility
public


Aktivitäten
Eclipse
Do, 26. Apr, 09:46:52
de
workspace-FM18 - PersonFirmaOhneAssozKlasse/model.di - Eclipse
File Edit Refactor Navigate Search Papyrus Project Run Window Help
Run
Config - First.kt
100%
Quick Access
Project
model
di
notation
uml
Model
Outline
model.uml
model.di
Person
+/ alter: Integer [1]
+ angestellter
1
+ job
*
Job
+ job
1..*
+ firma
1
Firma
1..*
+/ angestellter
*
+/ firma
Palette
Nodes
Edges
Association (Directed)
Association Branch
Association Class
Containment Link
Class Diagram
Properties
Model Validation
Documentation
Console
/ A_firma_angestellter
Style
Appearance
Rulers And Grid
Advanced
Property
Value
Default
Is Derived
true
Is Derived Union
false
Is ID
false

Workaround für qualifizierte Assoziationen:

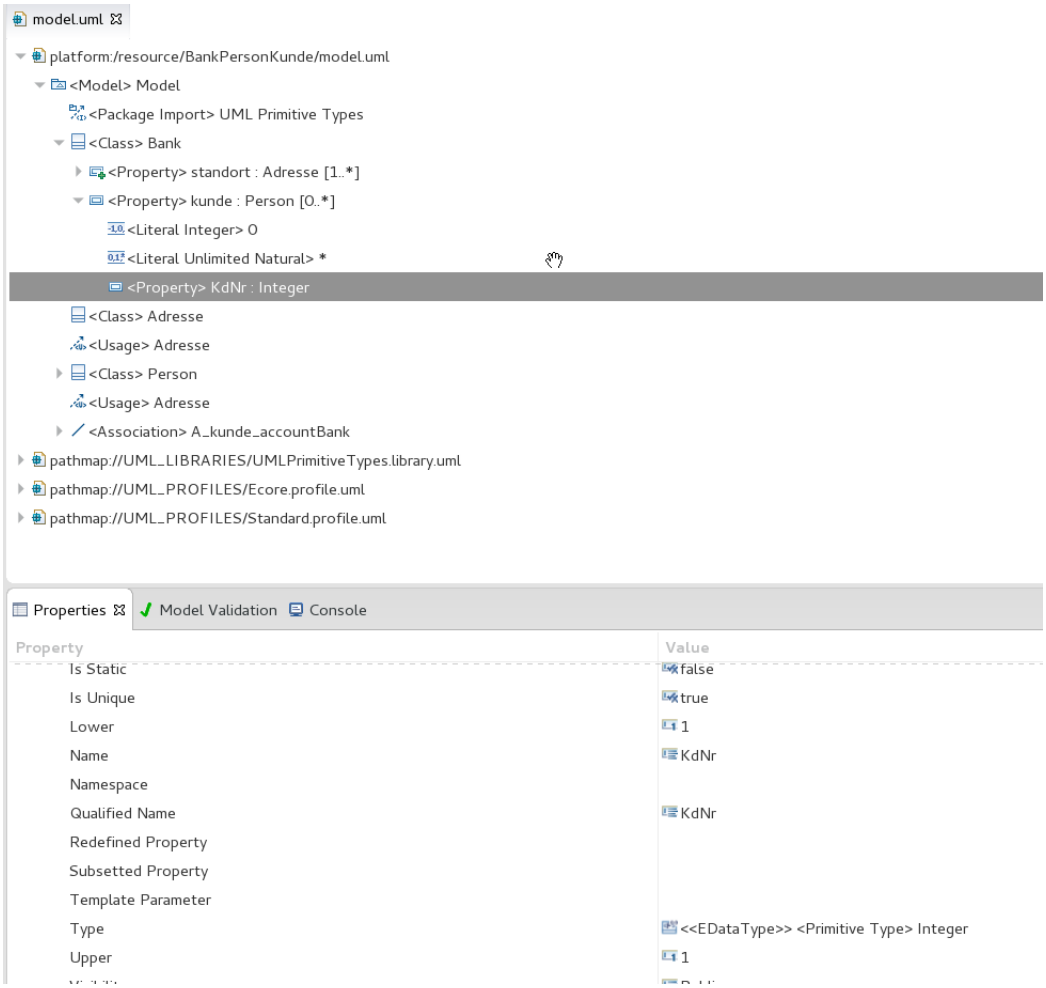
Der Rollenname `kunde` ist im Kontext `Bank` noch nicht mit einem `Integer`-Ausdruck qualifizierbar:



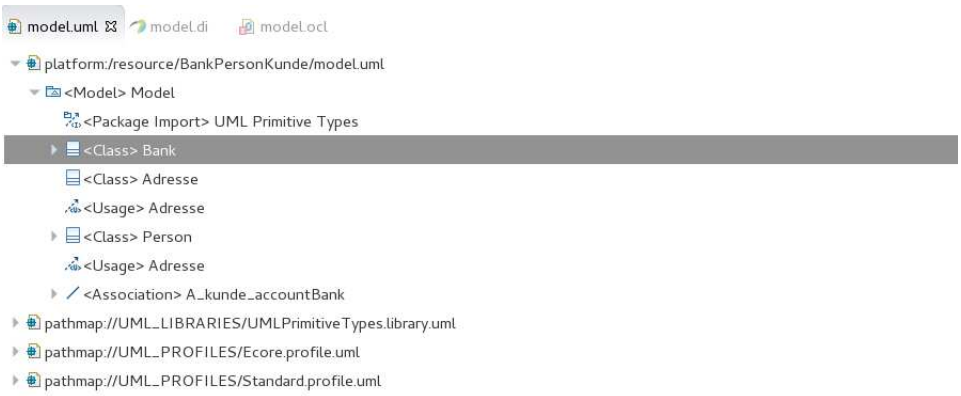
model.di model.uml model.ocl

```
1  import 'model.uml'
2
3  context Model::Bank
4  inv standortNichtLeer: standort->size() > 0
5  inv referenzKundeOK: kunde[1001].lastName = 'Meyer'
6
```

Nachdem das Attribut (Poperty) `kunde` der Klasse `Bank` die Kind-Qualifier-Property mittels der unten beschriebenen Aktionen erhalten hat (`KdNr: Integer`),



kann man ihn auch mittels einer Integer-Qualifizierung nutzen:



Aktion für `KdNr: Integer`:

Im UML-Editor rechten Mausklick auf der Property `kunde` der Klasse `Bank`, dann `NewChild, Qualifier, Property` anklicken und die neue Kind-Property im Property-Fenster (unten in Eclipse) mit dem Namen `KdNr` (für KundenNummer) und dem Typ `PrimitiveTypes::Integer` ausstatten.

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

http://foldoc.org/

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)

Aufgabe:
Suchen Sie im UML 2.5-Handbuch nach OCL-Präzisierungen der Erläuterungen (z.B. auf Seite 226, ...).

Formale Spezifikationssprachen benuzten anfangs mathematische Formelschreibweise (Latex-Stil):

$Queue = Qelem^*$
$q_0 = [\ ]$

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

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

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

VDM, VDM++

Heute geht man aber immer mehr zu reiner ASCII (oder Unicode)-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}}
```

Wir benuten die Object Constraint Language als integralen Bestandteil von UML: OCL-Handbuch:

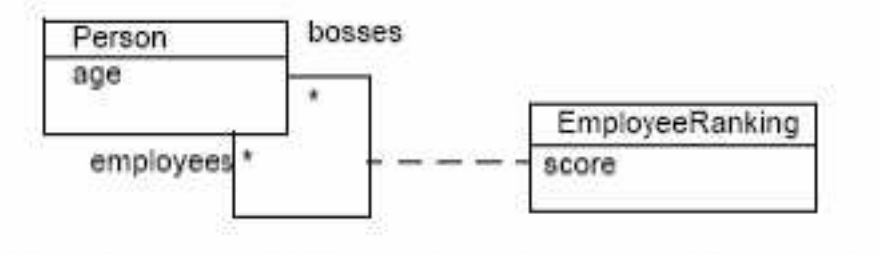


Figure 7.3 - Navigating recursive association classes

```
context Person inv bossesRankingPositive:
self.EmployeeRanking[bosses]->sum() > 0
⋮
... set of EmployeeRankings belonging to the collection of bosses
```

Softwarefehler

Beispiele

Nests Neujahrs-Bug lässt Kunden bibbern
Software Design as Trial and Error?

TomTom Navigatoren - Bug
Car computer directs couple into river
Panne beim Online-Banking der Deutschen Bank
Software-Fehler: Mars-Rover Curiosity im Sicherheitsmodus
Boeing Dreamliner 787 should be reboot every 21 days
Facebook- und Messenger-App saugten Akkus leer

Forum on Risks to the Public in Computers and Related Systems
⋮

Softwarekatastrophen als Rechtfertigung für den Aufwand formaler Spezifikation

- Top Ten Most Infamous Software Bugs Of All Time

- **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.

Risks Digest 24.88, suche nach ”German Telephone-Network Partial Outage”

Aktueller (19.04.2015 15:11):
Mindestens 100.000 IP-Telefone von Telekomkunden stundenlang stumm

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.

- 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.

- Haswell ohne transactional Memory

- Errata prompts Intel to disable TSX in Haswell, early Broadwell CPUs

Zur Jahrtausendwende aufgetretene Bugs:

2.1.00: Der Y2k-Direktor der United Nations ruft am 2.1.00 einen weltweiten Alarm vor der Benutzung von Gambro Dialysegeräten aus.
3.1.00: Schwere Störungen im Flugverkehr in Chicago und an der gesamten US- Ostküste.
3.1.00: Massive Störungen im Flugverkehr von Neuseeland durch Rechnerprobleme
3.1.00: Kreditkartensysteme bei amerikanischen Tankstellen ausgefallen.
4.1.00: FBI kann die Datenbank für Waffenlizenzen wegen Y2k-Fehler nicht mehr nutzen.
4.1.00: Radioaktivitätsüberwachung in Atomwaffenfabrik in Tennessee ausgefallen.
4.1.00: Stromausfall in Los Angeles
4.1.00: Justizcomputer verlängert Haftstrafen in Italien um 100 Jahre
4.1.00: 28 Kernkraftwerke melden Y2k-Probleme
4.1.00: Führerscheine in Indiana und New Mexico werden falsch ausgestellt.
4.1.00: Kunde einer US-Videothek soll 177.000 DM Strafgebühr bezahlen.
4.1.00: In Schweden und Deutschland "Zahlensalat" auf den Konten von Online-Kunden
5.1.00: Viertgrößter Autoversicherer der USA hat Y2k-Problem in der Policenverwaltung.
5.1.00: Pentagon hat zwei Stunden lang keine Kontrolle über Spionagesatelliten
5.1.00: Verwaltungscomputer der Feuerwehr von Washington DC mit Y2k-Fehler
6.1.00: Die amerikanische Datenbank für Chemieunfälle ist nicht y2k-fähig.
6.1.00: Pentagon stellt 230 falsche Schecks aus.
6.1.00: Homebanking-Programm BankUp von Macintosh hat Y2k-Problem
6.1.00: 40.000 Händler haben Y2k-Probleme mit Kreditkarten.
7.1.00: In Arkansas sind die Verwaltungscomputer von 22 Landkreisen betroffen.
7.1.00: Unbekannte Anzahl von 112.000 Cash Cards der US-Post fehlerhaft
7.1.00: Chicago-Bank kann in 8 US-Staaten keine Medicare-Überweisungen tätigen.
7.1.00: Utah Food Bank in Salt Lake City Total-Crash für einen Tag
7.1.00: Die Personalverwaltung der US-Notfallmanagementbehörde ausgefallen.
8.1.00: Chevy Chase Bank Window-Versionen von Quicken 99 und 2000 fehlerhaft
8.1.00: MCS Spectrum Buchhaltungssoftware nicht y2k-fest
8.1.00: First Union Bank bezahlt Arbeiter doppelt: insgesamt 2,3 Mio. US-$
8.1.00: Scheckverkehr in Oregon gestört
10.1.00: Quicken Tool der Financial Times wegen Y2k zusammengebrochen
10.1.00: Wasserversorgungssystem in amerikanischer Stadt wegen Y2k ausgefallen
11.1.00: Datenbanken in 157 staatliche Alkoholläden wegen Y2k gestört
11.1.00: Medizinischer Notfallservice der Feuerwehr im Staate Washington ausgefallen
12.1.00: Datenbank der Wahlbezirke in Maryland gestört
12.1.00: Satellitenausfall des Pentagon ernster als zuerst gemeldet
12.1.00: 50.000 Zeugenaussagen in Topeka durch Y2k-Fehler zerstört
12.1.00: Y2k-Fehler in Auszahlungssoftware der Deutschen Oper in Berlin
13.1.00: LOTUS warnt vor Anwendung der DOMINO-Software wegen Y2k

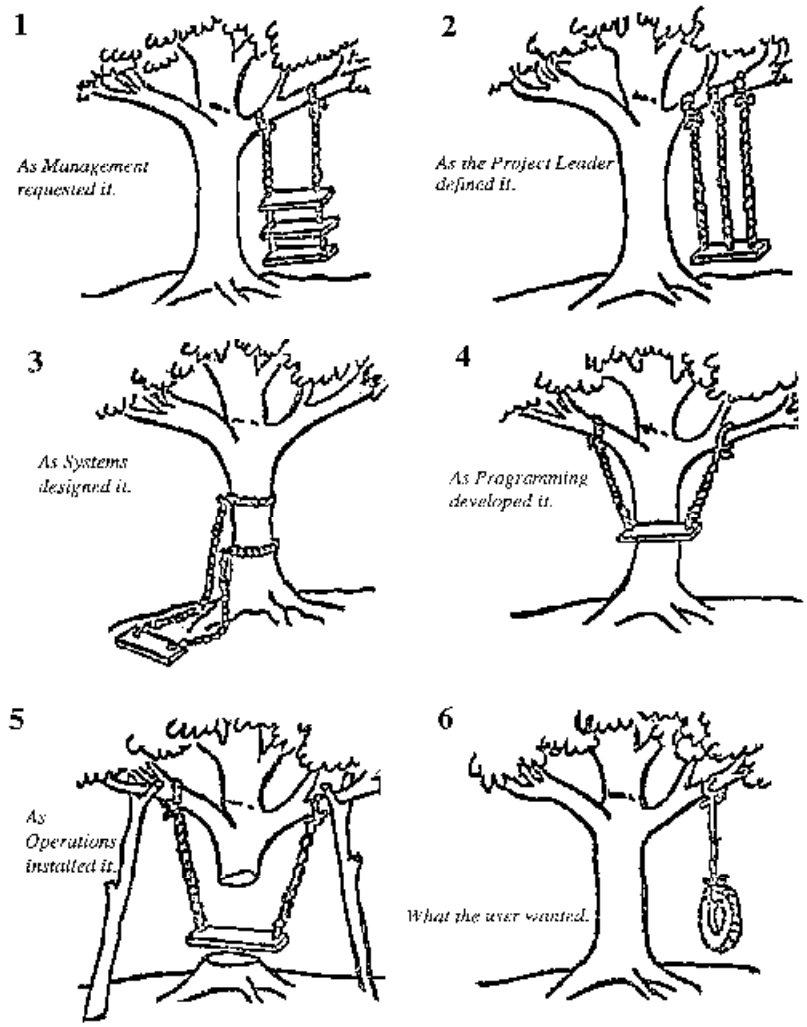
(aus: Das weltweite Y2k-Überwachungssystem meldet)

Weitere interessante Fundstellen:

- 20 Famous Software Disasters
- Kleine BUGs, große GAUs
- Top 25 Most Dangerous Programming Errors

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

Impedanzverlust bei der Softwareentwicklung:



UML und OCL

Klassen-Attribute mit Vielfachheiten, ... in UML 2.5

UML-Klassendiagramm

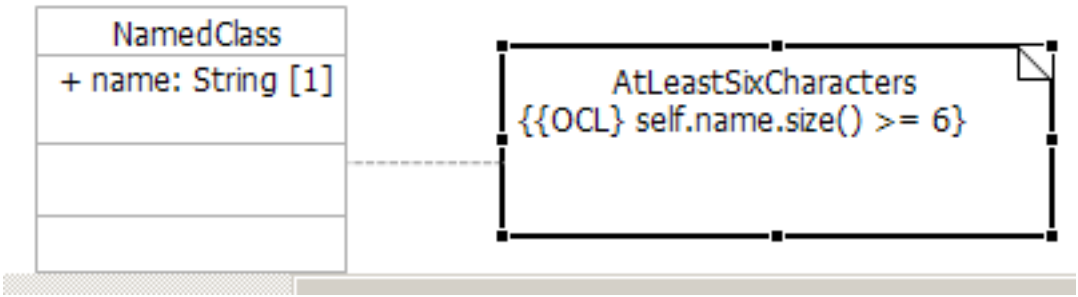
UML Constraint

UML Datatype

Eclipse Shortcuts
Eclipse Shortcuts - Tutorial
eclipse shortcuts

OCL 2.4 – Object Constrint Language

OCL in UML:



(eclipse: Nodes: Constraint, Specification +: OpaqueExpression, Language +: OCL, ...)

Ein erster (verbesserungsbedürftiger) Vertrag für den Konstruktor von `Datum`:

```
context Model::Datum::datum( t : Integer,
                             m : Integer,
                             j : Integer ) : Model::Datum
pre tagGueltig: 1 <= t and t <= 31
pre monatGueltig: 1 <= m and m <= 12
pre jahrGueltig: 1800 <= j and j <= 2500
post neuErzeugt: result.oclIsNew()
post tagRichtigUebernommen: result.tag = t
post monatrichtigUebernommen: result.monat = m
post jahrRichtigUebernommen: result.jahr = j
```

> UML doesn't model constructors as such. Your best approximation to a Java constructor is a static operation that has the same name as the class and the class as its return type. The semantics of constructor chaining etc. are language-specific and outside of the scope of UML.

(C++/Java-Constructors)

oclIsNew() : Boolean

Can only be used in a postcondition. Evaluates to true if the *self* is created during performing the operation (for instance, it didn't exist at precondition time).

post: self@pre.oclIsUndefined()

oclIsUndefined() : Boolean

Evaluates to true if the *self* is equal to *invalid* or equal to null.

post: result = self.isTypeOf(OclVoid) or self.isTypeOf(OclInvalid)

(aus 11.3.1 des OCL-Manuals)

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

What is OCL?

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

Runqiu Son: Einführung in die Object-Contraint-Language OCL, 2003

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) )
```

...

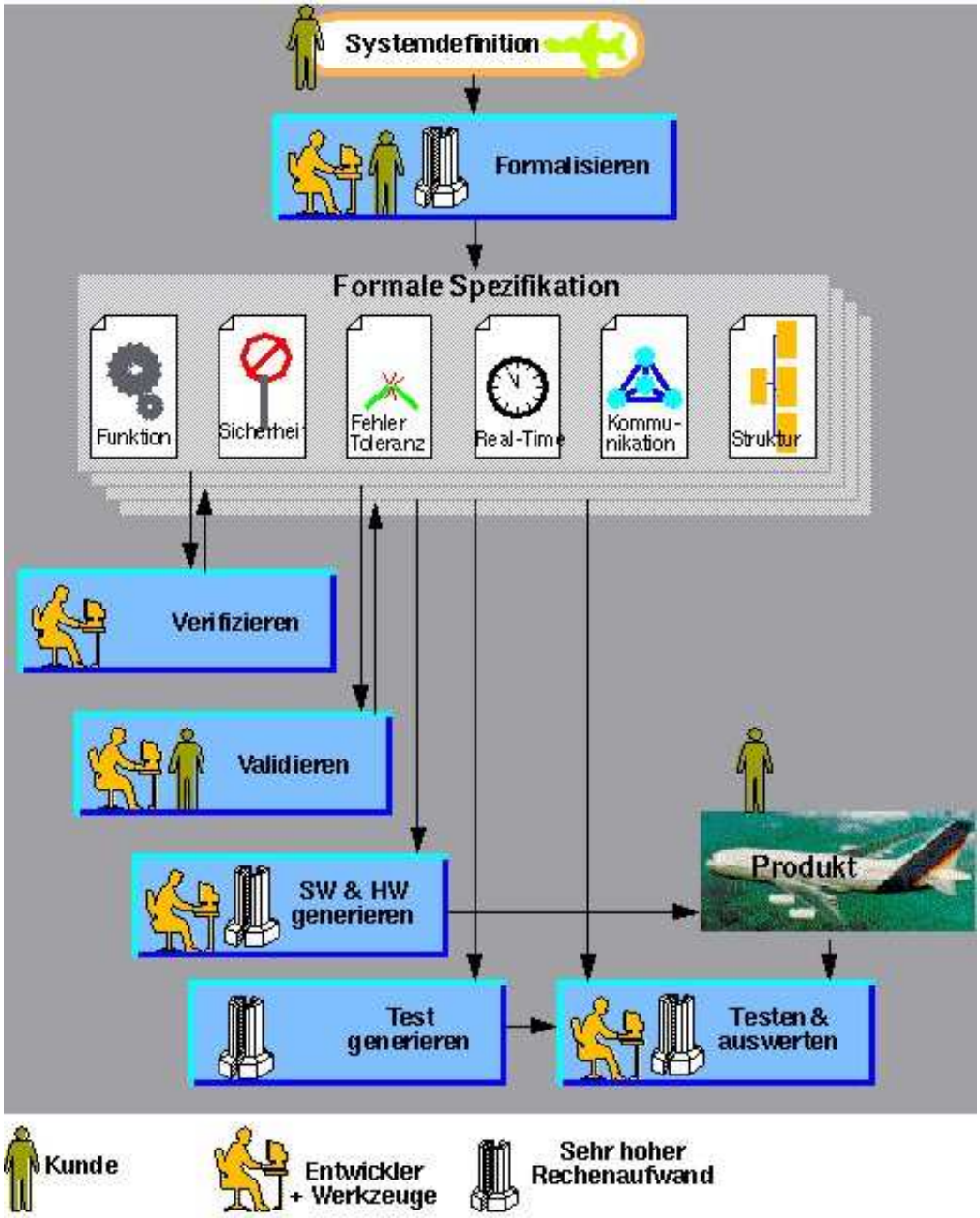
(Vergleiche Seite 157... der OCL-Spezifikation.)

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
```

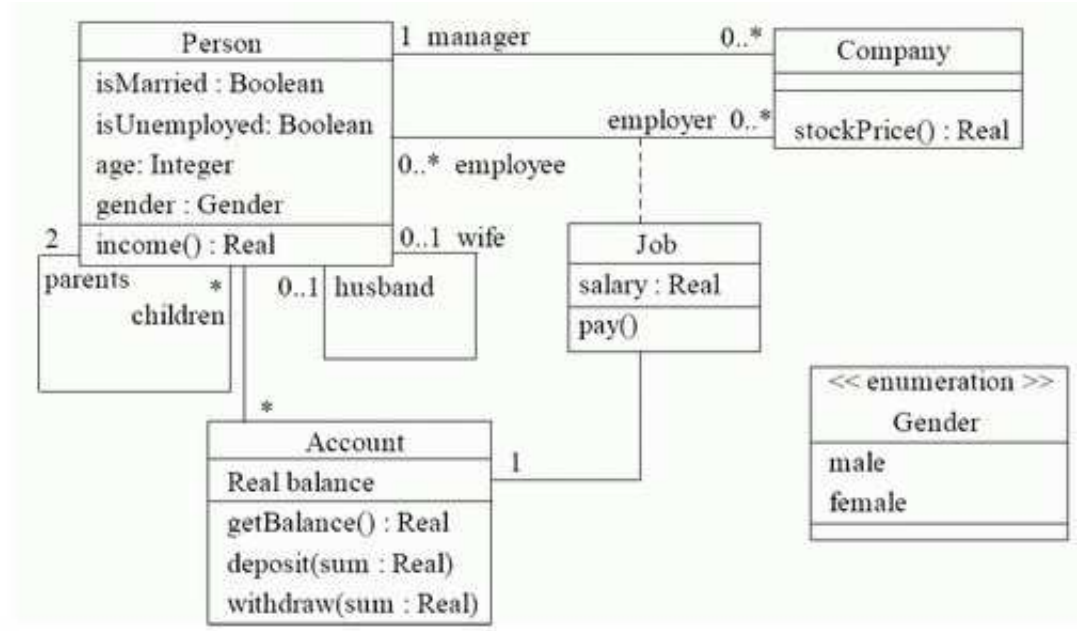
Vor- und Nachbedingungen erlauben die eindeutige Verantwortlichkeitszuordnung: Im Fehlerfall Vorbedingung verletzt (Aufrufender verantwortlich), Nachbedingung bei eingehaltener Vorbedingung verletzt (Software-Produzent verantwortlich).

- „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)



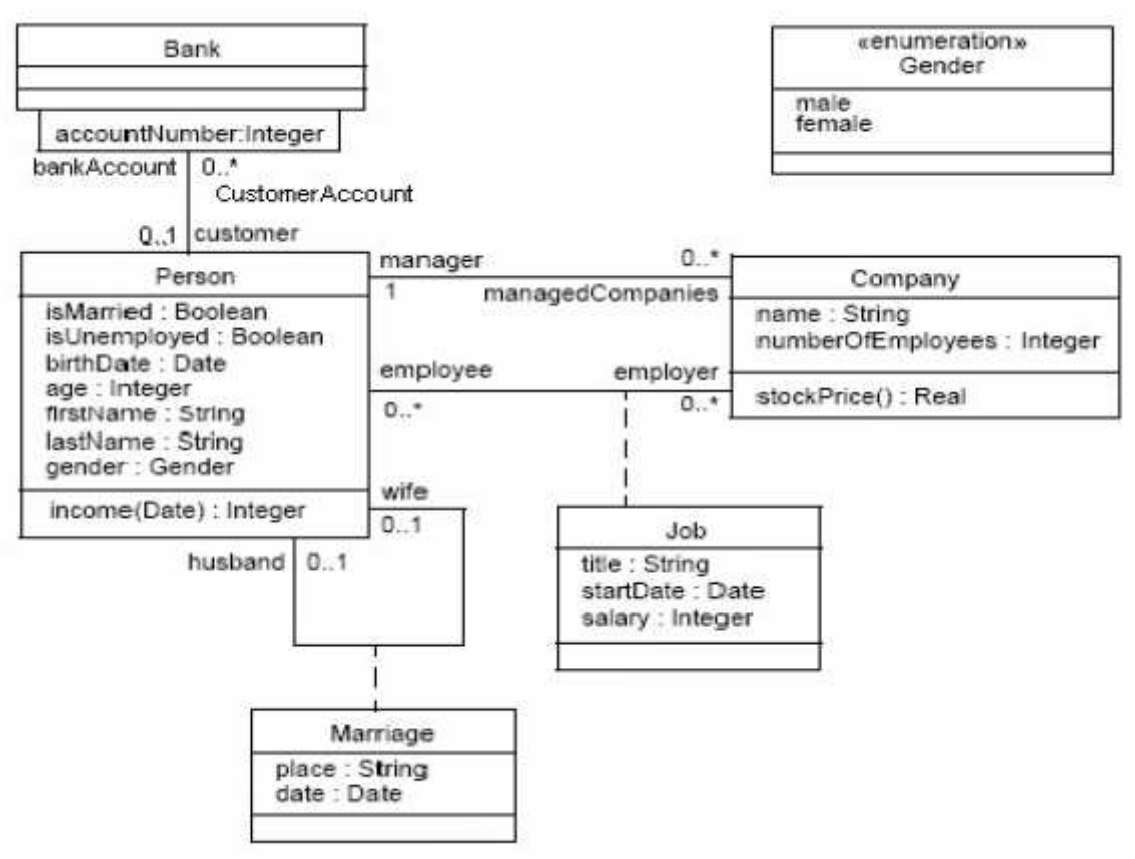
Beispiele:

- PersonAuto, siehe OCL-Beispiele (Wikipedia) und einleitende Beispiele der Materialsammlung sowie Übungsblatt 1.

- 

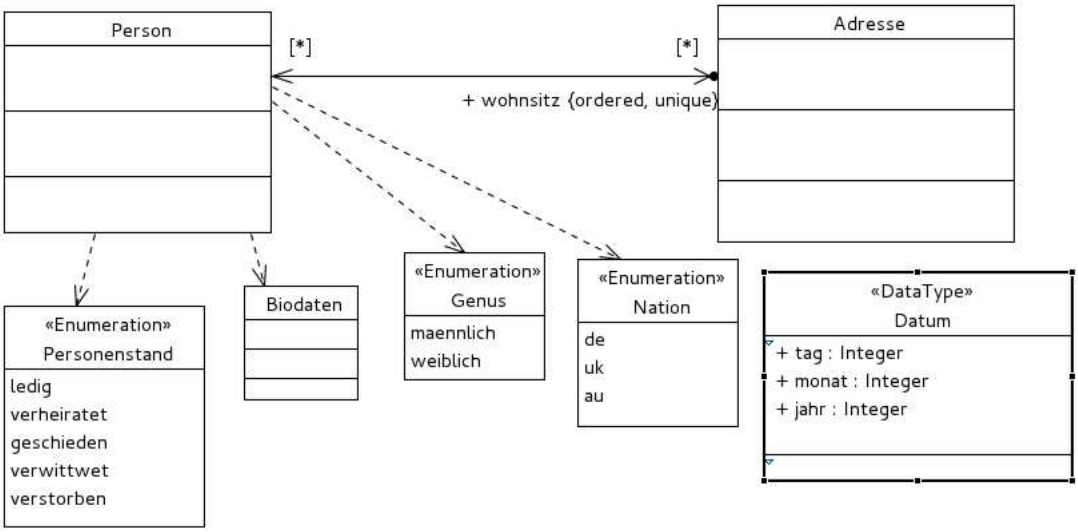
(ähnlich zu Seite 7-4 in Object Constraint Language-Specification)

- Siehe die Beispiele in der Object Constraint Language-Spezifikation (Seite 7 oben, 19, 21, ...).



und:

- Personenstandsdaten (im zweiten Teil dieser Materialsammlung):



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

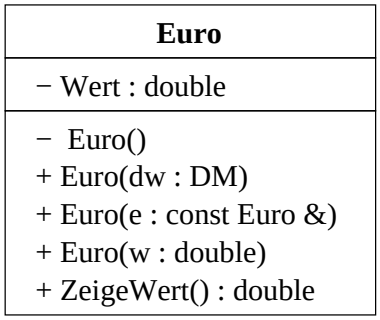


Abbildung 0.1.: Klasse Euro

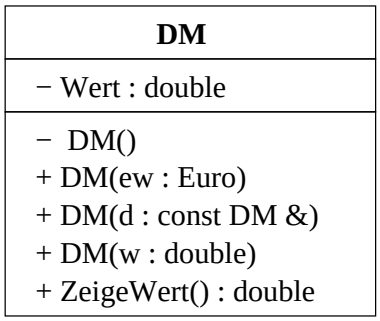


Abbildung 0.2.: Klasse DM

als instantiierbare Kinder einer abstrakten Klasse `Waehrung`.

Eine Anwendung Sparbuch:

Original mit anonymer Geldeinheit (double)

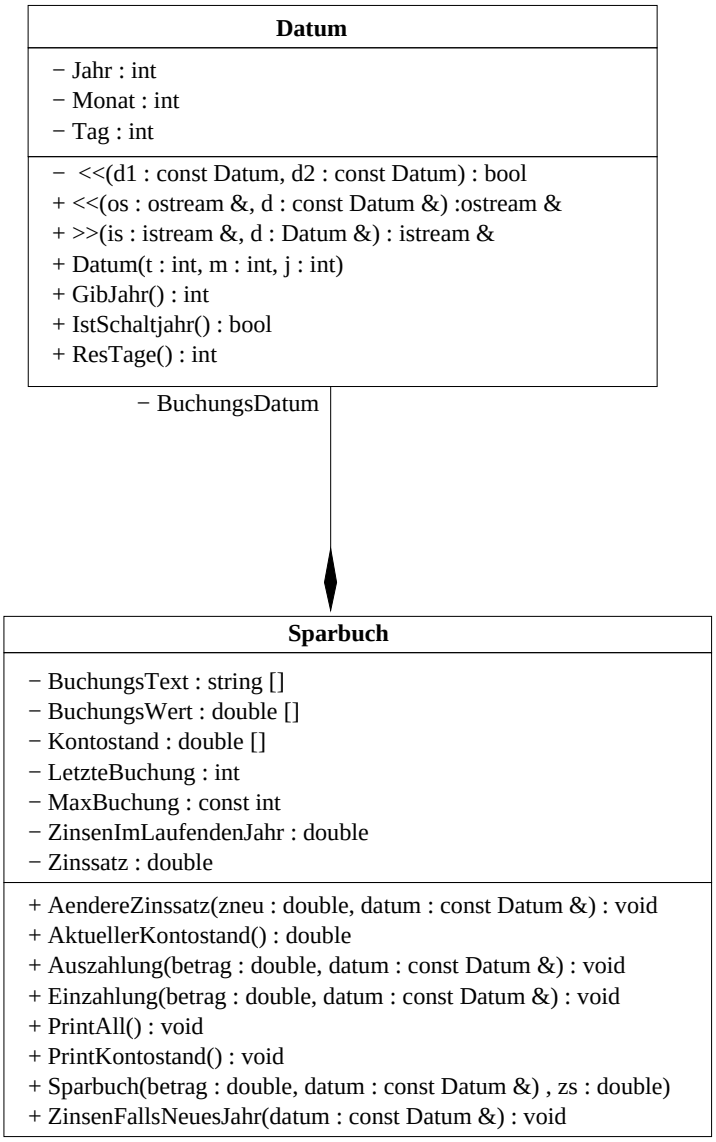


Abbildung 0.3.: Klassen Datum und Sparbuch

und besser: Klasse Sparbuch mit Klasse DM und Klasse Euro:

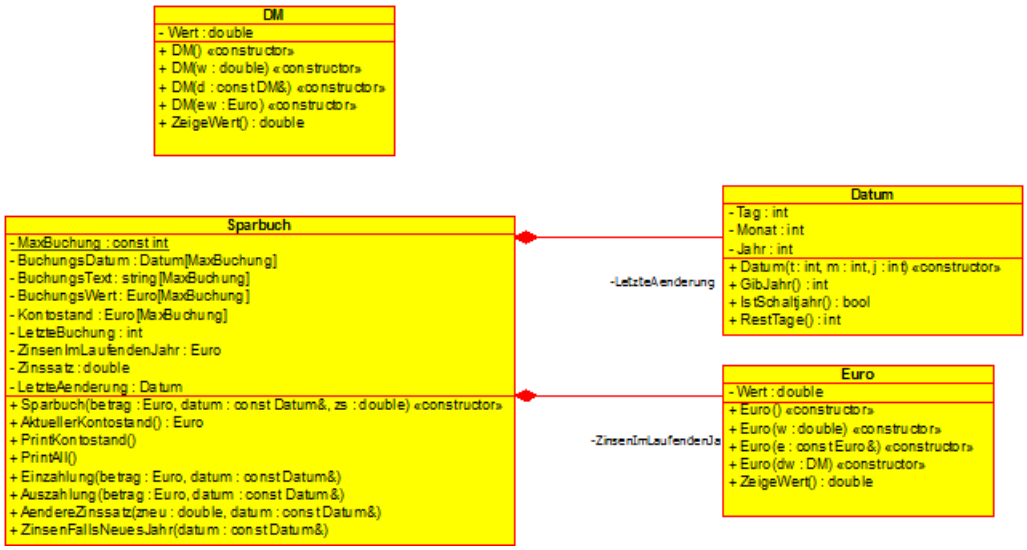
```
//////////////////////////////////////////
// Datei:    Sparbuch.cc
// Version:  1.2 mit DM/Euro
// Zweck:    Implementierung eines Sparbuchs
//           Loesung zu Uebungsaufgabe ...
// Autor:    Holger Arndt / HJB
// Datum:    25.01.2001 / 18.04.2006
//////////////////////////////////////////

#include <iostream>
//#include <strstream>
#include <sstream>
#include <iomanip>
#include <string>

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; };
  friend Euro operator+(Euro a, Euro b);
  friend Euro operator-(Euro a, Euro b);
  friend Euro operator*(Euro a, double d);
  friend Euro operator/(Euro a, double d);
  friend bool operator<(Euro a, Euro b);
  friend ostream&  operator<<(ostream& os, const Euro& e);
};

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 operator+(Euro a, Euro b)
{
    return Euro(a.Wert + b.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;
}

// einfache Datumsklasse ohne Ueberpruefung der Gueltigkeit
    von Daten
// ...


class Sparbuch
{
private:
  static const int MaxBuchung = 100;
  Datum BuchungsDatum[MaxBuchung];
  string BuchungsText[MaxBuchung];                    //
     Einzahlung, Zinsen o.ae.
  Euro BuchungsWert[MaxBuchung];                      //
     negativ bei Auszahlung
  Euro Kontostand[MaxBuchung];                            //
      Kontostand in Euro
  int LetzteBuchung;              // Nummer der letzten
     Buchung in [0,MaxBuchung-1]
```

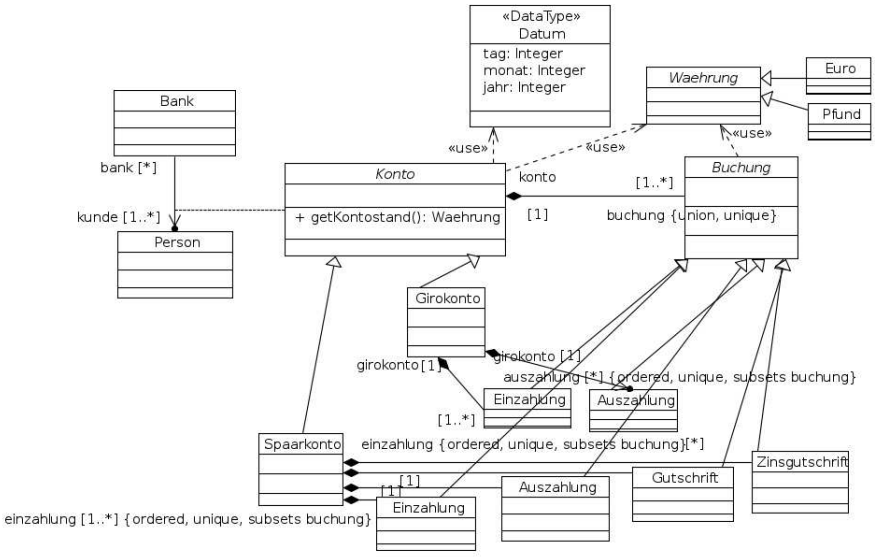
```
  Euro ZinsenImLaufendenJahr;     // Zinsen fuer aktuellen
     Stand bis Jahresende
  double Zinssatz;
                                                    // in
     Prozent
  Datum LetzteAenderung;                          // Buchung
     oder Zinssatzaenderung
public:
  Sparbuch(Euro betrag, const Datum &datum, double zs);
  Euro AktuellerKontostand() const { return Kontostand[
     LetzteBuchung]; };
  void PrintKontostand() const
  { cout << "Aktueller Kontostand: " << fixed <<
     setprecision(2)
         << AktuellerKontostand() << endl; };
  void PrintAll() const;
  void Einzahlung(Euro betrag, const Datum &datum);
  void Auszahlung(Euro betrag, const Datum &datum);
  void AendereZinssatz(double zneu, const Datum &datum);
  void ZinsenFallsNeuesJahr(const Datum &datum);
};

Sparbuch::Sparbuch(Euro betrag, const Datum &datum, double
   zs)
{
  // legt am Datum datum ein neues Sparbuch an mit
     Startkapital betrag Euro und
  // Zinssatz zs
  if (betrag < Euro(0))
    {
      cerr << "FEHLER: negativer Betrag bei Eroeffnung des
         Sparbuchs" << endl;
      throw 1;
    }
  Zinssatz = zs;
  LetzteBuchung = 0;
  BuchungsDatum[0] = datum;
  BuchungsText[0] = "Einzahlung";
  BuchungsWert[0] = betrag;
  Kontostand[0] = betrag;
                                        // Rundung wo und wie
                                           Haeufig?
  ZinsenImLaufendenJahr = betrag * Zinssatz * datum.
     RestTage() / 36000.0;
```

```
    LetzteAenderung = datum;
}

// ...
    return 0;
}
```

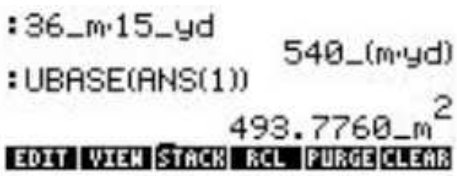
Klassendiagramm der ersten Verbesserung:



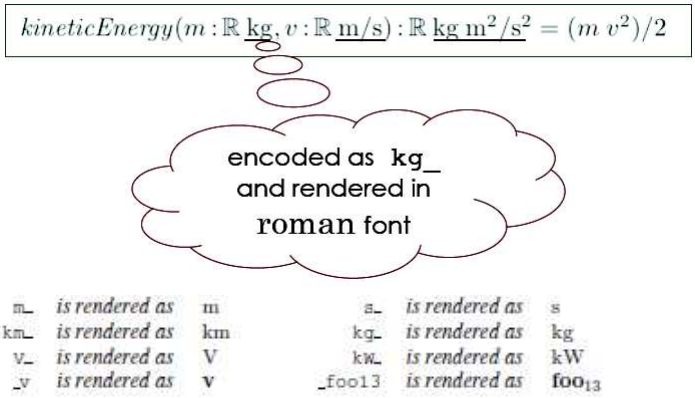
Noch besser wäre ein vollständiges objektorientiertes Design:

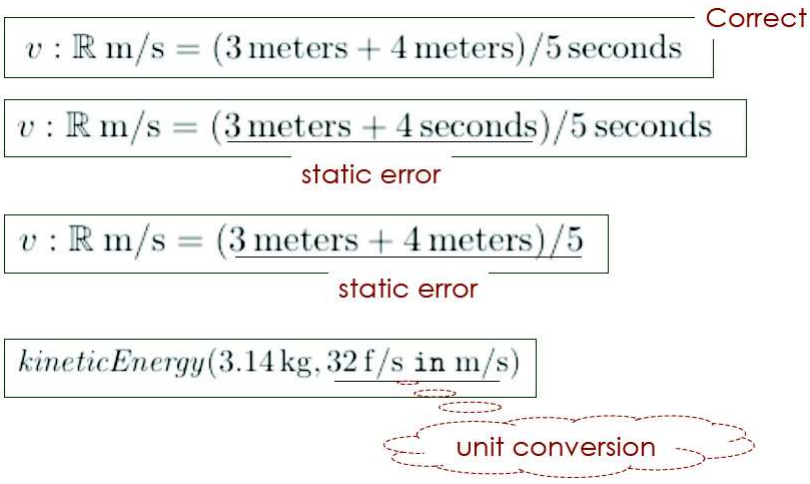


- Einheiten und Dimensionen in neueren Programmiersprachen:
  Arbeite nicht mit dimensionslosen skalaren Attributen sondern mit Maßeinheiten (units) und Dimensionsrechnung:
  - HP 50g: Working with Units



  - 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 ...

Autonomes Fahren: Tödlicher Uber-Unfall, Entscheidung, nicht zu reagieren – „False Positives“

Benutzung aggressiver Laufzeit-Zusicherungen zur Qualitätsverbesserung von Software:

(search for:) Heartbleed and Formal Methods
How to Prevent the next Heartbleed

03.05.2016 erneut: OpenSSL schließt Abkömmling der Lucky-13-Lücke

OS-Upgrade am 03.05.2016:

```
# apt list --upgradable
Auflistung... Fertig
libssl1.0.0/xenial-updates,xenial-security 1.0.2g-1ubuntu4.1 amd64
[aktualisierbar von: 1.0.2g-1ubuntu4]
openssl/xenial-updates,xenial-security 1.0.2g-1ubuntu4.1 amd64
[aktualisierbar von: 1.0.2g-1ubuntu4]
...
# apt upgrade
Die folgenden Pakete werden aktualisiert (Upgrade):
  libssl1.0.0 openssl
...
Holen:1 http://de.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libssl1.0.0 amd64 1.0.2g-1
    ubuntu4.1 [1.122 kB]
Holen:2 http://de.archive.ubuntu.com/ubuntu xenial-updates/main amd64 openssl amd64 1.0.2g-1ubuntu4.1
     [491 kB]
...
Vorbereitung zum Entpacken von
.../libssl1.0.0\_1.0.2g-1ubuntu4.1\_amd64.deb ...
Entpacken von libssl1.0.0:amd64 (1.0.2g-1ubuntu4.1) ueber (1.0.2g-1ubuntu4) ...
Vorbereitung zum Entpacken von .../openssl\_1.0.2g-1ubuntu4.1\_amd64.deb ...
Entpacken von openssl (1.0.2g-1ubuntu4.1) ueber (1.0.2g-1ubuntu4) ...
```

Acceptance of Formal Methods: Lessons from Hardware Design — the FDIV bug
20 Jahre FDIV-Bug: Ein Prozessor-Rechenfehler macht Geschichte
20 Jahre FDIV-Bug: Die Hintergründe

Haswell : Intel deaktiviert TSX per Microcode
Microcode in Debian Linux
Should I activate the additional driver: Processor microcode firmware for Intel CPUs for intel-microcode

FAQ zu Meltdown und Spectre
Spectre-NG

Doppelte Abbuchungen bei Aldi Süd — Hunderttausende Kunden in ganz Deutschland betroffen

Fehler in Standardsortieralgorithmus mit formalen Methoden aufgedeckt

Bag, Set, Sequence and OrderedSet
UML/OCL Collection Types

1. UML und SdV

1.1. Rekapitulation: UML-Klassendiagramme

1.1.1. Klassen und Objekte/Instanzen

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

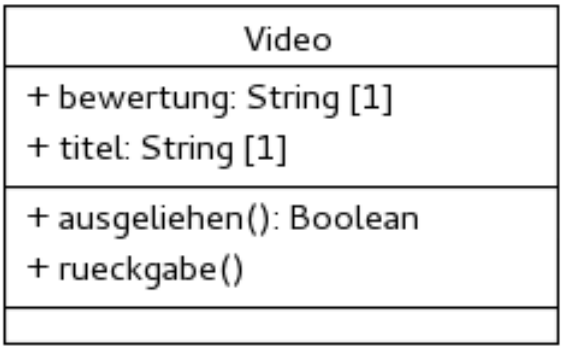


Abbildung 1.1.: Eine Klasse

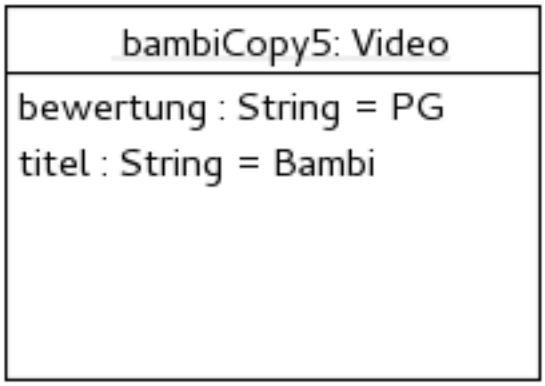
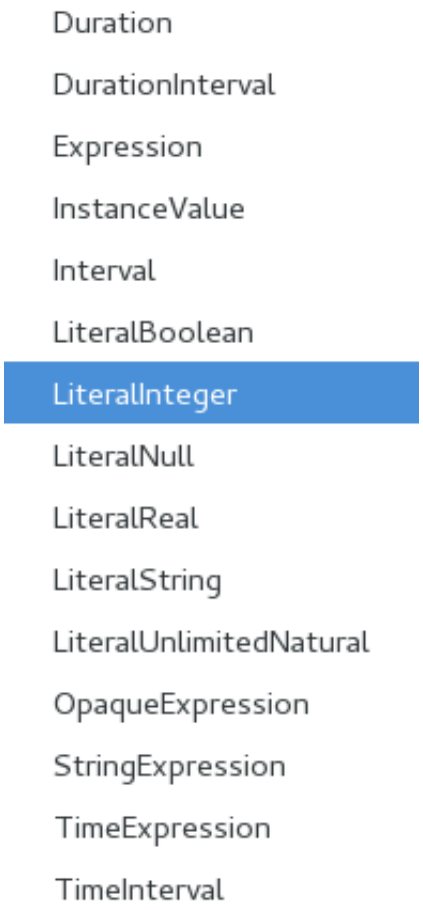


Abbildung 1.2.: Ein Objekt dieser Klasse (Instance-Specification, Slot)

Siehe:
Klasse, Objekt

```
<<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/2.5.1/PDF.

1.1.2. Klassenspezifikation

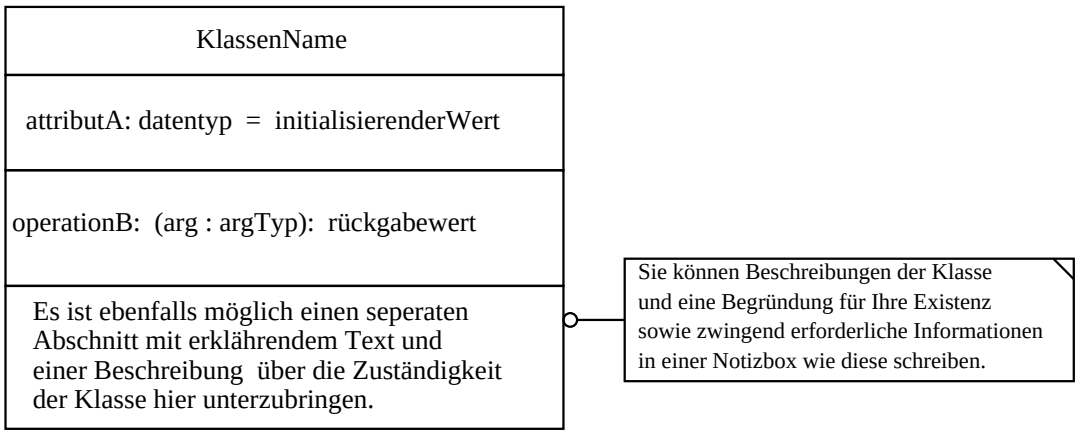


Abbildung 1.3.: Spezifikation einer Klasse

KlassenName

Normale Schrift = konkrete Klasse
kursiveSchrift **oder** << abstract >> = abstrakte Klasse
(kursive Schriften sind nicht bildschirmfreundlich; benutzen Sie die Stereotyp-Notation)

Klassen- oder Instanzenattribute

Normale Schrift = Instanzen-Bereich
Unterstrichen **oder** $ = Klassenobjekte ($ ist kein UML-Standard)
in der Regel mit kleinem Anfangsbuchstaben

Methoden/Operationen

Für abstrakte Methoden benutzen Sie = 0 oder <<abstract>> oder `{abstract}`
(=0 ist kein UML-Standard)
in der Regel mit kleinem Anfangsbuchstaben

Attribut- und Methodensichtbarkeit

+ public (öffentliche Sichtbarkeit)
- private (private Sichtbarkeit)
protected (geschützte Sichtbarkeit)
∼ package

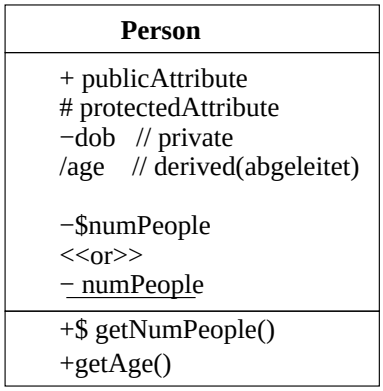


Abbildung 1.4.: Eine Klasse: Person

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

```
+ 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**

UML 2.5

Attribut:
`<<Stereotyp>> Sichtbarkeit Attributname : Paket::Typ [Multiplizität] = Initialwert {Eigenschaftswerte}`

Operation/Methode:
`<<Stereotyp>> Sichtbarkeit Operationsname (Richtung Argumentname: Argumenttyp [Multiplizität] = Standardwert {Eigenschaftswerte}, ...):Rückgabetyp {Eigenschaftswerte}`

1.1.3. Links und Assoziationen

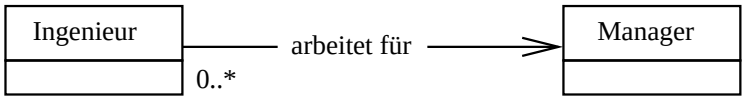


Abbildung 1.5.: Assoziationen verbinden Klassenexemplare

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.

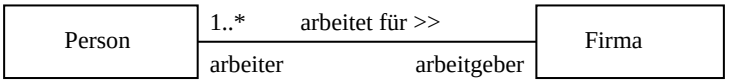


Abbildung 1.6.: Rollen in Klassen

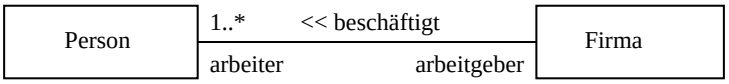
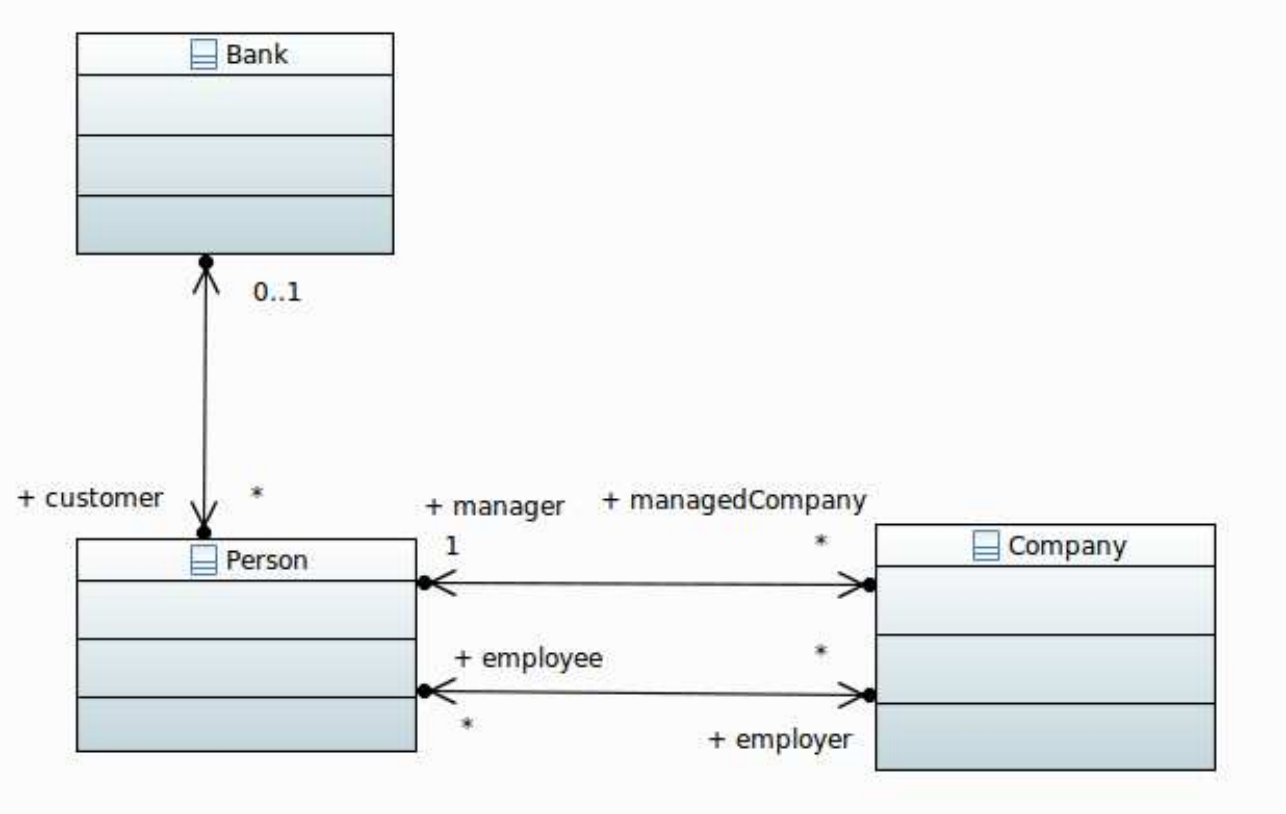


Abbildung 1.7.: Rollen in Klassen (Fortsetzung)

Einige Beispiele:



Beispiel TeamMeetingPerson (Seite 10)

Role owned by Classifier:

Ownership of Association ends by an associated Classifier may be indicated graphically by a small filled circle, which for brevity we will term a *dot*. The dot is to be drawn integral to the graphic path of the line, at the point where it meets the Classifier, inserted between the end of the line and the side of the node representing the Classifier. The diameter of the dot shall not exceed half the height of the aggregation diamond, and shall be larger than the width of the line. This avoids visual confusion with the filled diamond notation while ensuring that it can be distinguished from the line. The dot shows that the model includes a Property of the type represented by the Classifier touched by the dot. This Property is owned by the Classifier at the other end. In such a case it is normal to suppress the Property from the attributes compartment of the owning Classifier.

The dot may be used in combination with the other graphic line-path notations for Properties of Associations and Association ends. These include aggregation type and navigability.

Explicit end-ownership notation is not mandatory, i.e., a conforming tool may not support it. Where the dot notation is used, it shall be applied consistently throughout each diagram, so that the absence of the dot signifies ownership by the Association. Stated otherwise, when applying this notation to a binary Association in a user model, the dot will be omitted only for ends which are not owned by a Classifier. In this way, in contexts where the notation is used, the absence of the dot on certain ends does not leave the ownership of those ends ambiguous.

The dot is illustrated in Figure 11.26, at the maximum allowed size. The diagram shows endA to be owned by Classifier B, and because the notation must be applied consistently throughout the diagram, this diagram also shows unambiguously that endB is owned by BinaryAssociationAB.

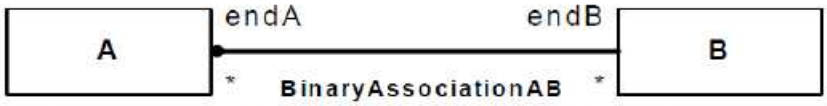


Figure 11.26 Graphic notation indicating exactly one Association end owned by the Association

Navigability notation was often used in the past according to an informal convention, whereby non-navigable ends were assumed to be owned by the Association whereas navigable ends were assumed to be owned by the Classifier at the opposite end. This convention is now deprecated. Aggregation type, navigability, and end ownership are separate concepts, each with their own explicit notation. Association ends owned by classes are always navigable, while those owned by associations may be navigable or not.

An AssociationClass is shown as a Class symbol attached to the Association path by a dashed line. The Association path may include a diamond, in which case the Class symbol shall be shown attached to the diamond by a dashed line. The Association path and the AssociationClass symbol represent the same underlying model element, which has a single name. The name may be placed on the path, in the Class symbol, or on both, but they must be the same name.

(aus: http://www.omg.org/spec/UML/2.5/PDF, Seite 200 oder aus http://www.omg.org/spec/UML/2.5.1/PDF, Seite 202)

Hinweis:
Getting It Right on the Dot.

1.1.5. Multiplizitäten (Kardinalitäten)

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

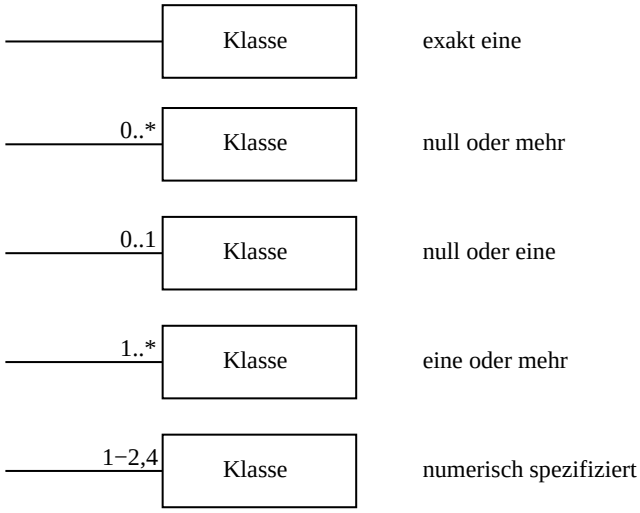


Abbildung 1.8.: Multiplizität

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 `<<history>>` 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.

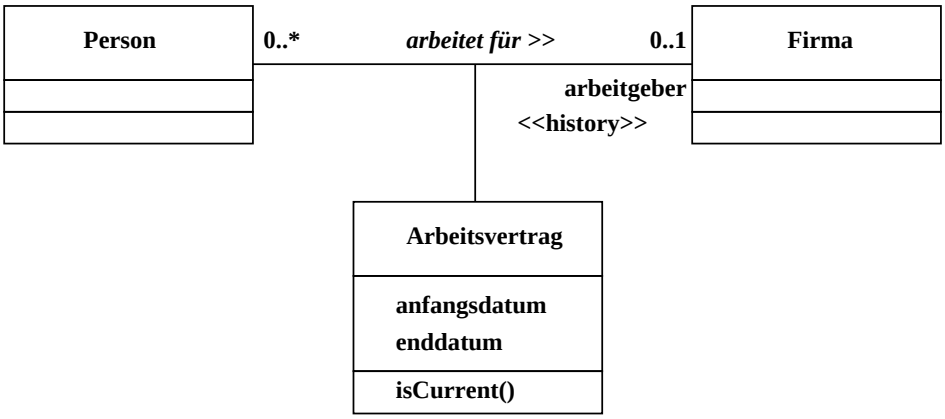


Abbildung 1.9.: Assoziierte Attribute

- 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 ("/").

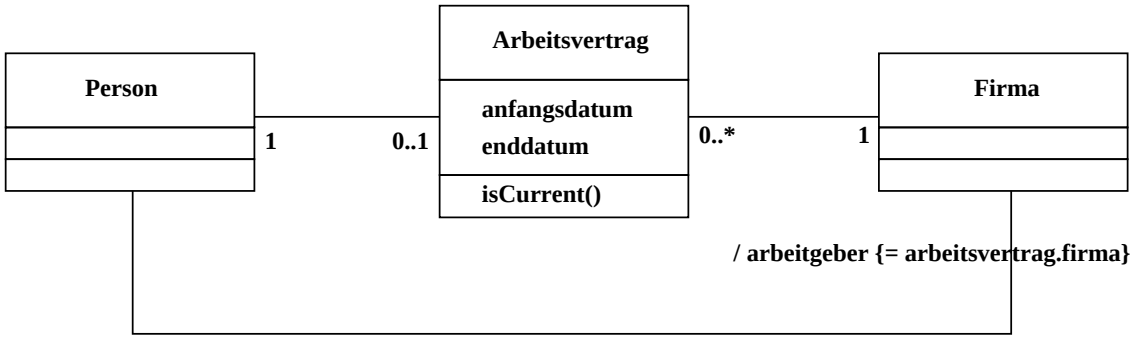
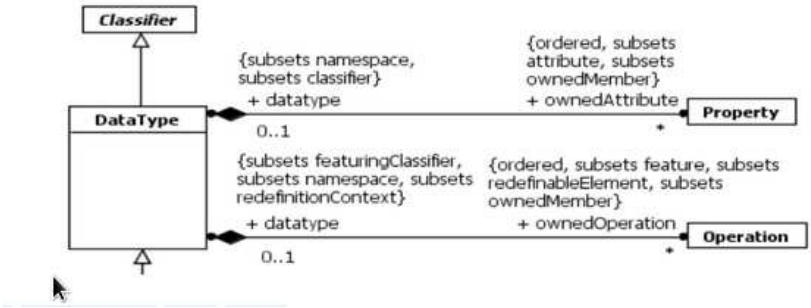


Abbildung 1.10.: Assoziiertes Attribut (Fortseztung)

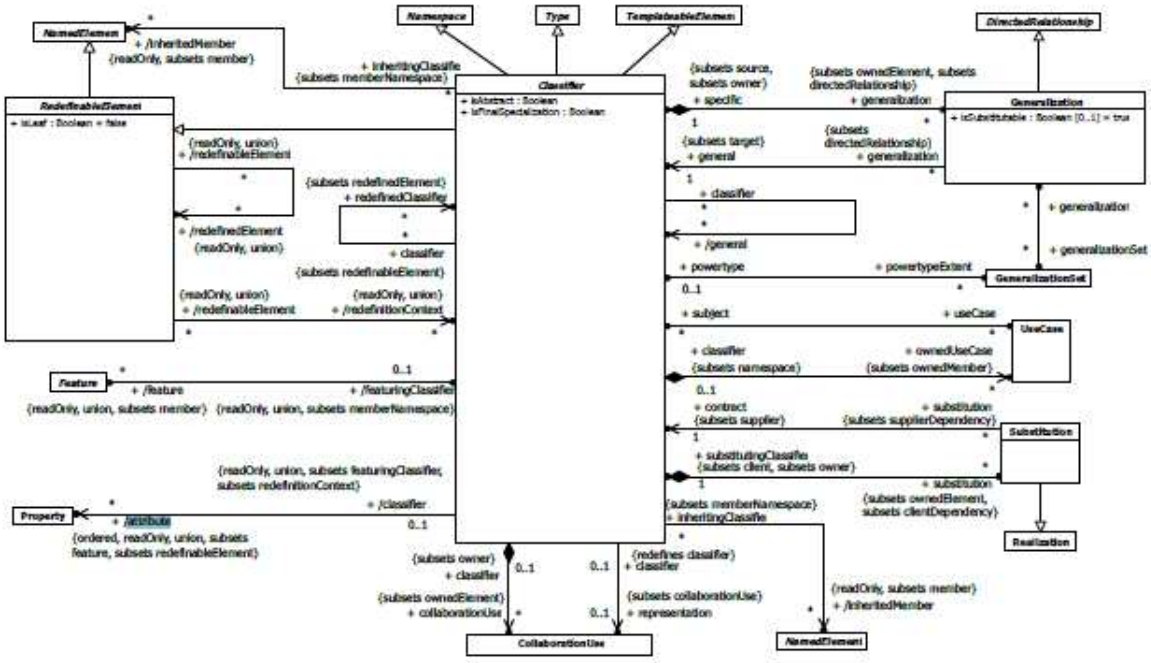
1.1.7. Subsets und Unions in OO-Modellen

Subsetted Assoziationsenden, ...:

`DataType.ownedAttribute`



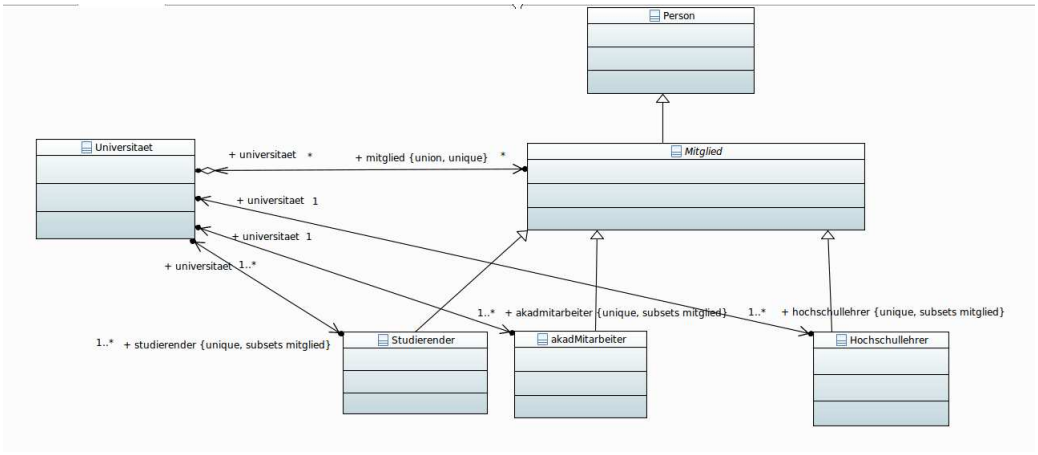
subsets `Classifier.attribute`



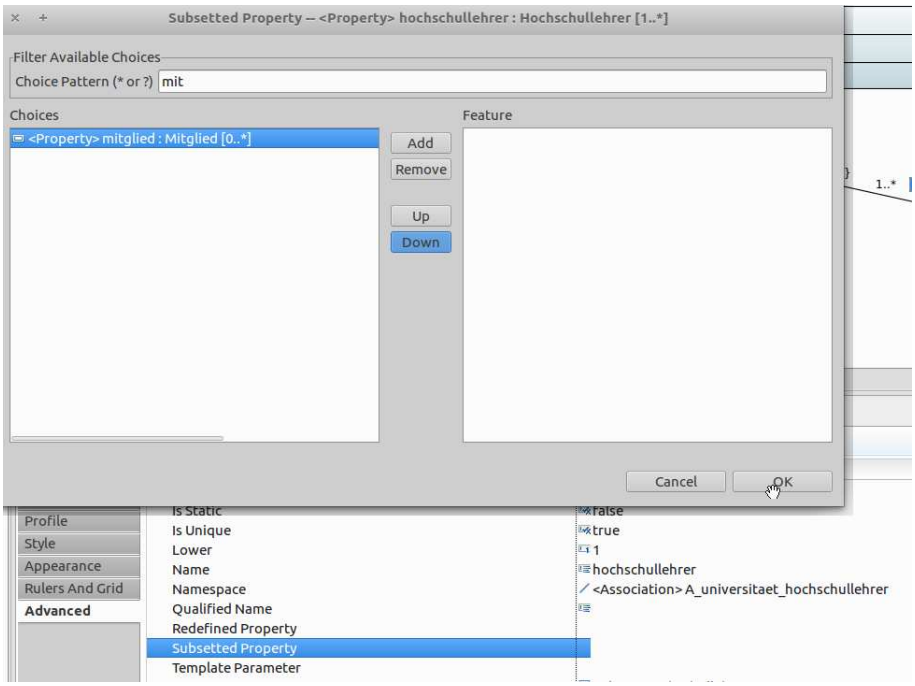
which in turn is a `union` of ...

(aus: UML 2.5-Handbuch)

In eigenen OO-Designs etwa:



Dabei wurde die Eigenschaft `subsets` folgendermaßen der Rolle `studierender` der Klasse `Universitaet` zugeordnet:



`Add`

1.1.8. 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**.

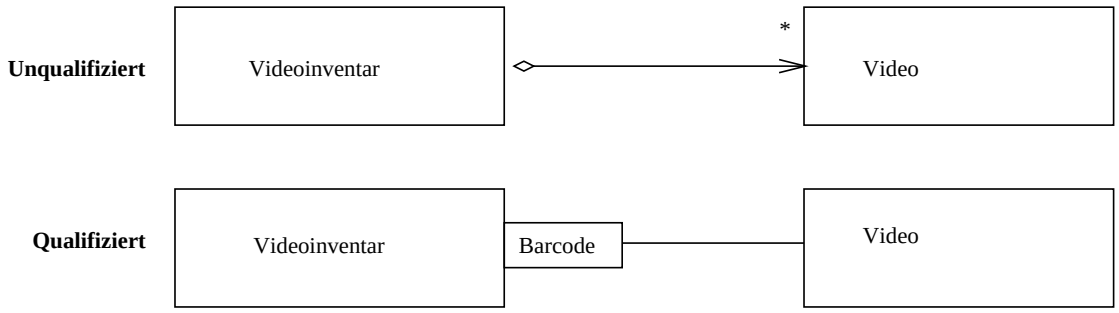
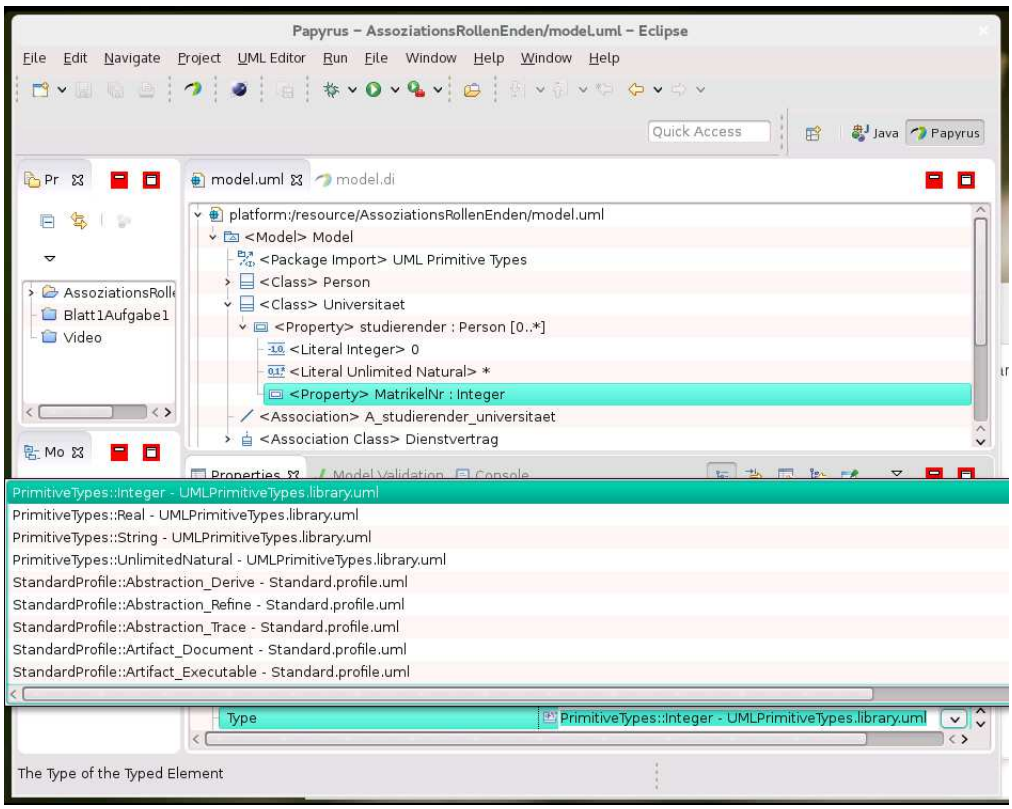
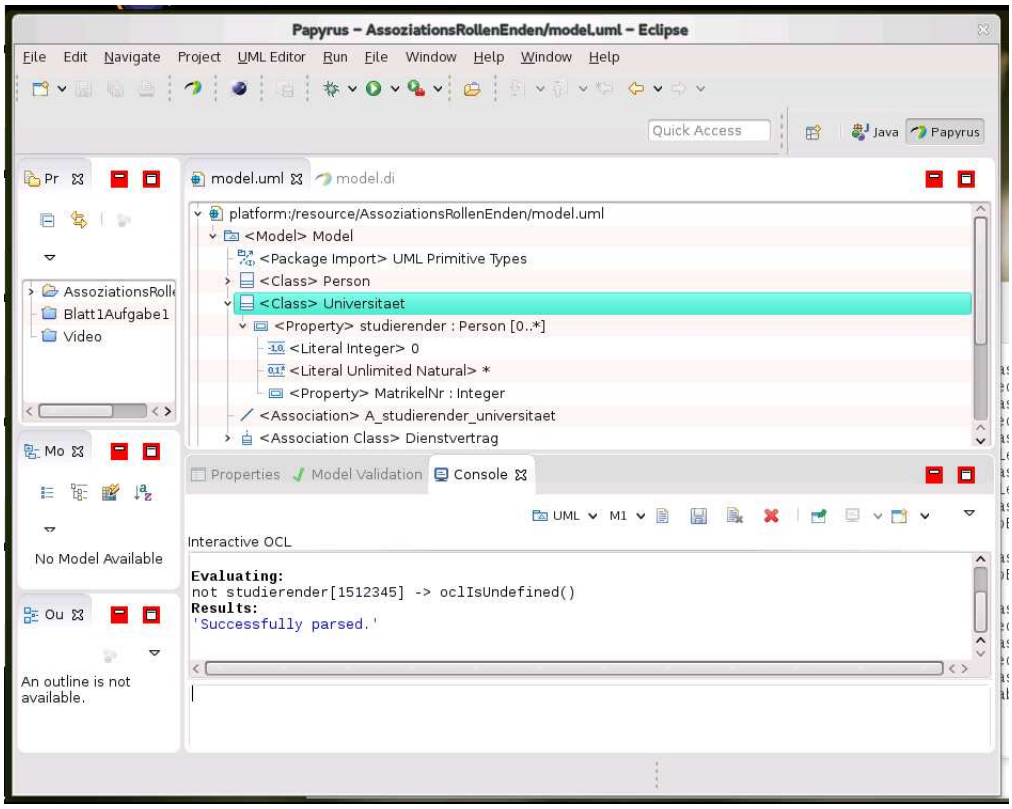


Abbildung 1.11.: Qualifizierte Assoziation

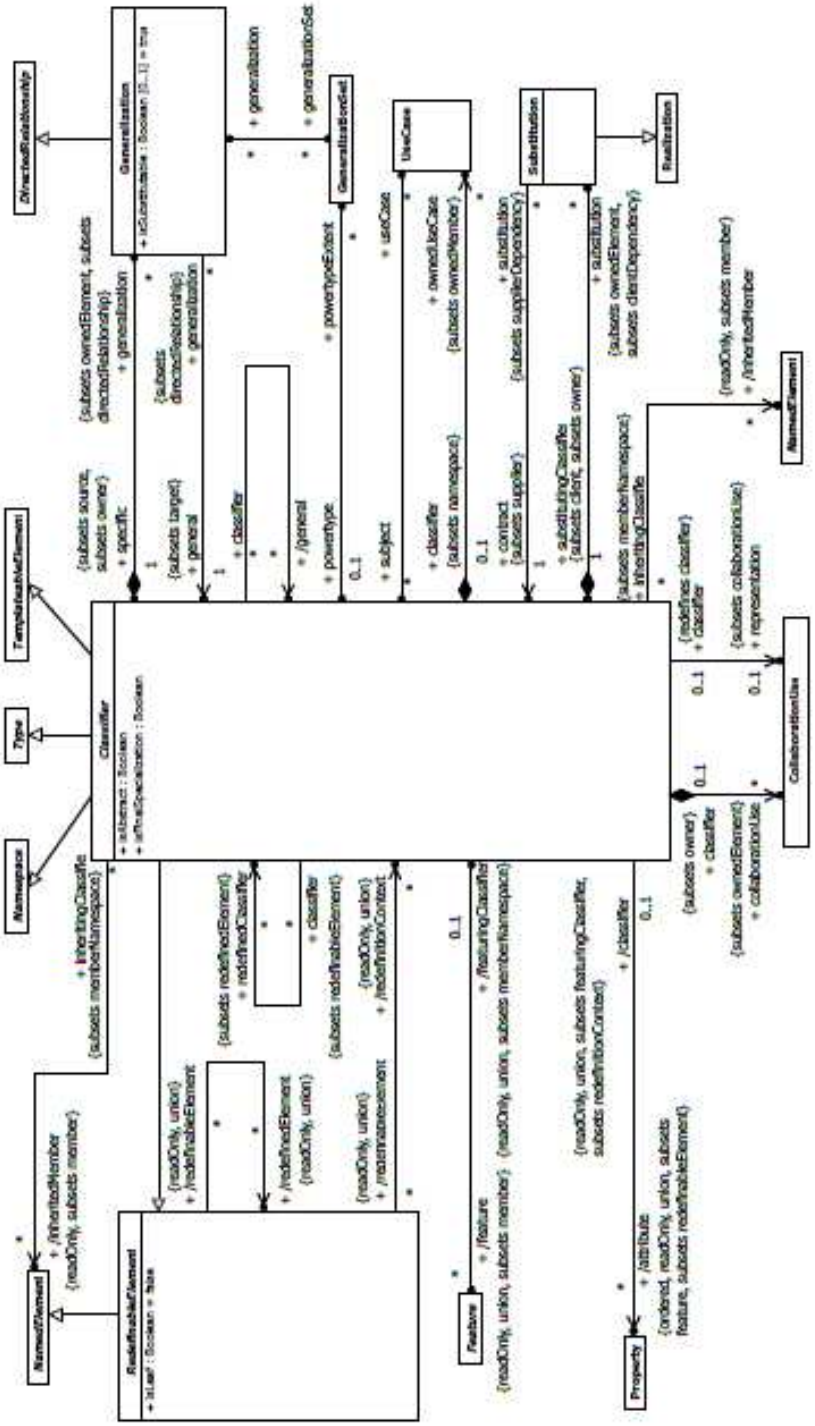
In den UML-Diagrammen werden Rollenendnamen mit `Qualifiern`, also qualifizierte Assoziationsenden leider noch nicht unterstützt. In OCL können wir aber bereits damit arbeiten, wenn wie die Qualifizierungseigenschaft (`Name` und `Typ` des Qualifieres) in die UML-Datei direkt eingeben:

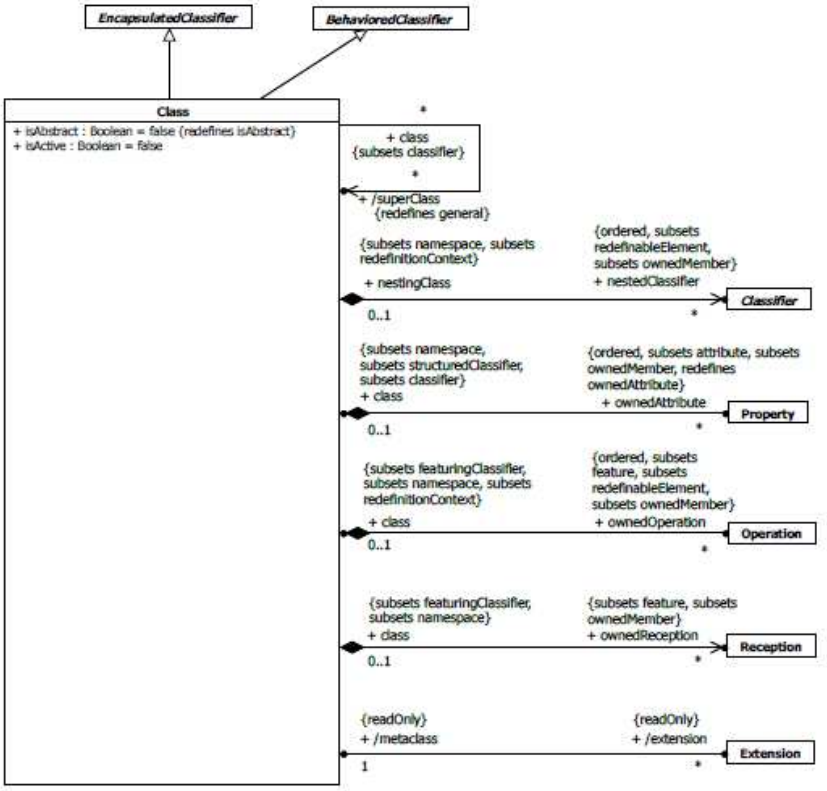
```
in der Klasse Universitaet
   auf dem Rollennamen studierender das Kontextmenue wählen
      New Child
         Qualifier
            Property
Die Eigenschaften des neuen Index-Attributs vervollständigen:
Name: matrikelNr
Typ:  Primitive Types::Integer - UMLPrimitiveTypes.library.uml
```

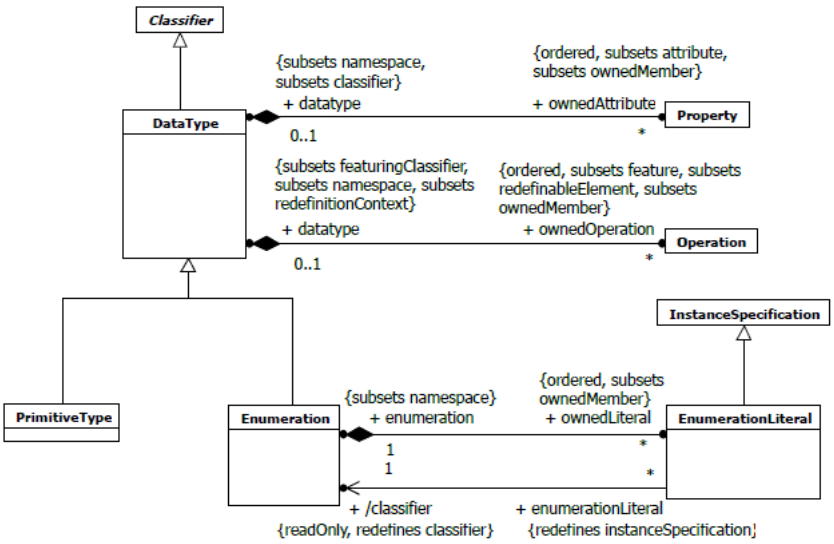

Papyrus – AssoziationsRollenEnden/model.uml – Eclipse
File Edit Navigate Project UML Editor Run File Window Help Window Help
Quick Access
Java
Papyrus
Pr
model.uml
model.di
platform:/resource/AssoziationsRollenEnden/model.uml
<Model> Model
<Package Import> UML Primitive Types
<Class> Person
<Class> Universitaet
<Property> studierender : Person [0..*]
<Literal Integer> 0
<Literal Unlimited Natural> *
<Property> MatrikelNr : Integer
<Association> A_studierender_universitaet
<Association Class> Dienstvertrag
AssoziationsRoll
Blatt1Aufgabe1
Video
Mo
Properties
Model Validation
Console
PrimitiveTypes::Integer - UMLPrimitiveTypes.library.uml
PrimitiveTypes::Real - UMLPrimitiveTypes.library.uml
PrimitiveTypes::String - UMLPrimitiveTypes.library.uml
PrimitiveTypes::UnlimitedNatural - UMLPrimitiveTypes.library.uml
StandardProfile::Abstraction_Derive - Standard.profile.uml
StandardProfile::Abstraction_Refine - Standard.profile.uml
StandardProfile::Abstraction_Trace - Standard.profile.uml
StandardProfile::Artifact_Document - Standard.profile.uml
StandardProfile::Artifact_Executable - Standard.profile.uml
Type
PrimitiveTypes::Integer - UMLPrimitiveTypes.library.uml
The Type of the Typed Element


Papyrus – AssoziationsRollenEnden/model.uml – Eclipse
File Edit Navigate Project UML Editor Run File Window Help Window Help
Quick Access
Java
Papyrus
Pr
model.uml
model.di
platform:/resource/AssoziationsRollenEnden/model.uml
<Model> Model
<Package Import> UML Primitive Types
<Class> Person
<Class> Universitaet
<Property> studierender : Person [0..*]
<Literal Integer> 0
<Literal Unlimited Natural> *
<Property> MatrikelNr : Integer
<Association> A_studierender_universitaet
<Association Class> Dienstvertrag
AssoziationsRoll
Blatt1Aufgabe1
Video
Mo
No Model Available
Ou
An outline is not available.
Properties
Model Validation
Console
UML
M1
Interactive OCL
Evaluating:
not studierender[1512345] -> oclIsUndefined()
Results:
'Successfully parsed.'

1.1.9. UML Structure: Classifier, Class und DataType







Classifier (UML)
Classifier (UML), in Englisch
UML Classifier
What do you mean by classifiers in unified modeling language?

1.1.10. 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 >>
<< ... >>
```

http://de.wikipedia.org/wiki/Stereotyp_(UML)
Beispiele für Stereotypen
UML Standard Profile (Kapitel 22)
PAPYRUS USER GUIDE SERIES — About UML profiling, version 1.0.0

1.1.11. 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:

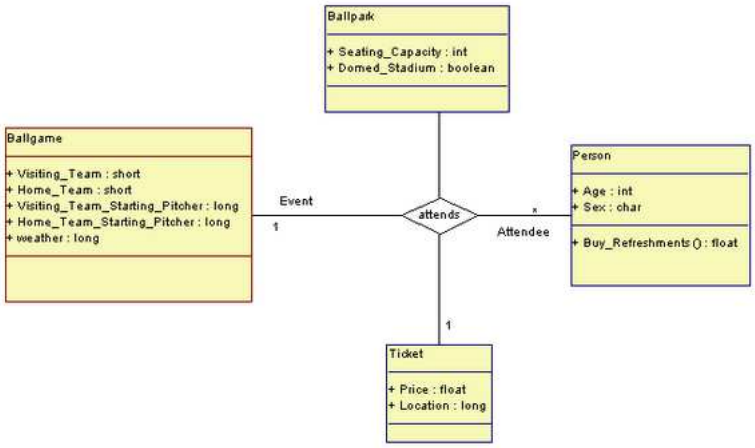
- {Autor = (Dave,Ron)}
- {Versionsnummer = 3}
- {Location = d:\Location\uml\examples}
- {Location = Node: Middle Tier}

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: „ML tagged value with papyrus“)

1.1.12. Mehrgliedrige Assoziationen

n-äre Assoziation



(aus: „Fußballstadion – Fussballspiel — Person – Eintrittskarte“)

Artists, song writers, recording companies and recording studios

1.1.13. 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.

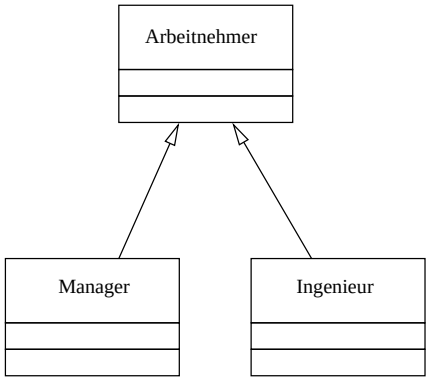


Abbildung 1.12.: Generalisierung, Spezialisierung und Vererbung

1.1.14. 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

1.1.15. 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.

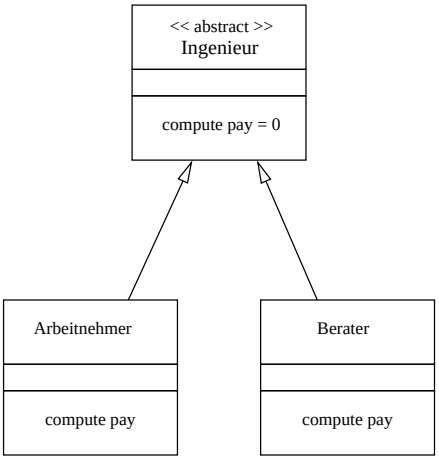


Abbildung 1.13.: Abstrakte Klassen

1.1.16. 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.

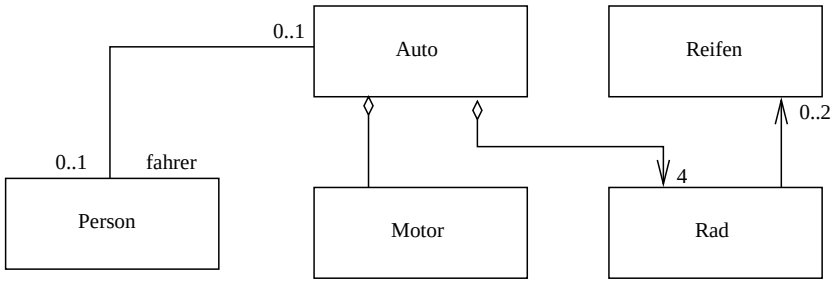


Abbildung 1.14.: Komposition / Aggregation

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.

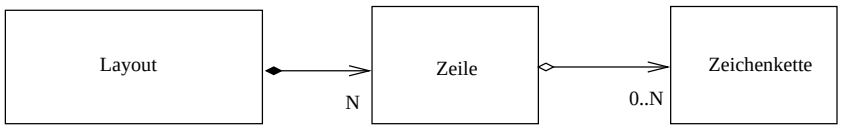
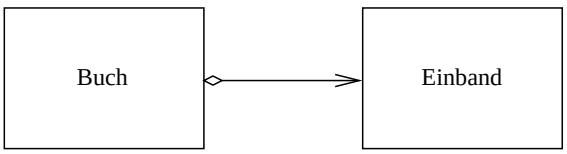


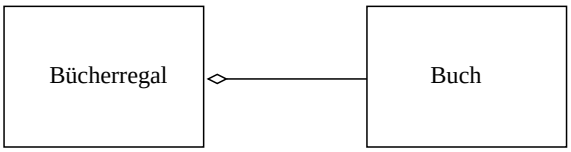
Abbildung 1.15.: Komposition zwischen Layout und Zeile

- 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).



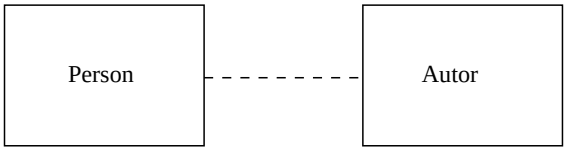
Aggregation:

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



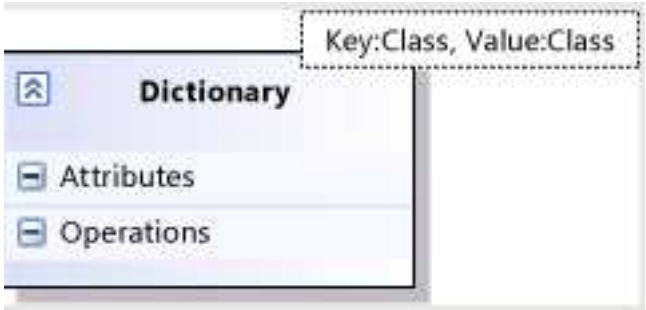
Dependancy:

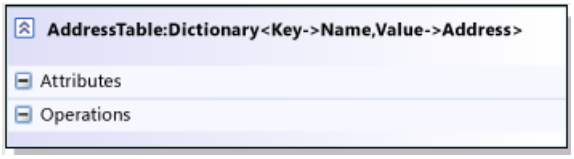
- Instanzen der Klasse Person haben vorübergehende Beziehungen zu Instanzen der Klasse Autor
- Beispiel: Eine Person liest ein Buch, dann gibt sie es einem Freund.



1.1.17. template classes

Template Types:





1.1.18. Modell und Metamodell

UML 4-Schichten-Architektur

1.1.19. UML 2.5: 26.09.2013

Meta Object Facility MOF 2.4.1
Unified Modeling Language (UML) Version 2.5.1
UML 2.5 ist verabschiedet: 27. September 2013

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

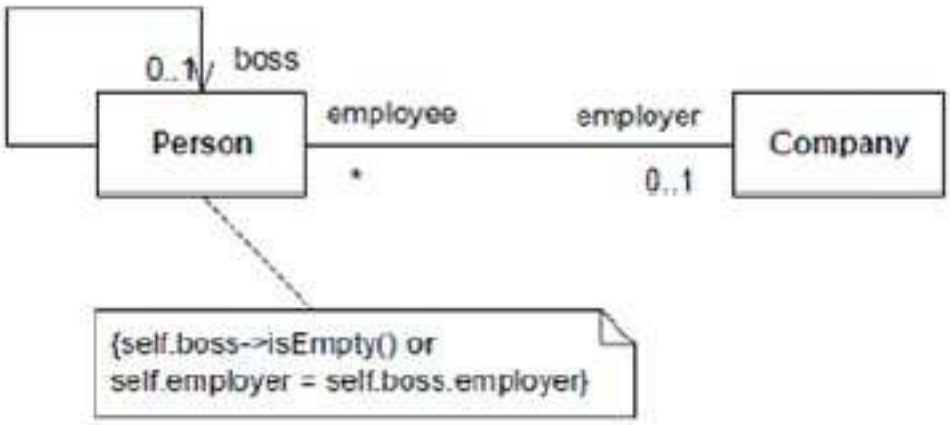


Figure 7.14 Constraint in a note symbol

Figure 7.15 shows a constraint string attached to an attribute.

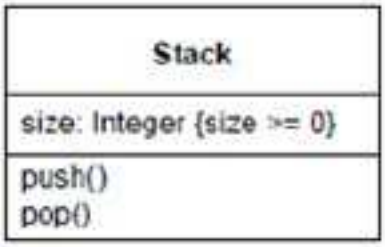


Figure 7.15 Constraint attached to an attribute

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
```

1.1.20. UML 2 Style Guidelines

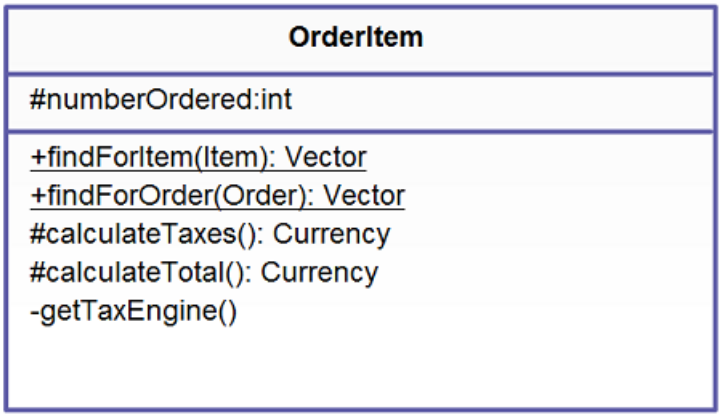
UML Style Guidelines

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.

(aus: http://www.cs.rit.edu/~cs4/UML_style.html)

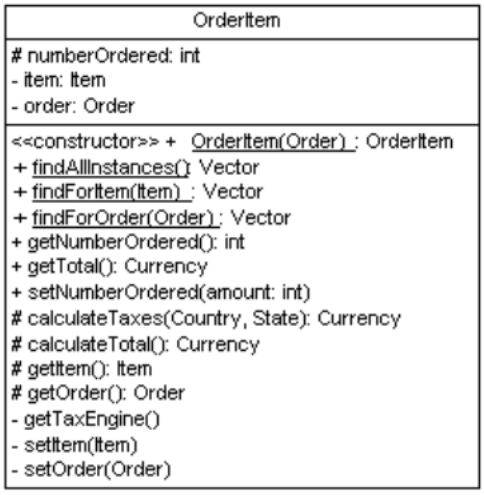


Put common terminology for names
Choose complete singular nouns over class names

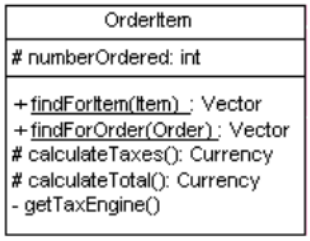
http://creately.com/diagram-type/article/simple-guidelines-drawing-uml-class-diagrams
UML 2 Class Diagramming Guidelines

with and without scaffolding

With Scaffolding



Without Scaffolding



1.2. Spezifikation einfacher Klassen nach Prinzipien der SdV

1.2.1. Ein einfaches Beispiel ...

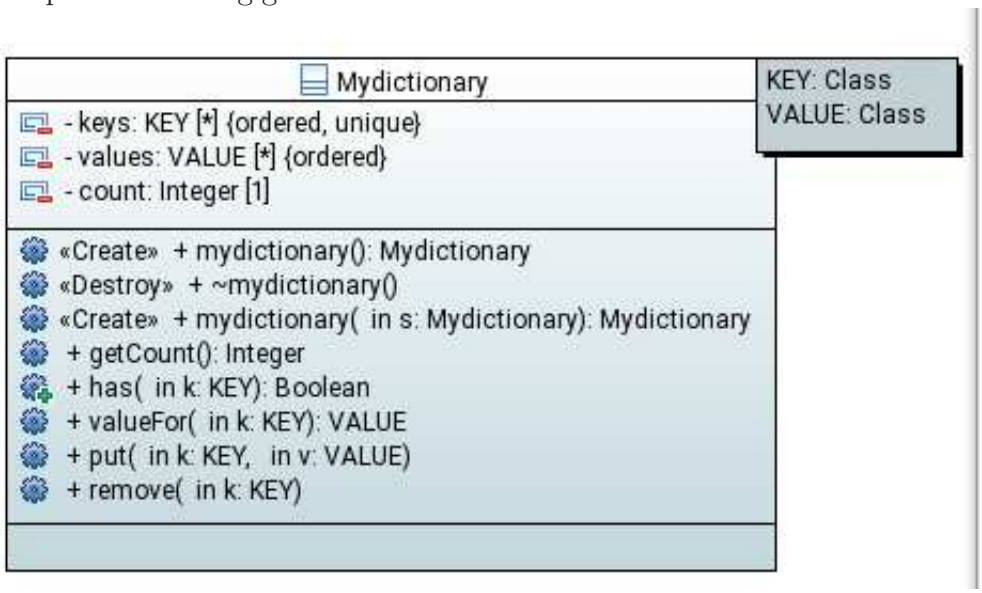
KEY:Class, VALUE:Class

Mydictionary

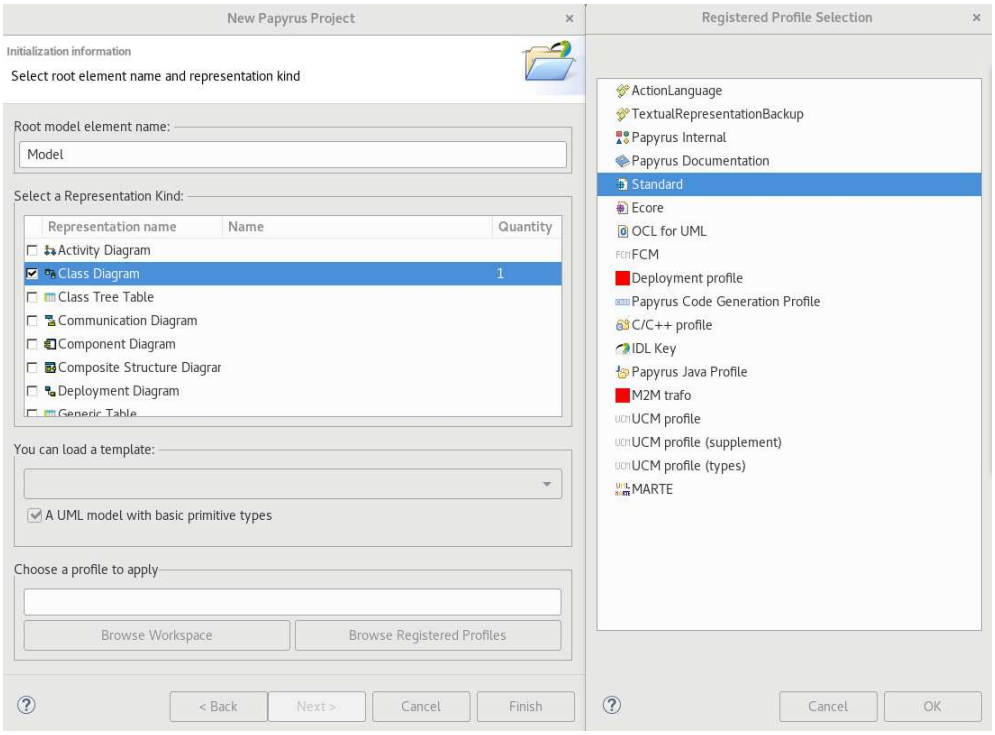
- keys: vector<KEY>*
- values: vector<VALUE>*
- count: unsigned int

```
/* basic queries */
+ get_count() : unsigned int
+ has(k: const KEY &) : bool
+ value_for (k: const KEY &) : VALUE

/* constructors */
+ << Create >>$ mydictionary()
+ << Create >>$ 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
```

oder besser in sprachunabhängigerer Notation:



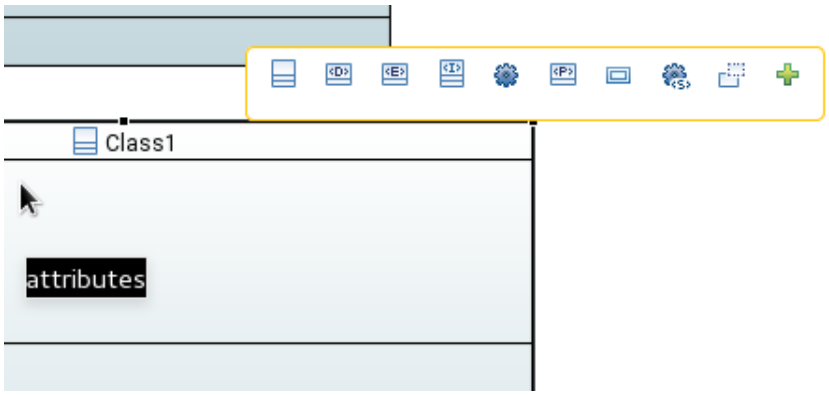
Bemerkung:



nach Anwendung des UML Standard Profiles und Auswahl des Stereotypen <<Create>> für den Konstruktor, ...

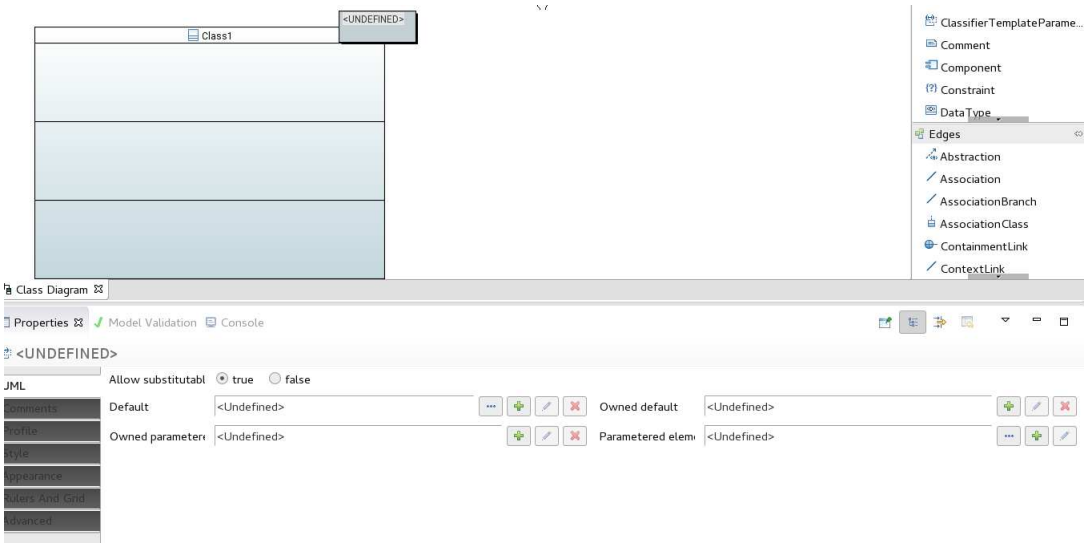
Eine generische Klasse in einem Papyrus UML-Diagramm einrichten

Klasse im Model.di-Diagramm erzeugen,
die Maus ins Klassenfenster schieben,
im erscheinenden gelb umrandeten Auswahl-Popupfenster die vorletzte
Position "Add Refinable Template Signature" anklicken:

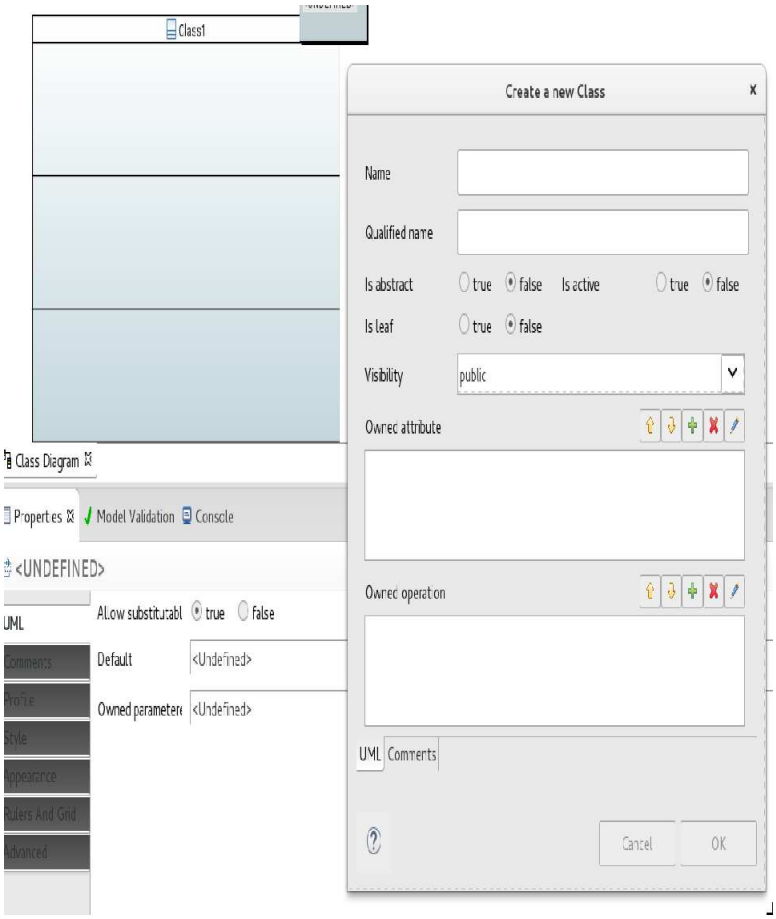


Es erscheint das Klassentemplate-Kästchhen am oberen rechten Rand der Klasse:

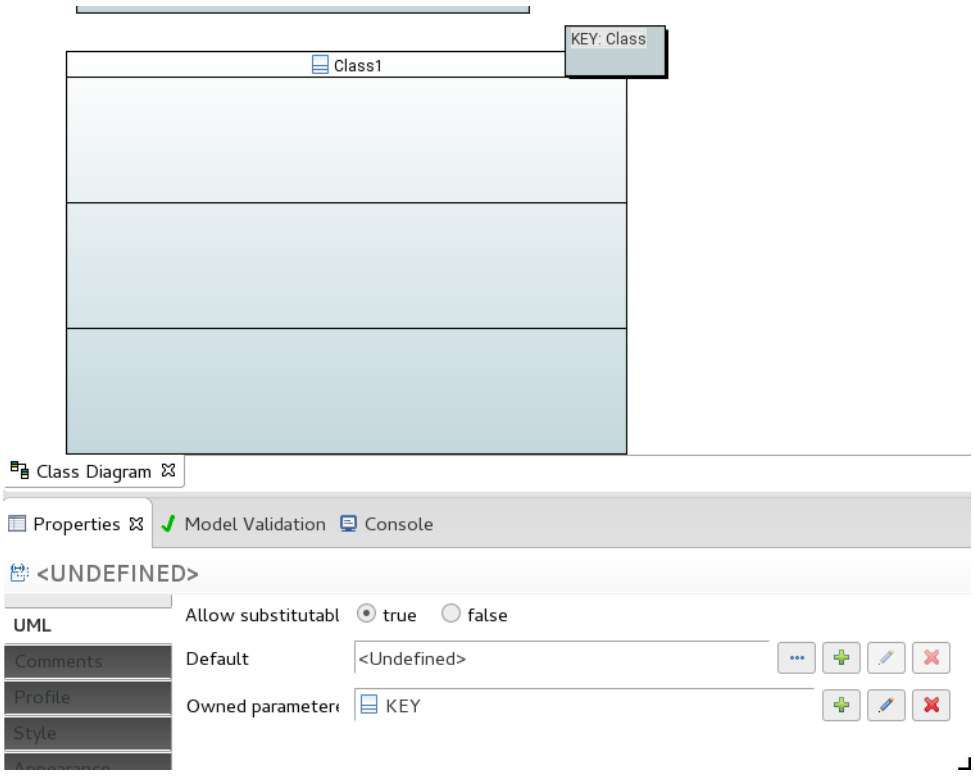
Nun das Node-Tool "Classifier Template Parameter" anklicken
und in dem Class1-Signatur-Kästchen erneut klicken:



Im Papyrus-Properties-Fenster "UNDEFINED" das "+" nach "Owned Parameter" anklicken und im erscheinenden Popup-Fenster Class auswählen.
Es erscheint das "Create a new Class"-Fenster,



mit dem Sie die neue Klasse "KEY: Class" Ihrem Model
hinzufügen können, indem Sie ins Name-Feld des "Create a new Class"-Fenster
KEY eintippen und das Fenster dann durch Anklicken des OK-Knopfes verlassen:



Analog können Sie bei Bedarf weitere generische Parameter der Klasse
Class1 hinzufügen.

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:

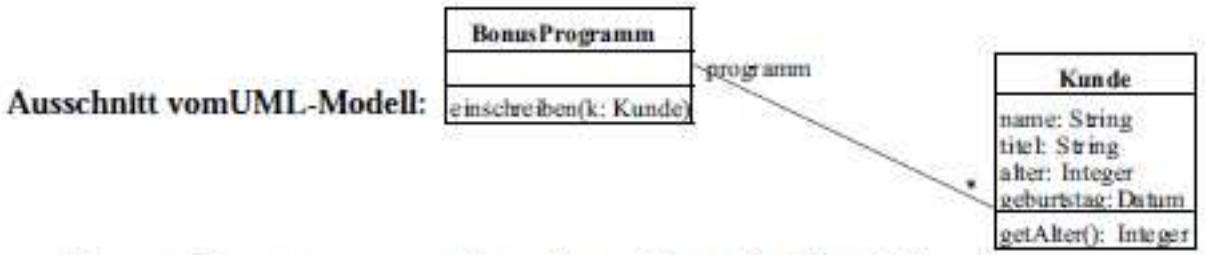
Spezifikation durch Vertrag — eine Basistechnologie für eBusiness

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:

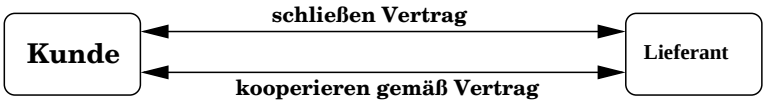


Abbildung 1.16.: Kunden-Lieferanten-Modell

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)

Stroustrup: Thoughts on C++17 - An Interview
P0380: Contracts as C++ attributes
P0542: Contract based programming in C++

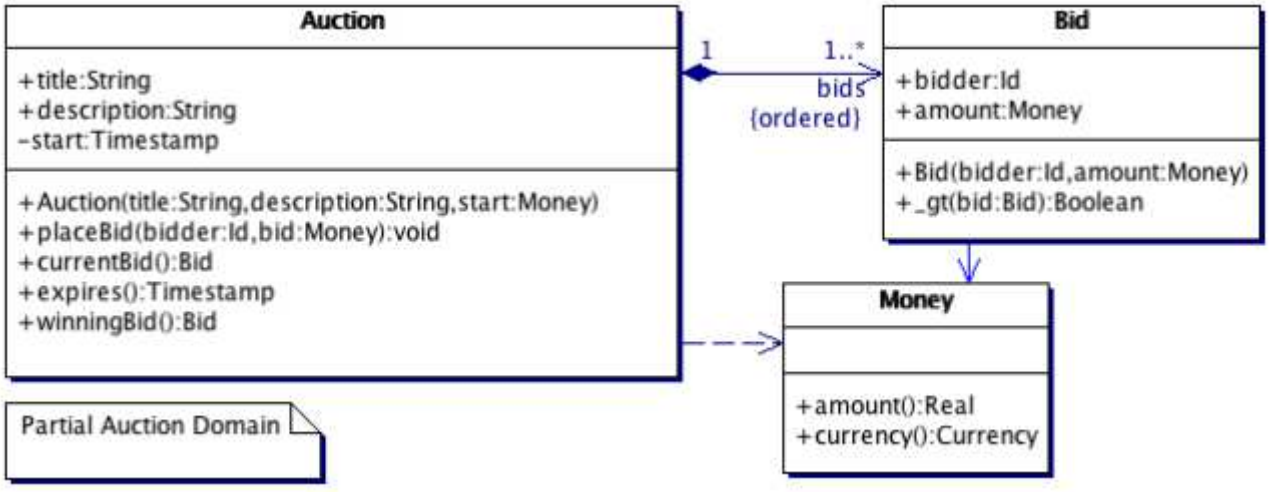
C++20 Codeverträge

Um solche „formalen“ Spezifikationen auch maschienenüberprüfbar zu machen, wurde für C++20 in Contracts programming for C++20 (J. Daniel Garcia) eine C++-Sprachergänzung für Codeverträge mit Hilfe der Attribute `expects, ensures` und `assert` vorgeschlagen:

```
template <tyename Iter>
  requires RandomAccessIterator<Iter>()
Iter bsearch(Iter p, Iter q, std::iterator_traits<Iter>::value_type w){
  [[expects default: p <= q]]
  [[expects audit:   is_sorted(p, q)]]
  [[ensures result:  !(result != q) || (*result == w
                    && (!(result != p) || *(result-1) != w))  ]]
  [[ensures axiom result:
   !(std::none_of(p, q, [](auto v){return v == w;}))|| result == q ]]
{
   // ...
}
```

1.2.5. OCL2 Codeverträge

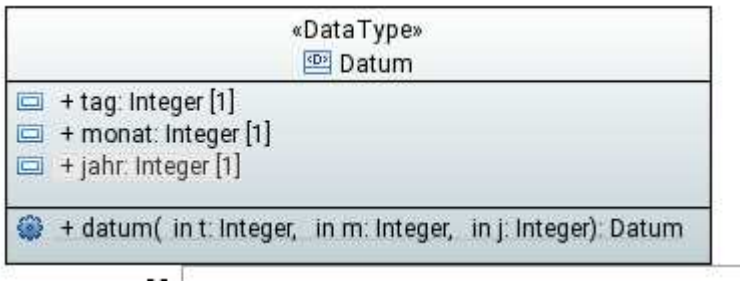
Dan Massey: „Object Constraint Language: Design by Contract and Queries.“




```
context Auction::placeBid(bidder:Id, bid:Money)
pre bidderValid: not bidder.oclIsUndefined()
pre bidValid: not bid.oclIsUndefined()
pre currencyValid: bid.currency =
        self.bids->first().amount.currency
pre bidHigher: bid > self.currentBid().money
post amountUpdated: self.currentBid().amount = bid
post bitIncreased: self.currentBid() > self.currentBid@pre()

context Auction
inv lifetimeValid: self.expires() > self.start

-- ...
```

1.2.6. Beispiel-Codeverträge

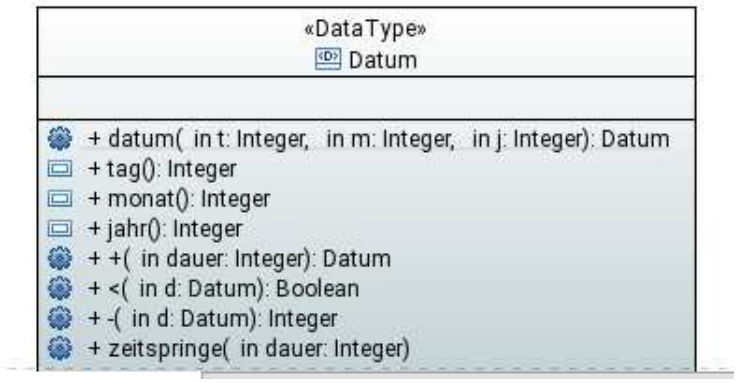
Zunächst eine SdV-konforme Umkonzeption des `<<datatype>> Datum` im Sinne der methodischen Restriktion:
aus



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

-- ...
```

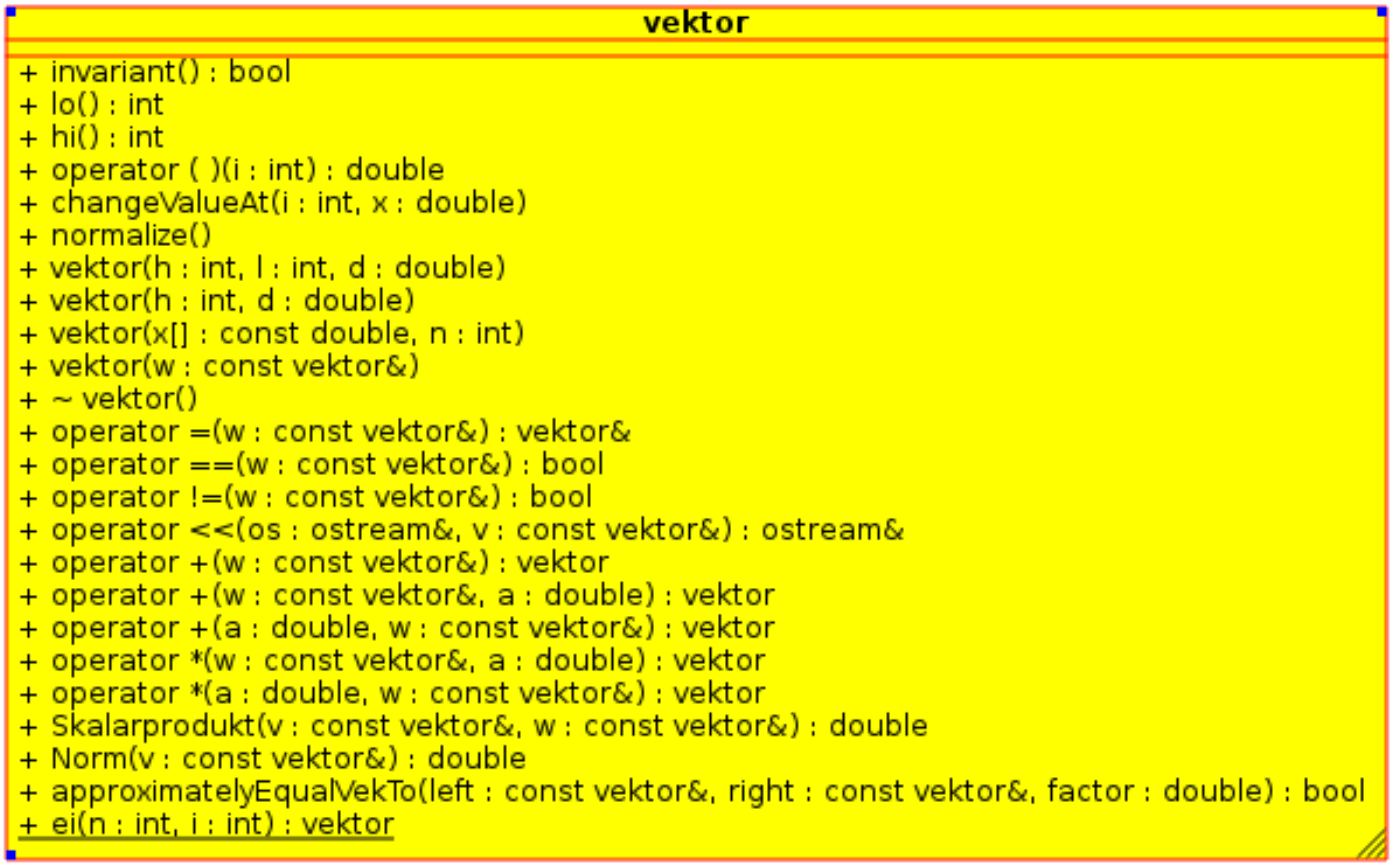
Aufgabe:
Überlegen Sie sich Codeverträge für die Operationen `+, <, -, zeitspringe(.)`.

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

) als Workaround für die noch fehlenden SdV-Sprachkonstrukte von C++17:

Klasse vektor:



Grundlegende Observatoren:
`int lo() const` (unterer Index eines `vektor`-Objekts),
`int hi() const` (oberer Index eines `vektor`-Objekts),
`double operator()(i : int) const` (Komponenten-Getter eines `vektor`-Objekts).

Klasseninvariante und Vorbedingungen der grundlegenden Observatoren der Klasse `vektor`:

- Klasseninvariante

```
virtual bool vektor::invariant() const     // Kein DO
{
  return lo() <= hi();
}
```

- `double vektor::operator()(int i) const`

```
double  vektor::operator()(int i) const
DO
  REQUIRE((lo()<=i) && (i<=hi()));
  ...
}
```

Beispiel-Codeverträge für die 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
std::numeric_limits::epsilon, siehe insbesondere „bool almost_equal(T x, T y, int ulp)“

- 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
        ())));
  ...
}
```

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;
}
```

1.2.7. 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.

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

1.2.7.1. 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:
{$VB_{Parameter}$} Konstruktor {Inv and $NB_{Konstruktor}$}

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

1.2.7.2. 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

1.2.7.3. Interface LoeseLGS

```
------------------------------ LoeseLGS-Elter
QUERIES
   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
QUERIES
   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 ||"
```

1.2.7.4. 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 Count = OLD(Count)
.
.
.
```

1.2.7.5. 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:

- Konzipiere `ueberquereBruecke(., .)` so um, dass die SdV-Regel, „ACTIONS haben keine Rückgabewerte“ eingehalten wird.

- Überlege Contracts und Subcontracts im Umfeld:
  - Kunde/Stammkunde
  - Firmenkonto/Privatkundenkonto
  - Vereinsmitglied /Vorstandsmitglied
  - ...

1.2.8. 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.

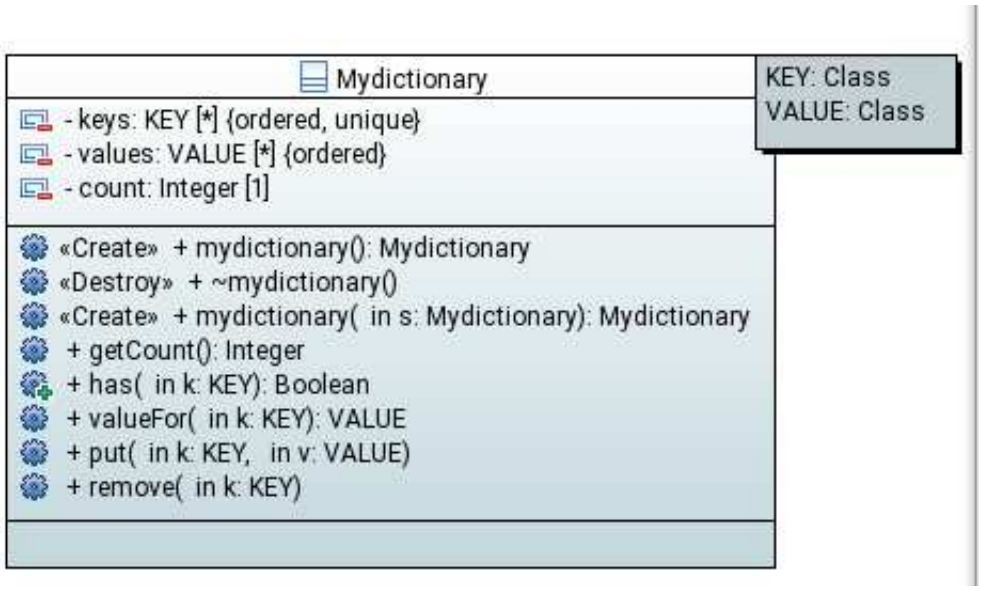
4. Für jeden Konstruktor/Modifikator schreibe eine Nachbedingung, die die Werte aller grundlegenden Observatoren am Ende einer Methode festlegt.

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.

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




```
import 'model.uml'

/* Invarianten der privaten Attribute fuer das
   Implementierungsteam */
context Model::Mydictionary
inv keysValid: keys <> null
inv valuesValid: values <> null
inv countValid: count >= 0
inv sameNumberOfKeysAndValues: keys->size() = values->size()
inv consistentCountValue: keys->size() = count
/* End Invarianten der privaten Attribute fuer das
   Implementierungsteam */


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

```
/* constructor */
context Model::Mydictionary::mydictionary(): Model::
   Mydictionary
post newDictionary: result.oclIsNew()
post emptyDictionary: result.getCount() = 0

context Model::Mydictionary::mydictionary(s: Model::
   Mydictionary): Model::Mydictionary
post newDictionary: result.oclIsNew()
post copiedDictionary: s.getCount() = result.getCount()
post copiedDictionary2:  Model::Mydictionary::KEY->forAll(
        k | s.has(k) implies (result.has(k) and result.valueFor
           (k)=s.valueFor(k)  )
)

/* modificators */
context Model::Mydictionary::put(k: Model::Mydictionary::KEY, v
   : Model::Mydictionary::VALUE): OclVoid
pre: not has(k)
post keyIncluded: has(k)
post countIncremented: getCount() = getCount@pre() + 1
post newValue: valueFor(k) = v
post frameCondition: Model::Mydictionary::KEY->forAll(
        kl | has@pre(kl) imlies valueFor@pre(kl) = valueFor(kl)
)

context Model::Mydictionary::remove(k: Model::Mydictionary::KEY
   ): OclVoid
pre: has(k)
post keyRemoved: not has(k)
post countDecremented: getCount() = getCount@pre() - 1
post frameCondition: Model::Mydictionary::KEY->forAll(
        kl | has@pre(kl) implies (kl = k or
                valueFor@pre(kl) = valueFor(kl)
        )
)
```

Wahrscheinlich muß es später

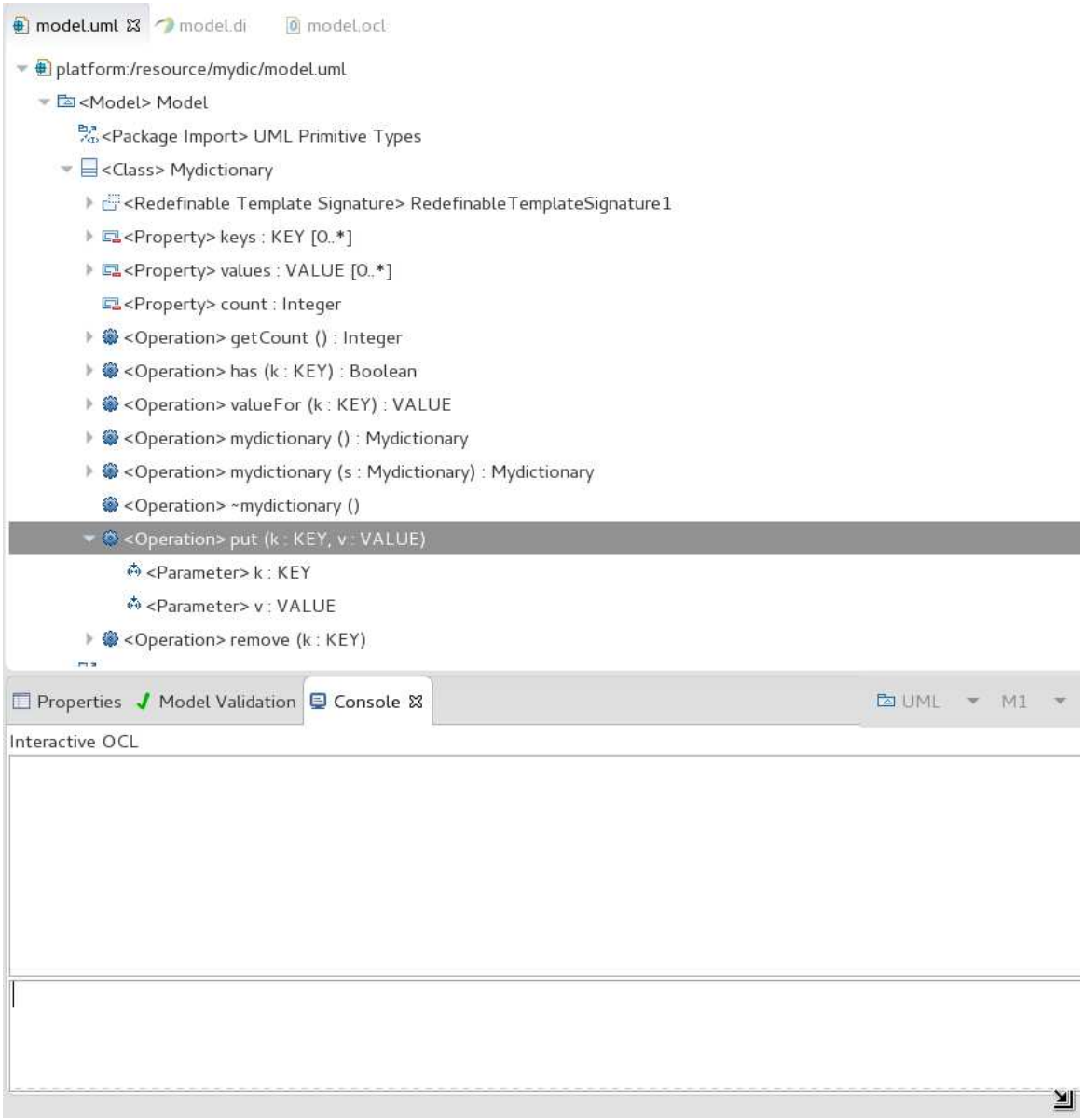
```
...
context Model::Mydictionary(KEY,VALUE)
inv: keys <> null
...
```

heißen.

Aufgaben:

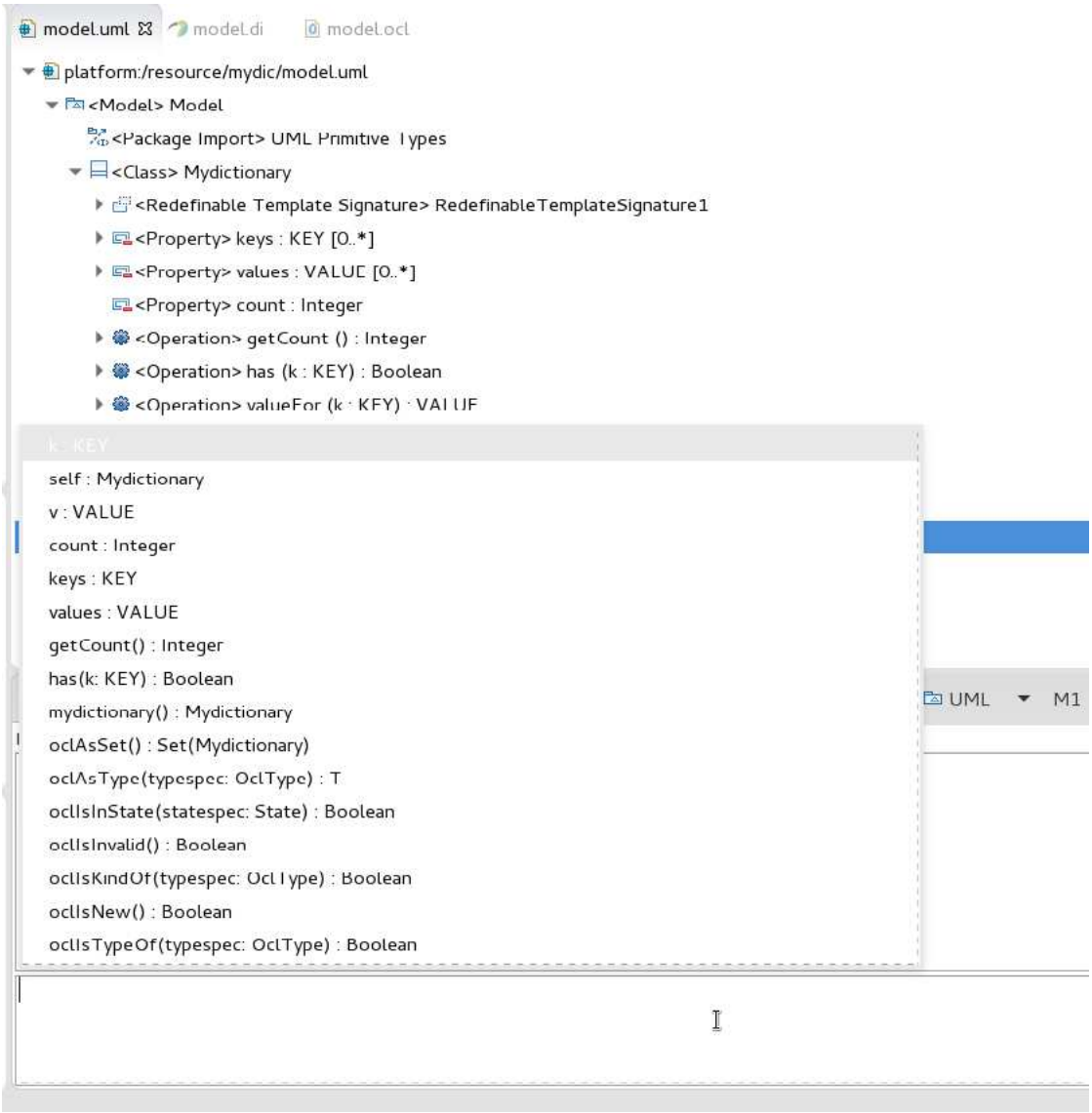
- Warum sind die oben genannten `forAll()`-Nachbedingungen und die `KEY->select`-Anweisung eventuell auszukommentieren?
- Ändern Sie das Design von `Mydictionary`, indem Sie `has()` zum abgeleiteten Observator machen und einen neuen grundlegenden Observator `usedKeys(): KEY [*]` einführen. Welchen Vorteil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

Im „Interactive OCL“-Editor der UML-Datei

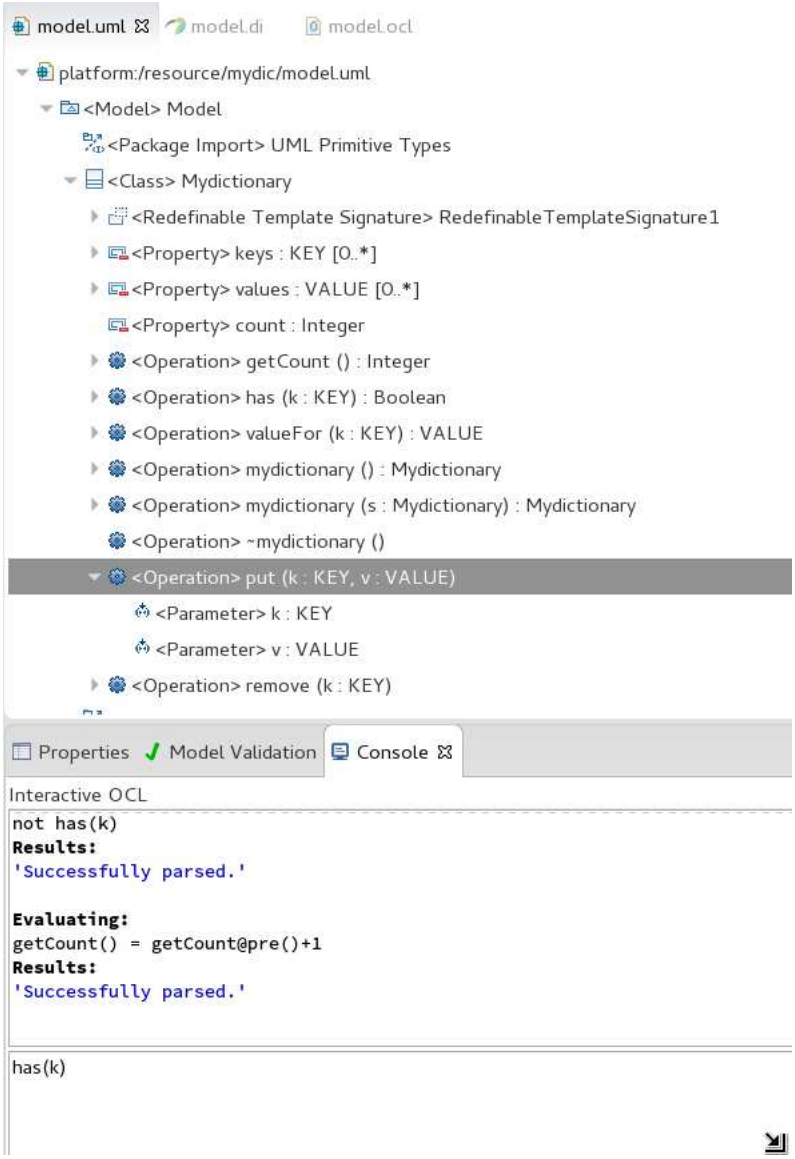


den Kontext durch Anklicken der entsprechneden UML-Datei-Zeile auswählen (hier etwa: <Operation> put(k: KEY, v: VALUE)),

dann stehen die entsprechenden kontextuell verfügbaren Eigenschaften des UML-Modells zur automatischen Vervollständigung (Ctrl-Leertaste) zur Verfügung:



und man kann, Vorbedingung, Nachbedingung, ... interaktiv syntaxüberprüfen lassen:



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:

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 assoziierter Klassen-Exemplare

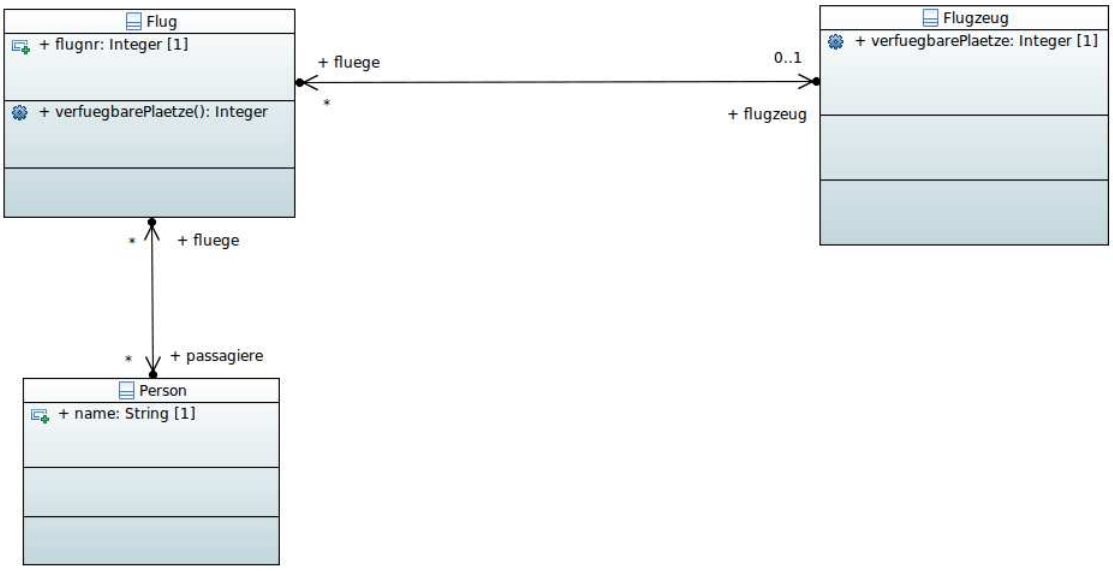


Abbildung 2.1.: Modell Flug/Flugzeug/Passagier

Erste Modell-Codeverträge:

```
import 'model.uml'

context Model::Flug
inv flugNrPositive: flugnr > 0

context Model::Flug::verfuegbarePlaetze(): Integer
post resultNonNegative: result >= 0

context Model::Flugzeug
inv verfuegbarePlaetzeNonNegative: verfuegbarePlaetze >= 0

context Model::Person
inv nameValid: name <> '' and name.size() >=2
```

OCL Overview
OCL 2.5 Plans: save navigation, ...

OCL 2.4: Navigating Association Classes (7.5.4, 7.5.5)

Weitere Semantikspezifikationen:

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

context Model::Flugzeug
inv ausreichendPlaetze: fluege.verfuegbarePlaetze()->sum() <=
    self.verfuegbarePlaetze
```

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.

und damit formulierte Semantik:

```
context Model::Flug
inv mVPNonNegative: momentanVerfuegbarePlaetze() >= 0

context Model::Flugzeug
inv mVPGeringAnzahlig: fluege.momentanVerfuegbarePlaetze()->sum
   () <= 0.05 * verfuegbarePlaetze
```

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

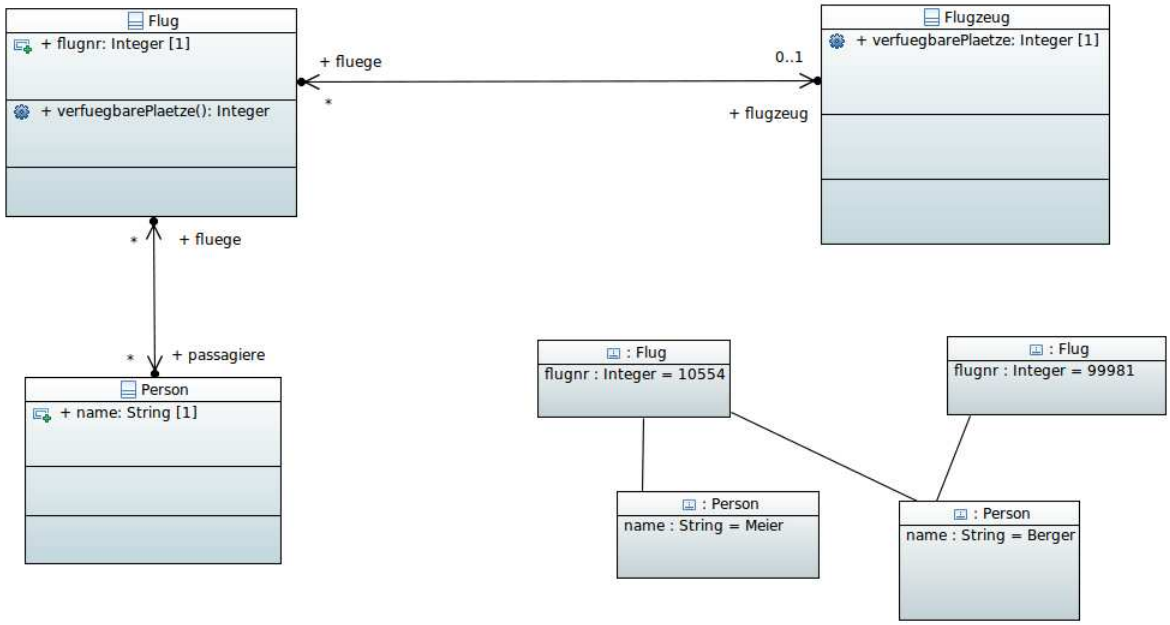
UML: how to implement Association class in Java
Java „Traditionelle“ und „neue“ Collections
Java8 Collections

C++(11) Containers library
How can I efficiently select a Standard Library container in C++11?

Umple — Syntax für Assoziationen, Vererbung, ...

Eclipse C++ Code-Generation (seit November 2014)

In Papyrus inklusive Objektdiagramm (Hinweise zur Konstruktion in Papyrus siehe folgende Seite)):



Einige (implementierungstechnische) OCL-Constraints:

```
context Model::Flug
inv konsistenteBidir1FluegeFlugzeug: flugzeug.fluege->includes(
    self)

context Model::Flugzeug
inv konsistenteBidir2FluegeFlugzeug: fluege.flugzeug->asSet() =
     Set{self}

context Model::Flug
inv konsistenteBidir1FluegePassagiere: passagiere.fluege->
    includes(self)

context Model::Person
inv konsistenteBidir2FluegePassagiere: fluege.passagiere->
    includes(self)
```

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

Types of Collection

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

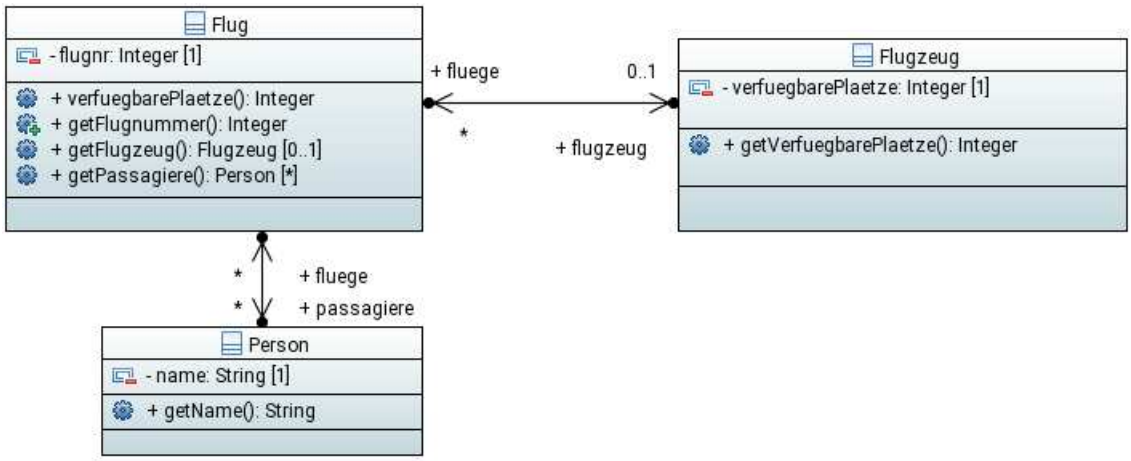
OrderedSet — eigentlich „unique Sequence“
An OCL Ordered Set is not an OCL Set but a speccial OCL Sequence

Zusammengefaßt:

```
import 'model.uml'

context Model::Flug
inv flugNrPositive: flugnr > 0

context Model::Flug::verfuegbarePlaetze(): Integer
post resultNonNegative: result >= 0

context Model::Flugzeug
inv verfuegbarePlaetzeNonNegative: verfuegbarePlaetze >= 0

context Model::Person
inv nameValid: name <> '' and name.size() >= 2

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

context Model::Flugzeug
inv ausreichendPlaetze: fluege.verfuegbarePlaetze()->sum() <=
   self.verfuegbarePlaetze

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

context Model::Flug
inv mVPNonNegative: momentanVerfuegbarePlaetze() >= 0

context Model::Flugzeug
inv mVPGeringAnzahlig: fluege.momentanVerfuegbarePlaetze()->sum
   () <= 0.05 * verfuegbarePlaetze


context Model::Flug
inv konsistenteBiDir1FluegeFlugzeug: flugzeug.fluege->includes(
   self)
context Model::Flugzeug
inv konsistenteBidir2FluegeFlugzeug: fluege.flugzeug->asSet() =
    Set{self}
```

```
context Model::Flug
inv konsistenteBidir1FluegePassagiere: passagiere.fluege->
    includes(self)
context Model::Person
inv konsistenteBidir2FluegePassagiere: fluege.passagiere->
    includes(self)
```

Oder mit Hilfe grundlegender Observatoren (Getter für private Attribute beziehungsweise Assoziationsenden):




```
24 context Model::Flug::getPassagiere(): Set(Model::Person)
25 body: passagiere
26
27 -- analog Klasse Flugzeug
28 -- analog Klasse Person
29 ----------------------------------------------------------------
30
31 -- reformulierte Constraints:
32
33 context Model::Person
34 inv nameValid: getName().size() >= 2
35
36 context Model::Flug
37 inv flugnrGueltig: getFlugnummer() > 0
38
39 context Model::Flug::verfuegbarePlaetze(): Integer
40 post konsistentesResult: not getFlugzeug().oclIsInvalid() implies result <=
41                                                    getFlugzeug().getVerfuegbarePlaetze()
42
43 context Model::Flug
44 inv nichtUeberbucht: getPassagiere()->size() <= verfuegbarePlaetze()
45
```

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

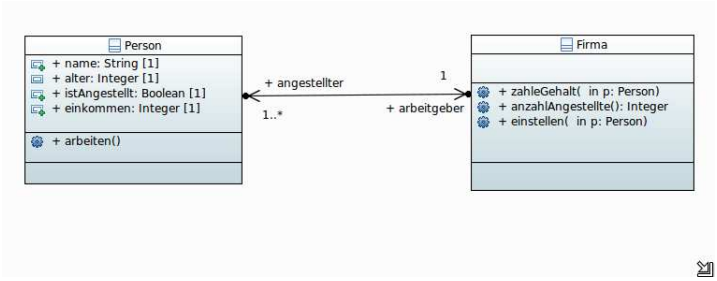


Abbildung 2.3.: Modell Person-Firma

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

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

context Model::Person::arbeiten(): OclVoid
  pre geschaeftsfaehig:    istAngestellt and alter >= 14

context Model::Firma::zahleGehalt(p: Person): OclVoid
  pre istAngestellt:    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 >= 14) 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 einkommenKonsistent:    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) **or** 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 und (impliziten) Collects

- 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 Collection-Objekts liefert einen Bag, falls die letzte Navigation zu einer *nicht* als `{ ordered }` gekennzeichneten Rolle führt oder die Vielfachheit 1 hat.

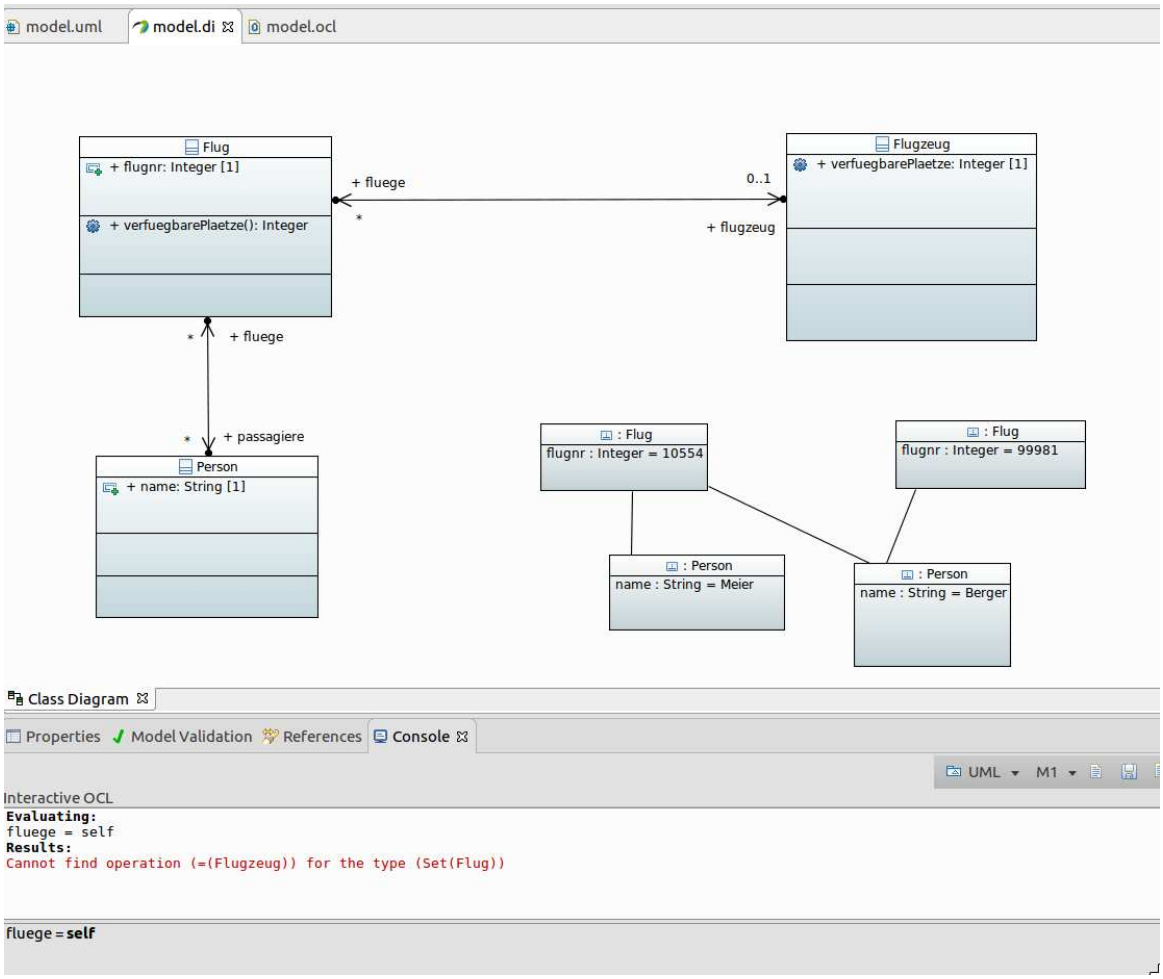
- Ist die letzte Rolle bei der Weiternavigation nach zwischenzeitlichem Erreichen eines Collectionobjekts als `{ ordered }` gekennzeichnet, so ist das Navigationsergebnis eine Sequence.

- 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

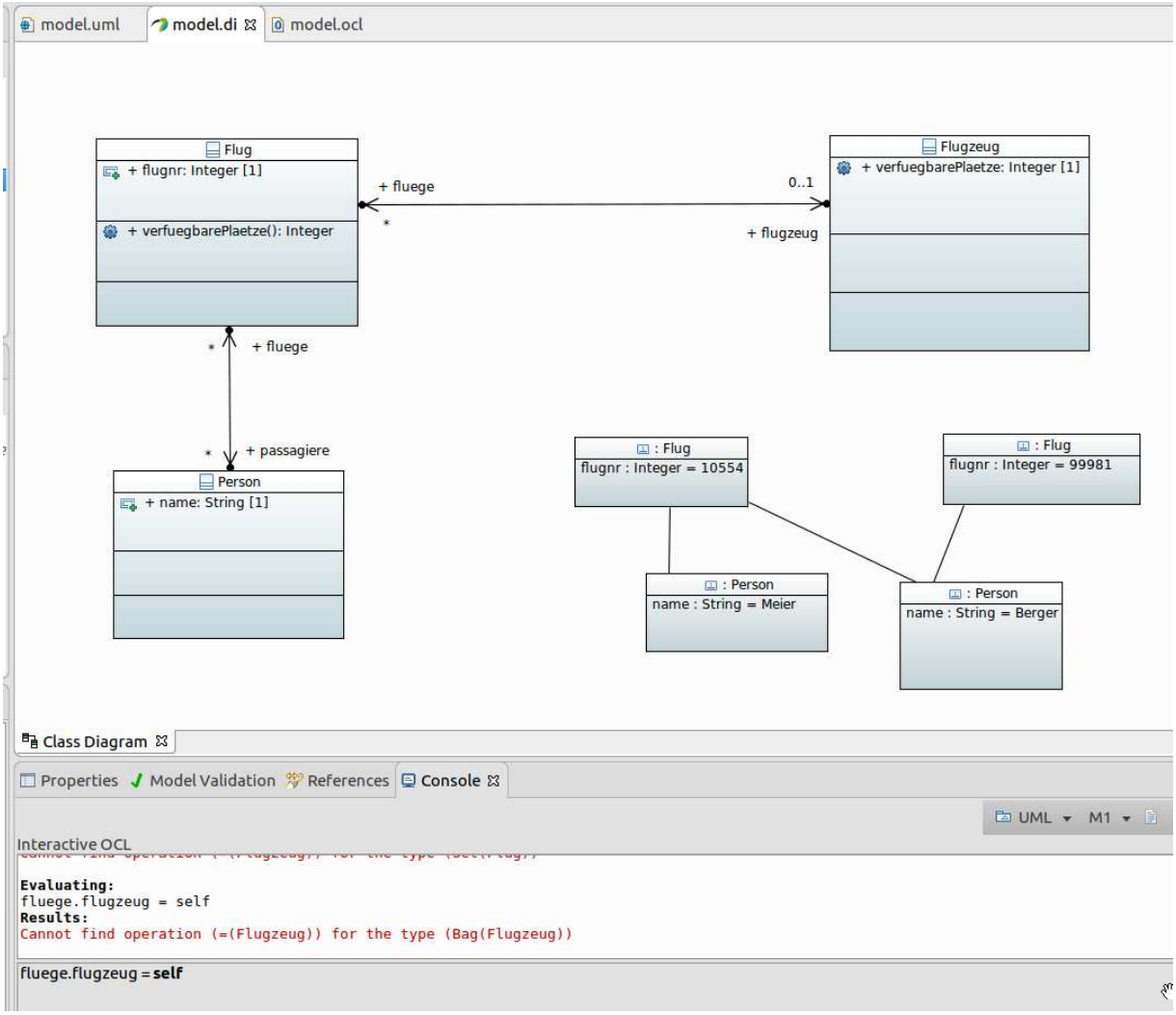
```
self.angestellter->collect(alter)
self.angestellter->collect(p | p.alter)
self.angestellter->collect(p : Person | p.alter)
```

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-Workaround

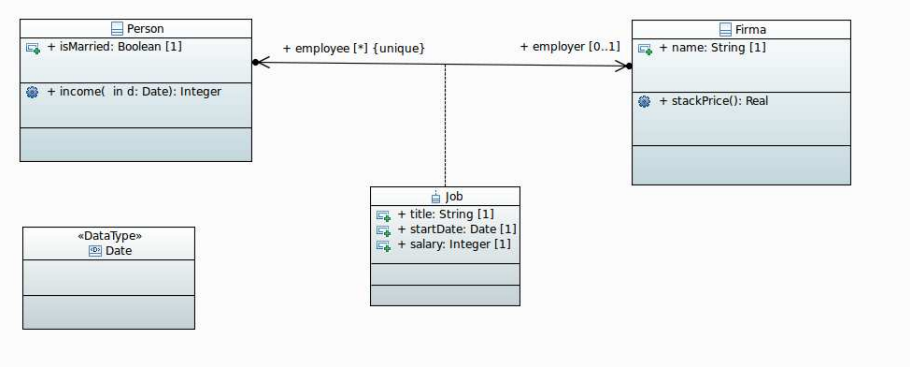


Abbildung 2.4.: Assoziationsklasse Job

wird als Workaround implementiert durch:

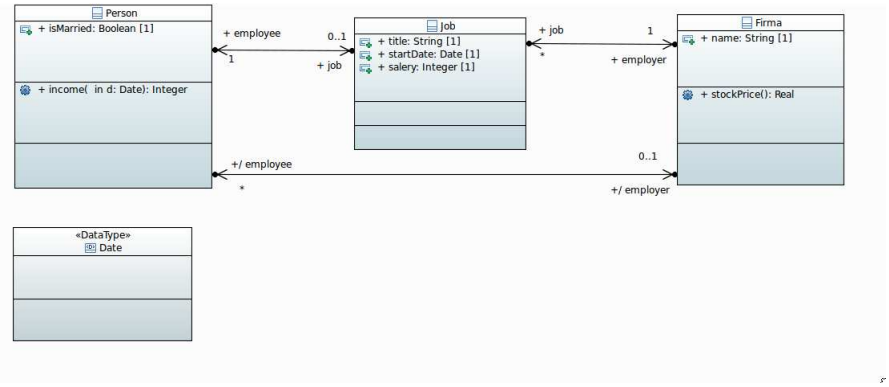


Abbildung 2.5.: Assoziationklasse im Workaround

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

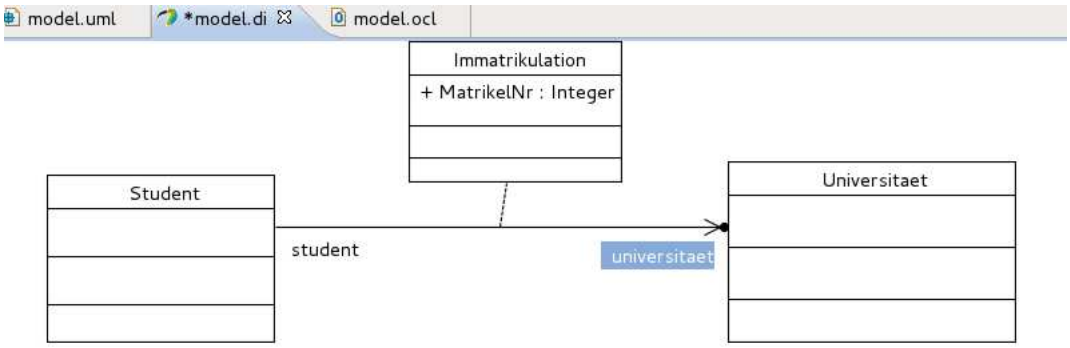
```
context Model::Firma
def: employee : Set(Model::Person) = job.employee->asSet()

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

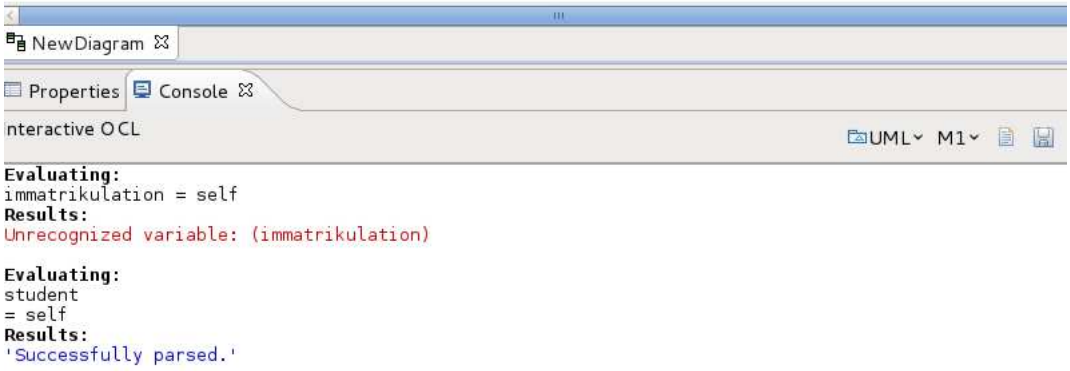
einführen.

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:

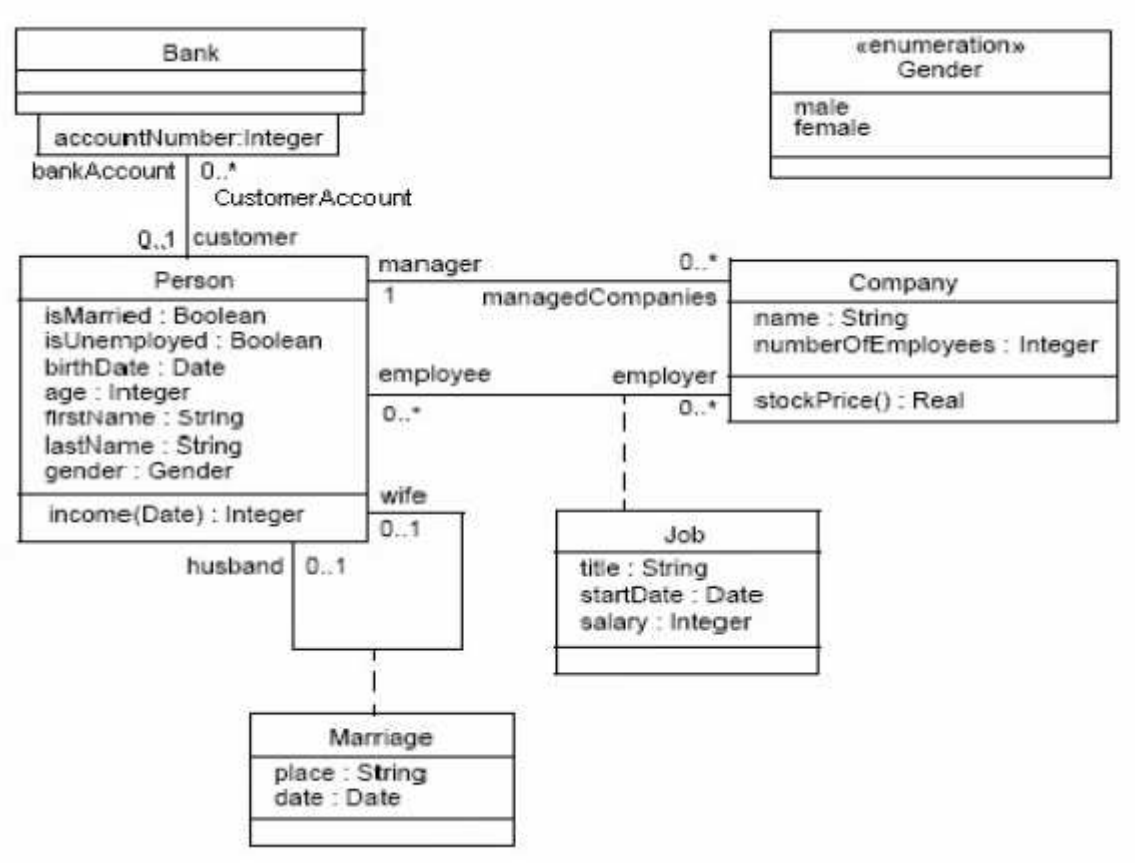




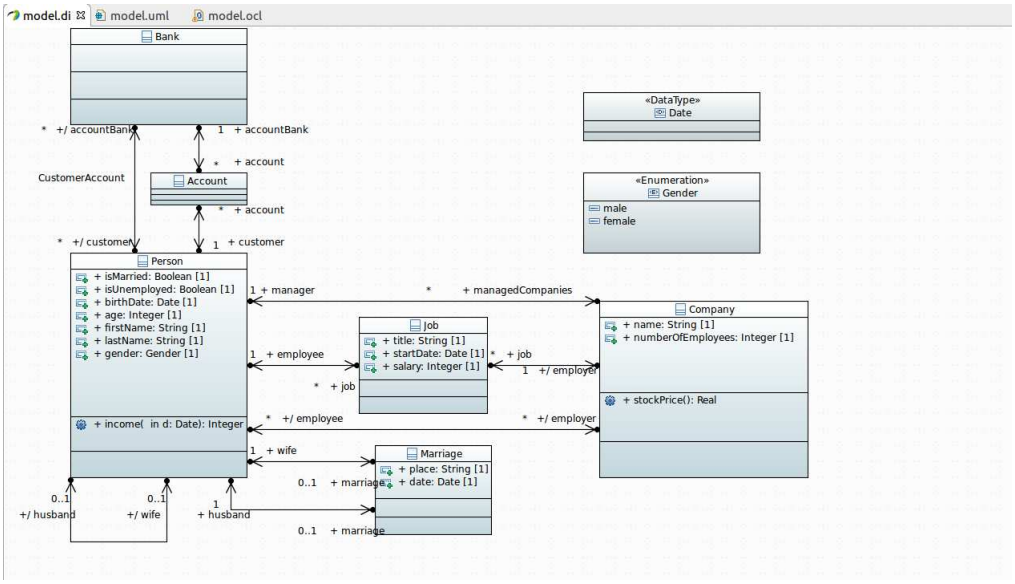
Man hat die Rolle `student` durch das Kontextmenü `Association End`, `owned by the classifier` noch umzustellen, um die übliche symmetrische Sichtweise zu erhalten.

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

Das Workaround angewandt auf das Beispiel-Modell des OCL-Manuals



sieht dann etwa folgendermaßen aus



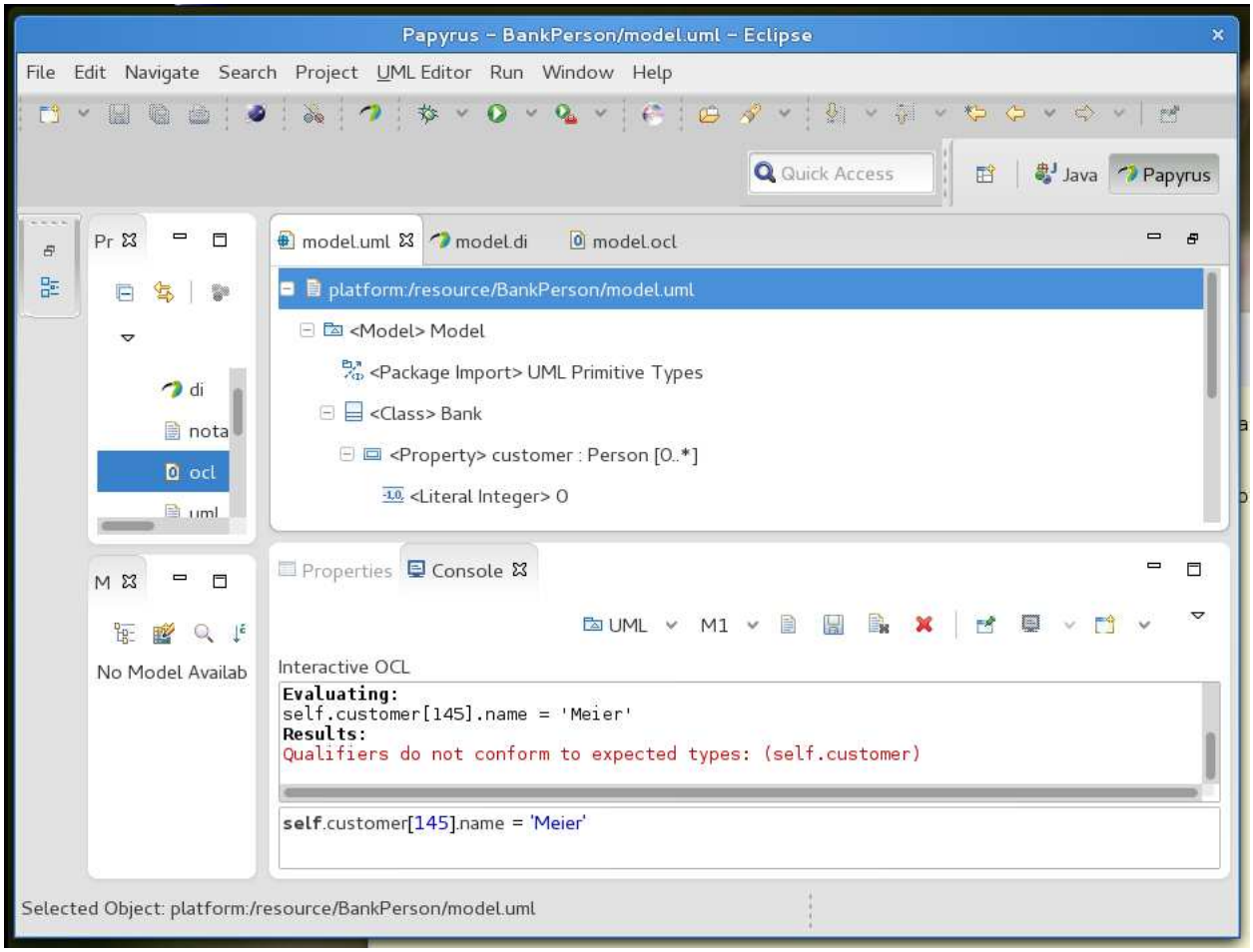
und sollte mit folgender OCL-Detaillierung ausgestattet werden:

```
1  import 'model.uml'
2
3  context Model::Person::wife: Model::Person
4  derive: if Marriage->notEmpty()
5          then
6            if gender = Gender::male
7            then Marriage.wife
8            else null
9            endif
10         else null
11         endif
12
13 context Model::Person::husband: Model::Person
14 derive: if Marriage->notEmpty()
15         then
16           if gender = Gender::female
17           then Marriage.husband
18           else null
19           endif
20         else null
21         endif
22
23 context Model::Person::employer: Set(Model::Company)
24 derive: Job.employer->asSet()
25
26 context Model::Company::employee: Set(Model::Person)
27 derive: Job.employee->asSet()
28
29 context Model::Person::accountBank: Set(Model::Bank)
30 derive: Account.accountBank->asSet()
31
32 context Model::Bank::customer: Set(Model::Person)
33 derive: Account.customer->asSet()
```

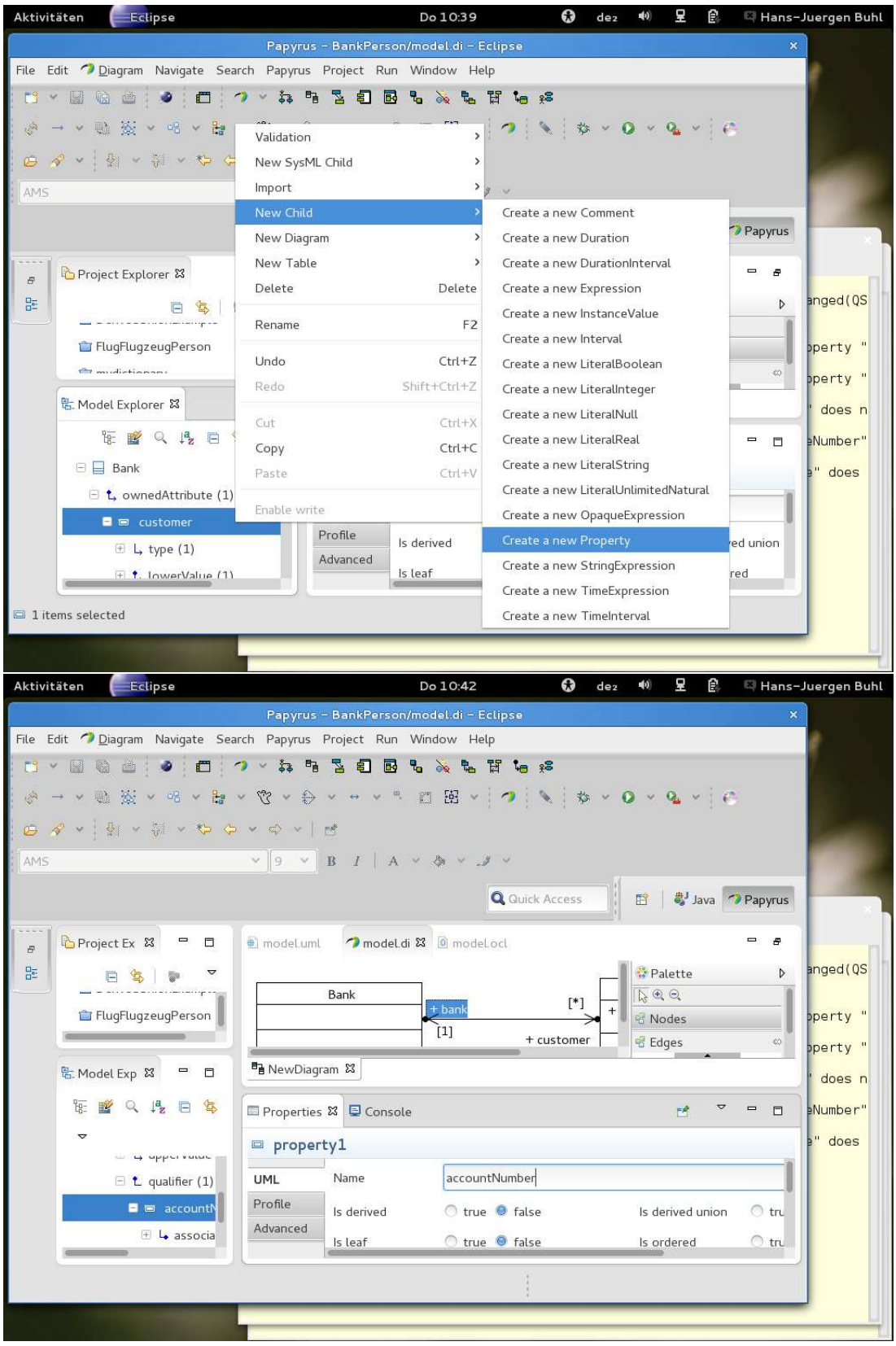
2.5. Workaround für qualifizierte Assoziationen

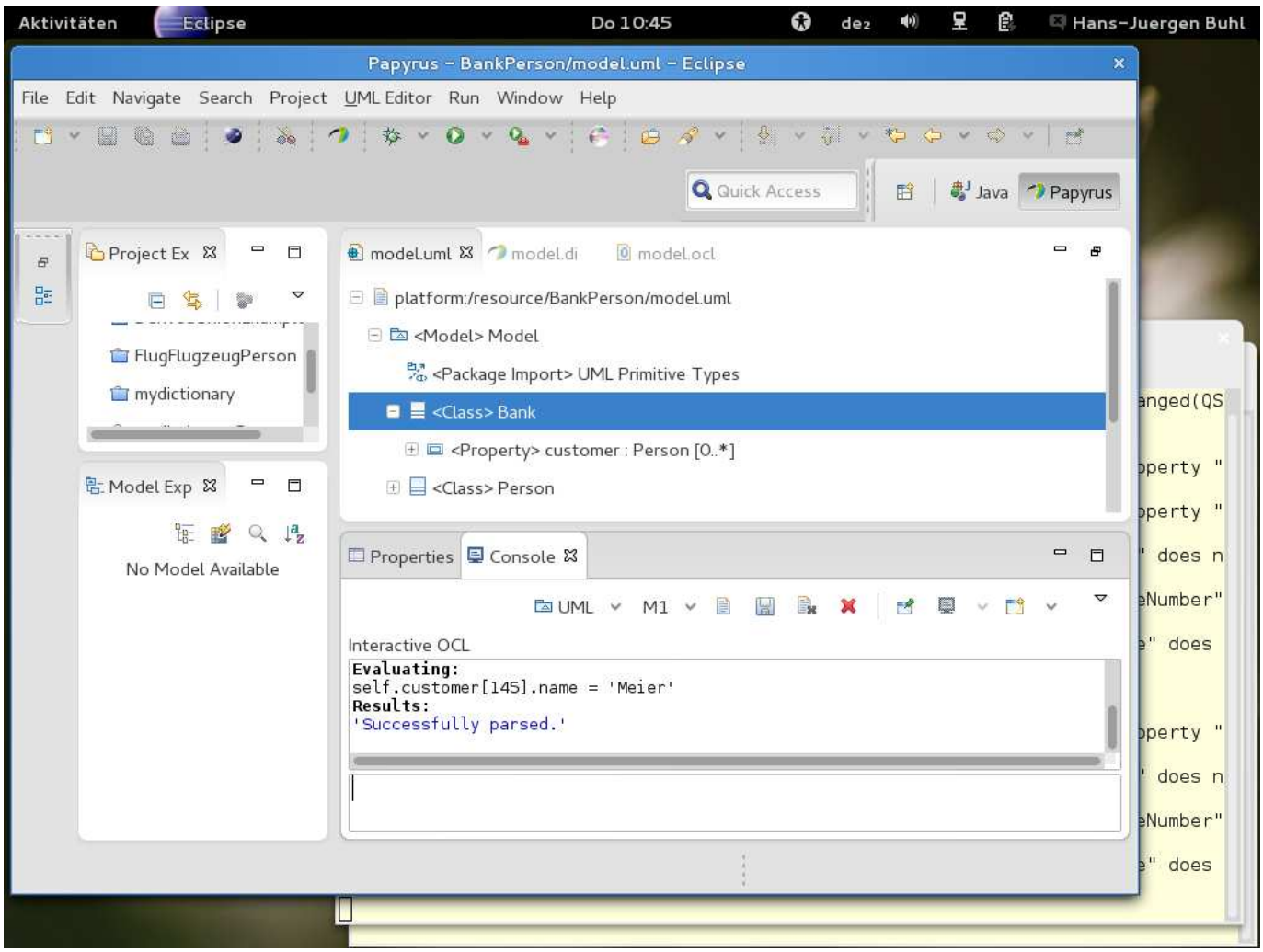
Abbildung 2.6.: qualifizierte Assoziation

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
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
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. Methoden für die unordered 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 bei 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. Collection(T)-Methoden

	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	-	-	-
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

Neben `=, <>` existieren für alle vier Collection-Ausprägungen die `Boolean`-wertigen Methoden `includes(), excludes(), includesAll(), excludesAll(), isEmpty()` und `notEmpty()`, die `Integer`-wertigen Methoden `size(), count(o: T)`, die `T`-wertigen Methoden `max(), min(), sum()` sowie die Collection-wertigen Methoden `product(), selectByKind(), asXXX(), flatten()` (Abschnitt 11.7 des OCL 2.4 Standards).
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!

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: Enumeration Gender

```
/*    Generated by Together    */

# ifndef GENDER_H
# define GENDER_H

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

Listing 2.3: Klasse Person

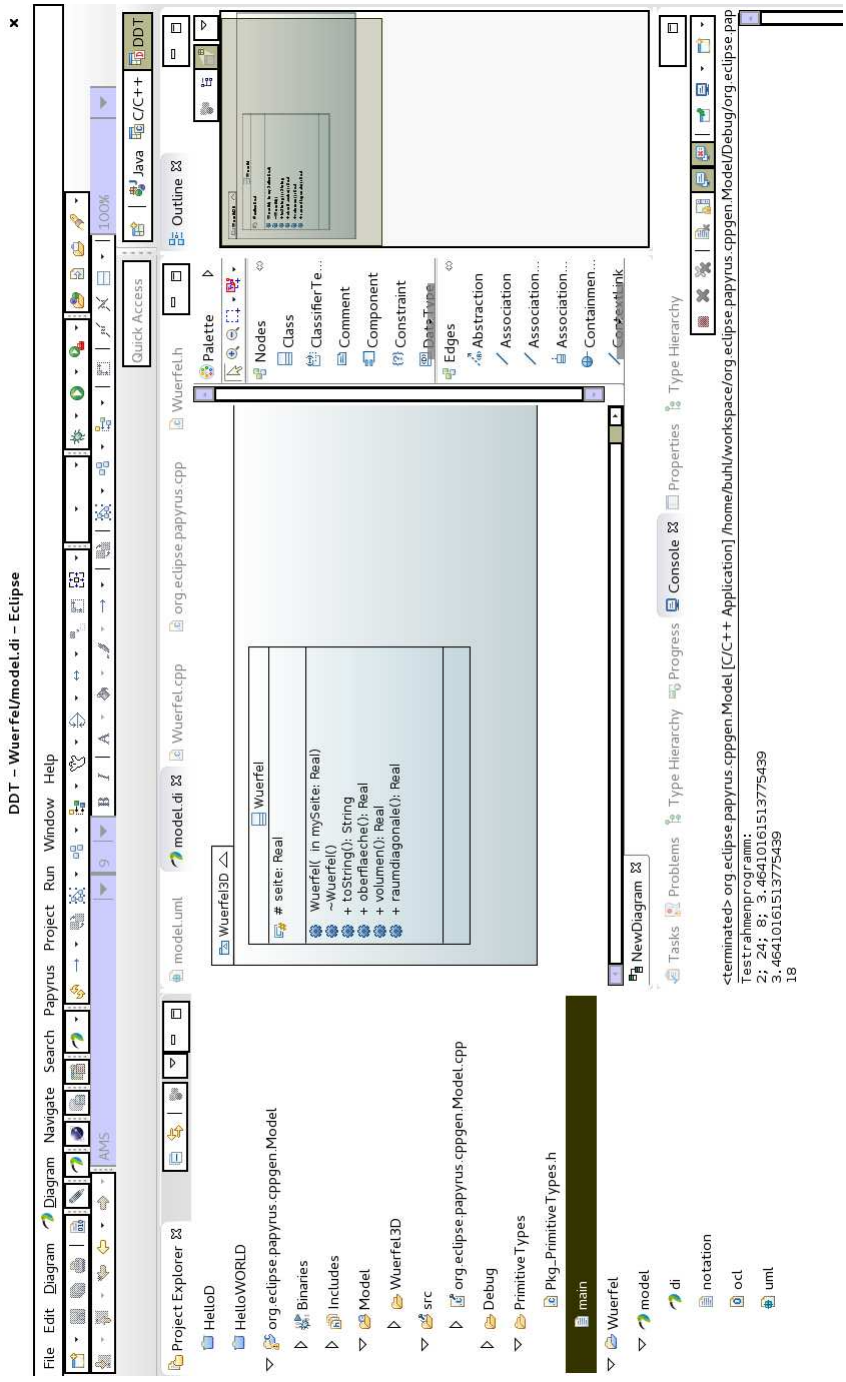
```
/*    Generated by Together    */

# ifndef PERSON_H
# define PERSON_H
```

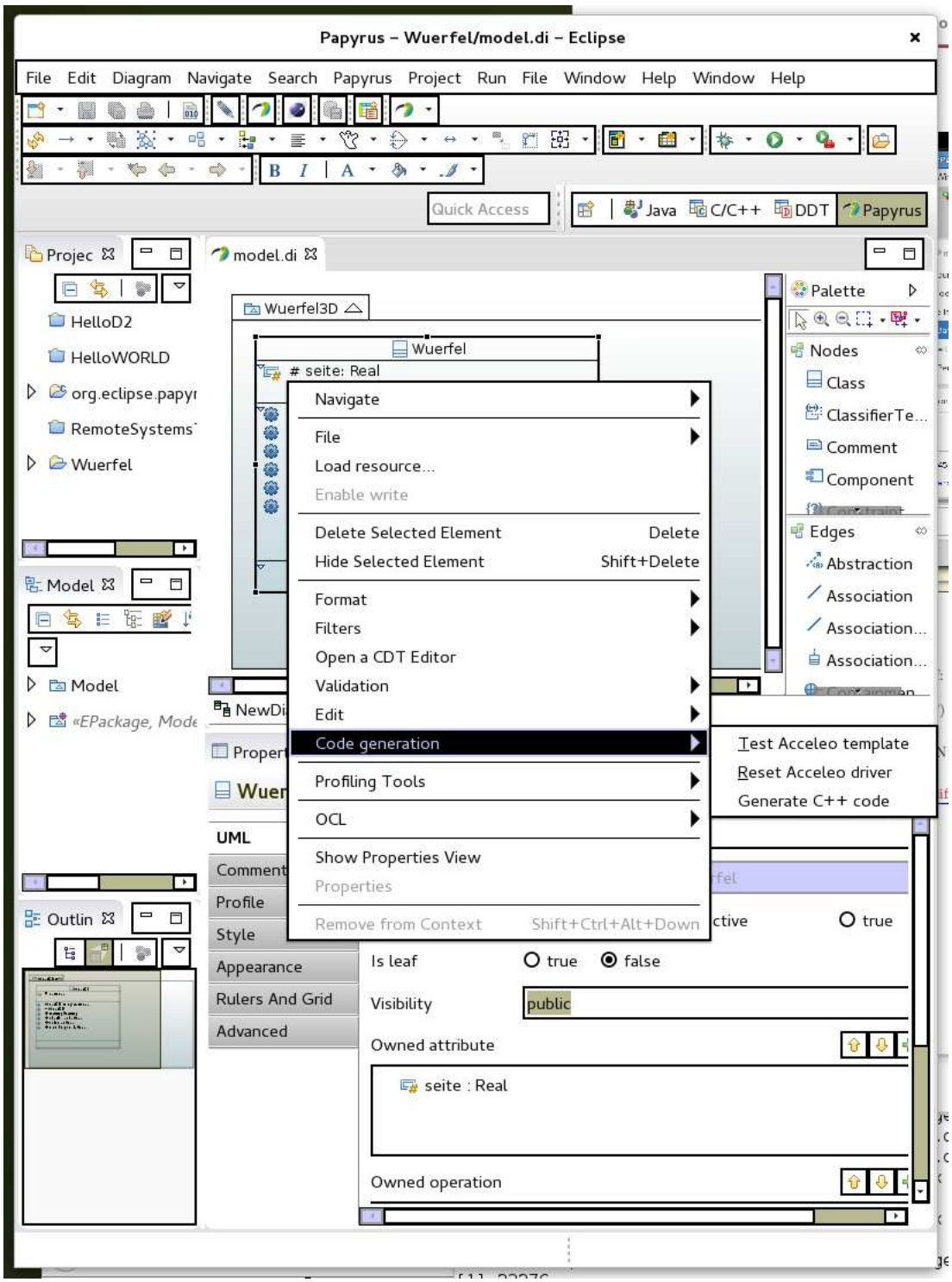
```
class Bank;
class Company;

class Person {
public:

    Integer income(Date);

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

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

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

    /**
```

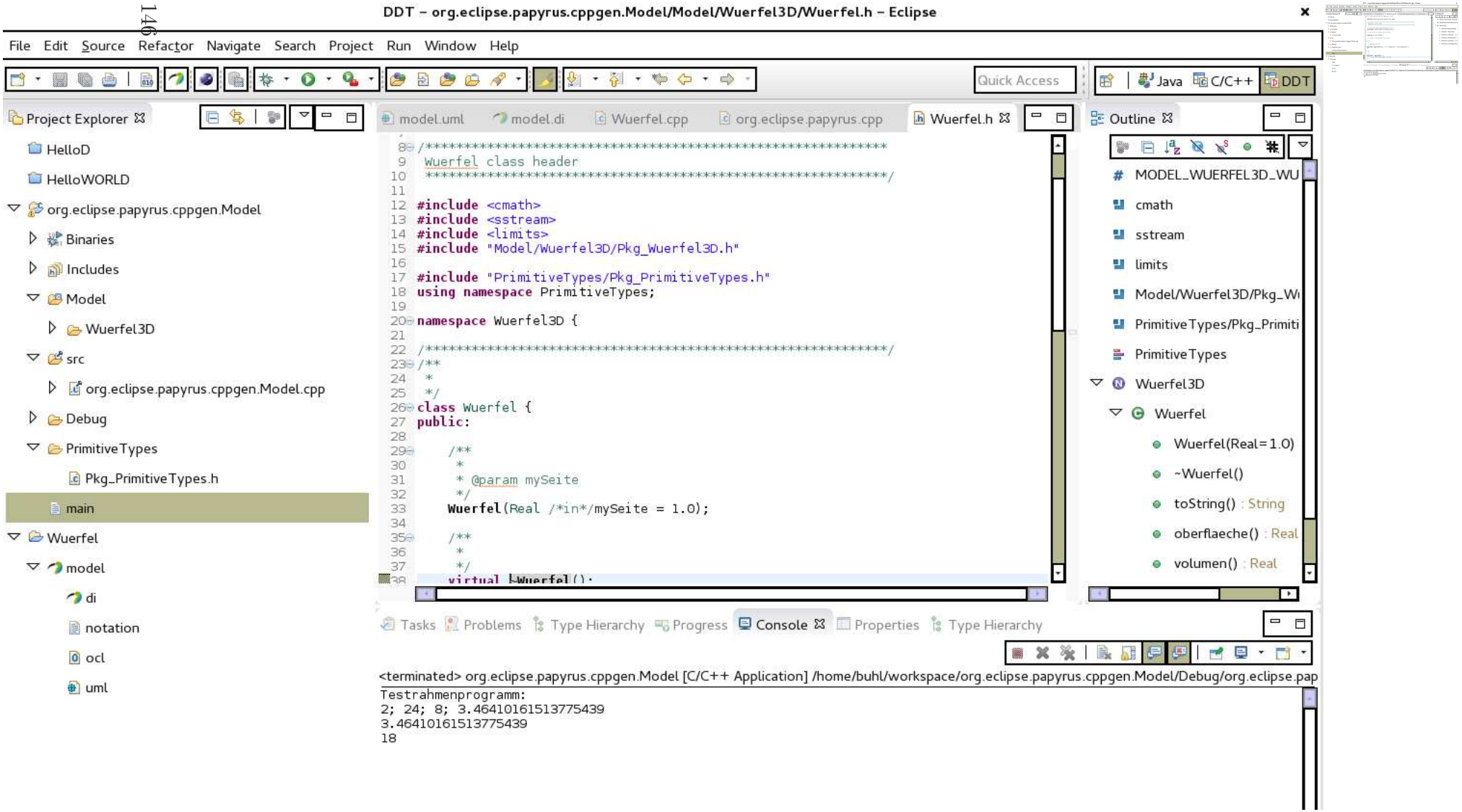
```
     * @bidirectional Person#wife
     * @clientCardinality 0..1
     * @supplierQualifier husband
     * @supplierQualifier wife
     * @associationAsClass Marriage
     */
Person * husband;
/** bidirectional */
Person * wife;

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

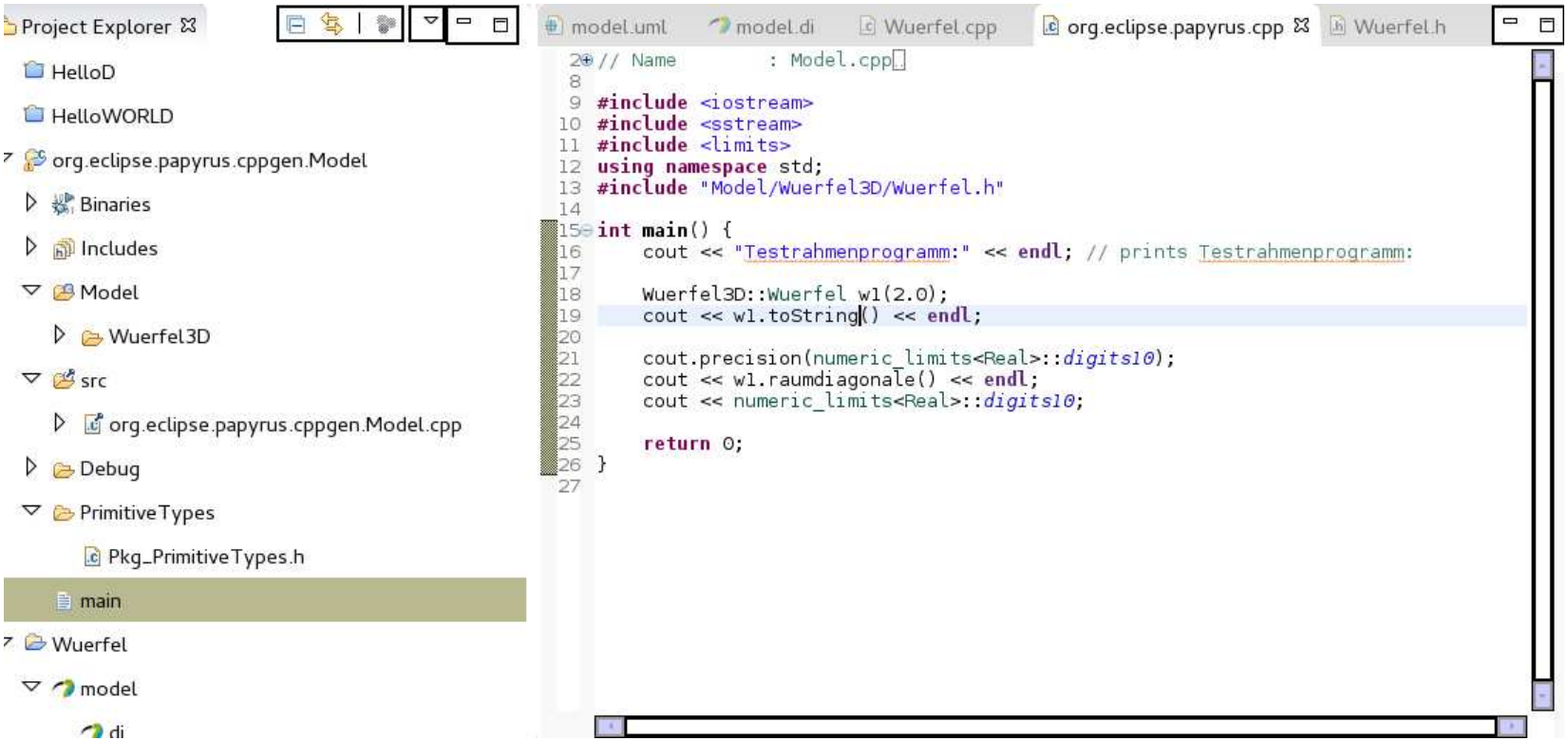
2.10. CDT Papyrus-Codeerzeugung

Seit Herbst 2014 funktioniert auch `eclipse-papyrus` (zur Erzeugung von UML-Diagrammen) mit C++-Quellcode-Erzerzeugung (zur Installation von `eclipse-papyruso3` siehe Anhang B):

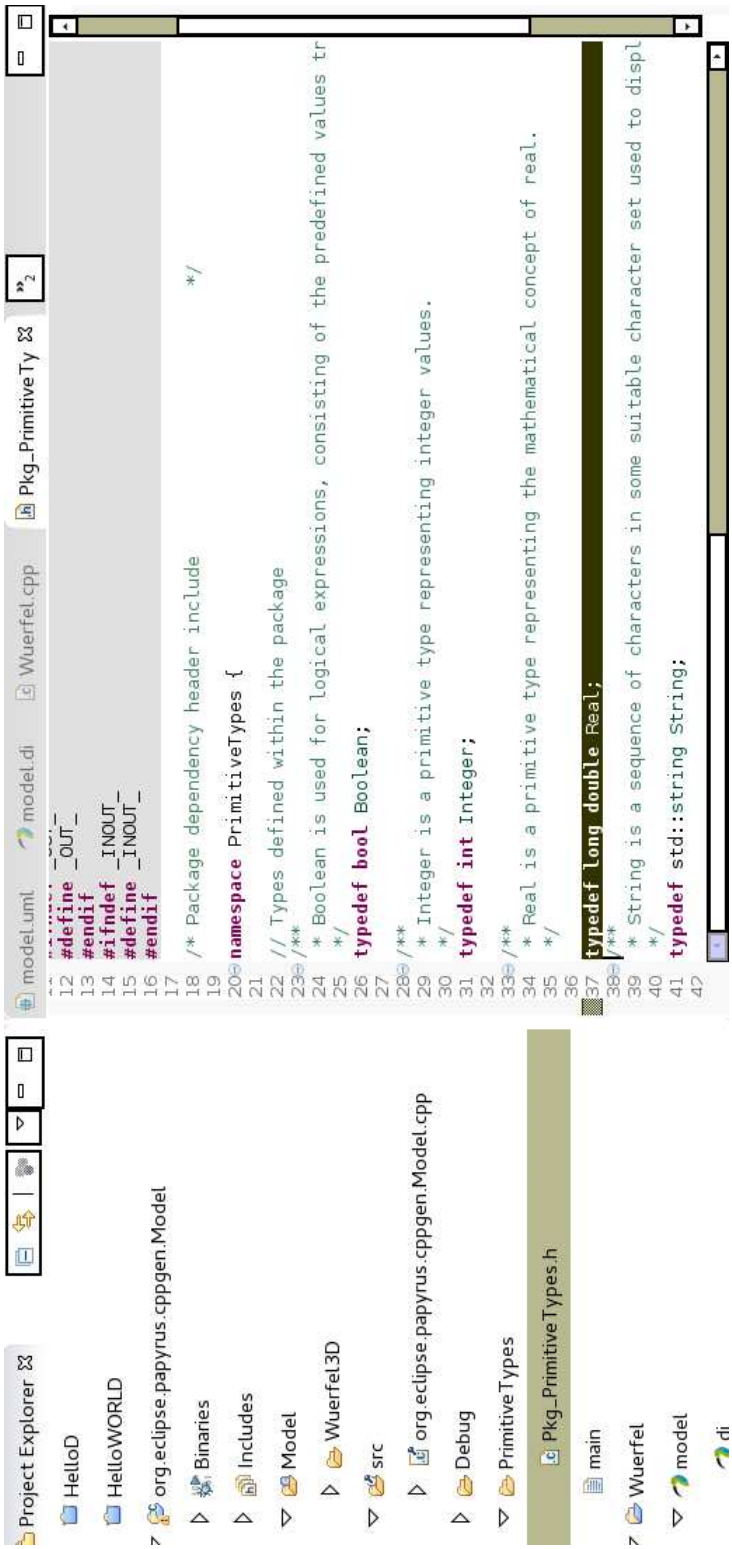


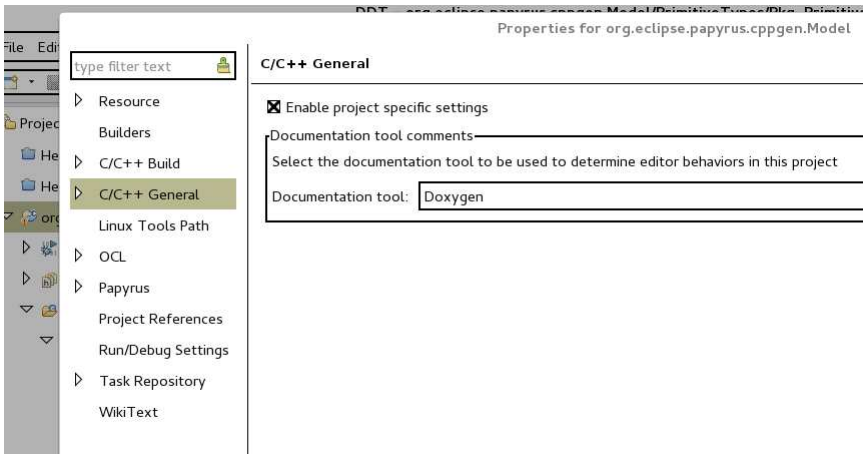

Papyrus – Wuerfel/model.di – Eclipse
File Edit Diagram Navigate Search Papyrus Project Run File Window Help Window Help
Quick Access
Java C/C++ DDT Papyrus
Projec
HelloD2
HelloWORLD
org.eclipse.papyr
RemoteSystems
Wuerfel
Model
Model
«EPackage, Mode
Outlin
model.di
Wuerfel3D
Wuerfel
seite: Real
Navigate
File
Load resource...
Enable write
Delete Selected Element Delete
Hide Selected Element Shift+Delete
Format
Filters
Open a CDT Editor
Validation
Edit
Code generation
Profiling Tools
OCL
Show Properties View
Properties
Remove from Context Shift+Ctrl+Alt+Down
Test Acceleo template
Reset Acceleo driver
Generate C++ code
Palette
Nodes
Class
ClassifierTe...
Comment
Component
Edges
Abstraction
Association
Association...
Association...
UML
Comment
Profile
Style
Appearance
Rulers And Grid
Advanced
true
Is leaf true false
Visibility public
Owned attribute
seite : Real
Owned operation


DDT – org.eclipse.papyrus.cppgen.Model/Model/Wuerfel3D/Wuerfel.h – Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java C/C++ DDT
Project Explorer
HelloD
HelloWORLD
org.eclipse.papyrus.cppgen.Model
Binaries
Includes
Model
Wuerfel3D
src
org.eclipse.papyrus.cppgen.Model.cpp
Debug
PrimitiveTypes
Pkg_PrimitiveTypes.h
main
Wuerfel
model
di
notation
ocl
uml
model.uml
model.di
Wuerfel.cpp
org.eclipse.papyrus.cpp
Wuerfel.h
/**********
Wuerfel class header
**********/
#include <cmath>
#include <sstream>
#include <limits>
#include "Model/Wuerfel3D/Pkg_Wuerfel3D.h"
#include "PrimitiveTypes/Pkg_PrimitiveTypes.h"
using namespace PrimitiveTypes;
namespace Wuerfel3D {
/**********/
/**
*
*/
class Wuerfel {
public:
/**
* @param mySeite
*/
Wuerfel(Real /*in*/mySeite = 1.0);
/**
*
*/
virtual ~Wuerfel();
Outline
MODEL_WUERFEL3D_WU
cmath
sstream
limits
Model/Wuerfel3D/Pkg_W
PrimitiveTypes/Pkg_Primiti
PrimitiveTypes
Wuerfel3D
Wuerfel
Wuerfel(Real=1.0)
~Wuerfel()
toString() : String
oberflaeche() : Real
volumen() : Real
Tasks Problems Type Hierarchy Progress Console Properties Type Hierarchy
<terminated> org.eclipse.papyrus.cppgen.Model [C/C++ Application] /home/buhl/workspace/org.eclipse.papyrus.cppgen.Model/Debug/org.eclipse.pap
Testrahmenprogramm:
2; 24; 8; 3.46410161513775439
3.46410161513775439
18


Project Explorer
HelloD
HelloWORLD
org.eclipse.papyrus.cppgen.Model
Binaries
Includes
Model
Wuerfel3D
src
org.eclipse.papyrus.cppgen.Model.cpp
Debug
PrimitiveTypes
Pkg_PrimitiveTypes.h
main
Wuerfel
model
di
model.uml
model.di
Wuerfel.cpp
org.eclipse.papyrus.cpp
Wuerfel.h
// Name : Model.cpp
#include <iostream>
#include <sstream>
#include <limits>
using namespace std;
#include "Model/Wuerfel3D/Wuerfel.h"
int main() {
cout << "Testrahmenprogramm:" << endl; // prints Testrahmenprogramm:
Wuerfel3D::Wuerfel w1(2.0);
cout << w1.toString() << endl;
cout.precision(numeric_limits<Real>::digits10);
cout << w1.raumdiagonale() << endl;
cout << numeric_limits<Real>::digits10;
return 0;
}

Die Datei `Pkg_Primitive Typed.h` muss ein bisschen modifiziert werden (`Real` sollte als Alias auf `double` oder `long double` gelegt werden, `String` statt auf `char*` besser auf `std::string`, ...)





UML2 to C++-Quellcode (seit November 2014)

2.11. Fallstudie: Person/Haus/Hypothek/Verpfändung

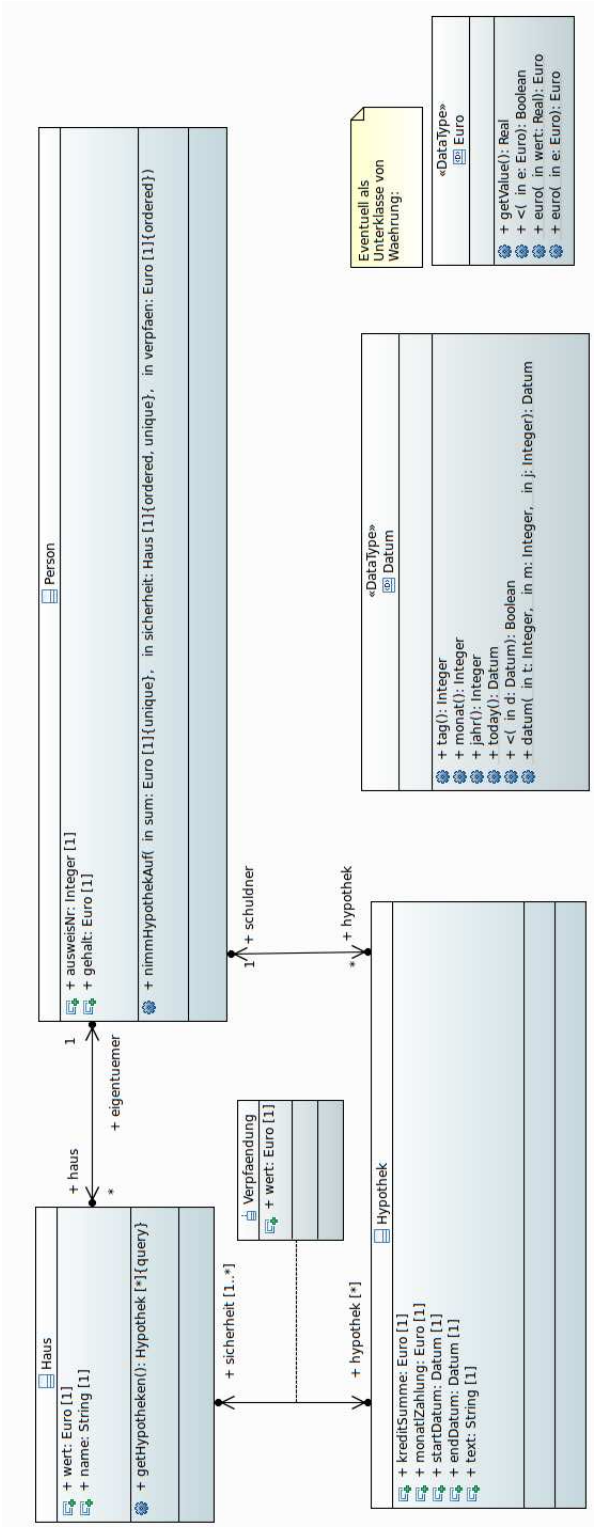
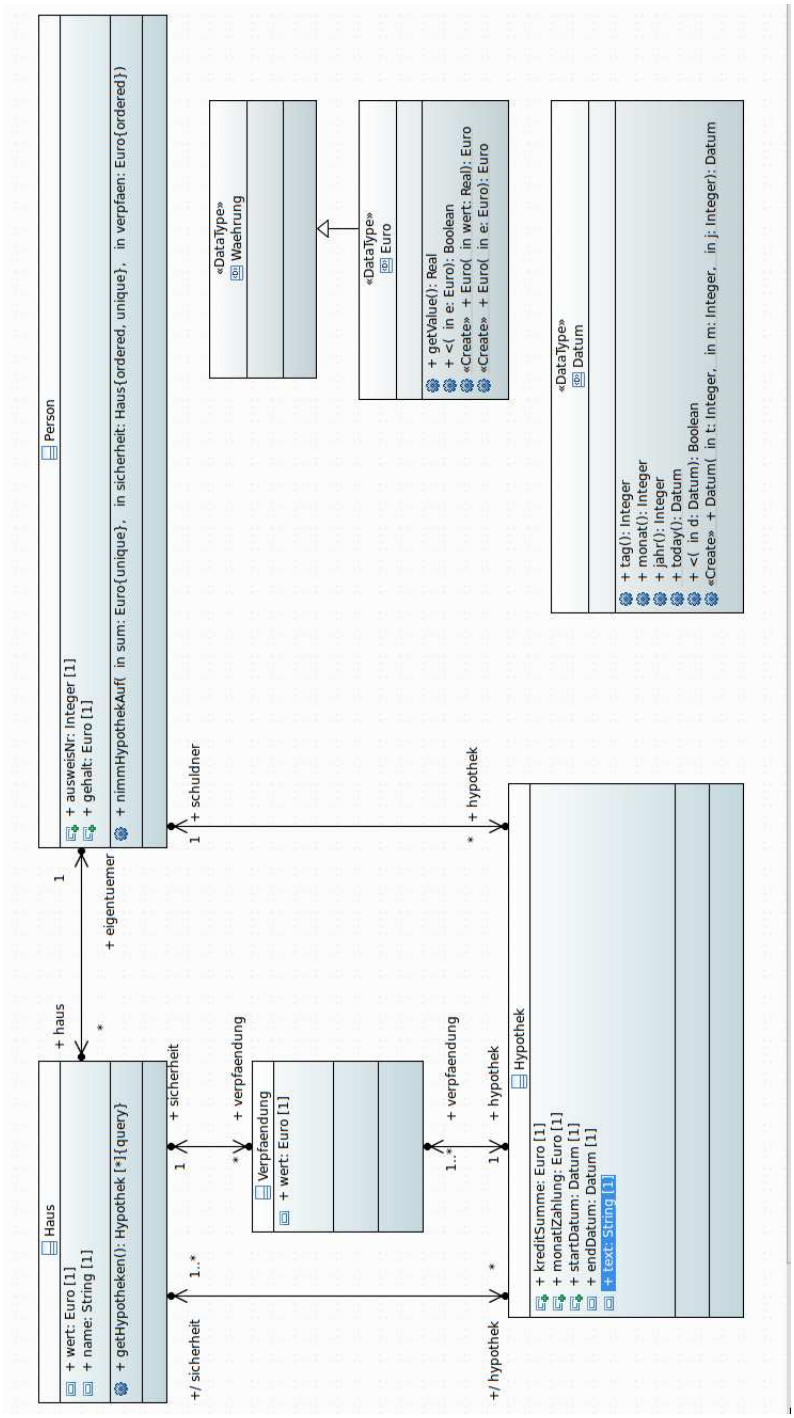


Abbildung 2.7.: Klassendiagramm Hypothek


Person
+ ausweisNr: Integer [1]
+ gehalt: Euro [1]
+ nimmHypothekAuf(in sum: Euro{unique}, in sicherheit: Haus{ordered, unique}, in verpfaen: Euro{ordered})
«DataType»
Waehrung
«DataType»
Euro
+ getValue(): Real
+ <(in e: Euro): Boolean
«Create» + Euro(in wert: Real): Euro
«Create» + Euro(in e: Euro): Euro
«DataType»
Datum
+ tag(): Integer
+ monat(): Integer
+ jahr(): Integer
+ today(): Datum
+ <(in d: Datum): Boolean
«Create» + Datum(in t: Integer, in m: Integer, in j: Integer): Datum
Haus
+ wert: Euro [1]
+ name: String [1]
+ getHypotheken(): Hypothek [*]{query}
Verpfaendung
+ wert: Euro [1]
Hypothek
+ kreditSumme: Euro [1]
+ monatlZahlung: Euro [1]
+ startDatum: Datum [1]
+ endDatum: Datum [1]
+ text: String [1]
+ haus
1
+ eigentuemer
*
+ schuldner
1
+ hypothek
*
+ sicherheit
1
+ verfpaendung
*
+ verfpaendung
1..*
+ hypothek
1
+/ sicherheit
1..*
+/ hypothek
*

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

context Model::Hypothek
inv startVorEnde: startDatum < endDatum

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

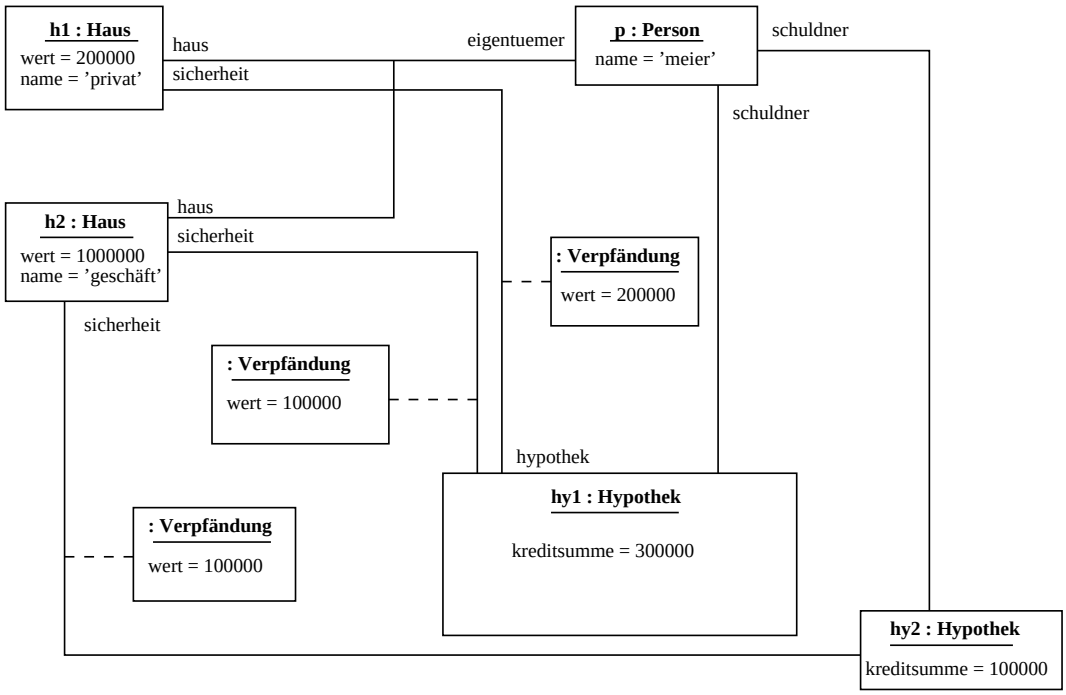


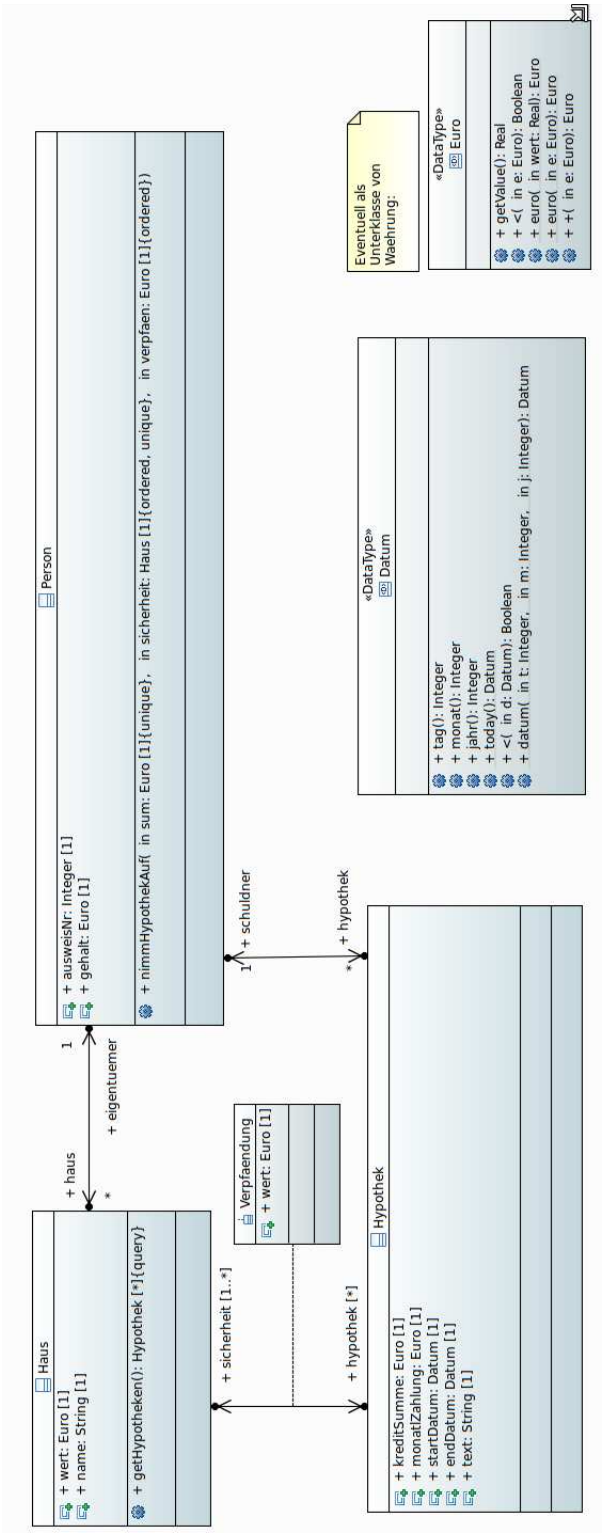
Abbildung 2.8.: Hypothek mit zwei Häusern

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 wertNichtNegativ: Euro::euro(0.0) < wert
inv nameGueltig: name.size()>2

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

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

context Model::Hypothek
inv startVorEnde: startDatum < endDatum

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

-- ...
```

In Papyrus nach Ergänzung des query-Operators `+` in der Klasse `Euro`:



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

-- constructor Euro
context Model::Euro:euro(wert: Real): Model::Euro
pre: wert <> null and wert <> invalid
post istKonstruktor: result.oclIsNew()
post wertRichtigKonstruiert: 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 = Model::Euro:euro(self.getValue() + e.getValue())
```

Beachte für die Operatoren-Codeverträge von `Euro::<, Euro::+` Abschnitt „*7.4.9 Use of Infix Operators*“ des OCL-Manuals:

7.4.9 Use of Infix Operators

The use of infix operators is allowed in OCL. The operators '+', '-', '*', '/', '=', '<>', '<', '>', '<=', '>=' are used as infix operators. If a type defines one of those operators with the correct signature, they will be used as infix operators. The expression:

```
a + b
```

is equal to the expression:

```
a._'+'(b)
```

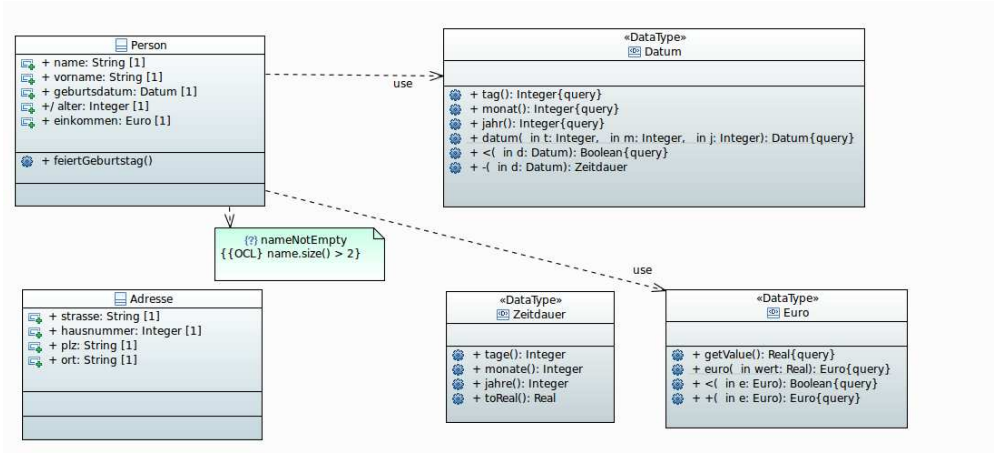
that is, invoking the "+" operation on a with b as the parameter to the operation.

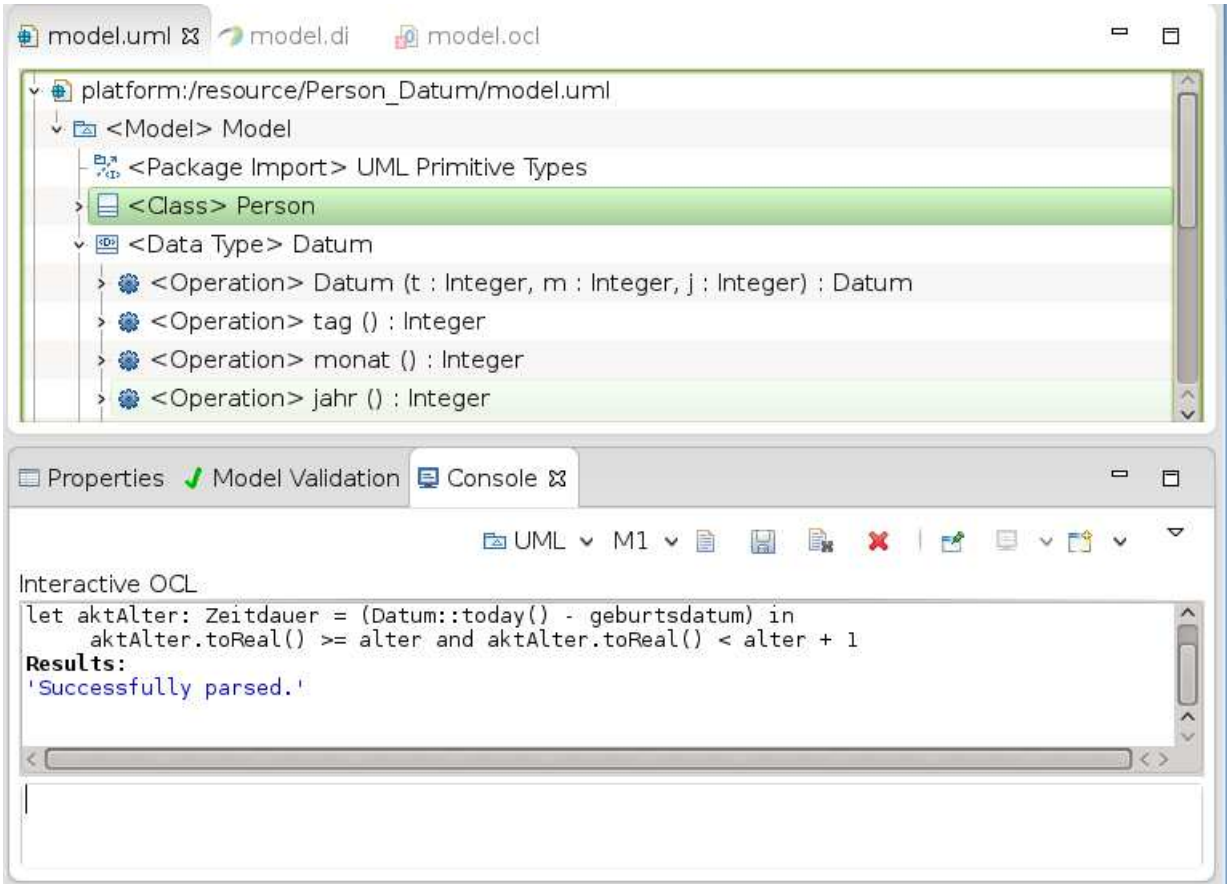
The infix operators defined for a type must have exactly one parameter. For the infix operators '<,' '>,' '<=,' '>=,' '<>,' 'and,' 'or,' and 'xor' the return type must be Boolean.

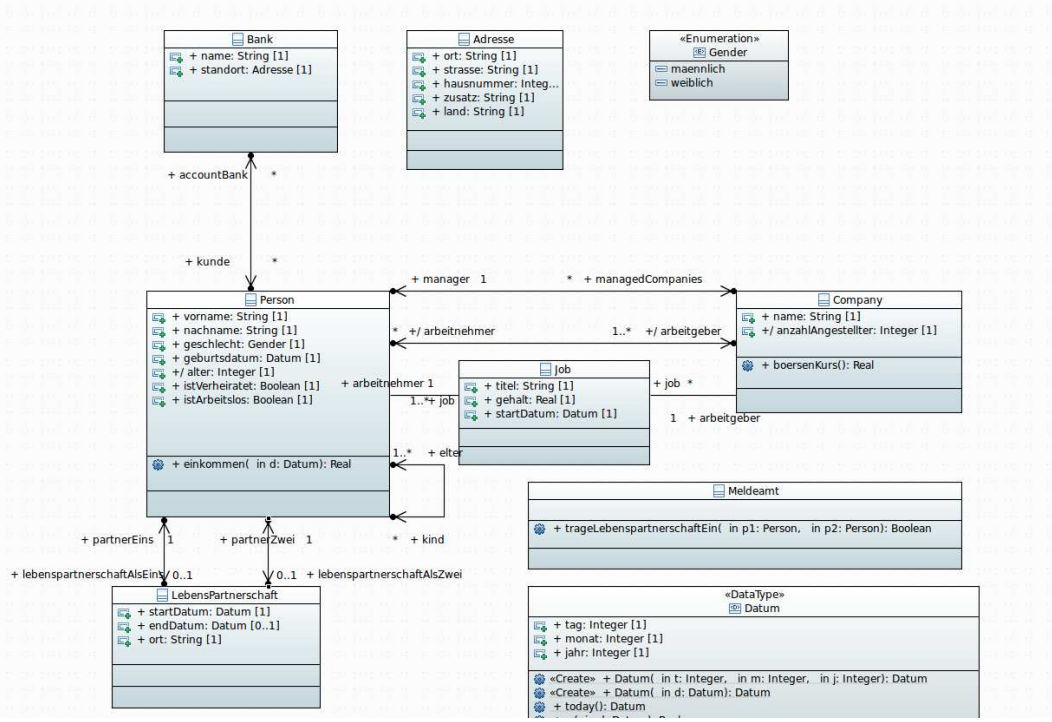
```
-- Constraints
context Model::Haus::getHypotheken(): Set(Model::Hypothek)
body: hypothek

context Model::Hypothek
inv validKreditSumme: Euro::euro(0.0) < kreditSumme
inv kreditSummeAusreichendAbgesichert: kreditSumme = self.
   Verpfaendung.wert->sum()
inv textValid: text <> ''

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

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

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

Das redundante Attribut `alter`:






Bank
+ name: String [1]
+ standort: Adresse [1]
Adresse
+ ort: String [1]
+ strasse: String [1]
+ hausnummer: Integ...
+ zusatz: String [1]
+ land: String [1]
«Enumeration»
Gender
maennlich
weiblich
+ accountBank
*
+ kunde
*
+ manager 1
* + managedCompanies
Person
+ vorname: String [1]
+ nachname: String [1]
+ geschlecht: Gender [1]
+ geburtsdatum: Datum [1]
+/ alter: Integer [1]
+ istVerheiratet: Boolean [1]
+ istArbeitslos: Boolean [1]
+ einkommen(in d: Datum): Real
* +/ arbeitnehmer
1..* +/ arbeitgeber
Company
+ name: String [1]
+/ anzahlAngestellter: Integer [1]
+ boersenKurs(): Real
Job
+ titel: String [1]
+ gehalt: Real [1]
+ startDatum: Datum [1]
+ arbeitnehmer 1
1..*+ job
+ job *
1 + arbeitgeber
1..* + elter
Meldeamt
+ trageLebenspartnerschaftEin(in p1: Person, in p2: Person): Boolean
+ partnerEins 1
+ partnerZwei 1
* + kind
+ lebenspartnerschaftAlsEins 0..1
0..1 + lebenspartnerschaftAlsZwei
LebensPartnerschaft
+ startDatum: Datum [1]
+ endDatum: Datum [0..1]
+ ort: String [1]
«DataType»
Datum
+ tag: Integer [1]
+ monat: Integer [1]
+ jahr: Integer [1]
«Create» + Datum(in t: Integer, in m: Integer, in j: Integer): Datum
«Create» + Datum(in d: Datum): Datum
+ today(): Datum

```
import 'model.uml'

--

context Model::Datum
inv tagGueltig:   1 <= tag and tag <= 31
inv monatGueltig: 1 <= monat and monat <= 12
inv jahGueltig:   1820 <= jahr and jahr <= 2500

context Model::Datum::Datum(t: Integer, m: Integer, j: Integer): Model::Datum
pre t_v: t <> null and t <> invalid
pre m_v: m <> null and m <> invalid
pre j_v: j <> null and j <> invalid
pre t: 1 <= t and t <= 31
pre m: 1 <= m and m <= 12
pre j: 1820 <= j and j <= 2500
post new: result.oclIsNew()
post resultVerified: result.tag   = t and
                     result.monat = m and
                     result.jahr  = j

context Model::Datum::Datum(d: Model::Datum): Model::Datum
pre d_v: d <> null and d <> invalid
post new: result.oclIsNew()
post resultVerified:  result.tag     = d.tag and
                      result.monat   = d.monat   and
                      result.jahr    = d.jahr

context Model::Datum::destructDatum(): OclVoid
-- pre: ...
-- post: ...


-- ...
```

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 istKonstruktor: result.oclIsNew()
post ergebnisTag: result.tag() = t
post ergebnisMonat: result.monat() = m
post ergebnisJahr: result.jahr() = j
```

Listing 2.6: OCL-Constraints Person

```
context Model::Person

inv nichtNegativ: alter >= 0
inv kleinerObergrenze: alter < 200
inv nameGueltig: name.size() > 2
inv alterGueltig: let aktAlter : Tage = (Datum::today() -
   geburtsDatum) in
   aktAlter.toReal() >= alter and aktAlter.toReal() < alter + 1
```

oder insgesamt:

```
import 'model.uml'

context Model::Person
-- ...
inv vornameGueltig: vorname.size() >0
-- ...
inv einkommenNichtNegativ: Euro::Euro(0.0) < einkommen or Euro::
    Euro(0.0) = einkommen
-- ...

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

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

2.13. OclHelper für Verwandschaftsbeziehungen

Als Anwendung der OCL Collection-Methoden OCLs ein Paar OclHelper-Methoden und -Attribute:

```
context Model::Person
def: istKind(p: Model::Person): Boolean =
      elter->includes(p)
def: istSchwester(s: Model::Person): Boolean =
      elter.kind->includes(s) and
      self <> s and
      s.geschlecht = Gender::weiblich
def: onkel: Set(Model::Person) =
            elter.elter.kind->excludingAll(elter)
                             ->select(geschlecht = Gender::
                                 maennlich)
                             ->asSset()
```

⋮

2.14. Alle Instanzen einer Klasse: allInstances()/ M2 Modell-Queries und WFRs

Die Methode

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

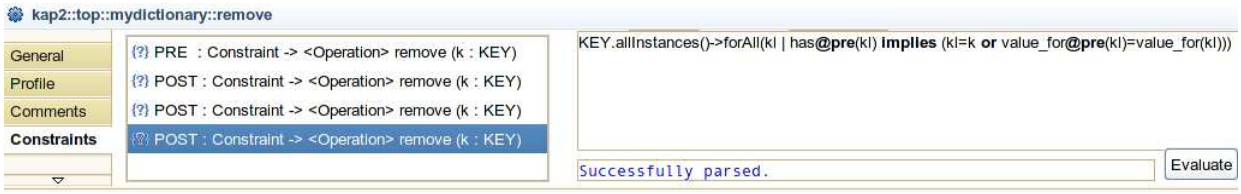
liefert alle (endlich viele) Instanzen einer User-definierten Klasse. Sie darf nicht benutzt werden für String, Integer und Real.
Ein Beispiel:

```
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/ (Download Chapter)
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')
```

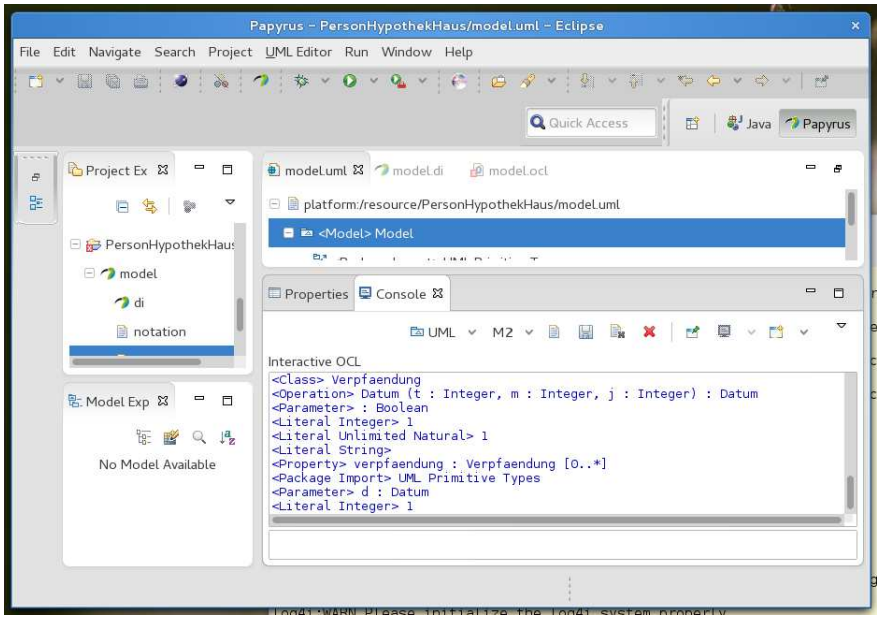
Get the dates of all Konsultations

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

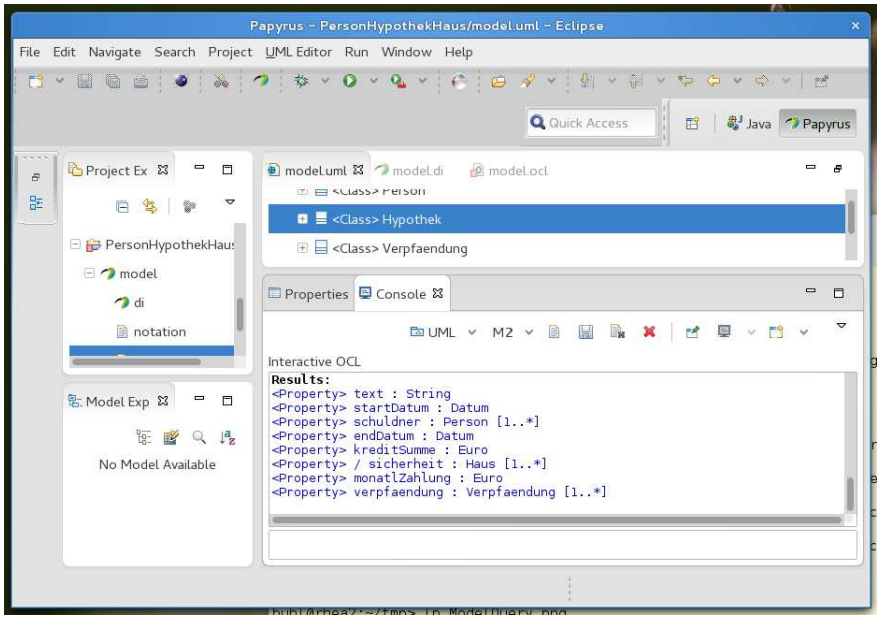
aus: allInstances-Query

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):`

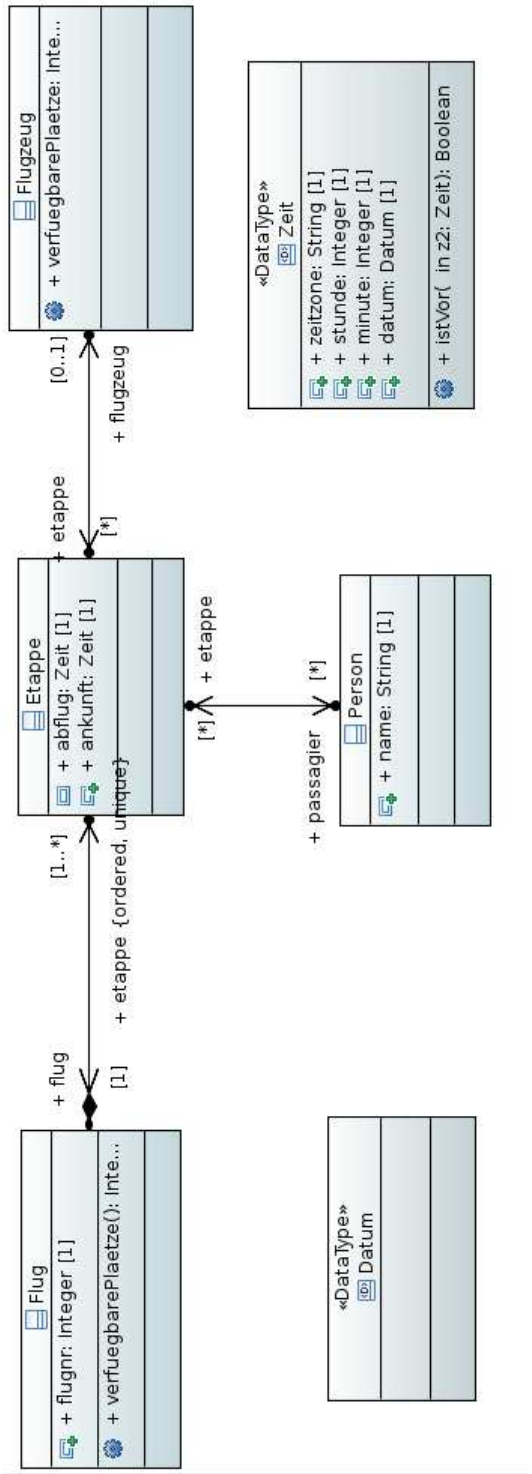


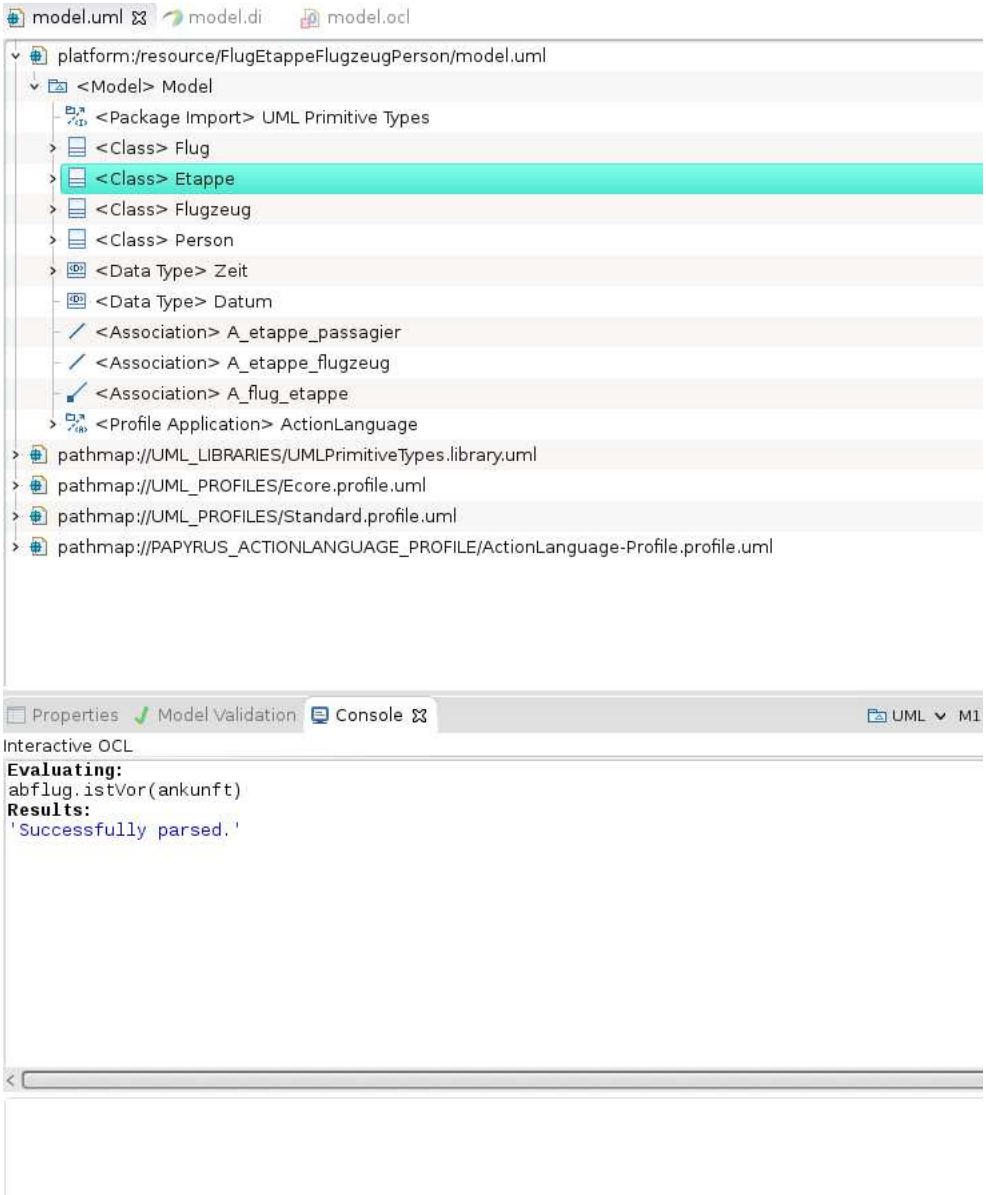
`allFeatures():Set(Features):`

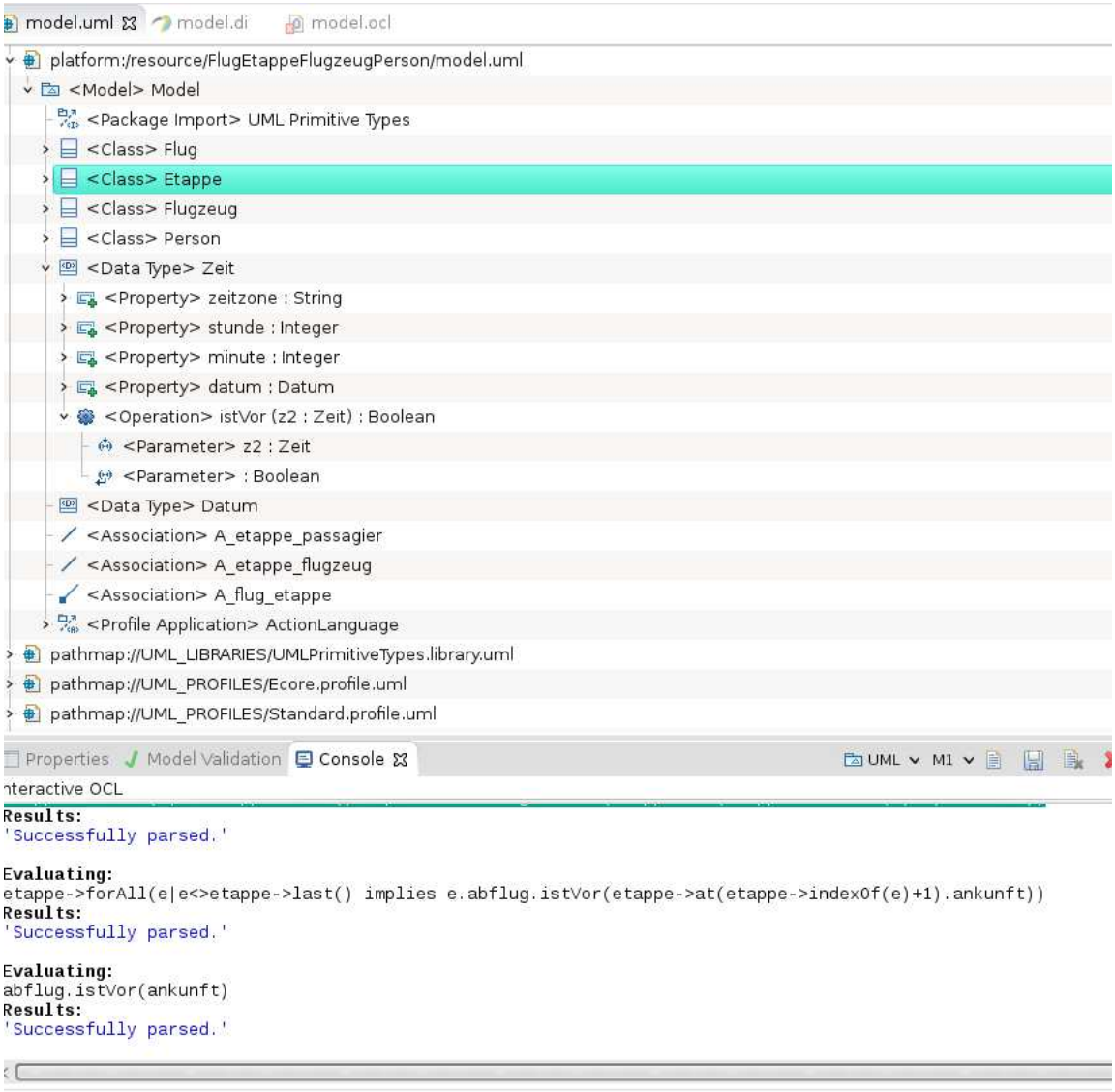


(Beachte den Abfrage-Level: `M2`)

2.15. Modell-Constraints für Flüge mit Zwischenlandungen




model.uml
model.di
model.ocl
platform:/resource/FlugEtappeFlugzeugPerson/model.uml
<Model> Model
<Package Import> UML Primitive Types
<Class> Flug
<Class> Etappe
<Class> Flugzeug
<Class> Person
<Data Type> Zeit
<Data Type> Datum
<Association> A_etappe_passagier
<Association> A_etappe_flugzeug
<Association> A_flug_etappe
<Profile Application> ActionLanguage
pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml
pathmap://UML_PROFILES/Ecore.profile.uml
pathmap://UML_PROFILES/Standard.profile.uml
pathmap://PAPYRUS_ACTIONLANGUAGE_PROFILE/ActionLanguage-Profile.profile.uml
Properties
Model Validation
Console
UML
M1
Interactive OCL
Evaluating:
abflug.istVor(ankunft)
Results:
'Successfully parsed.'


model.uml
model.di
model.ocl
platform:/resource/FlugEtappeFlugzeugPerson/model.uml
<Model> Model
<Package Import> UML Primitive Types
<Class> Flug
<Class> Etappe
<Class> Flugzeug
<Class> Person
<Data Type> Zeit
<Property> zeitzone : String
<Property> stunde : Integer
<Property> minute : Integer
<Property> datum : Datum
<Operation> istVor (z2 : Zeit) : Boolean
<Parameter> z2 : Zeit
<Parameter> : Boolean
<Data Type> Datum
<Association> A_etappe_passagier
<Association> A_etappe_flugzeug
<Association> A_flug_etappe
<Profile Application> ActionLanguage
pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml
pathmap://UML_PROFILES/Ecore.profile.uml
pathmap://UML_PROFILES/Standard.profile.uml
Properties
Model Validation
Console
UML
M1
nteractive OCL
Results:
'Successfully parsed.'
Evaluating:
etappe->forAll(e|e<>etappe->last() implies e.abflug.istVor(etappe->at(etappe->indexOf(e)+1).ankunft))
Results:
'Successfully parsed.'
Evaluating:
abflug.istVor(ankunft)
Results:
'Successfully parsed.'

model.uml model.di model.ocl

```
1  import 'model.uml'
2
3  context Model::Flug
4  inv: flugnr > 0
5  inv etappenZeitRelation: etappe->forAll(e|e<>etappe->last() implies
6               e.abflug.istVor(etappe->at(etappe->indexOf(e)+1).ankunft))
7  inv konsistenteBidir: etappe.flug=Bag{self}
8
9  context Model::Flug
10 def: next(s: OrderedSet(Model::Etappe), index: Integer): Model::Etappe =
11 s->at(index+1)
12 context Model::Flug
13 inv etappeVorEtappe: etappe->forAll(e |e<>etappe->last() implies
14                       e.abflug.istVor(next(etappe,etappe->indexOf(e)).ankunft))
15
16 context Model::Etappe
17 inv: abflug.istVor(ankunft)
18 inv konsistenteBidir1: flugzeug->notEmpty() implies flugzeug.etappe->includes(self)
19 inv konsistenteBidir2: passagier.etappe->includes(self)
20 inv konsistenteBidir3: flug.etappe->includes(self)
21
22 context Model::Flugzeug
23 inv nichtNegativ: verfuegbarePlaetze >= 0
24 inv verfuegbarePlaetze: etappe.flug.verfuegbarePlaetze()->sum() <= self.verfuegbarePlaetze
25 inv konsistenteBidir1: etappe.flugzeug->asSet() = Set{self}
26
27 context Model::Person
28 inv: name <> '' and name.size() >= 2
29 inv konsistenteBidir: etappe.passagier->includes(self)
30
31 context Model::Etappe
32 def: momentanVerfuegbarePlaetze(): Integer =
33      flug.verfuegbarePlaetze() - passagier->size()
34 inv nichtNegativ: momentanVerfuegbarePlaetze() >= 0
35
36 context Model::Flugzeug
37 inv wenigVergeudung: etappe.momentanVerfuegbarePlaetze()->sum() <= 0.05 *
38      verfuegbarePlaetze
39
```

2.16. pre-Ausdrücke in Nachbedingungen

7.5.14 Previous Values in Postconditions

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

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

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

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

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

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

⋮

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

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

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

(OCL 2.4-Manual)

2.17. SdV in C++20(?):

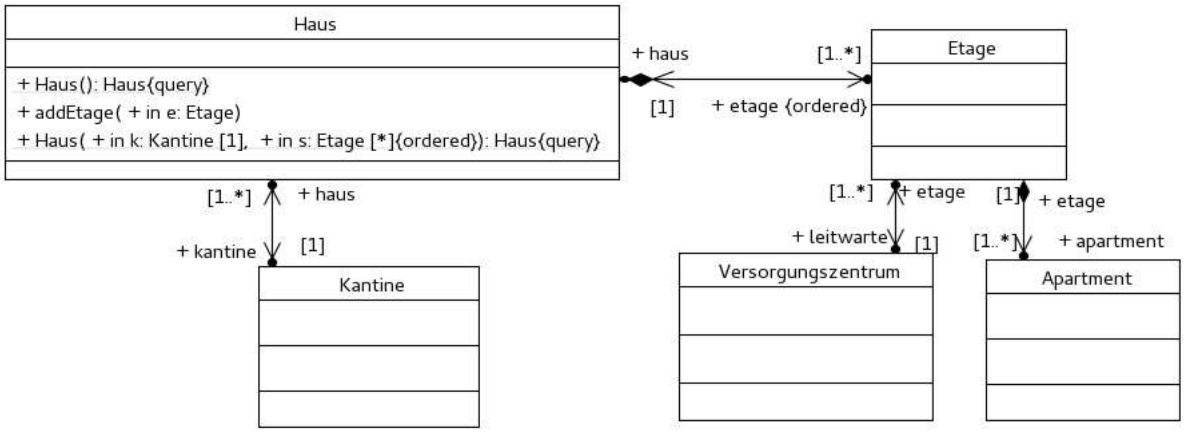
Codeverträge mit Hilfe der Attribute `expects, ensures` und `assert` vorgeschlagen:

```
template <tyename Iter>
  requires RandomAccessIterator<Iter>()
Iter bsearch(Iter p, Iter q, std::iterator_traits<Iter>::value_type w){
  [[expects default: p <= q]]
  [[expects audit:   is_sorted(p, q)]]
  [[ensures result:  !(result != q) || (*result == w
                    && (!(result != p) || *(result-1) != w))  ]]
  [[ensures axiom result:
   !(std::none_of(p, q, [](auto v){return v == w;}))|| result == q ]]
{
   // ...
}
```

N4110: Exploring the design space of contract specications for C++ (J. Daniel Garcia)
N4415: Simple Contracts for C++ (J. Daniel García et. al.)
Contracts programming for C++20

attribute specifier

2.18. 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 2.7: 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 and e <> invalid
pre: etage->excludes(e)
post hausMitMindestensEinerEtage: etage->size() >= 1
post hausEnthaeltE: etage->includes(e)
post anzEtagenIncrementet: etage->size() - etage@pre->size
    () = 1
```

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

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

- Formal vollständiger inlusive Framebedingung:

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

- Alternative Vorbedingungen:

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

Klassen-Invarianten:

- Kein Haus darf mehr als 10 Etagen besitzen.

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

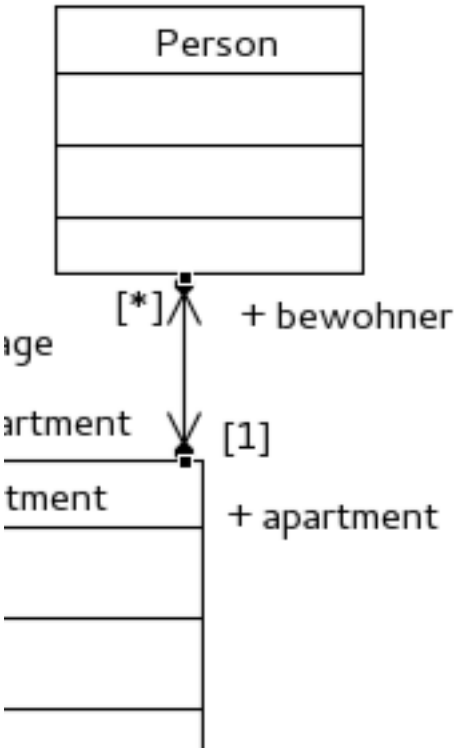
- Jede Etage hat 1..20 Apartments.

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

- Jedes Apartment hat 0..4 Bewohner.

```
context Model::Apartment
inv maxAnzahlBewohner1: bewohner->size() <= 4
                    -- oder
inv maxAnzahlBewohner2: 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 Apartment besitzt eine Identifikationsnummer `apartmentID`, die in der gesamten Wohnanlage eindeutig ist.

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

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

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

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

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

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

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

  oder als explizites Collect:

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

Destruktor-Spezifikation:

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

  ```
  context 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 destruiereEtagen: Model::Etage.allInstances()->
      excludesAll(etage@pre)
  post destruiereApartments: Model::Apartment.allInstances()
      ->excludesAll(etage@pre.apartment@pre)
  ```

  (Beachte die mehrmalige 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 clearupUnnecessaryVZ: 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))
```

2.19. `operator-` der Klasse `Datum`

Spezifikation des OclHelpers `toChronoJD(): Integer`
(Ideen nach:
Chronologisches julianisches Datum, Berechnung
Julian date calculation
Chronological Julian Date):

```
-- ...
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

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

context Model::Datum
def: minusZinstage(d: Model::Datum): Integer =
-- nach der E30/360-Methode der Sparkassen
  (jahr() - d.jahr()) * 360 +
  (monat() - d.monat()) * 30 +
  tag().min(30) - d.tag().min(30)
-- ...
```

2.20. Startwerte von Attributen, abgeleitete Attribute, Bodies von query-Operationen

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

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

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

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

2.21. Virtuelle OCL Variablen/Operationen / OclHelper

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

2.22. Typ-Konformität

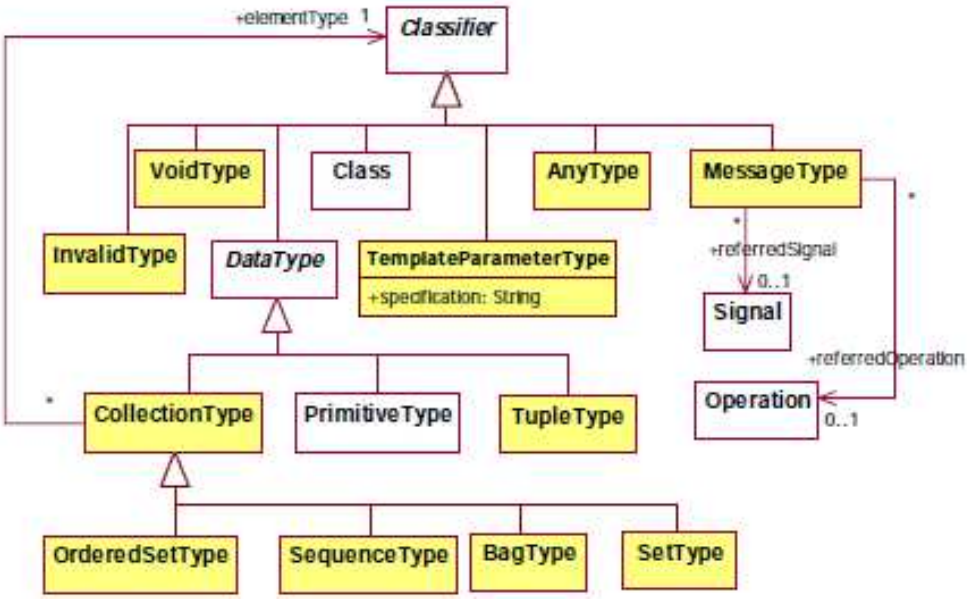
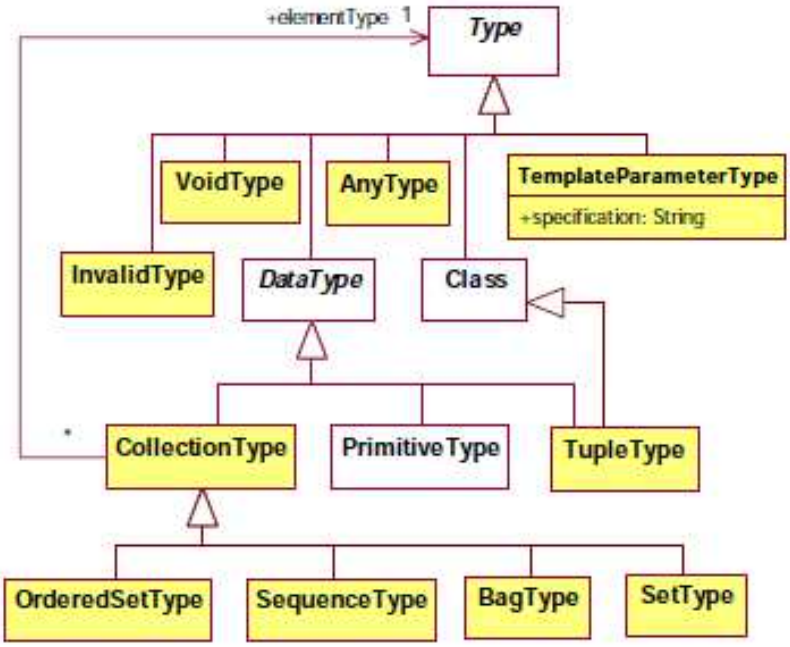


Abbildung 2.9.: Die Typen der OCL-Standard-Bibliothek

(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.

2.23. Operator-Vorrangsregeln

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

2.24. oclIsUndefined()

Auf Objekten des Typs OclAny kann mittels

not obj.oclIsUndefined()

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

2.25. 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.

7.4.7 Re-typing or Casting Collections

A Collection may be retyped in a similar way, but using the collection navigation operator.

```
aCollection->oclAsType(Set(String))
```

This will return *invalid* if either *aCollection* is not a Set or the elements of *aCollection* are not conformant with String.

The elements of a collection may be retyped individually using a collect iteration.

```
aCollection->collect(oclAsType(String))
```

This preserves the kind of collection (Set or Sequence or ...) but retypes the elements.

The selectByKind operation may be used to select a type conformant sub-collection.

```
aCollection->selectByKind(Person)
```

This returns a sub-collection of the same kind as *aCollection* containing all the non-null elements that are conformant to Person. Similarly selectByType returns a sub-collection of the non-null elements with the exact type.

oclIsTypeOf(Graduate)
oclIsKindOf() vs. oclIsTypeOf()

Im OCL 2.4-Handbuch:

```
Invalid values: 7.4.13
OCL Typ-Hierarchie: A.2.7
OclVoid/null: 11.2.3
```

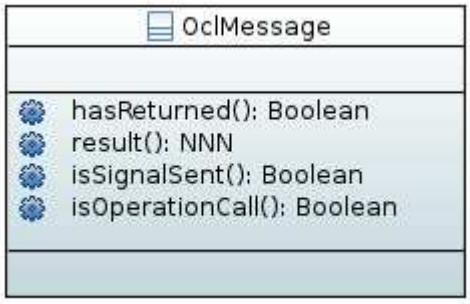
2.26. OclMessage/Signal/Observer und UML-Statusdiagramme

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)

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

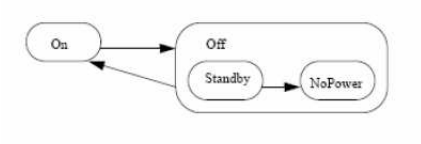


Figure 7.5 - Statemachine Example

In the example statemachine above, values for *s* can be *On*, *Off*, *Off::Standby*, *Off::NoPower*. If the classifier of *object* has the above associated statemachine, valid OCL expressions are:

```
object.oclIsInState(On)
object.oclIsInState(Off)
object.oclIsInState(Off::Standby)
object.oclIsInState(Off::NoPower)
```

If there are multiple statemachines attached to the object's classifier, then the statename can be prefixed with the name of the statemachine containing the state and the double colon '::,' as with nested states.

The operation *oclIsNew* evaluates to true if, used in a postcondition, the object is created during performing the operation (i.e., it didn't exist at precondition time).

The operation *oclAsType(t)* casts the source to the type *t*, which must be a subtype or supertype of the source type.

event driven programming

2.27. 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 2.11):

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

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` Hash-Varianten bereit.

In OCL 2.5 sind `Map(.,.)` und `OrderedMap(.,.)` zu erwarten (Wo bleiben die Multimaps?).

2.28. Ausblick OCL 2.5

OCL2.5 Plans (EdWillink)

Implied Functionality - Reflection

```
Classifier::allInstances() : Set(T)      in OCL 2.4
OclAny::oclType() : Classifier           in OCL 2.4
```

- informal declarations, magic T

```
MyType::allInstances().myProperty
```

- unspecified apparent type utility

```
OclAny::oclType() : typeof(OclSelf)      in OCL 2.5
OclAny::allInstances() : Set(OclSelf)    in OCL 2.5
```

- OclSelf - the apparent type of self
- typeof(X) - Class with instance lowerbound of X

```
myType.oclType().ownedOperations
```

Implied functionality - multi-returns

- UML supports multiple function returns
  - OCL 2.4 doesn't
- Solution, wrap multiple returns up as a Tuple

'UML': `f(in a:A, in b:B, inout c:C, out d:D)`

becomes

OCL:
```
f(a:B, b:B, c:C) : Tuple{c:C, d:D}
body: Tuple{c = ..., d = ...}

post: result.c = ...
post: result.d = ...

post: result = Tuple{c = ..., d = ...}
```

Syntax sugar - elseif

- OCL 2.4 cumbersome endifs

```
if c1 then v1
else if c2 then v2
     else if c3 then v3
          else v4
          endif
     endif
endif
```

- OCL 2.5 simpler

```
if c1 then v1
elseif c2 then v2
elseif c3 then v3
else v4
endif
```

- just a CS rewrite

Syntax sugar - typesafe if

- OCL 2.4 cumbersome re-typing

```
if x.oclIsKindOf(MyType) then
    let t : MyType = x.oclAsType(MyType) in f(t)
else ...
endif
```

- OCL 2.5 simpler

```
if t : MyType = x then
    f(t)
else ...
endif
```

- special case of a pattern match

Syntax sugar - safe navigation

- OCL 2.4 vulnerable to navigation on nulls

```
x.ys.z          x.ys->collect(z)
```

is invalid if x or any x.ys is null

```
if x <> null then x.ys->excluding(null).z else null endif
```

- OCL 2.5 offers safe navigation

```
x.?ys.?z        x.?ys->?collect(z)
```

- simple CS rewrite

Syntax sugar - iterator alternatives

- OCL 2.4 `source->iteration(iterators | body)`

```
e.g. s->forAll(body)
     s->forAll(i | body)
     s->forAll(i : Integer | body)
```

- OCL 2.5 iterator may instead have domain

```
forAll(i in s | body)
forAll(i : Integer in s | body)
```

- OCL 2.5 iterator may be allInstances

```
forAll(c in Class | body)
Class::allInstances()->forAll(c | body)
```

  - type/property ambiguity resolved to type

Iteration co-indexes

- OCL 2.4 cumbersome to obtain iteration index

```
parameters->forAll(p |
    let i = parameters->indexOf(p),
        a = arguments->at(i)
    in a.type->conformsTo(p.type))
```

- For ordered collections

```
parameters->forAll(p[i] |
    let a = arguments->at(i)
    in a.type->conformsTo(p.type))
```

- Implementation can be better than syntax sugar
  - add coIndexName to AS

Syntax sugar - collection comprehension

- OCL 2.4 iterations

```
source->iteration(iterator : T | body)
```

- OCL 2.5 iteration alternative with initializer

```
iteration(iterator : T in source | body)
```

- OCL 2.4 collection literal

```
Set{1, 4, 9, 16}
```

- OCL 2.5 collection comprehension

```
Sequence{1..4}->Set{i | i*i}
Set{i in Sequence{1..4} | i*i}
```

rewrite of

```
Sequence{1..4}->collect(i | i*i)->asSet()
```

Library - Map

- OCL 2.4 cumbersome

```
let myMap = Set{Tuple{key:String=...,value:Integer=...}}
in myMap->includes{Tuple{key='five',value=5})
```

  - inaccurate Set{Tuple} not Set{String}

- OCL 2.5 Map (and OrderedMap)

```
let myMap : Map(String,Integer) = ...
in myMap->includes('five',5)
```

- Obvious, but not quite so obvious
  - OCL-like coherent interface design welcome

Complete OCL - Package invariants

- OCL 2.4 only classes have invariants
  - all invariants must be allocated to a class
- OCL 2.5, packages may have invariants too

```
package Mine
inv PackageInvariant: ....
```

2.29. N4126: explicitly defaulted comparison operators(C++20?)

Explicitly defaulted comparison operators:

...

It is vital that equal/unequal, less/more-or-equals and more/less-or-equal pairs behave as boolean negations of each other. After all, we are building total ordering and the world would make no sense if both `operator==()` and `operator!=()` returned *false*!

As such, it is common to implement these operators in terms of each other.

Inequality for Regular types:

```
bool operator!=(const user &a, const user &b)
{
    return !(a == b);
}
```

Relational operators for Totally Ordered types:

```
bool operator>=(const user &a, const user &b)
{
    return !(a < b);
}

bool operator>(const user &a, const user &b)
{
    return b < a;
}

bool operator<=(const user &a, const user &b)
{
    return !(a > b);
}
```

...

Oder kurz:

```
struct Thing
{
    int a, b, c;
    std::string d;
};

bool operator==(const Thing &, const Thing &)= default;
bool operator!=(const Thing &, const Thing &)= default;

class AnotherThing
{
    int a, b;

public:
    // ...

    friend bool operator<(Thing, Thing) = default;
    friend bool operator>(Thing, Thing) = default;
    friend bool operator<=(Thing, Thing) = default;
    friend bool operator>=(Thing, Thing) = default;
};
```

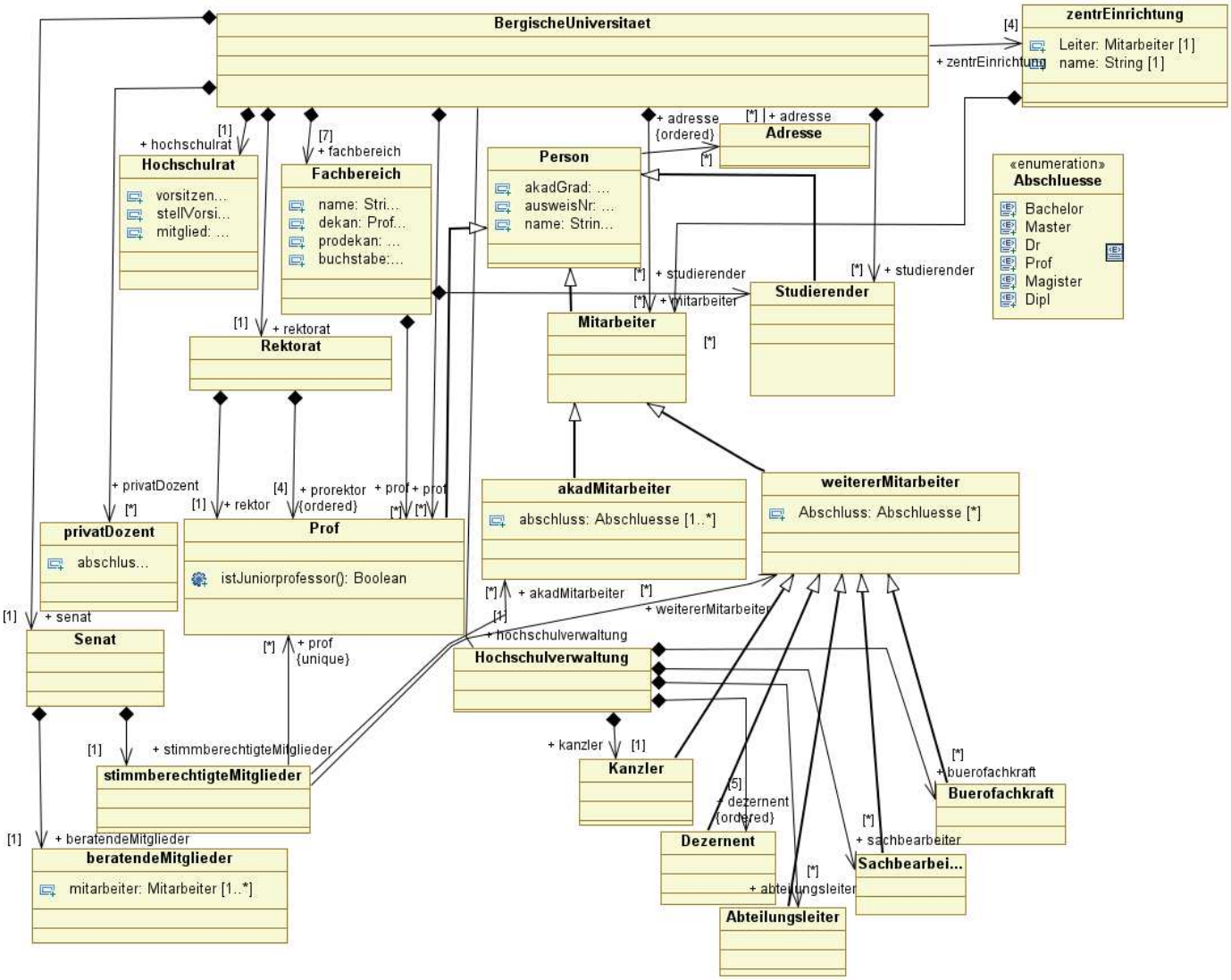
2.30. std::optional, std::variant und std::any in C++17

C++ reference
std::optional::optional
`make_optional`

```
#include <string>
#include <iostream>
#include <optional>

// optional can be used as the return type of a factory that may fail
std::optional<std::string> create(bool b) {
    if(b)
        return "Godzilla";
    else
        return {};
}

int main()
{
    std::cout << "create(false) returned "
              << create(false).value_or("empty") << '\n';

    // optional-returning factory functions are usable as conditions of while and if
    if(auto str = create(true)) {
        std::cout << "create(true) returned " << *str << '\n';
    }
}
```

std::variant statt union
std::any

2.31. Modell Bergische Universität



und einige Constraints (seit 2015 muß `Fachbereich` durch `Fakultaet` erssetzt werden):

```
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)
```

...

2.32. OCL-Stilregeln

- 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 maximalZweiLeiblicheEltern: parents->size() <= 2
```

  und

```
context Model::Person
inv maximalZweiElter: 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 keineVerheiratetenAngestellten: employees.wife->
   intersection(self.employees)->isEmpty()
```

    einfacher als

```
context Model::Person
inv keineVerheiratetenAngestellten: wife->notEmpty()
    implies
        wife.employers->intersection(self.employers)
          ->isEmpty()
```

– Nutze viele einfache

```
inv label1: ...
inv label2: ...
```

statt einer einzelnen inv-Zeile komplizierterer Bauart.

- Vermeide der Lesbarkeit halber collect():

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

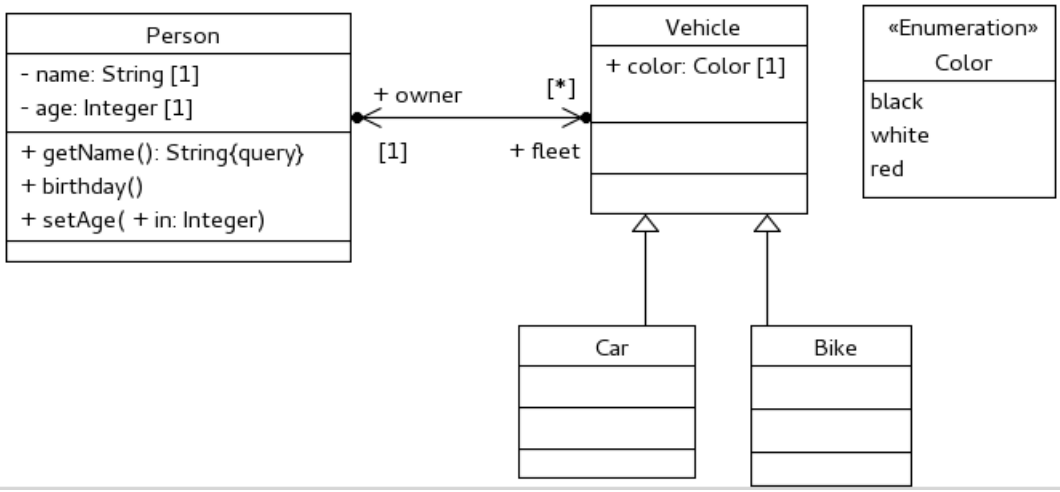
ist äquivalent zu:

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

- Gib allen Assoziationsenden einen Namen.

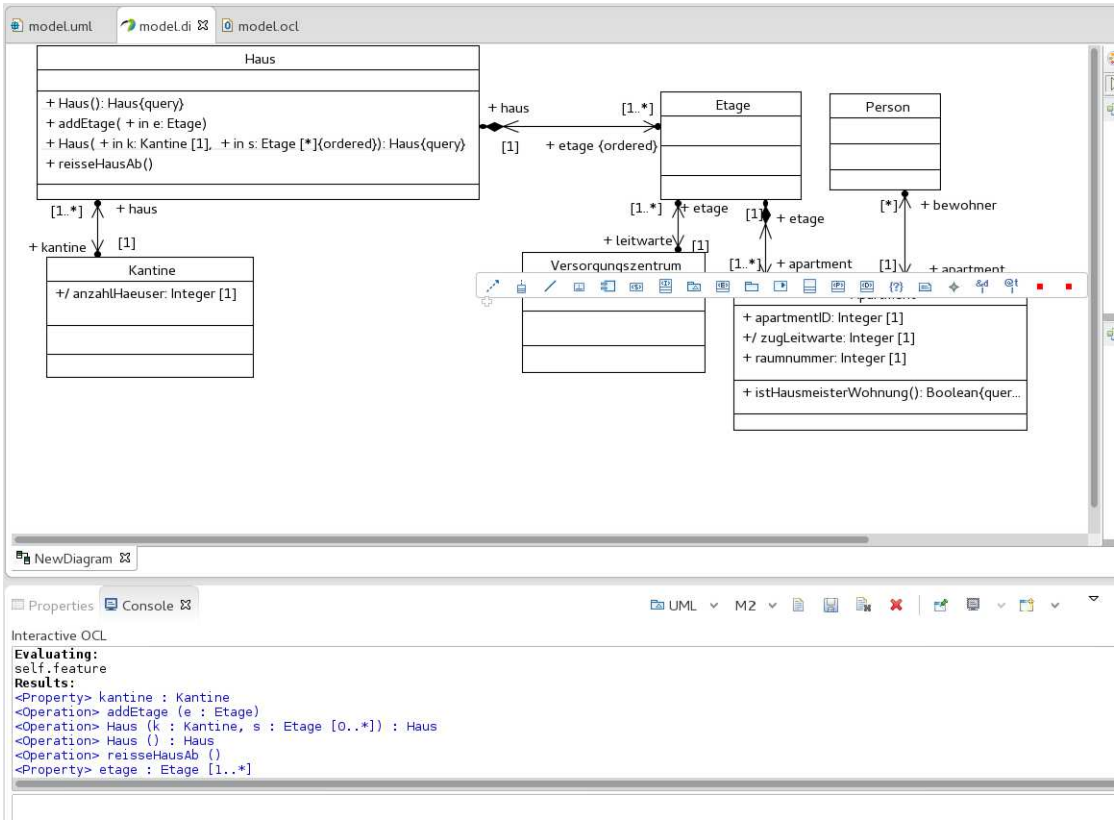
2.33. Ein einfacher Beispielvertrag für die geeignete Kontextwahl

Geeignete Kontextwahl macht semantisch äquivalente Constraints strukturell einfacher:




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


-- oder:


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

2.34. Metalevel2-Constraints = Wohldefiniertheitsregeln für Modelle

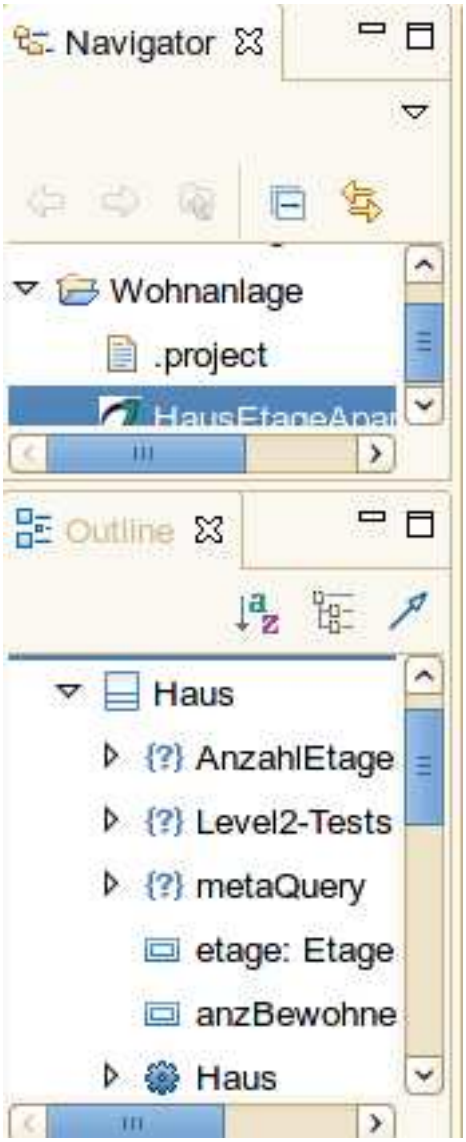


Im M2-Level des Constraints-Checkers von Papyrus kann man ein Modell algorithmisch mit OCL-Ausdrücken abfragen (Query-Sprache) und zum Beispiel Regeln des Programmiererteams als Invarianten definieren. Einige Queries:

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.
```

```
    oclIsUndefined())->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

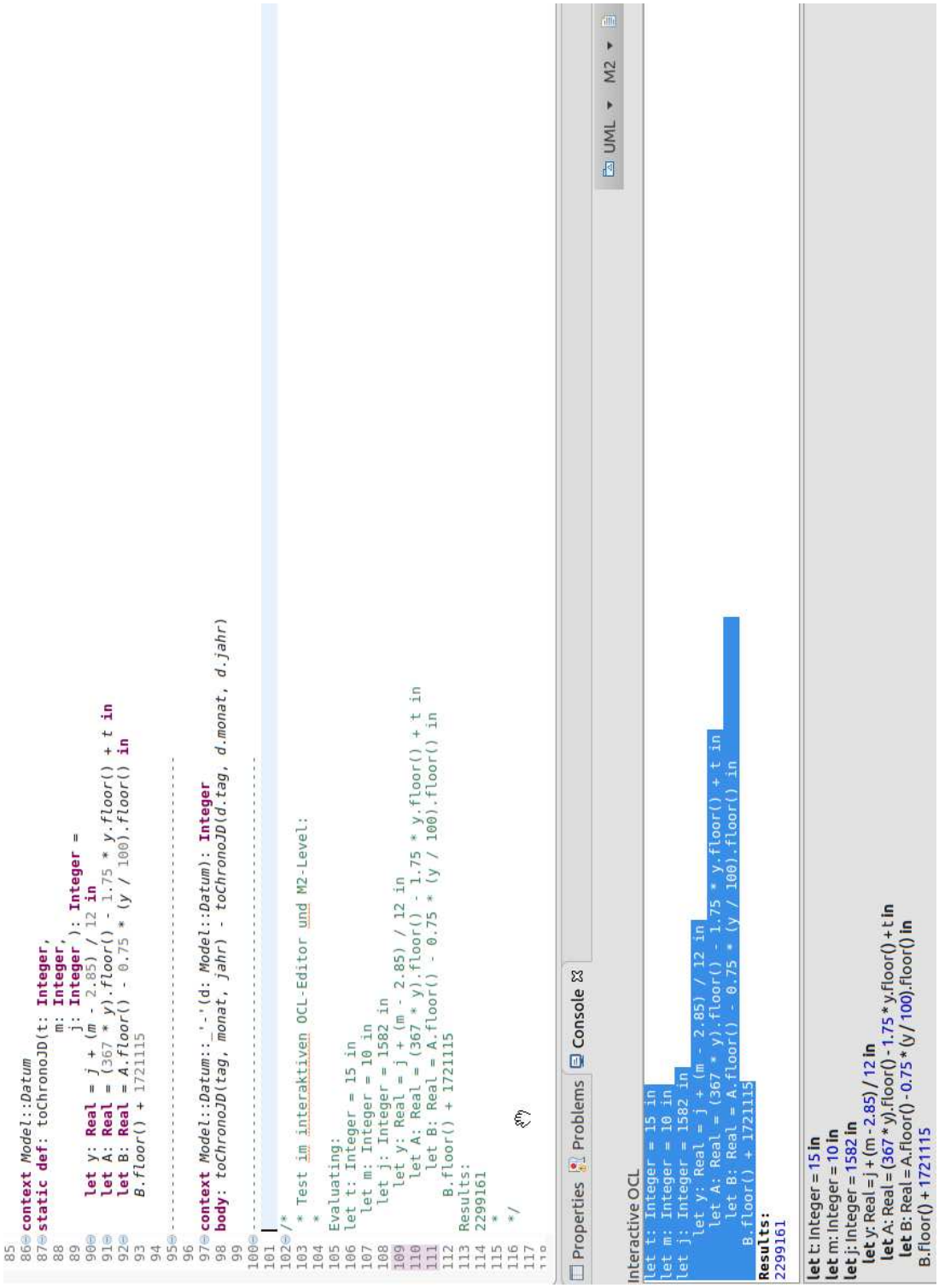
Aussagen über das aktuelle UML-Modell kann man so etwa über den folgenden M2 OCL-Ausdruck erhalten:

```
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)

2.35. toChronoJD-Test im M2-Level

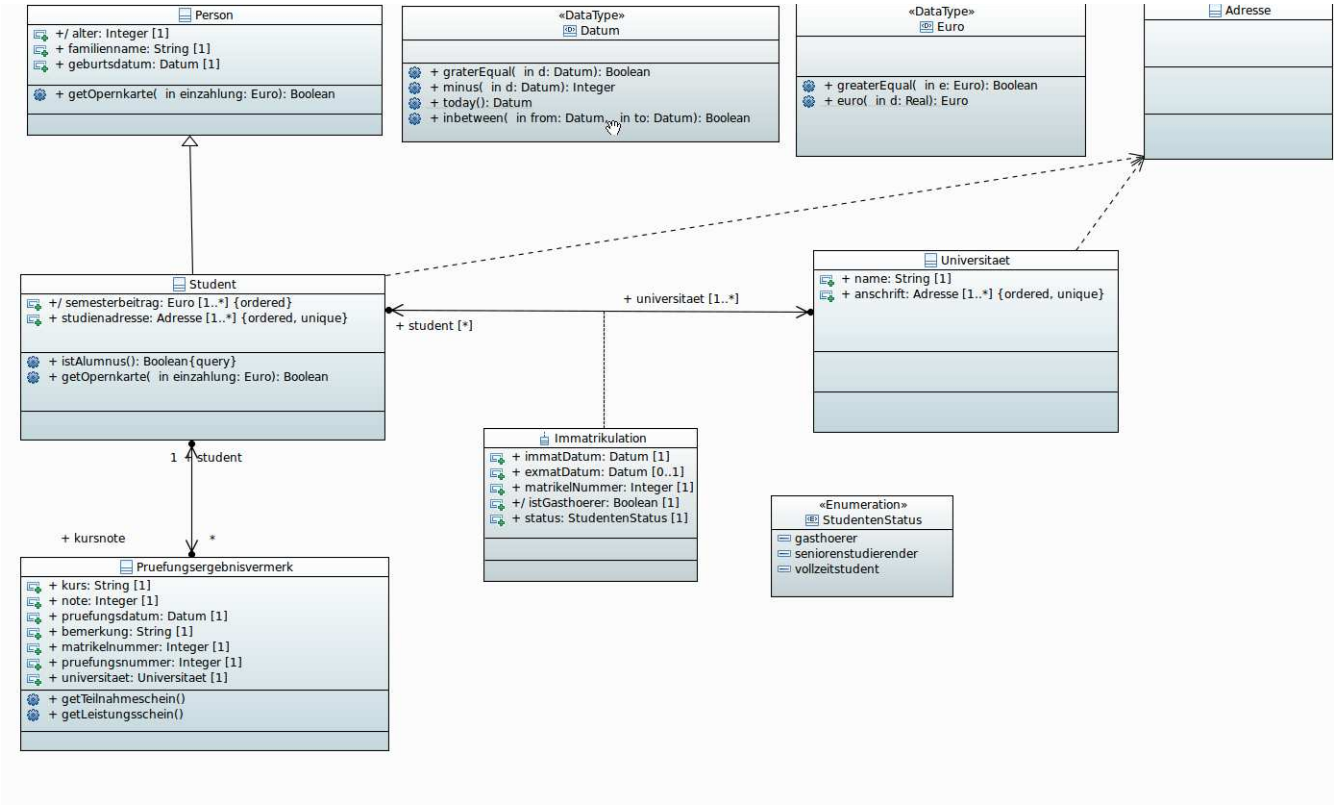


... stimmt mit dem julianischen Datum des 15. Oktobers 1582 aus https://de.wikipedia.org/wiki/Julianisches_Datum#Chronologisches_julianisches_Datum überein:

Gregorianischer Kalender	Julianisches Datum
15. Oktober 1582	2299161
1. Januar 1583	2299239
1. Januar 1990	2447893
1. Januar 2000	2451545
25. Juni 2018	2458295

2.36. Modell Student/Universitaet/Pruefungsergebnisvermerk

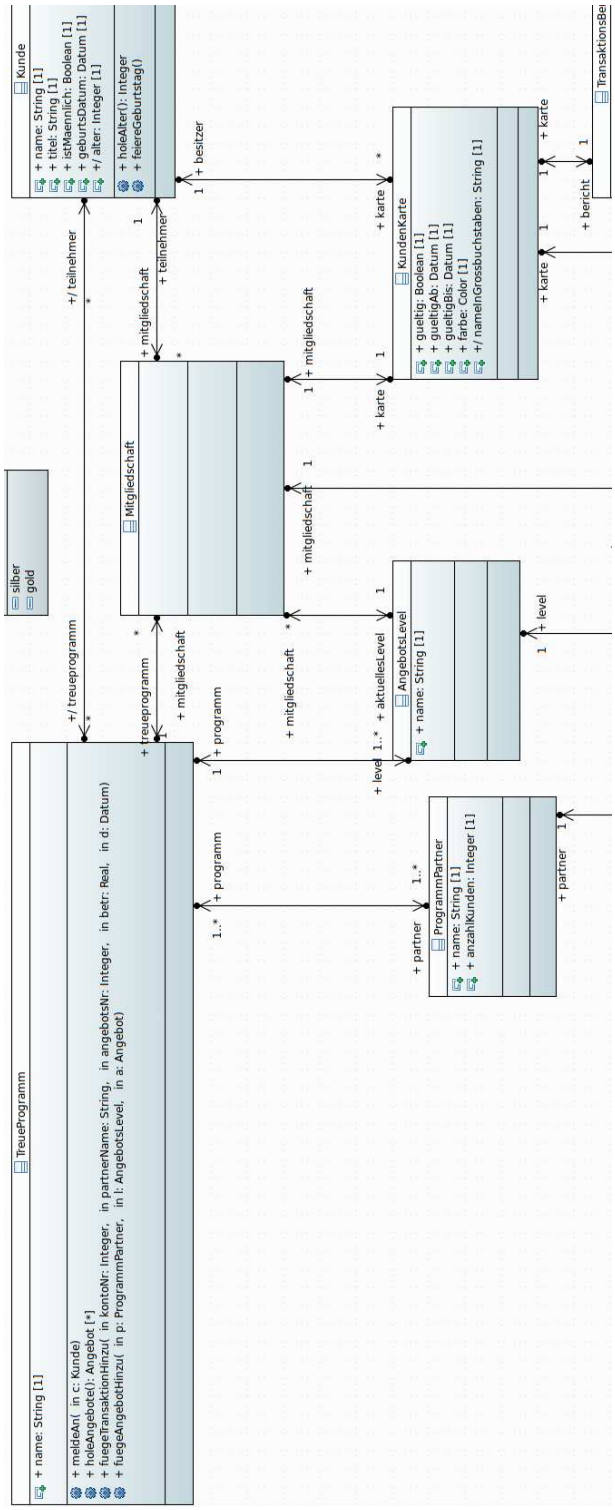
Ein UML-Modell mit Assoziationsklasse `Immatrikulation` und qualifizierter Assoziation `Model::Universitaet::student[matrikelnummer: Integer]` sowie `Model::Student::kursnote[pruefungsnummer: Integer]`

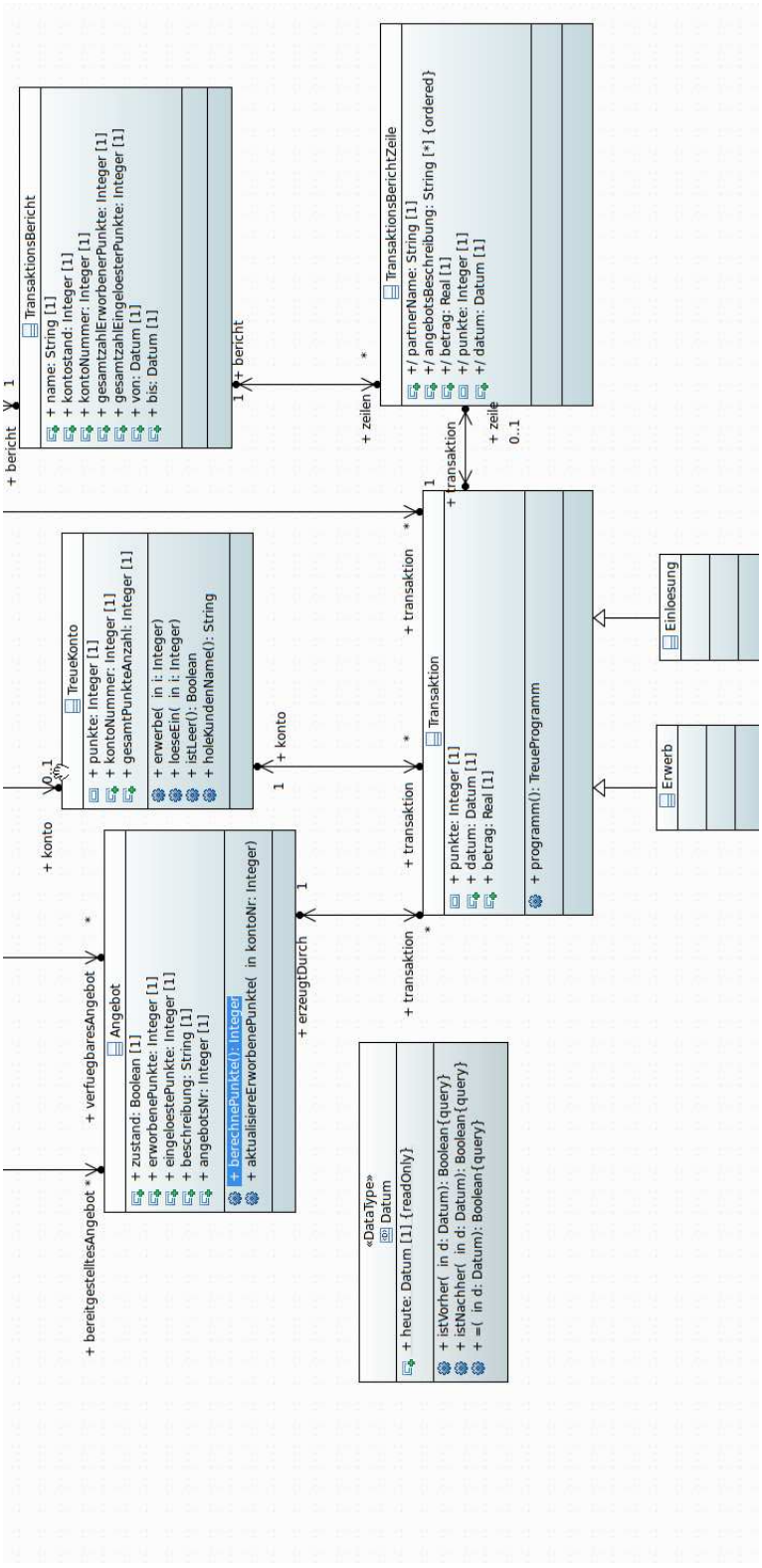


... und zugehörige Code-Contracts:

```
import 'model.uml'

context Model::Universitaet
inv nameNichtLeer: name <> ''
inv studentenExistent: student->size() > 0
inv matrikelnummerUnique: Immatrikulation->isUnique(i | i.matrikelNummer)
inv bidirektionalStudUni: student.universitaet->includes(self)
inv student_0312345: student[0312345].familienname = 'Bauer'
```

2.37. OCL-Fallstudie „Vorzugs- und Treuekunden“ / „Royal and Loyal Model“




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

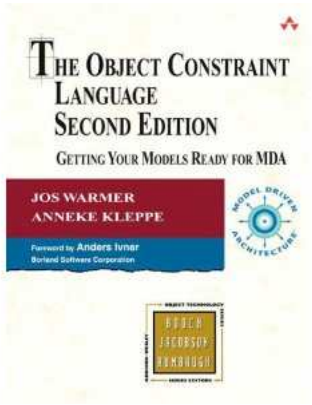
Erste Codeverträge:

```
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)

```
import 'model.uml'

--------------------------------------------------
context Model::TreueProgramm::teilnehmer: Set(Model::Kunde)
derive: mitgliedschaft.teilnehmer->asSet()

context Model::Kunde::treueprogramm: Set(Model::TreueProgramm)
derive: mitgliedschaft.treueprogramm->asSet()
--------------------------------------------------

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 gueltigkeitsDatenOk: 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 firstColor: level->first().name = 'silber'

context Model::TreueProgramm
inv secondColor: 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::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: Model::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 tBberichtOk: karte.transaktion->includesAll(zeilen.transaktion)
```

```
context Model::TreueProgramm::fuegeTransaktionHinzu(kontoNr: Integer,
                                                    partnerName: String,
                                                    angebNr: Integer,
                                                    betr: Real,
                                                    d: Model::Datum): OclVoid
pre kontoNrValid: mitgliedschaft.konto.kontoNummer->includes(kontoNr)
pre partnerNameValid: partner.name->includes(partnerName)
pre angebNrValid: angebNr > 0
pre betrValid: betr > 0.0
pre datumValid: not d.istVorher(Datum::heute)
post: let kto: TreueKonto =
            mitgliedschaft.konto->select(k| k.kontoNummer=kontoNr),
      neueT: Transaktion =
            partner->select(p| p.name=partnerName).
                        bereitgestelltesAngebot
                  ->select(a| a.angebotsNr=angebNr).transaktion
                  ->select(datum=d and betrag=betr),
```

```
        karte: KundenKarte =
            mitgliedschaft->select(m| m.konto.kontoNummer=kontoNr)
    in
        kto.punkte = kto.punkte@pre + neueT.punkte and
        neueT.oclIsNew() and
        betr = 0 implies neueT.oclIsTypeOf(Einloesung) and
        betr > 0 implies neueT.oclIsTypeOf(Erwerb) and
        kto.transaktion - kto.transaktion@pre = Set{neueT} and
        karte.transaktion - karte.transaktion@pre = Set{neueT}
```

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

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

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

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

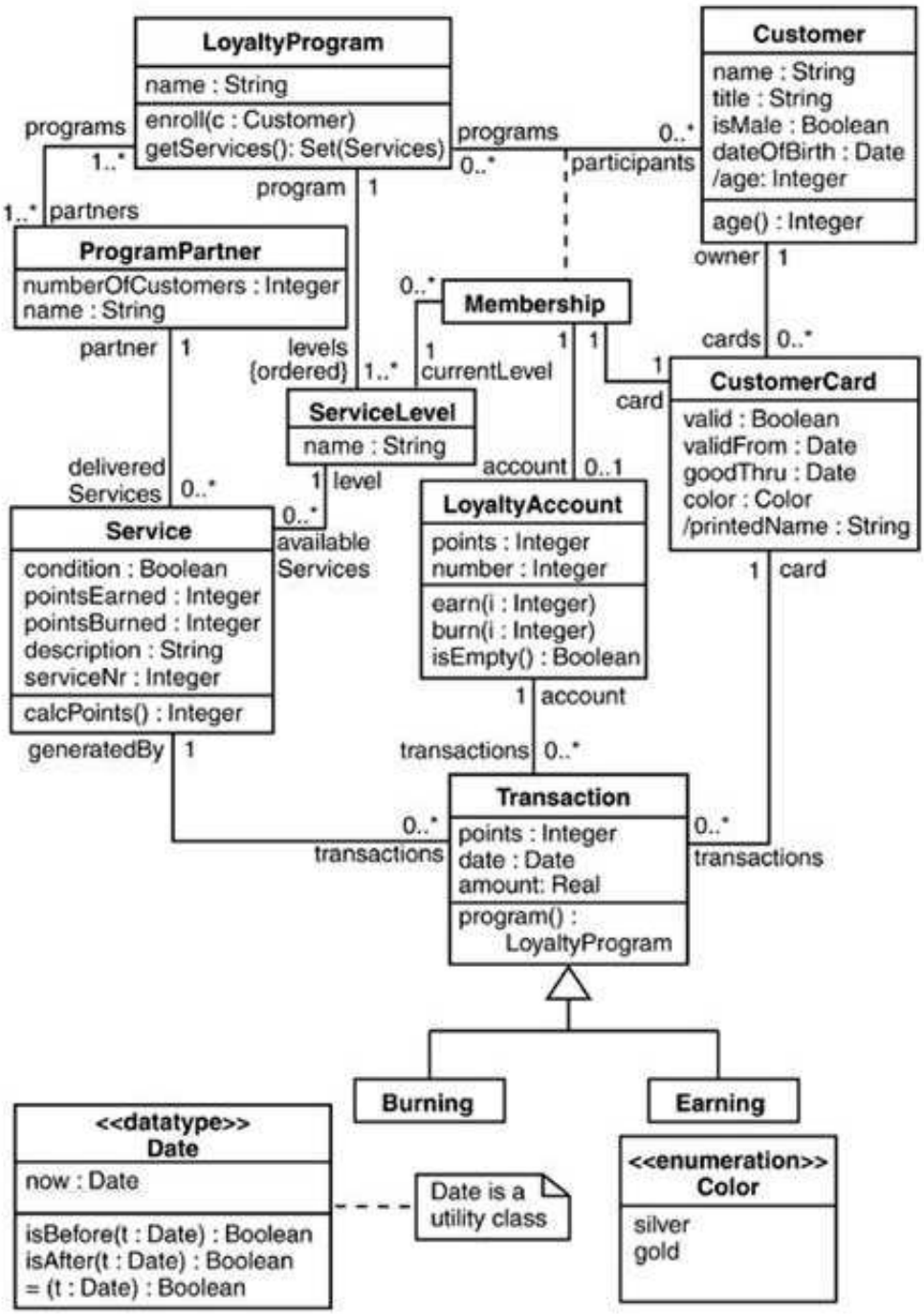
```
context Model::TreueProgramm
def: ohnePunktErwerbsMoeglichkeiten: Boolean =
    partner.bereitgestelltesAngebot->forAll(erworbenePunkte = 0)

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

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

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

Ana Moreira: Object Constraint Language: Seite 21...25, 36, 46, 47

Figure 2-1. The Royal and Loyal model



2.38. 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**.

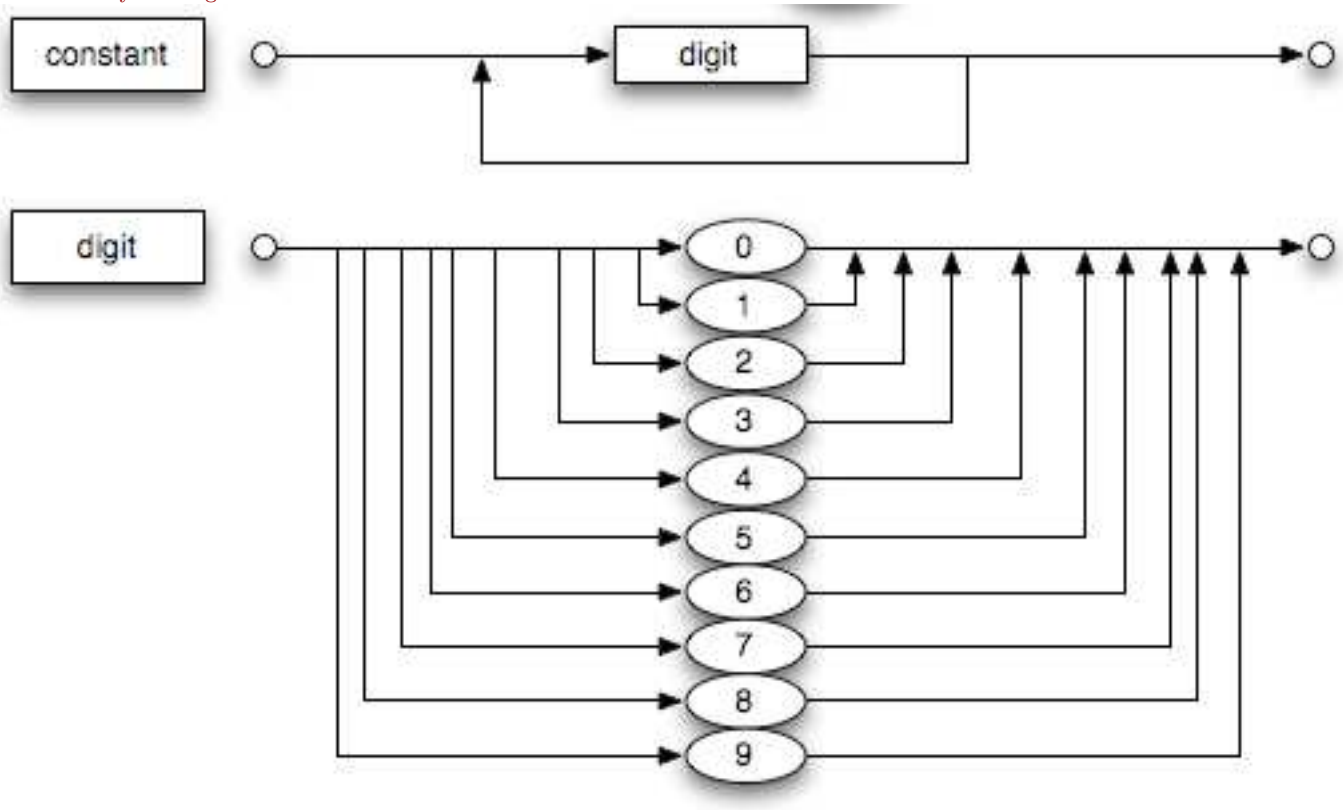
3.2.5.2 Canonical representation

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";
```

- Appendix: Extended Backus-Naur Form (EBNF)

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";
```

2.39. 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

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. UML/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

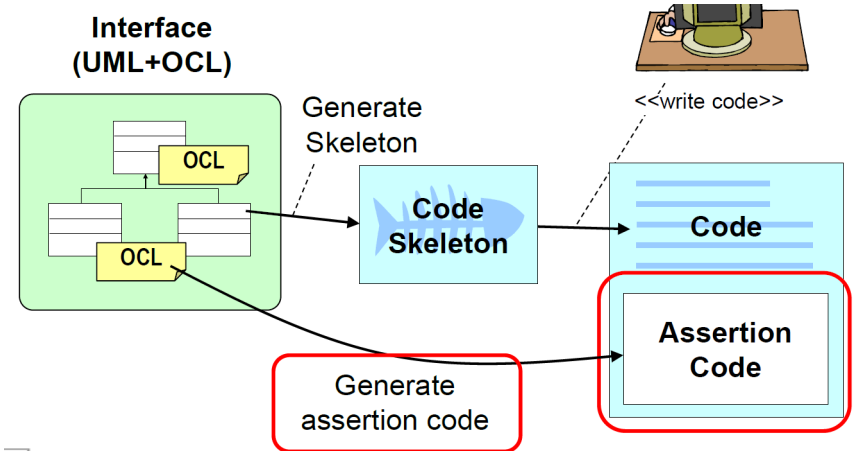
A.3. Generate Code

A.3.1. C++ code generation

Papyrus/Designer/getting-started
Papyrus/Codegen/Cpp description
Papyrus/Codegen/Adding a New Code Generator
Papyrus/Codegen/CppHelloWorld

A.3.2. Code generation for the OCL Constraints

Gererating Assertion Code from OCL:



A.4. OMGs C++ Language-Mapping für CORBA

CORBA C++ language mapping
Currency Service

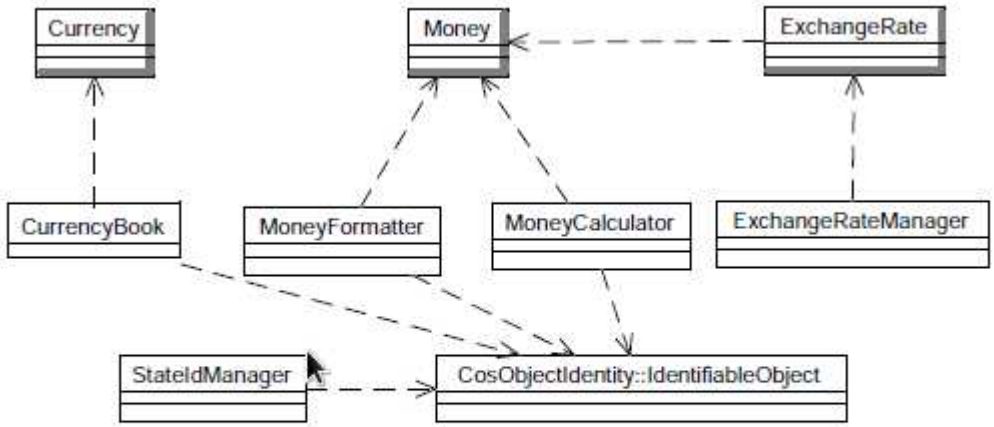
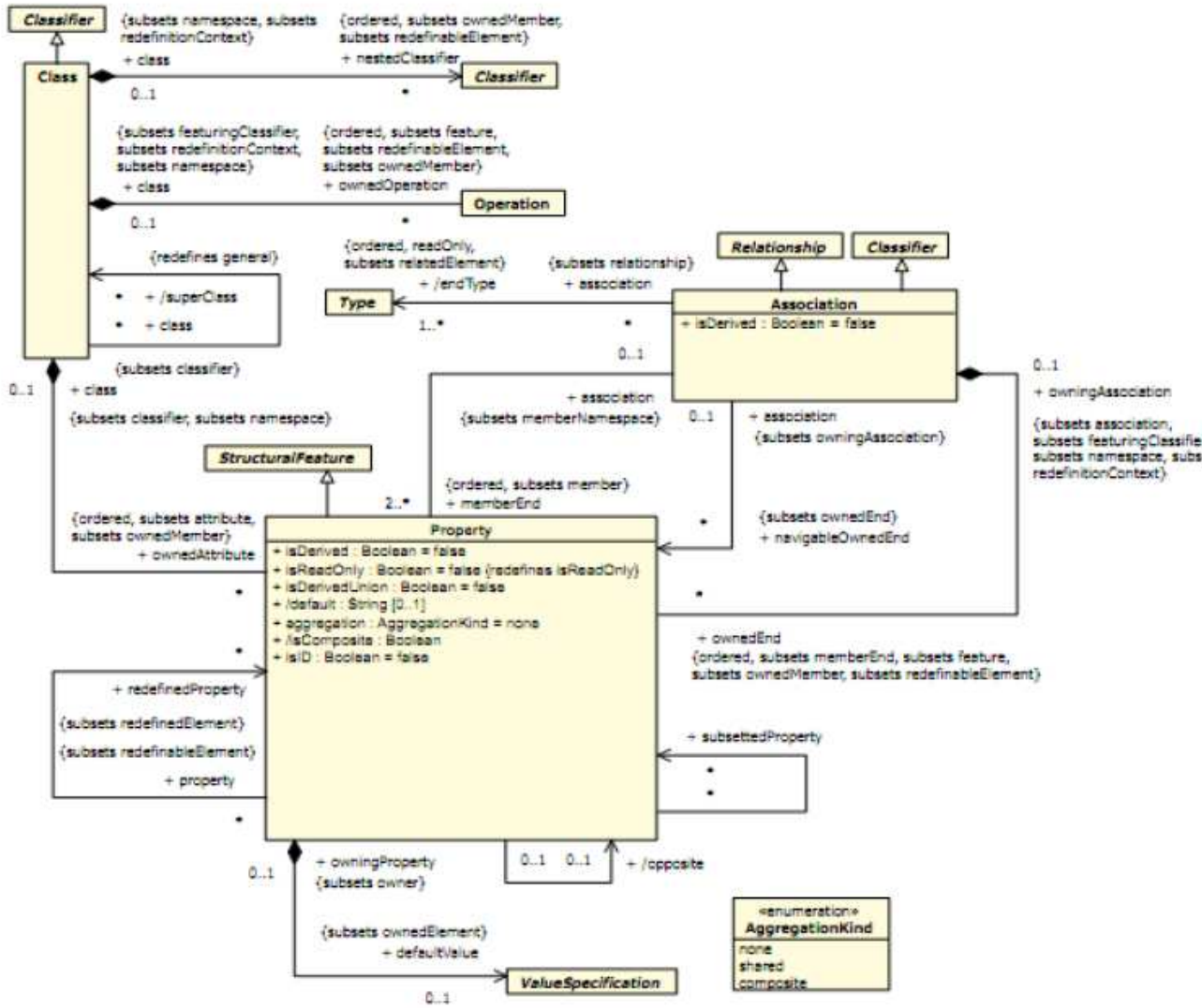
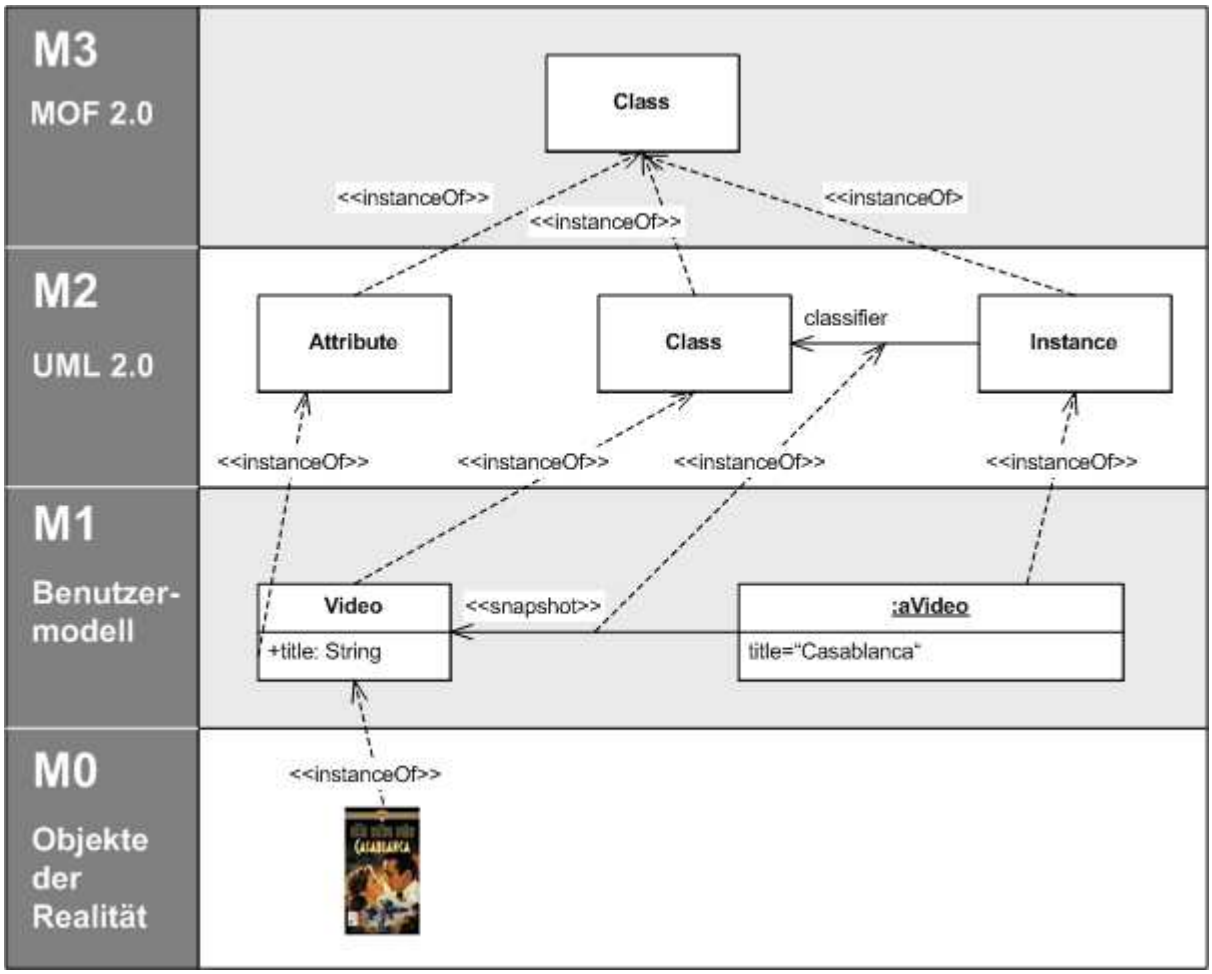


Figure 2-1 FbcCurrency Module

A.5. 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.6. OCL-Beispiele

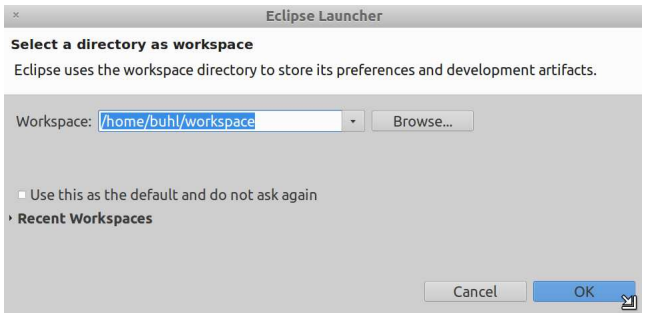
The university model
Object Constraint Language (OCL) tutorial
OCL2 to SQL transormation
...

A.7. Benutzungsanleitung: Erstellen eines neuen Papyrusprojekts

Starte `eclipse-papyrus03`:

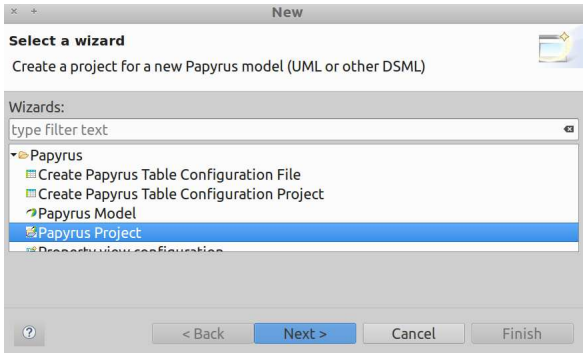
```
username@l104:~> eclipse -papyrus03
```

und wähle einen `Workspace`:



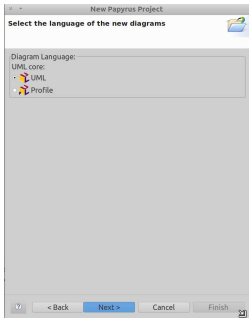
(OK), erstelle ein neues Projekt:

`File`
  `New`
    `Other`:

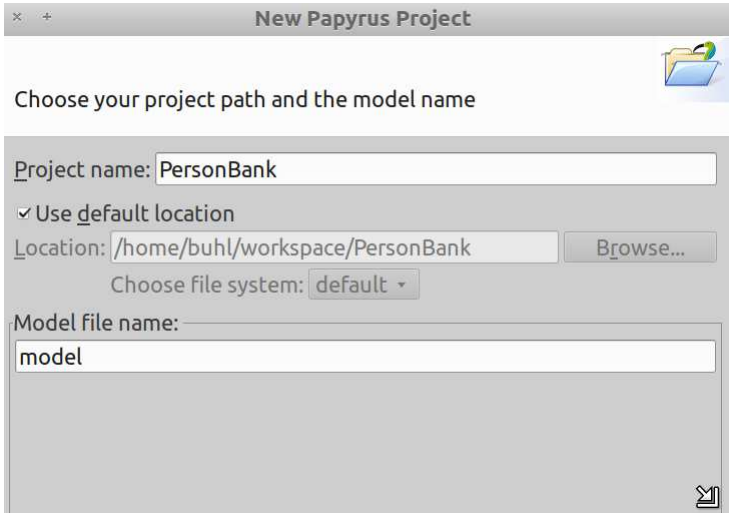


(Papyrus Projekt),

vom UML-Typ:

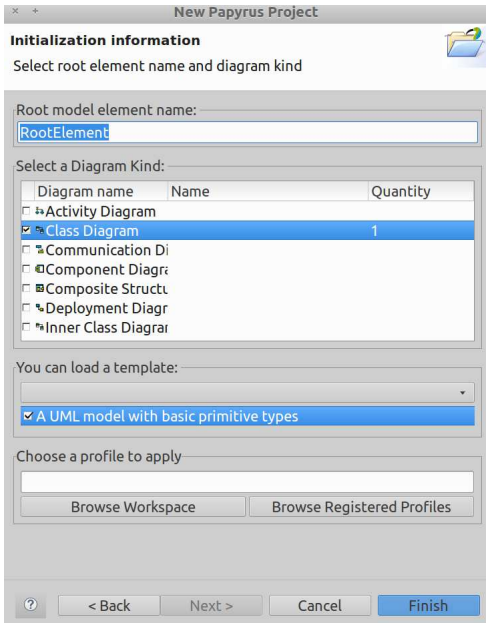


(Next) mit dem Projekt-Namen:

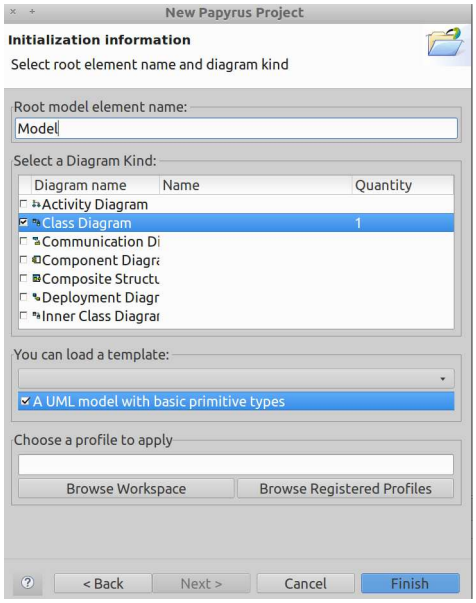


(PersonBank),

einem Klassendiagramm, Zugriff auf die UML „primitiven Typen“

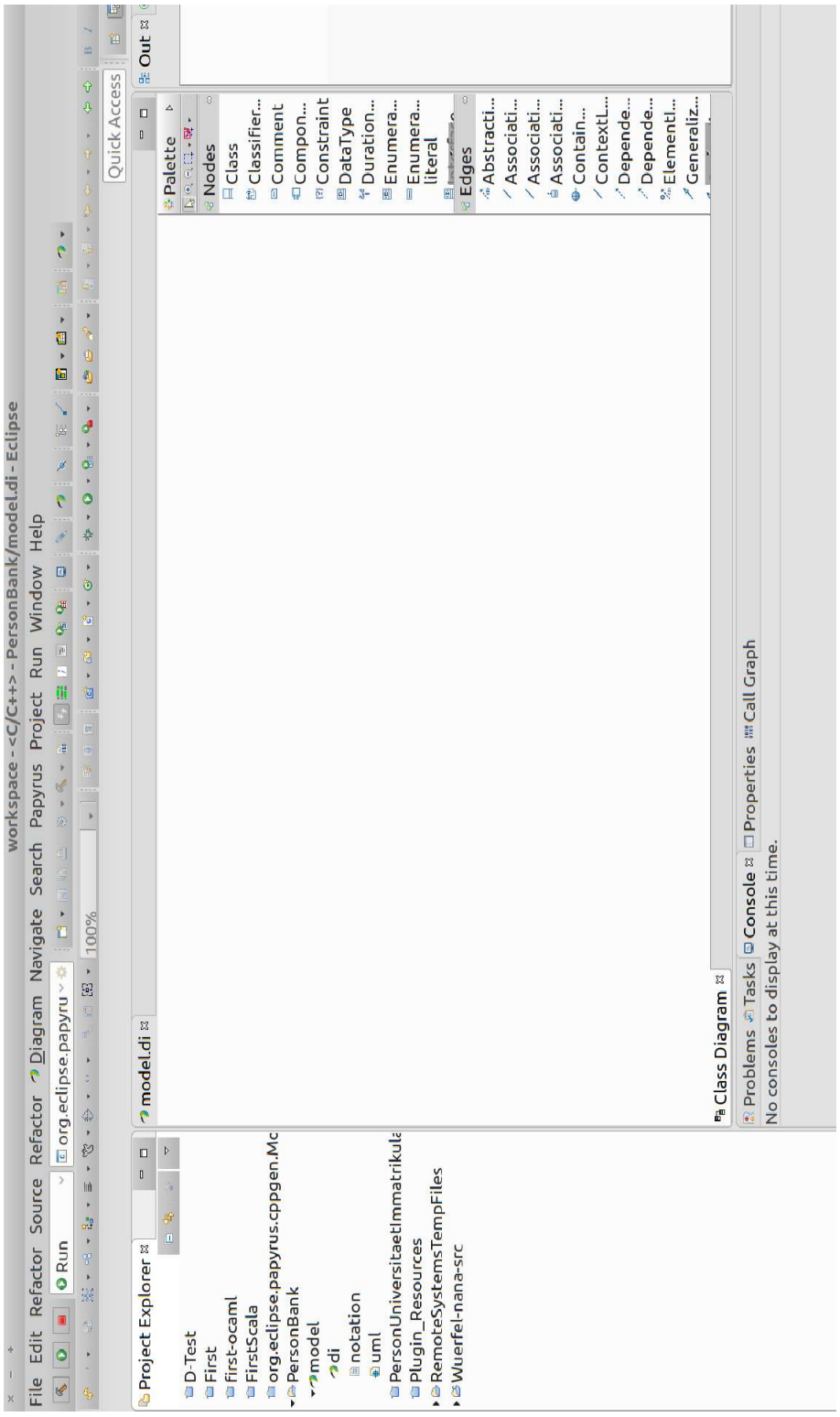


(ClassDiagram, A UML model with basic primitive types) und dem `RootElement` *Model*:



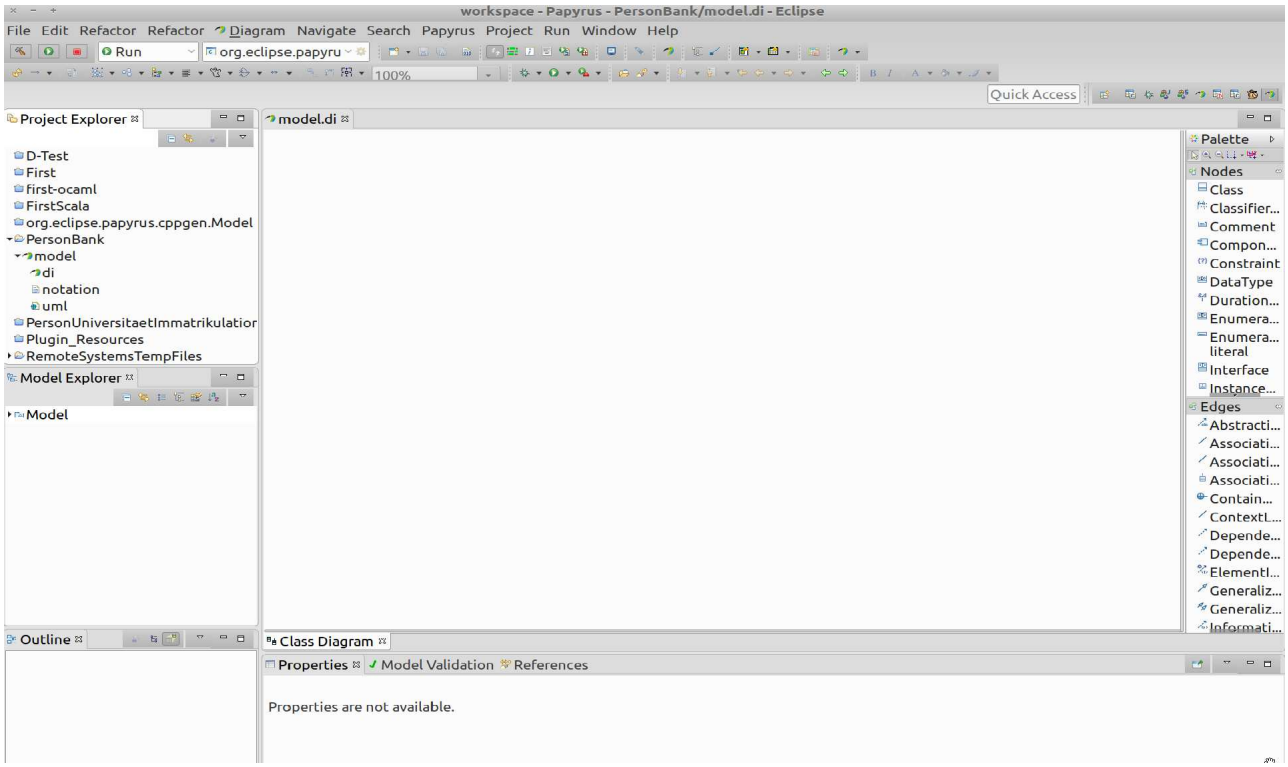
(Model, Finish).

Das neue Projekt muß nur noch in der Papyrus-Perspective



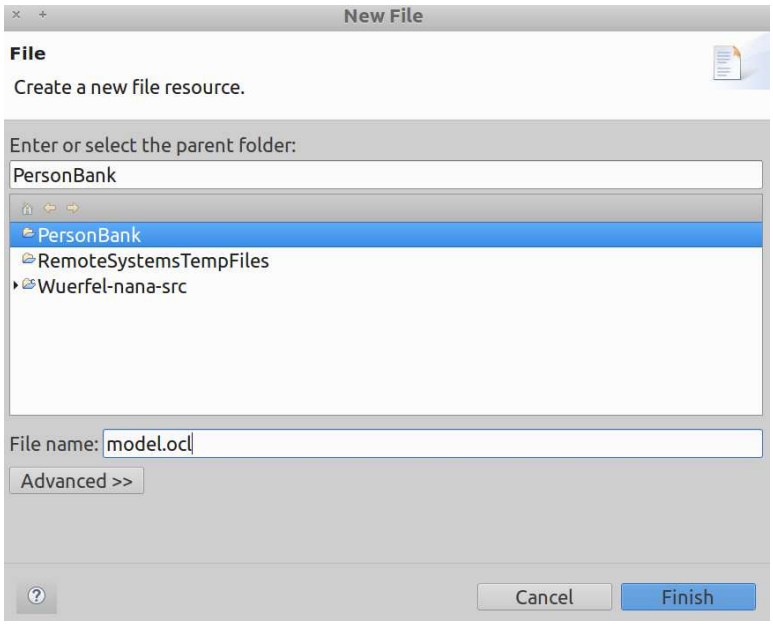
betrachtet werden:

```
Windows
  Perspective
     Open Perspective
       Other
         Papyrus
```

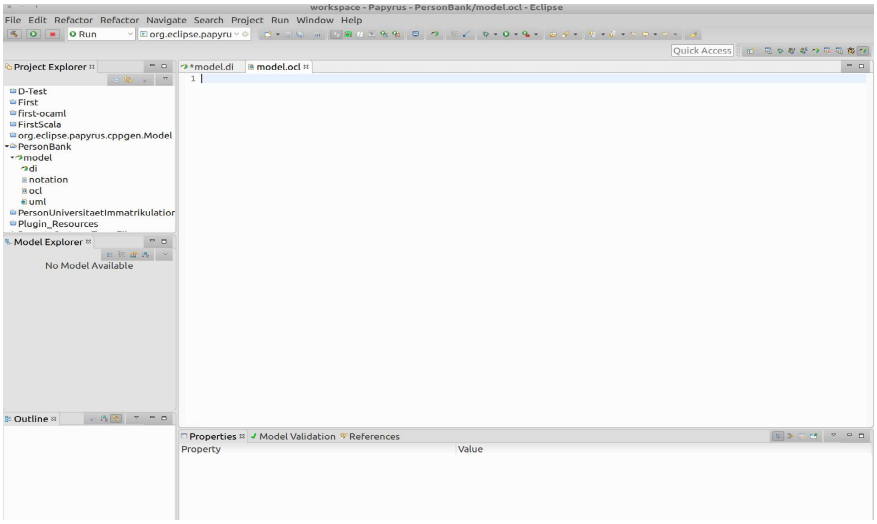


Erstellen Sie mit Hilfe der Palette/Auswahlfeld Nodes eine Klasse `Person` mit einem Attribut (`Property`) `name: Integer`:

Erzeugen Sie im Papyrus-Project `PersonBank` eine Datei (`NewFile`) `model.ocl`:



Statten Sie diese Datei mit ersten OCL-Constraints aus:



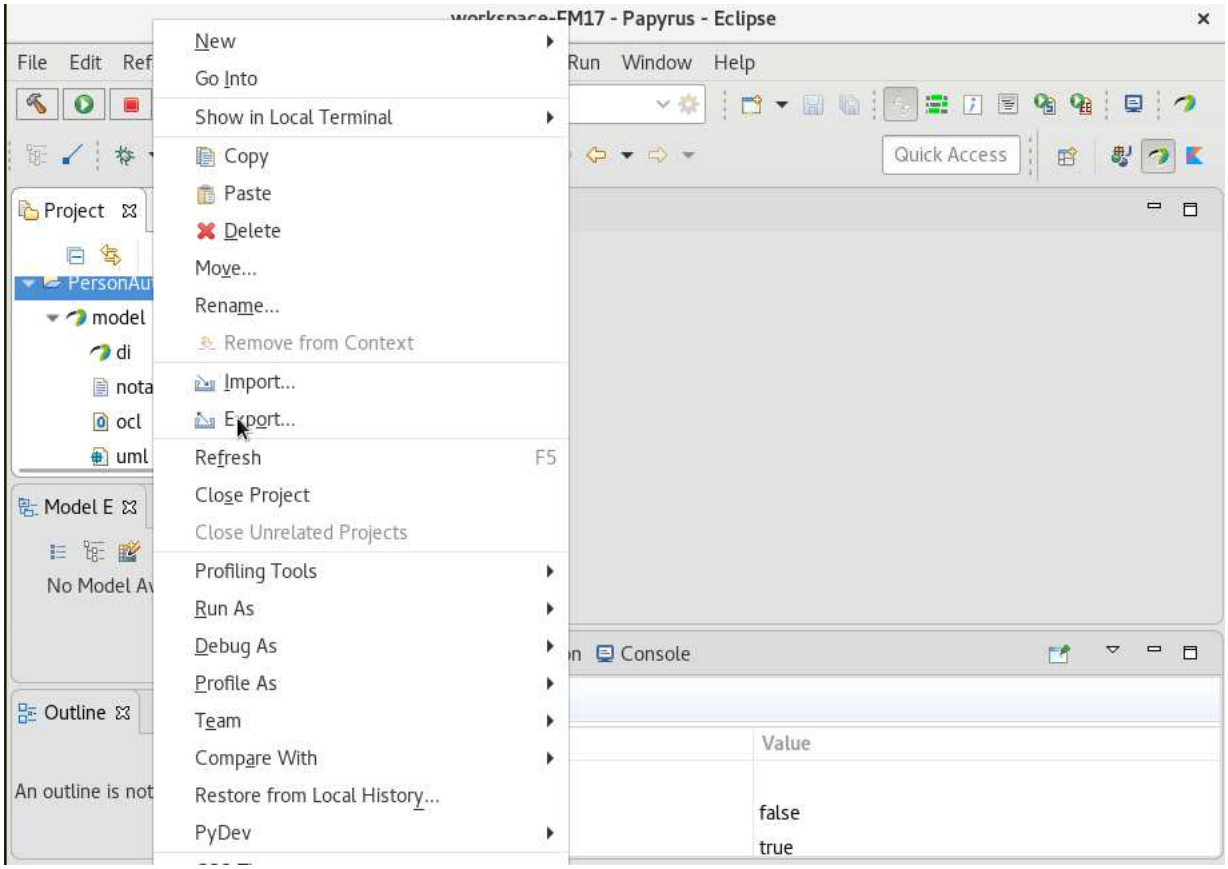
Importieren Sie das UML-Modell:

Das Attribut `name` der Klasse `Person` darf nicht leer sein:

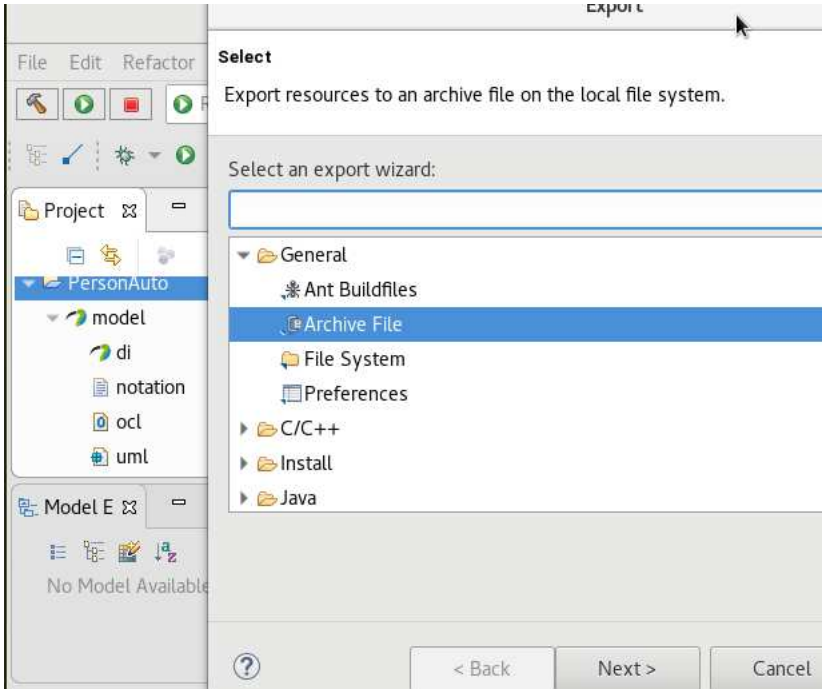
⋮

A.8. Benutzungsanleitung: Export eines Papyrusprojekts in eine Archivdatei (.zip oder .tar.gz)

Klicken Sie im ProjektExplorer auf das zu exportierende PapyrusProjekt, öffnen Sie das zu diesem Projekt gehörige PopUp-Menü (reche Maustaste) und wählen sie den Menü-Punkt **Export...** an:

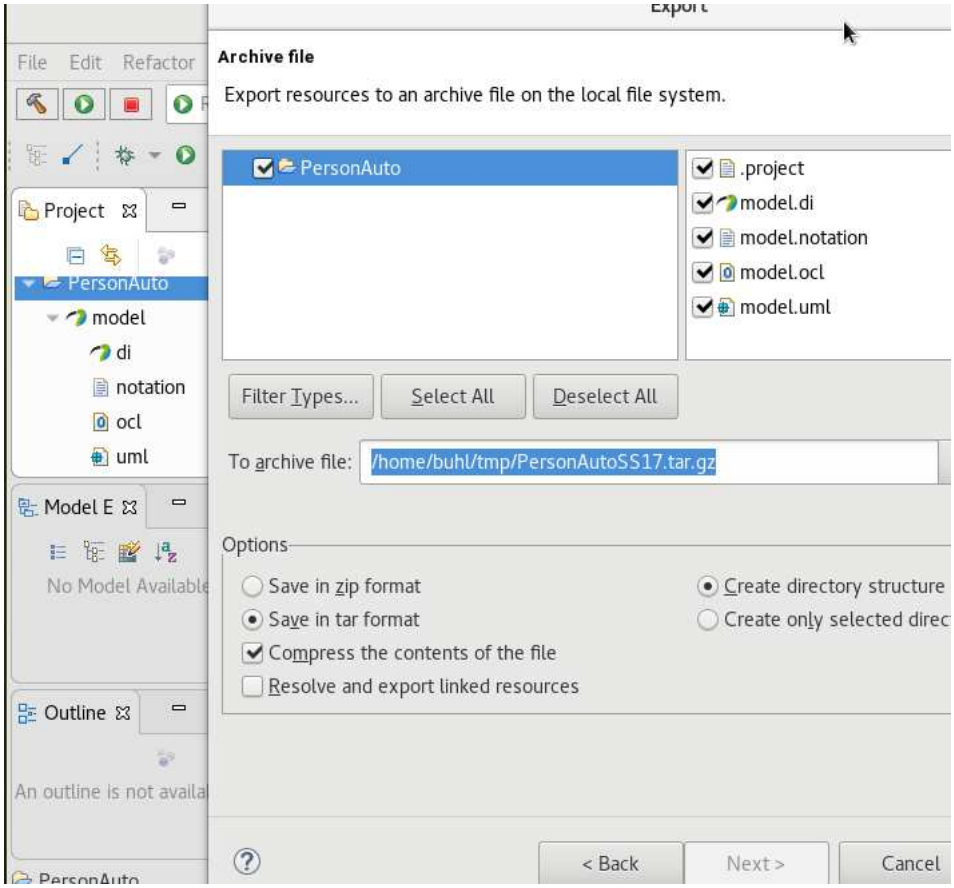


Selektieren Sie unter **General** die Auswahl **Archive File**



und klicken Sie auf **Next**.

Select All, Angabe des Dateinamens (voller Pfad) und Wahl der Optionen **Save in tar/zip format Compress the Contents of the file** sowie **Create directory structure**

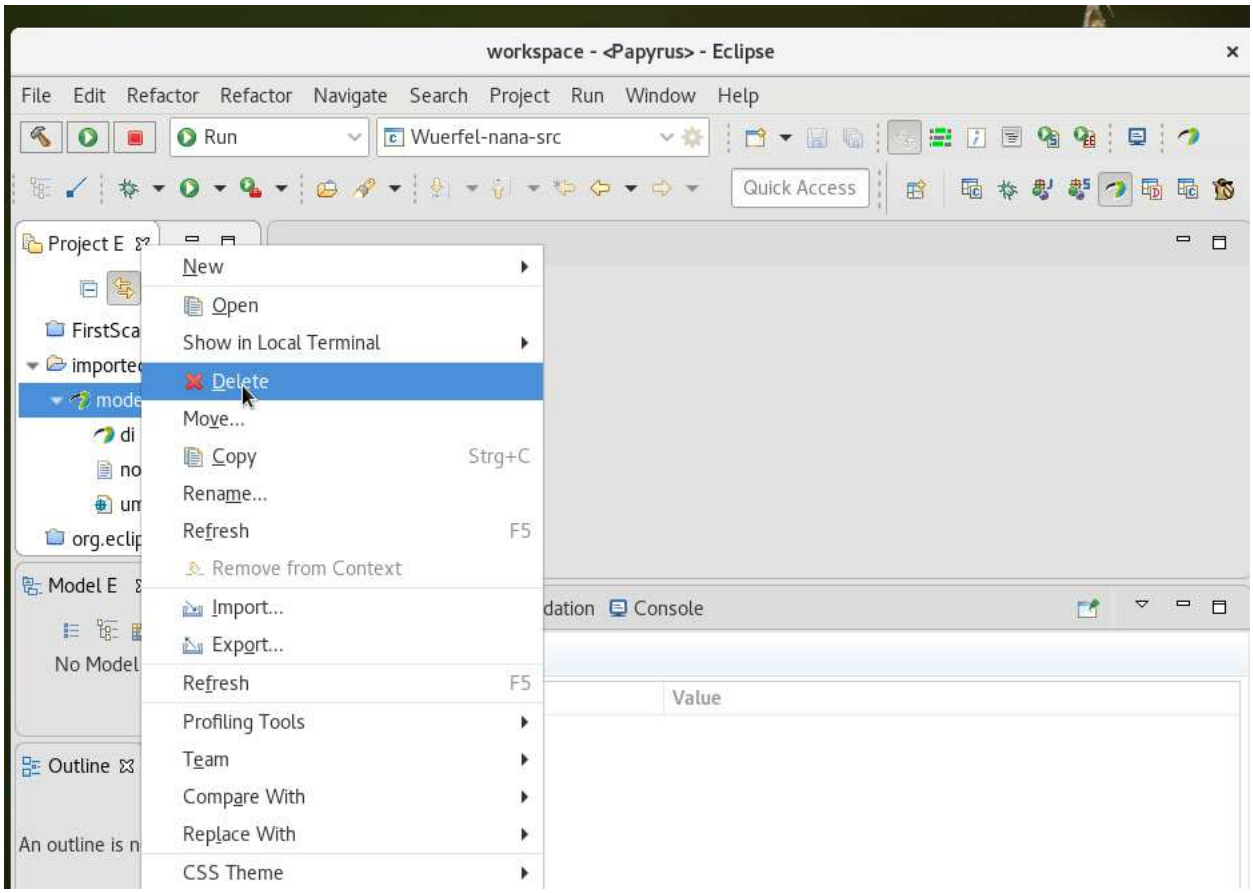


mit anschließendem **Finish** führt den Export aus.

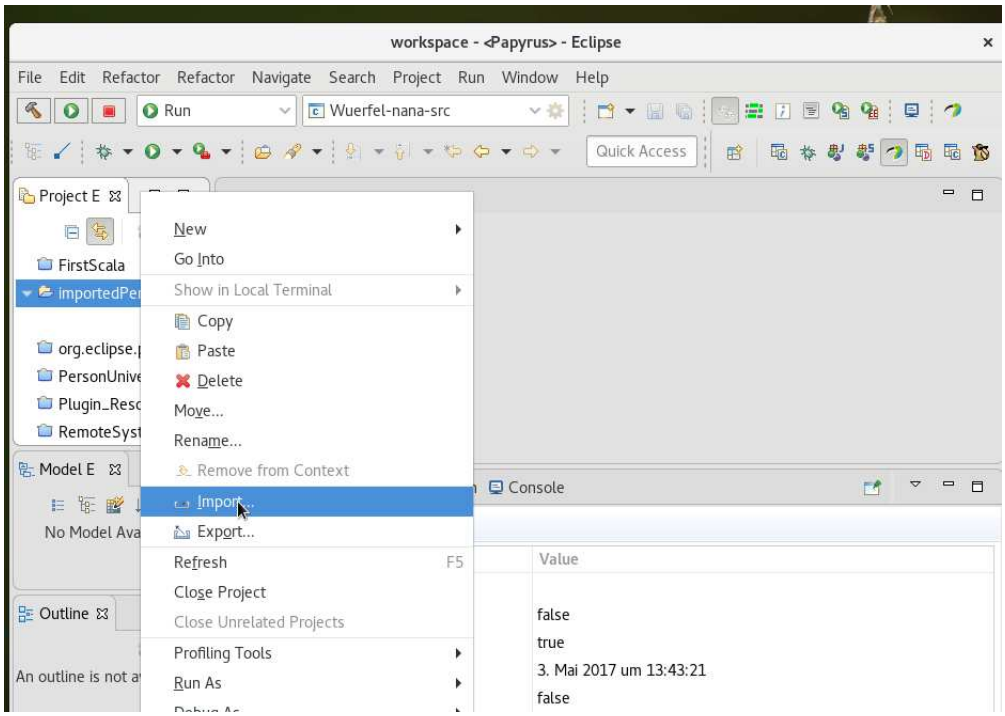
A.9. Benutzungsanleitung: Import eines früher exportierten Papyrusprojekts

Um zum Beispiel das archivierte Papyrusprojekt `personAuto03052017.tar.gz` in eclipse zu importieren, starten Sie eclipse und erzeugen ein Papyrus-Projekt, in das Sie `personAuto03052017.tar.gz` importieren können:

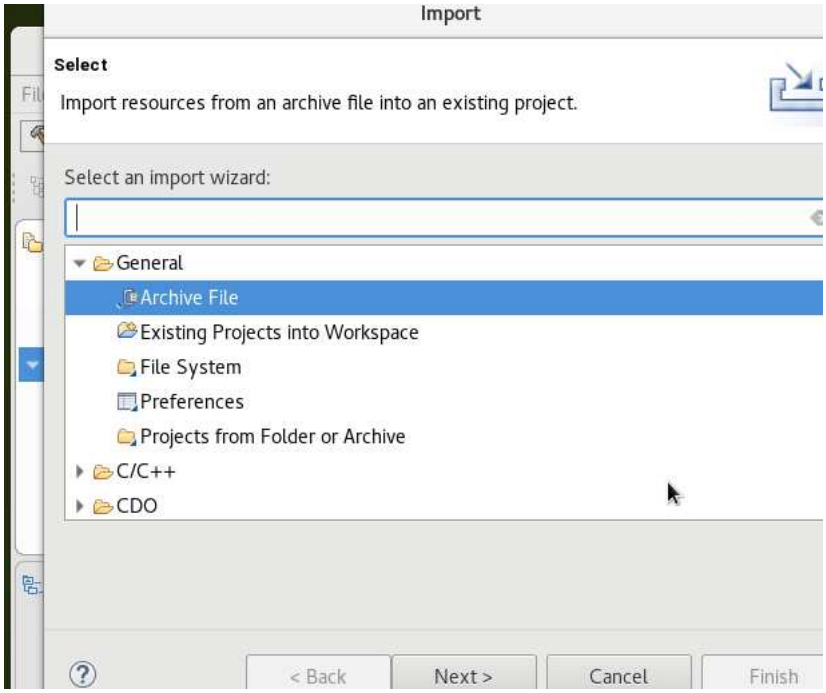
Wählen Sie die unnötigen drei model.*-Dateien an und löschen Sie sie:



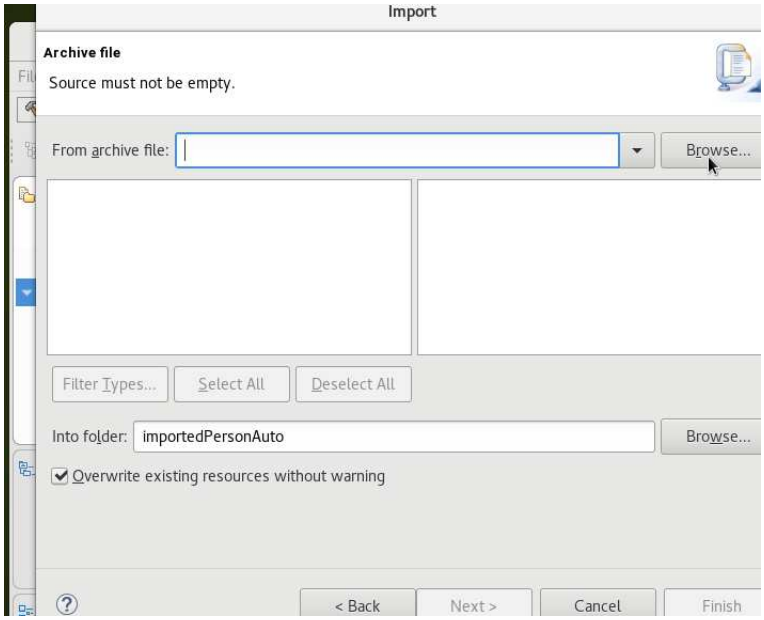
Wählen Sie `Import`



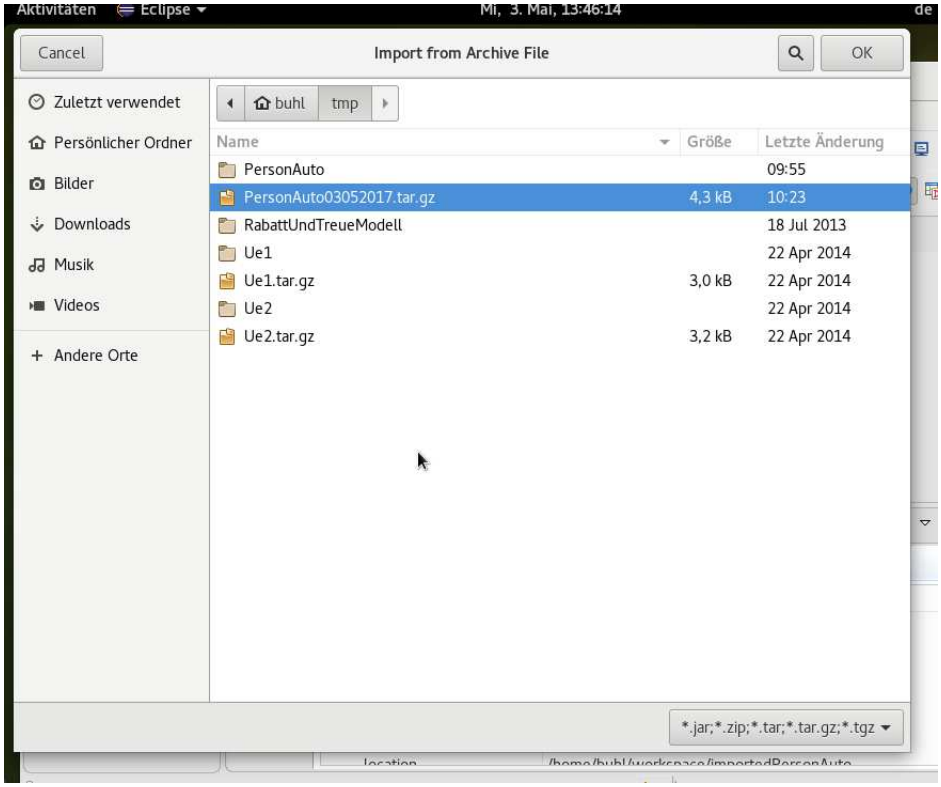
und `Achive File` an



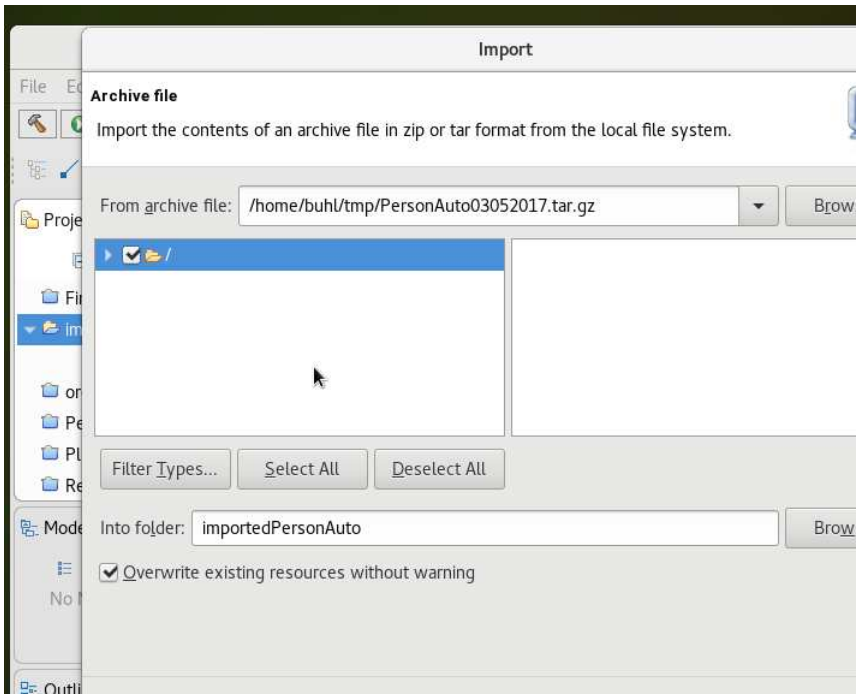
Im Quell-Auswahlfenster



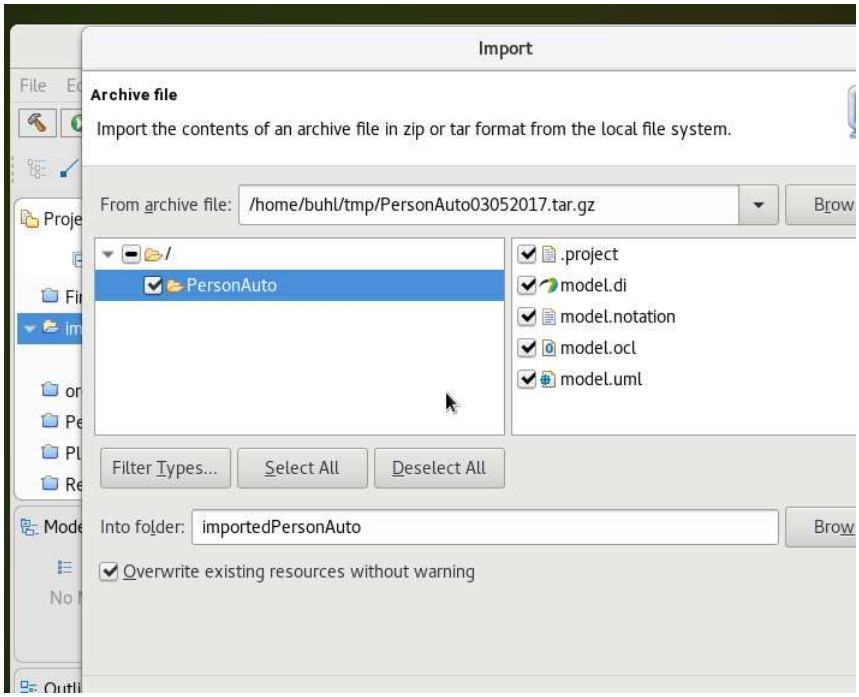
mittels `Browse`



wird das zu importierende Archiv ausgewählt und durch `OK` bestätigt.

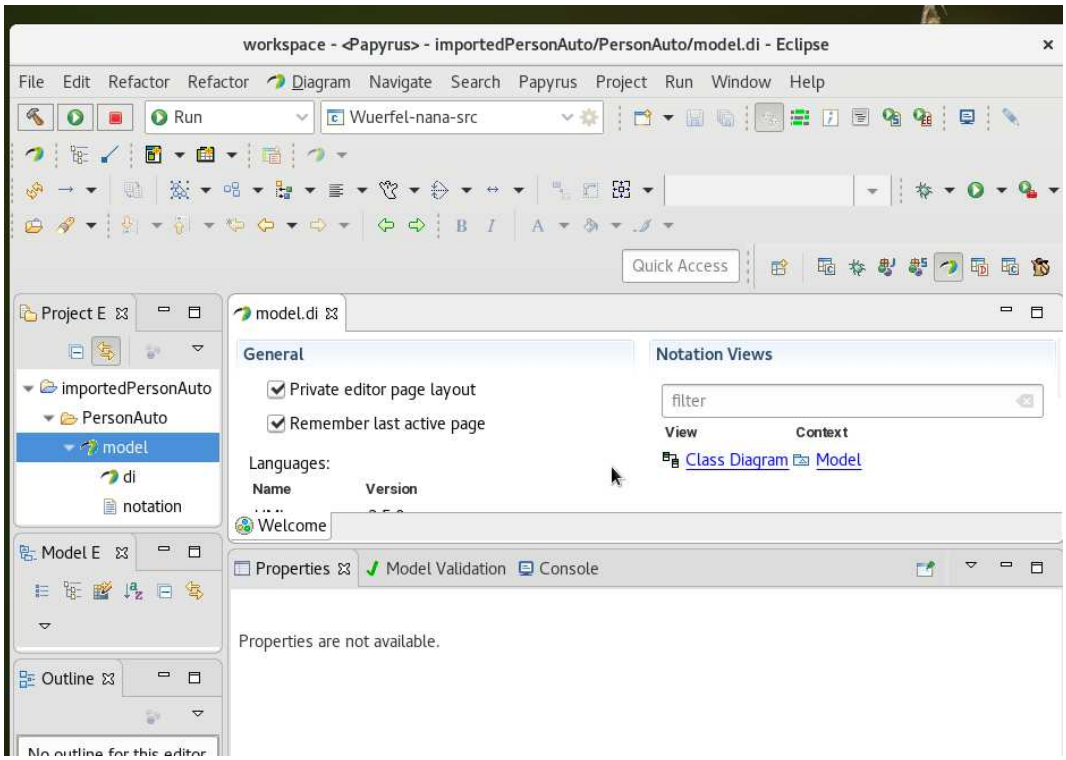


Durch Klicken auf `/`

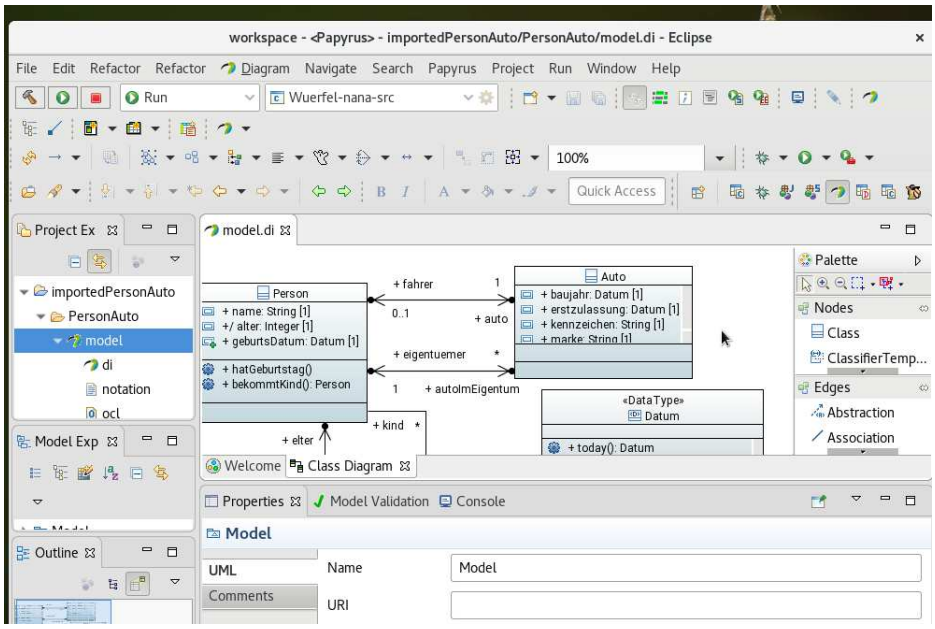


und dann z. B. auf `PersonAuto` erscheint der Inhalt des archivierten Projekts im rechten Fenster und der Zielordner kann ausgewählt werden.

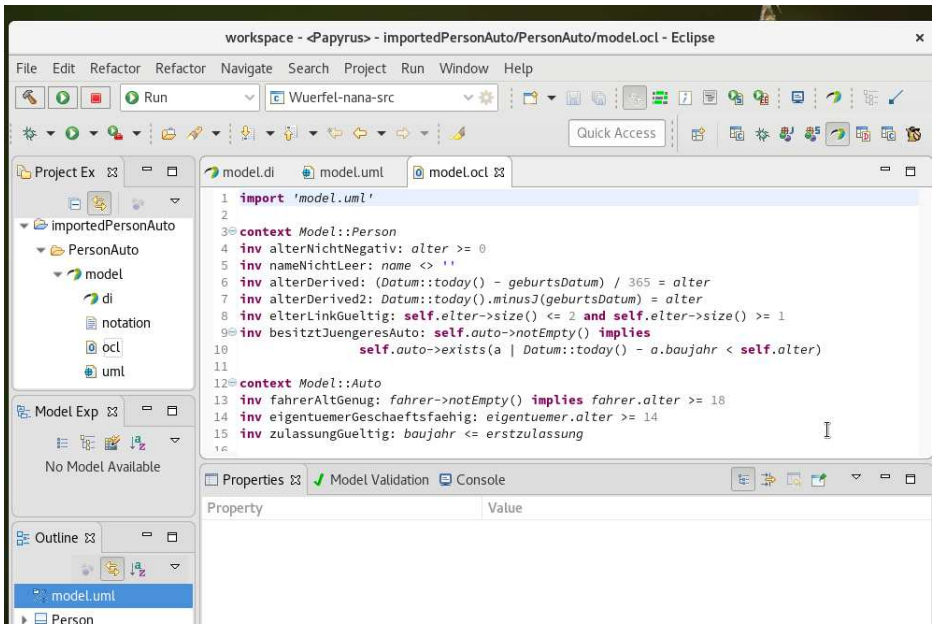
Das Projekt enthält den importierten Ordner `PersonAuto` mit den tt .di, .uml, .ocl .notation-Dateien.



Das Öffnen der `di-Datei` geschieht durch Anklicken von `Class Diagram`



Die `model.*`-Dateien können über den `Open`-Eintrag des zugehörige Kontextmenüs im Projektbrowser geöffnet werden:



B. Quelloffenes Eclipse mit UML2.5/OCL2.4-Tools/CDT zur C++ Programmentwicklung

Hilfsmittel (Tools) zur modernen Entwicklung von C++-Anwendungen:
Verfügbar (vorinstalliert) auf dem IT-Ausbildungsclustern (l101, ...) als `eclipse-papyruso3`.

Hinweis zur Installation auf dem eigenen Linux-Notebook:

IDE für C++-Programmierung: Eclipse Neon Modeling mit UML, OCL, CDT, Linux Tools, ...

Eclipse Downloads (wähle nach Klick auf „Download Packages“):

Installiere *Eclipse Modeling Tools*, (zur Zeit die Version Oxygen 3) durch Download der Datei
`eclipse-modeling-oxygen-3-linux-gtk-x86_64.tar.gz`
von `http://www.eclipse.org/downloads/`, installiere sie mittels:

```
~/Downloads> ls ec*
username users 405705165 15. Feb 10:56 eclipse-modeling-oxygen-3-linux-gtk-x86_64.tar.
    gz
~/Downloads> gunzip eclipse-modeling-oxygen-3-linux-gtk-x86_64.tar.gz
~/Downloads> ls -al ec*
-rw-r--r-- 1   eclipse-modeling-oxygen-3-linux-gtk-x86_64.tar
```

wechsle ins Zielverzeichnis für selbstinstallierte Software (etwa `$HOME/sw`) und entpacke `eclipse` dorthin:

```
~/sw> tar xf ~/Downloads/eclipse-modeling-oxygen-3-linux-gtk-x86_64.tar
~/sw> ls -al
drwxr-xr-x  9 username users  4096  8. Dez 09:10 eclipse
~/sw> mv eclipse eclipse-modeling-oxygen-3-linux-gtk-x86_64
~/sw> cd ~/bin
```

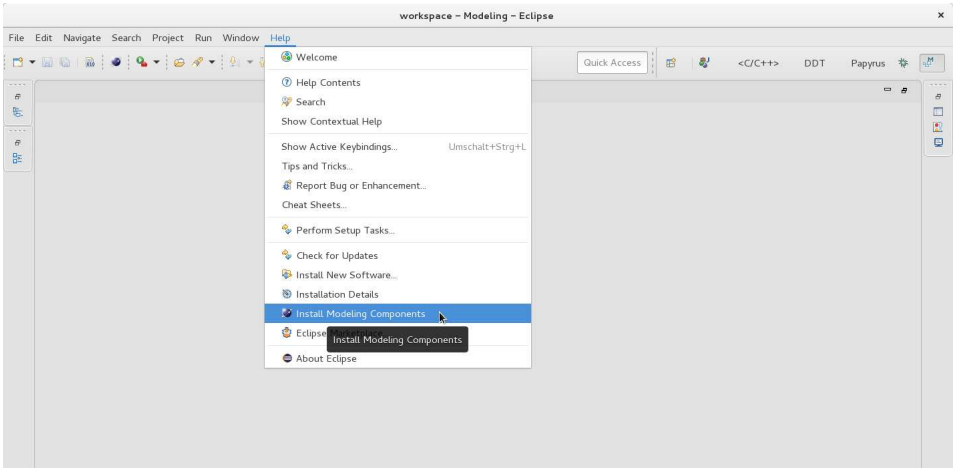
Erzeuge in `S$HOME/bin` ein Startskript `$HOME/bin/eclipse-papyruso3` mit dem Inhalt:

```
#!/bin/sh
#
$HOME/sw/eclipse-modeling-oxygen-3-linux-gtk-x86_64/eclipse $*
```

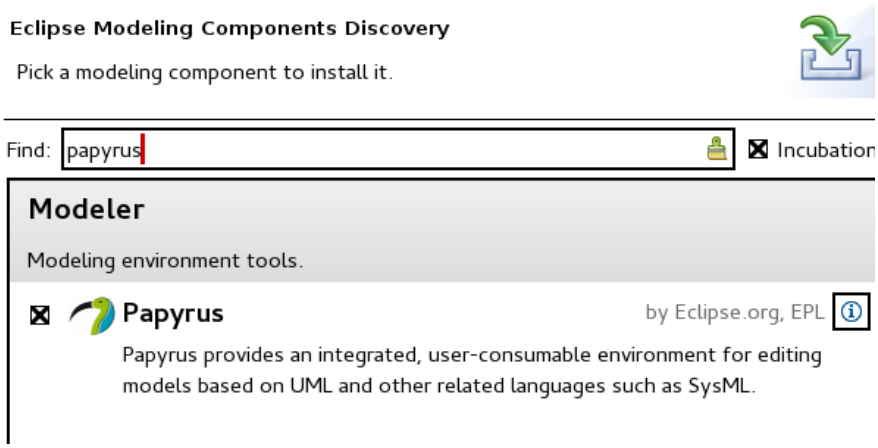
und gib ihm Ausführbarkeitsrechte:

```
~/bin> chmod 755 $HOME/bin/eclipse-papyruso3
```

(Eclipse oxygen 3 = 4.7.3)
ergänze dann unter `Help, Install Modelling Components`



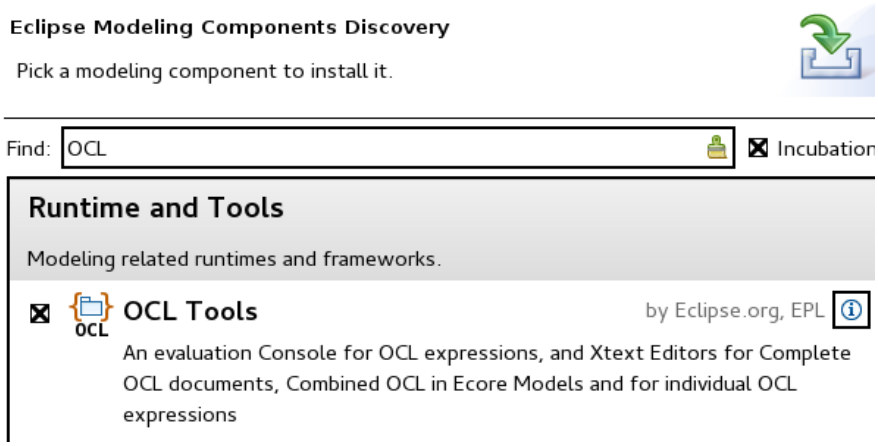
das UML-Tool[1] `Papyrus 3.3.0` (für die Erstellung von UML-Modellen):



⋮

sowie die `OCL Tools 6.3.0` zur Erstellung von Constraints an UML-Komponenten (formale Codeverträge):

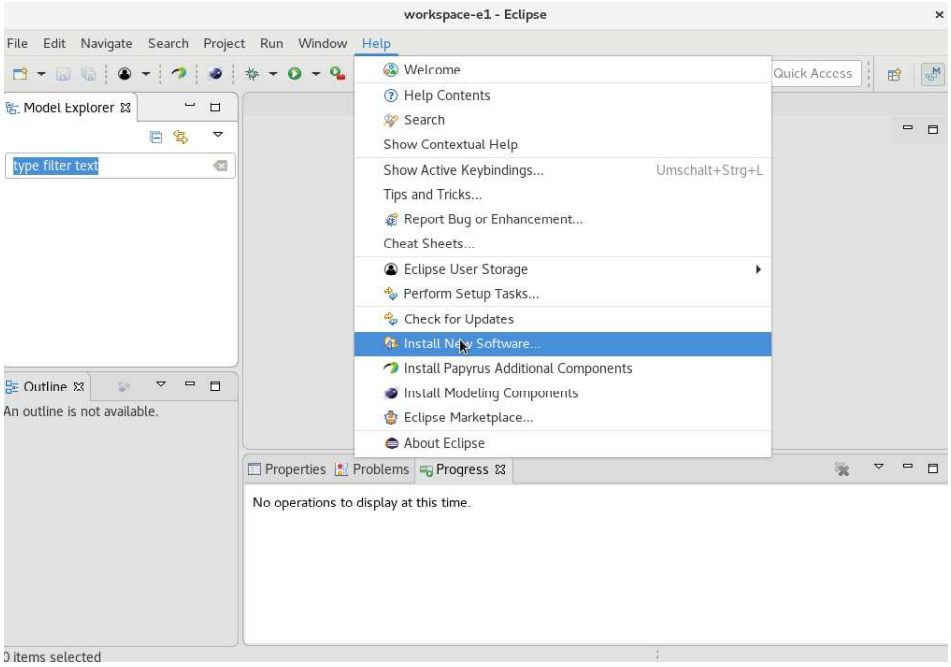
[1] Anwahl von Papyrus, Finish, Select All, Next, Next, I accept ... licence agreement, Finish, Anwahl von Eclipse.org Certificate, Accept selected ,restart now: Yes



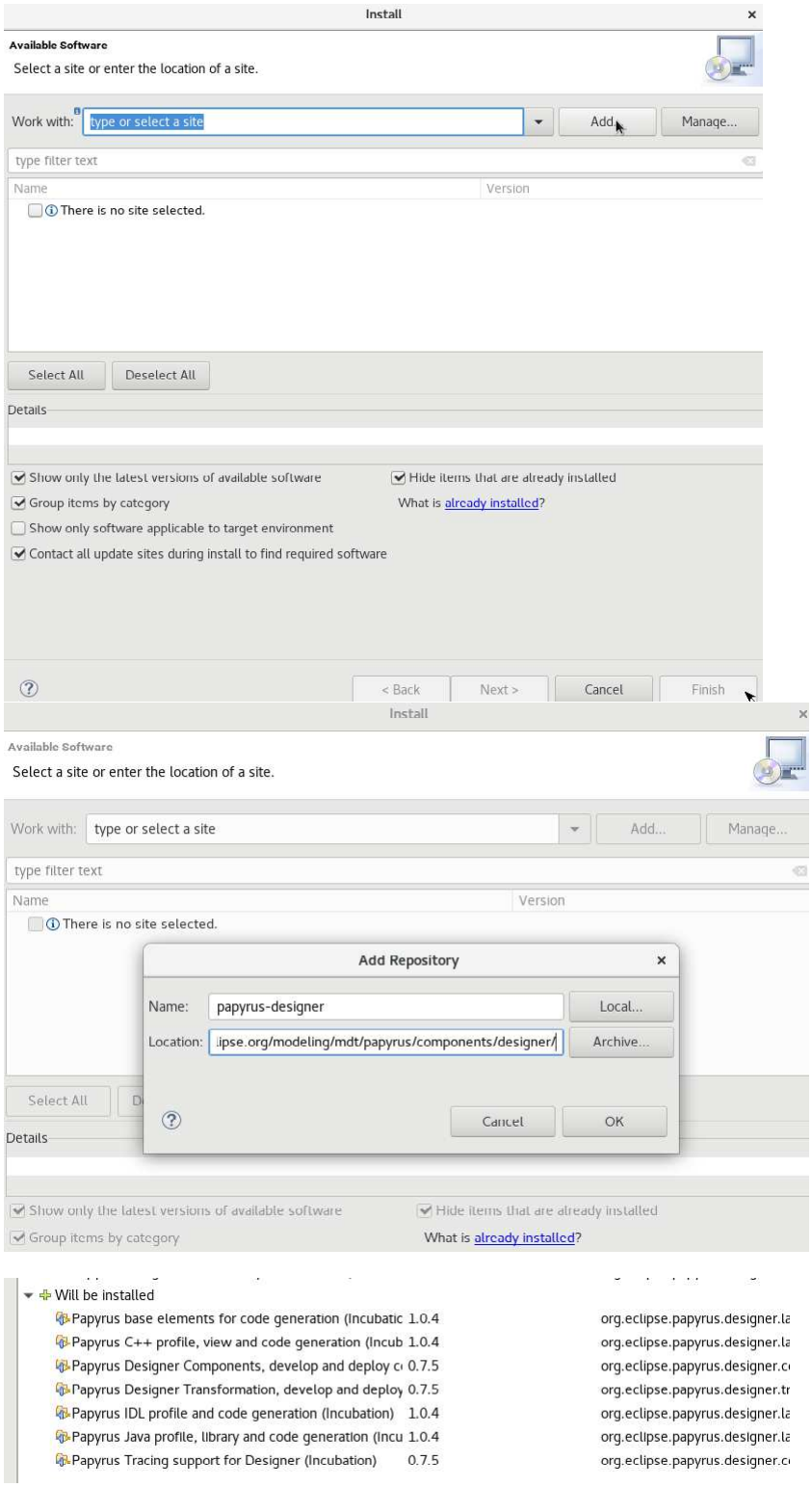
Aktualisiere Papyrus unter `Help, Check for updates`, falls nötig.

Ergänze unter `Help, Install New Software, Add`[2] den C/C++-Designer (0.7.6, zur Codeerzeugung unter CDT und JAVA):

```
papyrus-designer
http://download.eclipse.org/modeling/mdt/papyrus/components/designer/
```



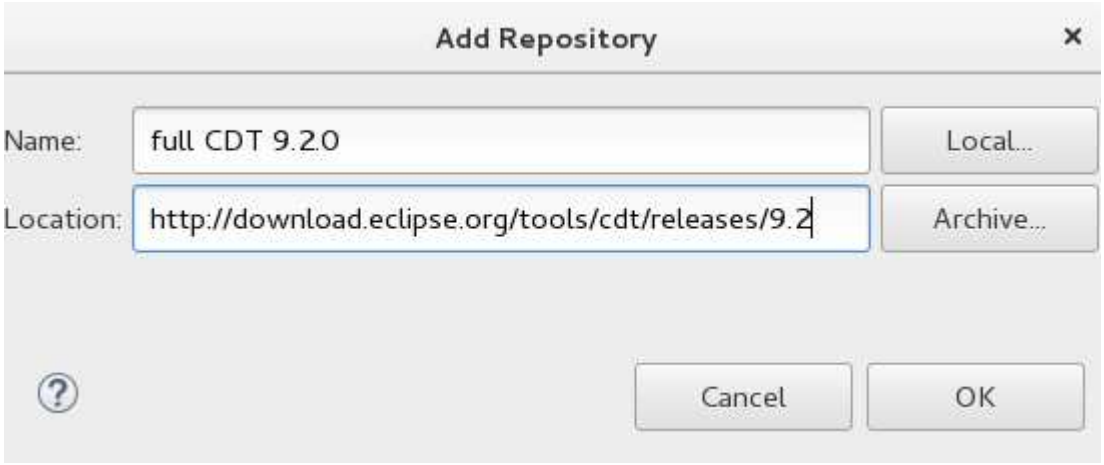
[2] ok, select All, next, next, I accept ... license agrements, finish, Eclipse.org Certificate, selext All, accept selected, restart now



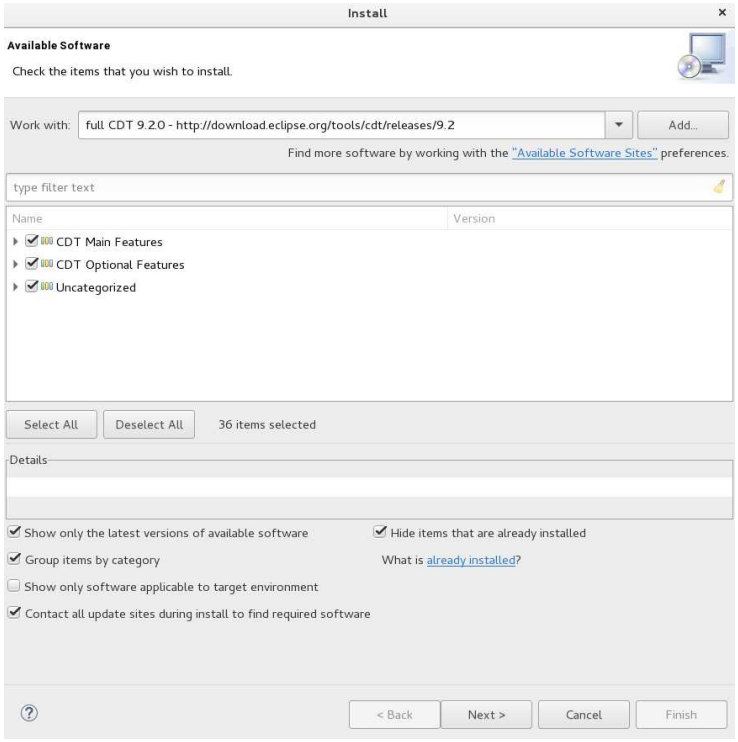
Ergänze unter `Help, Install New Software, Add`

`full CDT 9.4.3,`

`http://download.eclipse.org/tools/cdt/releases/9.4`

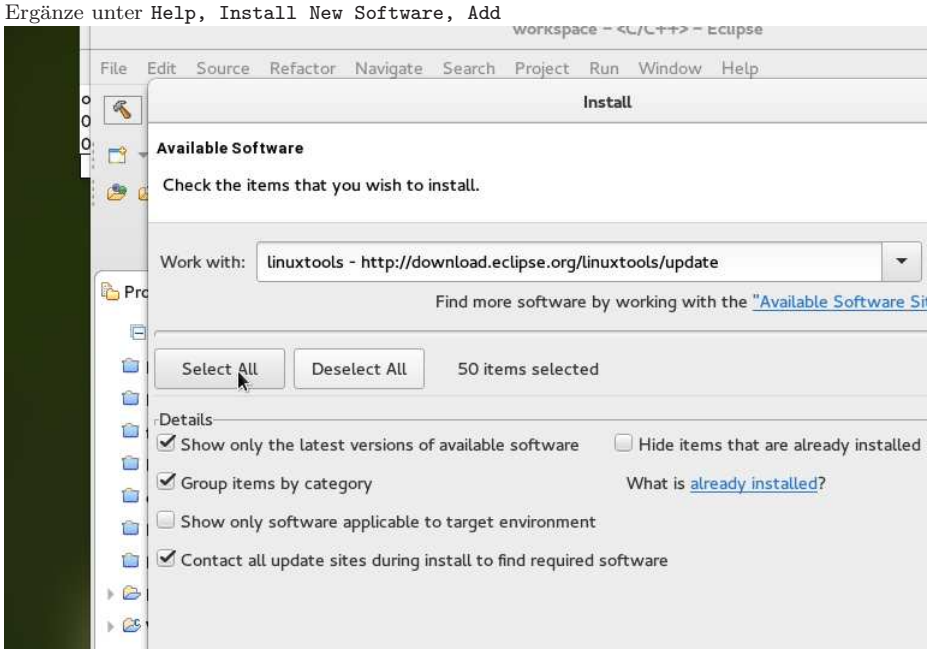


und wähle mittels `Select All` das volle CDT-Plugin aus

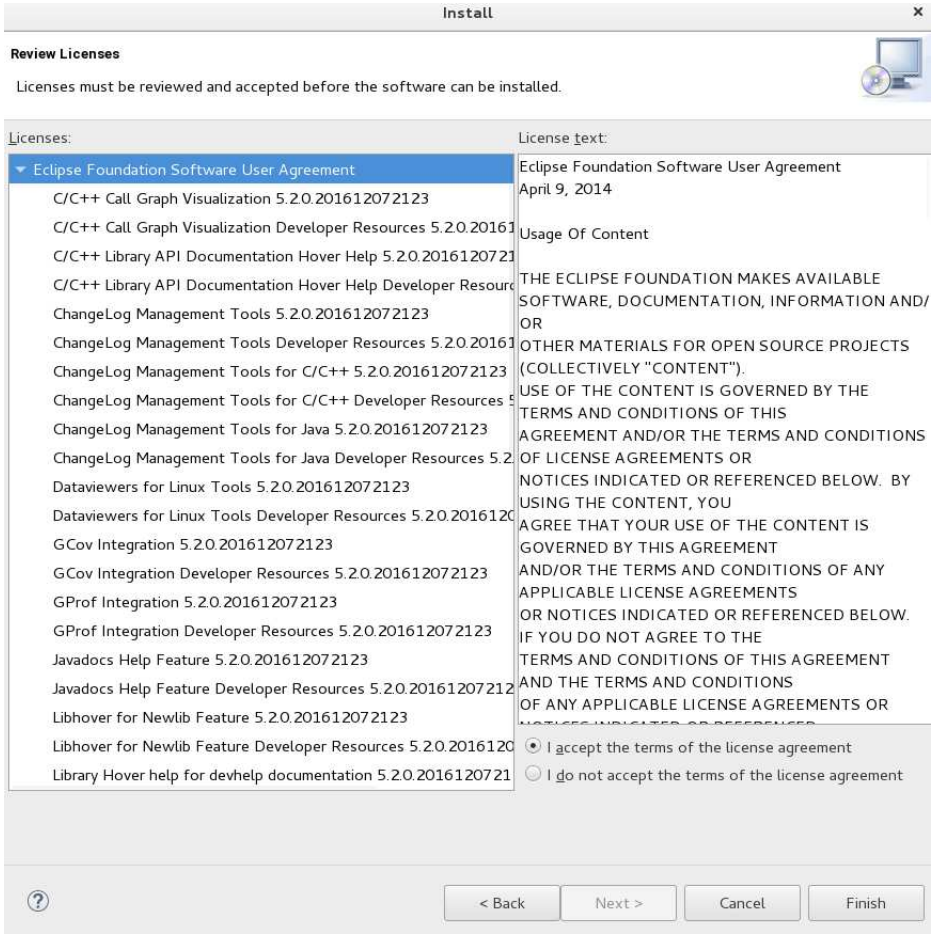


und installiere es,

Ergänze unter `Help, Install New Software, Add`



das Linuxtools-Repositorium (6.2.1) und wähle `Select All, Next` an. Es erscheint eine Übersicht aller zugehöriger Komponenten:

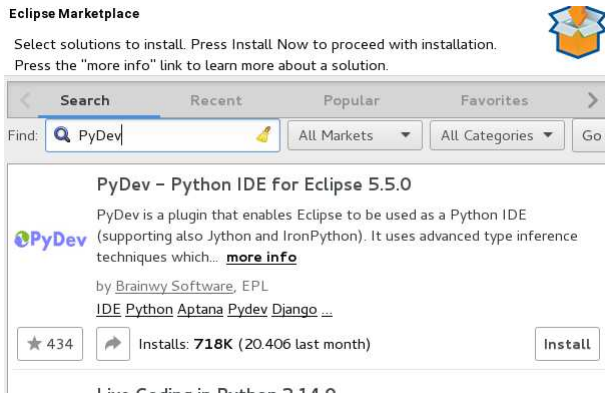


Die Linuxtools bieten Eclipse-Integration (Plugins) qualitätssteigernder Tools [3] für die C++-Entwicklung:

- Call Ggraph
- ChangeLog
- Dataviewers
- Gcov (oder lcov)
- GProf
- Man Page Viewer
- OProfile
- Perf
- ...
- RPM
- System Tap
- Valgrind

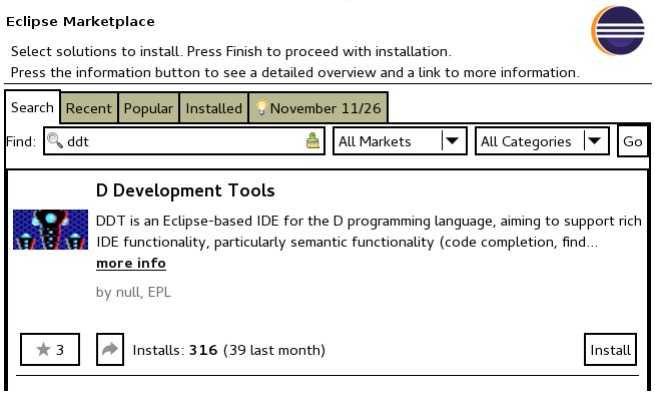
[3] diese müssen natürlich zuvor in Linux installiert sein

Bei Bedarf kann man unter `Help, Eclipse Marketpace` schließlich noch `Python`-Entwicklungsumgebung PyDev (6.3.2)

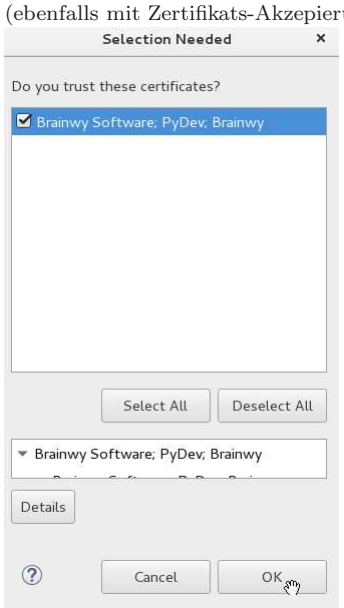


(Confirm, SelectAll des Zertifikats und OK)

und `D-Unterstützung ddt 1.0.3` (sofern auf Ihrer Maschine `dmd` und `dub` installiert sind)



(ebenfalls mit Zertifikats-Akzepierung:



)

und ... hinzuinstallieren.
(Vorinstalliert auf den Ausbildungsclustern der Fachgruppe als: `eclipse-papyruso3` sind zum Beispiel zusätzlich: `Scala`

4.7.x, Kotlin 0.8.2, ocaIDE 1.2.21 für Ocaml sowie Cute 5.2 für C++ Unittests.
Empfehlenswert sind darüber hinaus: EclipseFP 2.6.4 für Haskell, cppcheclipse 1.1.0 (Plugin für cppcheck, statische CodeAnalyse), ...)

Getting started with CDT development
CDT Documentation, Tutorials, ...
Eclipse CDT (C/C++ Development Tooling)
Eclipse für C/C++-Programmierer, dritte Auflage

Hinweise für die Nutzung der gnu/gcc-Tools unter Windows:
Cygwin für Windows, Cygwin
mingw-64
Windows 10 Linux-Subsystem
C++17 Features In Visual Studio 2017 Version 15.3 Preview

Microsoft Imagine (früher MSDNAA): VisualStudio 201x für Windows