Softwaretechnik
Model Driven Architecture
Einführung — OCL

Prof. Dr. Peter Thiemann

Universität Freiburg

11.07.2008

Einführung MDA

Material

- Thomas Stahl, Markus Völter. Modellgetriebene Softwareentwicklung. Dpunkt Verlag, 2. Auflage. 2007.

- Anneke Kleppe, Jos Warmer. MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson. 2003.
- Stephen J. Mellor, Axel Uhl, Kendall Scott, Dirk Weise. MDA Distilled: Solving the Integration Problem with the Model Driven Architecture. Pearson. 2004.

Was ist MDA?

- MDA = Model Driven Architecture
  - auch: MD (Software/Application) Development, Model Based [Development/Management/Programming]
  - Model Driven Engineering, Model Integrated Computing
- Initiative der OMG (Warenzeichen)
  - Object Management Group: CORBA, UML, ...
  - offenes Firmenkonsortium (ca. 800 Firmen)
- Ziel: Verbesserung des Softwareentwicklungsprozesses
  - Interoperabilität
  - Portabilität
- Ansatz: Verlagerung des Entwicklungsprozesses von der Codeebene auf die Modellebene
  - Wiederverwendbarkeit von Modellen
  - Transformation von Modellen
  - Codeerzeugung aus Modellen

Ziele von MDA
Höherer Abstraktionsgrad

Portabilität und Wiederverwendbarkeit

- Entwicklung abstrahiert von Zielplattform
- Technologieabbildung in wiederverwendbaren Transformationen
- Neue Technologie $\Rightarrow$ neue Transformation

Interoperabilität

- Systeme sind plattformübergreifend
- Informationsübertragung zwischen Plattformen durch *Brücken*
- Nebenprodukt von Modelltransformation

Ziele von MDA
Modelle und Modelltransformation

Produktivität

Jede Phase der Entwicklung leistet direkten Beitrag zum Produkt, nicht nur die Implementierung.

Dokumentation und Wartung

- Änderungen durch Änderung der Modelle
- Modelle sind Dokumentation $\Rightarrow$ Konsistenz
- Trennung von Verantwortlichkeit
- Handhabbarkeit von Technologiewandel

Spezialisierung

- Geschäftsprozesse
- Technologien

Ein Modellbegriff
(nach Herbert Stachowiak, 1973)

Repräsentation

Ein Modell ist Repräsentation eines Original-Objekts.

Abstraktion

Ein Modell muss nicht alle Eigenschaften des Original-Objekts erfassen.

Pragmatismus

Ein Modell ist immer zweckorientiert.

Ein Modellbegriff
(nach Herbert Stachowiak, 1973)

Repräsentation

Ein Modell ist Repräsentation eines Original-Objekts.

Abstraktion

Ein Modell muss nicht alle Eigenschaften des Original-Objekts erfassen.

Pragmatismus

Ein Modell ist immer zweckorientiert.

- Modellierung erzeugt eine Repräsentation, die nur die für einen bestimmten Zweck relevanten Eigenschaften beinhaltet.

Formale Modelle

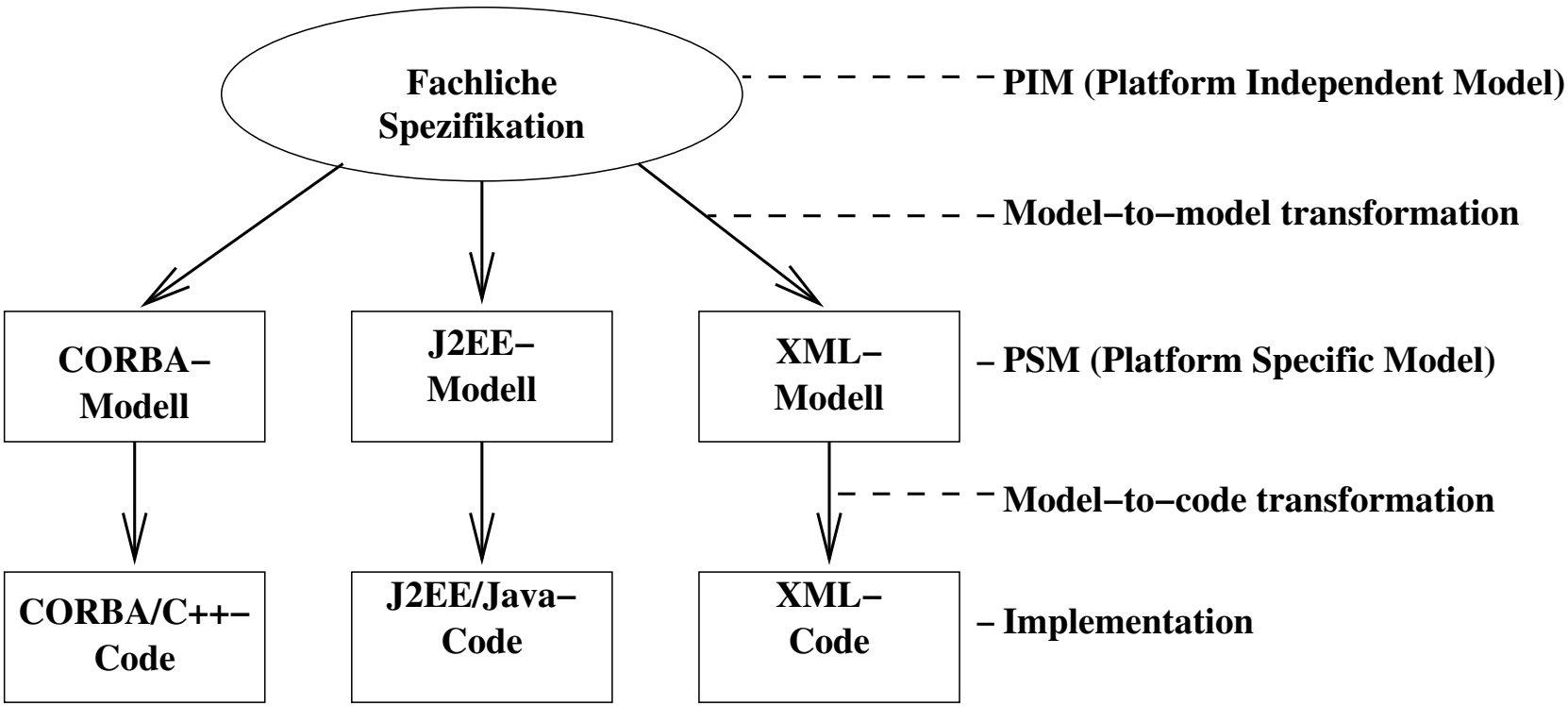
Modelle, die in einer formalen Sprache verfasst sind

- Textuell: definiert durch Grammatik, BNF, o.ä.
- Grafisch: definiert durch *Metamodell*
  - Welche Modellierungselemente?
  - Welche Kombinationen?
  - Welche Modifikationen?

Modelle, die eine formale Semantik besitzen

- Beispiel: logische Formel $\Rightarrow$ Wahrheitswert
- Beispiel: kontextfreie Grammatik $\Rightarrow$ Sprache
- Beispiel: Programm $\Rightarrow$ Programmausführung

Modelle in MDA



Modelle in MDA/2

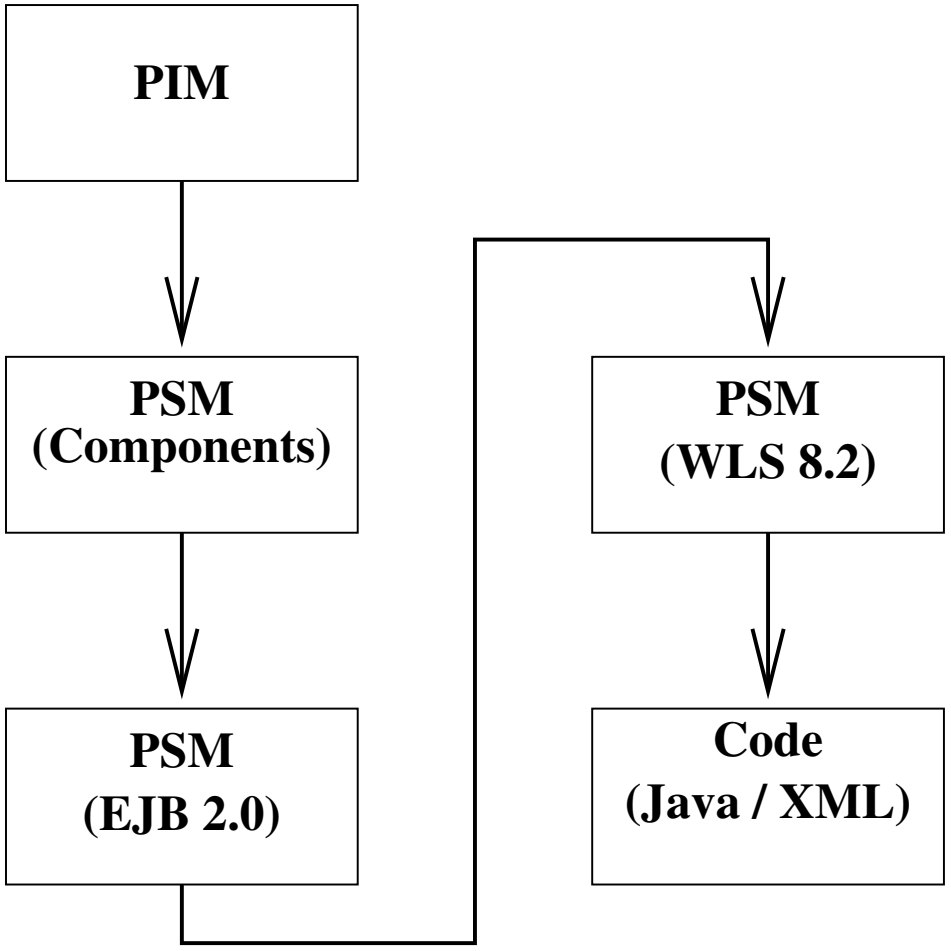
PIM vs PSM

- Relative Konzepte
- Übergang fließend
- Mehrere Modellebenen und Transformationsschritte möglich
- Rücktransformation PSM $\Rightarrow$ PIM kaum automatisierbar

Transformation

- Code ist ultimatives Modell (PSM)
- Model-to-Code ist Spezialfall

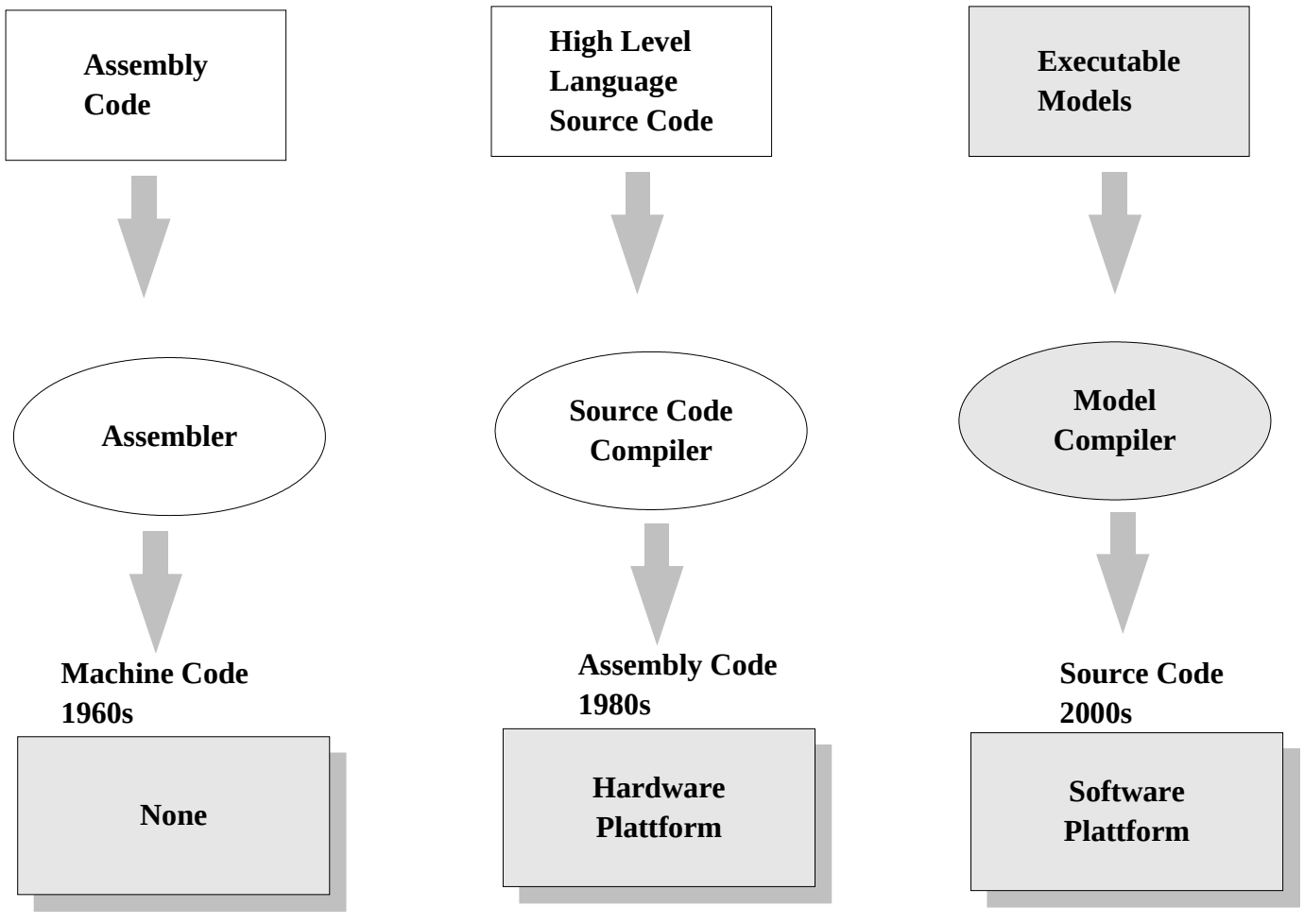
Modelle und Transformationen



Plattform

- Programmierschnittstelle, API
- Virtuelle Maschine
- Stellt verschiedene Dienste zur Verfügung
- Beispiele
  - Hardwareplattform
  - Betriebssystem $\Rightarrow$ Softwareplattform
  - Java VM $\Rightarrow$ Softwareplattform
  - EJB $\Rightarrow$ Komponentenplattform
  - CORBA, Webservices, ...
  - Anwendungsarchitektur, DSL (Domain Specific Language)

Plattformen im Beispiel



OCL

Was ist OCL?

- OCL = object constraint language
- Standard Anfragesprache UML 2
- Ausdrücke und constraints in Artefakten der Objekt-Modellierung

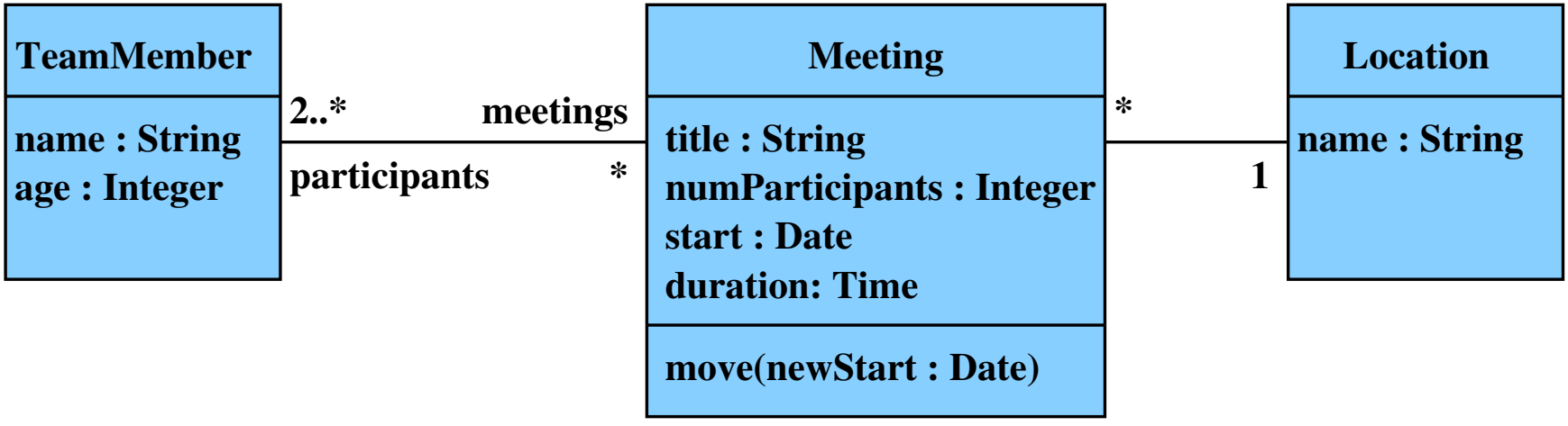
OCL/Ausdrücke und Constraints

- Ausdrücke
  - Initialwerte, abgeleitet Werte
  - Parameterwerte
  - Rumpf einer Operation (ohne Seiteneffekte $\Rightarrow$ nur Anfragen möglich)
  - mögliche Typen: `Real`, `Integer`, `String`, `Boolean`, oder Modelltypen
- Constraints schränken die möglichen Instanzen ein
  - Invariante (Klasse): Bedingung an den Zustand aller Objekte einer Klasse, die immer erfüllt sein muss
  - Vorbedingung (Operation): wahr, falls Operation anwendbar
  - Nachbedingung (Operation): muss am Ende der Operation gelten
  - Guard (Transition): Anwendbarkeit der Transition
- Auswertung bezüglich eines Schnappschusses des Instanzgraphen

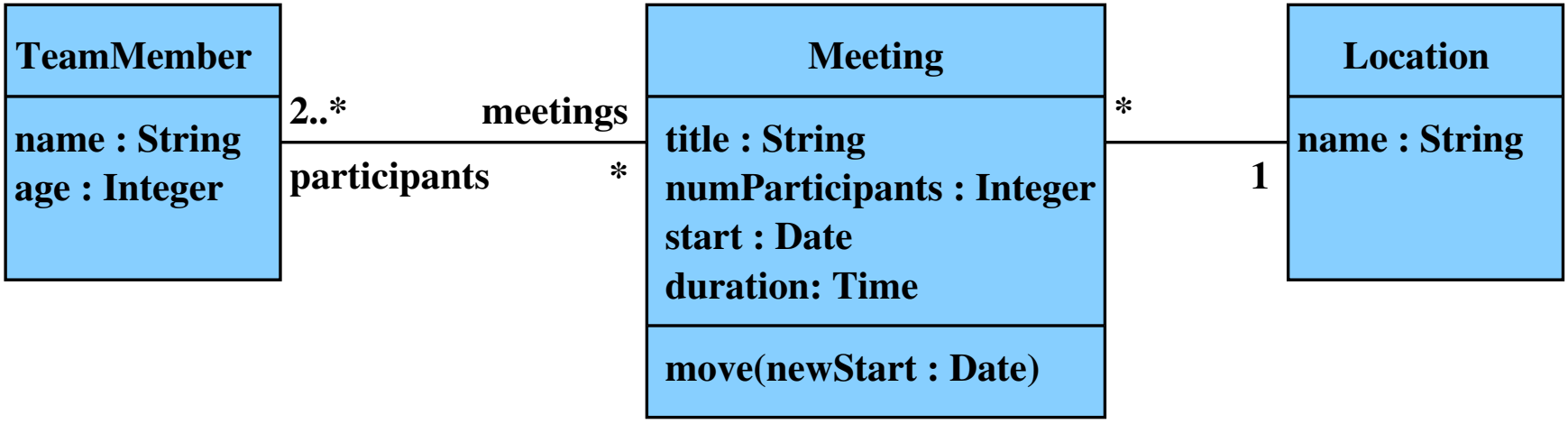
OCL/Kontext

- Jeder OCL-Ausdruck wird in einem Kontext ausgewertet.
  - Invariante: Klasse, Interface, Datatype, Komponente (ein *classifier*)
  - Vor-, Nachbedingung: Operation
  - Guard: Transition
- Kontext wird angezeigt durch
  - grafisch durch Ankleben als *Note*
  - textuell durch die `context` Syntax

OCL/Beispiel



OCL/Beispiel



- **context** TeamMember **inv:** age => 18
- **context** Meeting **inv:** duration > 0

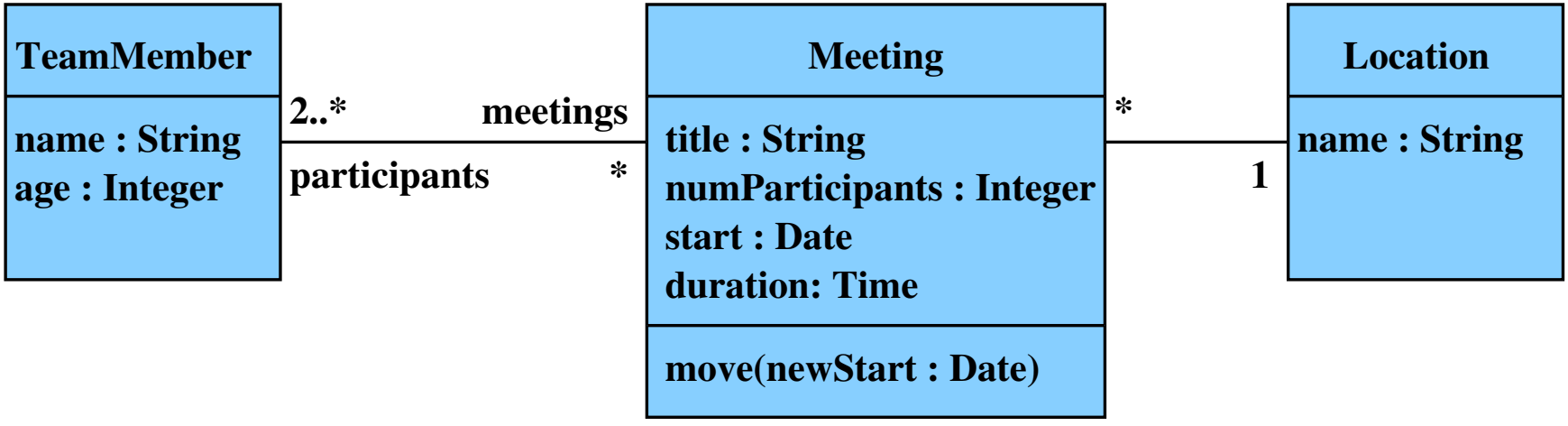
OCL/Invarianten

- Ausdrücke vom Typ **`Boolean`**
- Interpretiert in 3-wertiger Logik (**`true`**, **`false`**, undefined)
- Arithmetische/ logische Ausdrücke mit üblichen Operatoren
- Attribute des Kontextobjekts direkt zugreifbar
- Alternativ durch **`self.`**⟨***`attributeName`***⟩
- Andere Werte erreichbar durch Navigation

OCL/Navigation

- Navigation traversiert Assoziationen von einem classifier zu einem anderen
- Punkt-Notation ⟨*object*⟩.⟨*associationEnd*⟩ liefert
  - assoziiertes Objekt (oder undefined), falls obere Schranke der Vielfachheit ≤ 1
  - geordnete Menge der assoziierten Objekte, falls Assoziation `{ordered}` ist
  - sonst: Menge der assoziierten Objekte
- Falls Assoziationsende nicht benannt, verwende ⟨*object*⟩.⟨*classNameOfOtherEnd*⟩

OCL/Navigation/Beispiel



- `context Meeting`
  - `self.location` liefert das assoziierte Objekt
  - `self.participants` liefert Menge der Teilnehmer

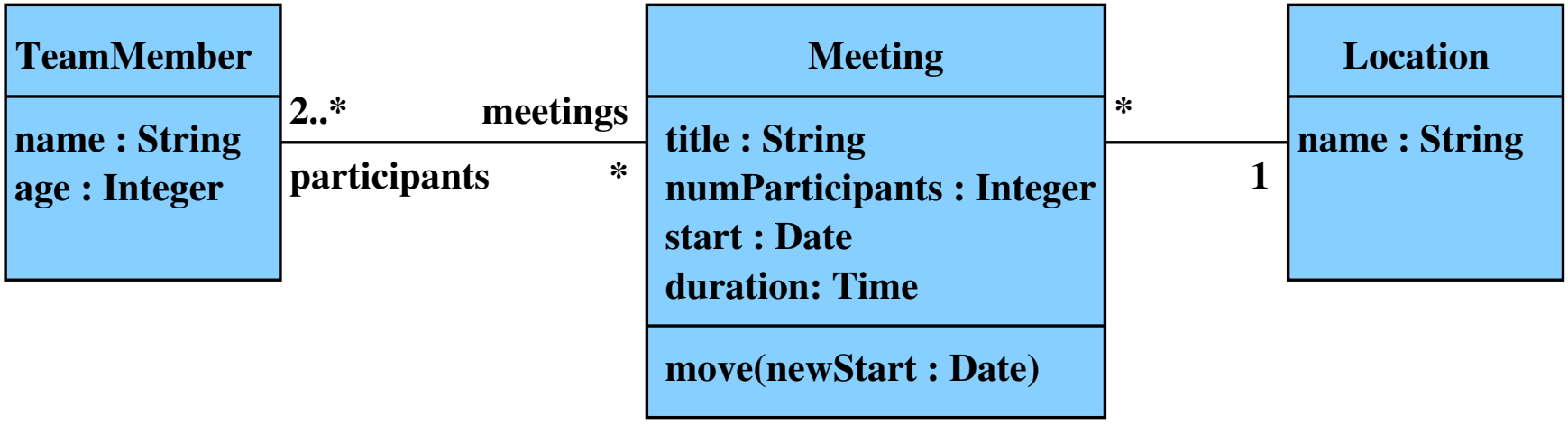
OCL/Mehr Navigation

- Falls Navigation ein Objekt liefert, fahre fort mit
  - Attributezugriffen
  - weiterer Navigation
  - Aufrufen von Operationen

OCL/Mehr Navigation

- Falls Navigation ein Objekt liefert, fahre fort mit
  - Attributezugriffen
  - weiterer Navigation
  - Aufrufen von Operationen
- Falls navigation eine Collection liefert, fahre fort mit collection-Operation ⟨*collOp*⟩:
  - Notation ⟨`collection`⟩`->`⟨`collOp`⟩`(`⟨`args`⟩`)`
  - Beispiele: `size()`, `isEmpty()`, `notEmpty()`, ...
- Einzelne Objekte können auch als collections verwendet werden
- Attribute, Operationen und Navigation nicht direkt anwendbar

OCL/Mehr Navigation/Beispiele



- **context** Meeting
  - **inv:** self.participants->size() = numParticipants
- **context** Location
  - **inv:** name="Lobby" **implies** meeting->isEmpty()

OCL/Zugriff Collections Elemente

- Aufgabe: navigiere ausgehend von einer collection
- Die **collect**-Operation
  - ⟨*collection*⟩**->collect(** ⟨*expression*⟩ **)**
  - ⟨*collection*⟩**->collect(** *v* **|** ⟨*expression*⟩ **)**
  - ⟨*collection*⟩**->collect(** *v* **:** ⟨*Type*⟩ **|** ⟨*expression*⟩ **)**

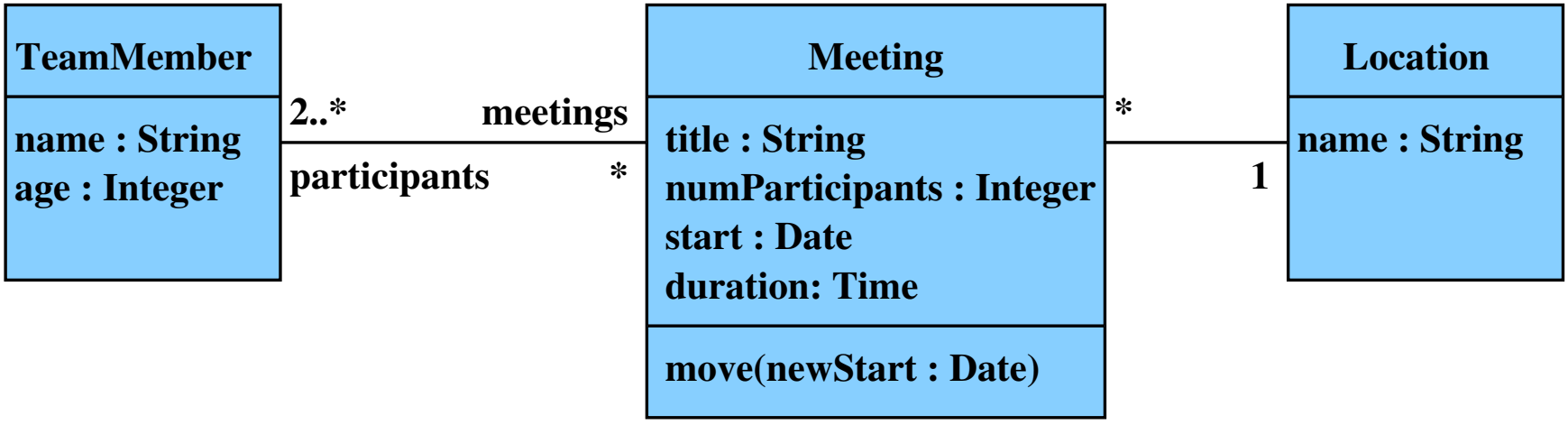
  wertet ⟨*expression*⟩ aus für jedes Element von ⟨*collection*⟩ (als Kontext, optional benannt)
- Ergebnis ist ein bag (Multi-Menge: ungeordnet mit wiederholten Elementen); gleiche Größe wie die ursprüngliche ⟨*collection*⟩
- Transformiere in Menge mit Operation **->asSet()**

OCL/Zugriff Collections Elemente

- Aufgabe: navigiere ausgehend von einer collection
- Die **collect**-Operation
  - ⟨*collection*⟩**->collect(** ⟨*expression*⟩ **)**
  - ⟨*collection*⟩**->collect(** *v* **|** ⟨*expression*⟩ **)**
  - ⟨*collection*⟩**->collect(** *v* **:** ⟨*Type*⟩ **|** ⟨*expression*⟩ **)**

  wertet ⟨*expression*⟩ aus für jedes Element von ⟨*collection*⟩ (als Kontext, optional benannt)
- Ergebnis ist ein bag (Multi-Menge: ungeordnet mit wiederholten Elementen); gleiche Größe wie die ursprüngliche ⟨*collection*⟩
- Transformiere in Menge mit Operation **->asSet()**
- Abkürzungen
  - ⟨*col*⟩**.**⟨*attribute*⟩ für ⟨*col*⟩**->collect(**⟨*attribute*⟩**)**
  - ⟨*col*⟩**.**⟨*op*⟩ **(**⟨*args*⟩**)** für ⟨*col*⟩**->collect(**⟨*op*⟩ **(**⟨*args*⟩**))**

OCL/Zugriff auf Elemente einer Collection



- **context** TeamMember
  - **inv:** meetings.start = meetings.start->asSet()->asBag()

OCL/Iterator Ausdrücke

- Aufgabe:
  - Bearbeiten einer Collection
  - Extrahieren einer Subcollection
- Werkzeug: der **iterate**-Ausdruck

  ⟨*coll*⟩**->iterate(**⟨*it*⟩**;** ⟨*res*⟩ **=** ⟨*init*⟩ **|** ⟨*expr*⟩**)**
- Value:

```
(Set {})->iterate
  (⟨it⟩ ; ⟨res⟩ = ⟨init⟩ | ⟨expr⟩)
  = ⟨init⟩


(Set {x1, ...})->iterate
  (⟨it⟩ ; ⟨res⟩ = ⟨init⟩ | ⟨expr⟩)
  = (Set {...})->iterate
      ( ⟨it⟩
      ; ⟨res⟩ = ⟨expr⟩[⟨it⟩ = x1, ⟨res⟩ = ⟨init⟩]
      | ⟨expr⟩)
```

OCL/Iterator Ausdrücke/Vordefiniert

exists es gibt ein Element, das ⟨***body***⟩ erfüllt

```
⟨source⟩->exists(⟨it⟩ | ⟨body⟩) =
⟨source⟩->iterate(⟨it⟩;r=false | r or ⟨body⟩)
```

forAll alle Elemente erfüllen ⟨***body***⟩

```
⟨source⟩->forAll(⟨it⟩ | ⟨body⟩) =
⟨source⟩->iterate(⟨it⟩;r=true | r and ⟨body⟩)
```

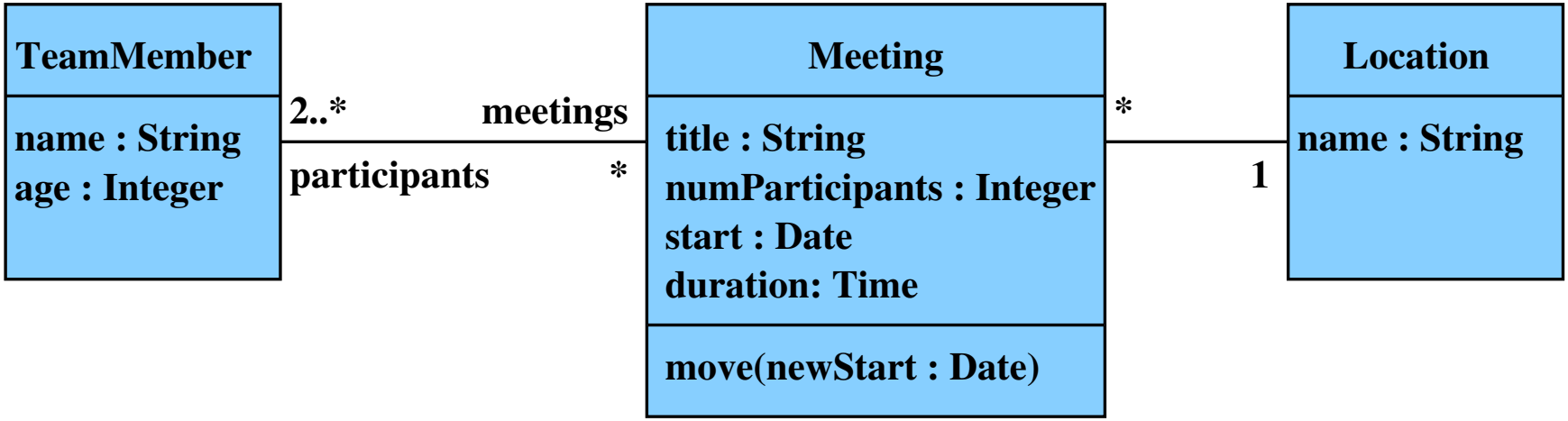
select Teilmenge der Collection-Elemente, für die ⟨***body***⟩ erfüllt ist

```
⟨source⟩->select(⟨it⟩ | ⟨body⟩) =
⟨source⟩->iterate(⟨it⟩;r=Set{}|
                        if ⟨body⟩
                        then r->including(⟨it⟩)
                        else r
                        endif)
```

OCL/Iterator Ausdrücke/Vordefiniert/2

- Abkürzung mit impliziter Variablenbindung
  ⟨***source***⟩**->select(**⟨***body***⟩**)**
- Weitere Iterator-Ausdrücke
  - Auf Collection: **exists**, **forAll**, **isUnique**, **any**, **one**, **collect**
  - Auf Set, Bag, Sequence: **select**, **reject**, **collectNested**, **sortedBy**

OCL/Iterator Ausdrücke/Beispiele




```
context TeamMember
inv: meetings->forAll (m1
       | meetings->forAll (m2
       | m1<>m2 implies disjoint (m1, m2)))
def: disjoint (m1 : Meeting, m2 : Meeting) : Boolean =
     (m1.start + m1.duration <= m2.start) or
     (m2.start + m2.duration <= m1.start)
```

- **def:** extends `TeamMember` by «`OclHelper`» operation

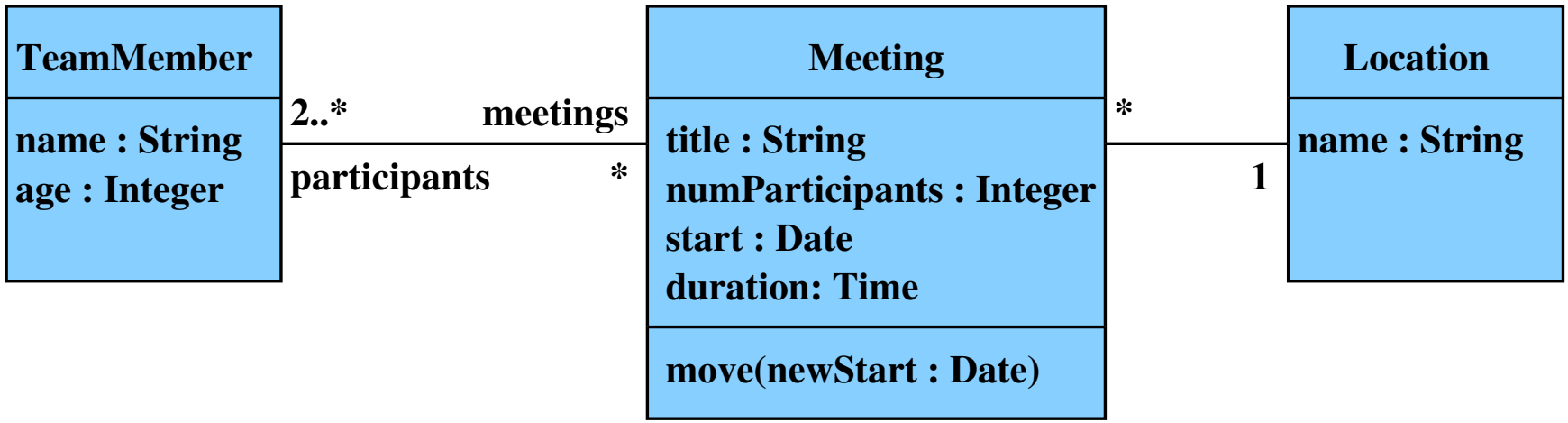
OCL/OclAny, OclVoid, Model Elements

- **`OclAny`** ist Supertyp aller Typen des UML Modells und aller primitiver Typen (aber nicht Supertyp der Collection-Typen)
- **`OclVoid`** its Subtyp jedes Typs
  - einzige Instanz dieses Typs: **`OclUndefined`**
  - jede Operation angewandt auf **`OclUndefined`** liefert **`OclUndefined`** (Ausnahme: **`oclIsUndefined()`**)
- **`OclModelElement`** Aufzählung aller Elemente eines UML-Modells
- **`OclType`** Aufzählung mit einem Literal für jeden Classifier im UML-Modell

OCL/Operationen auf OclAny

- `= (obj : OclAny) : Boolean`
- `<> (obj : OclAny) : Boolean`
- `oclIsNew() : Boolean`
- `oclIsUndefined() : Boolean`
- `oclAsType(typeName : OclType) : T`
- `oclIsTypeOf(typeName : OclType) : Boolean`
- `oclIsKindOf(typeName : OclType) : Boolean`
- `oclIsInState(stateName : OclState) : Boolean`
- `allInstances() : Set(T)` darf nur auf einen Classifier mit endlich vielen Instanzen angewandt werden
- `=` und `<>` sind verfügbar auf `OclModelElement` und `OclType`

OCL/Operationen auf OclAny/Beispiele




```
context Meeting inv:
  title = "general assembly" implies
    numParticipants = TeamMember.allInstances()->size()
```

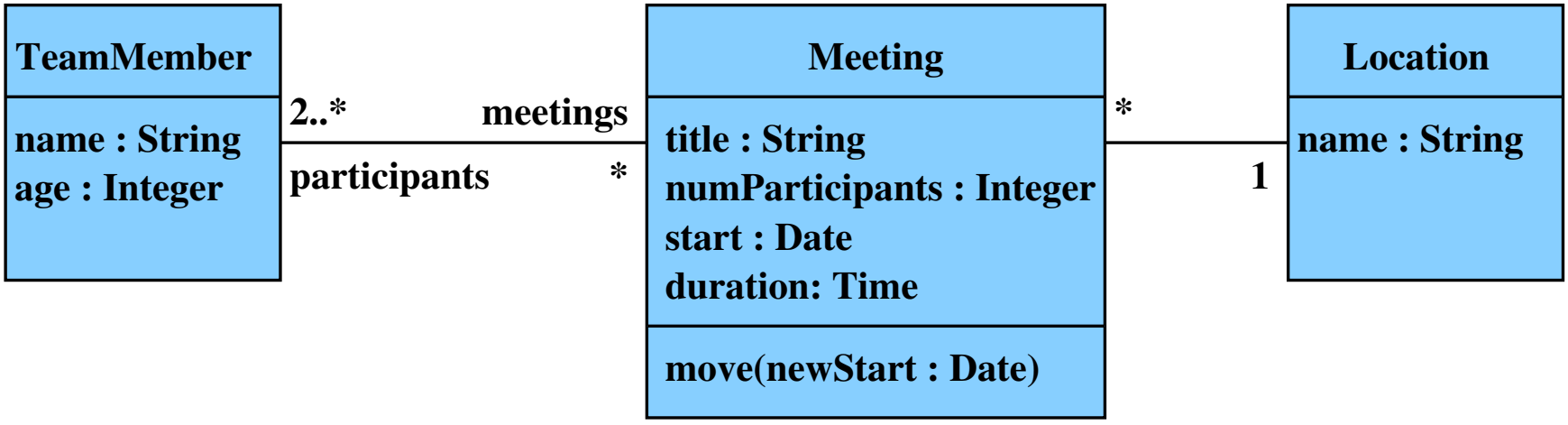
OCL/Vor- und Nachbedingungen

Spezifikation von Operationen durch

```
context ⟨Type⟩::⟨operation⟩(⟨param1⟩ : ⟨Type1⟩, ... ):
pre  ⟨parameterOk⟩: param1 > self.prop1
post ⟨resultOk⟩ : result = param1 - self.prop1@pre
```

- **pre** Vorbedingung mit optionalem Namen ⟨***parameterOk***⟩
- **post** Nachbedingung mit optionalem Namen ⟨***resultOk***⟩
- **self** Empfänger-Objekt der Operation
- **result** Ergebnis der Operation
- **@pre** Zugriff auf den Wert vor Ausführung der Operation
- **body:** ⟨***expression***⟩ definiert das Ergebnis der Operation
- **pre**, **post**, **body** sind optional

OCL/Vor- und Nachbedingung/Beispiele




```
context Meeting::move (newStart : Date)
pre: Meeting.allInstances()->forAll (m |
        m<>self implies
          disjoint(m, newStart, self.duration))
post: self.start = newStart
```

OCL/Vor- und Nachbedingung/Beispiele/2

```
context Meeting::joinMeeting (t : TeamMember)
pre:  not (participants->includes(t))
post: participants->includes(t) and
      participants->includesAll (participants@pre)
```