Softwaretechnik
Model Driven Architecture
Introduction — OCL

Prof. Dr. Peter Thiemann

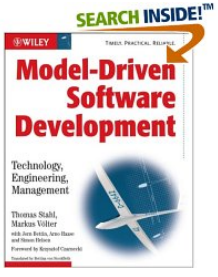
Universität Freiburg

11.07.2008

Introduction MDA

Material

- Thomas Stahl, Markus Völter. Model-Driven Software Development. Wiley & Sons. 2006.



- 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.

What is MDA?

- MDA = Model Driven Architecture
  - also: MD (Software/Application) Development, Model Based [Development/Management/Programming]
  - Model Driven Engineering, Model Integrated Computing
- Initiative of the OMG (trade mark)
  - Object Management Group: CORBA, UML, ...
  - open consortium of companies (ca. 800 Firmen)
- Goal: Improvement of software development process
  - Interoperability
  - Portability
- Approach: Shift development process from code-centric to model-centric
  - Reuse of models
  - Transformation of models
  - Code generation from models

Goals of MDA
Higher Degree of Abstraction

Portability and Reusability

- Development abstracts from target platform
- Technology mapping in reusable transformations
- New technology $\Rightarrow$ new transformation

Interoperability

- Systems span several platforms
- Information flow between platforms via *bridges*
- Byproduct of model transformations

Goals of MDA

Productivity

Every development phase directly contributes to the product, not just the implementation

Documentation and Maintenance

- Changes throught changes of the models
- Models are documentation $\Rightarrow$ consistency
- Separation of concern
- Better handle on changing technology

Specialization

- Business processes
- Technologies

The Concept “Model”
(after Herbert Stachowiak, 1973)

Representation

A model is a representation of an original object.

Abstraction

A model need not encompass all features of the original object.

Pragmatism

A model is always goal-oriented.

The Concept “Model”
(after Herbert Stachowiak, 1973)

Representation

A model is a representation of an original object.

Abstraction

A model need not encompass all features of the original object.

Pragmatism

A model is always goal-oriented.

- Modeling creates a representation that only encompasses the relevant features for a particular purpose.

Formal Models

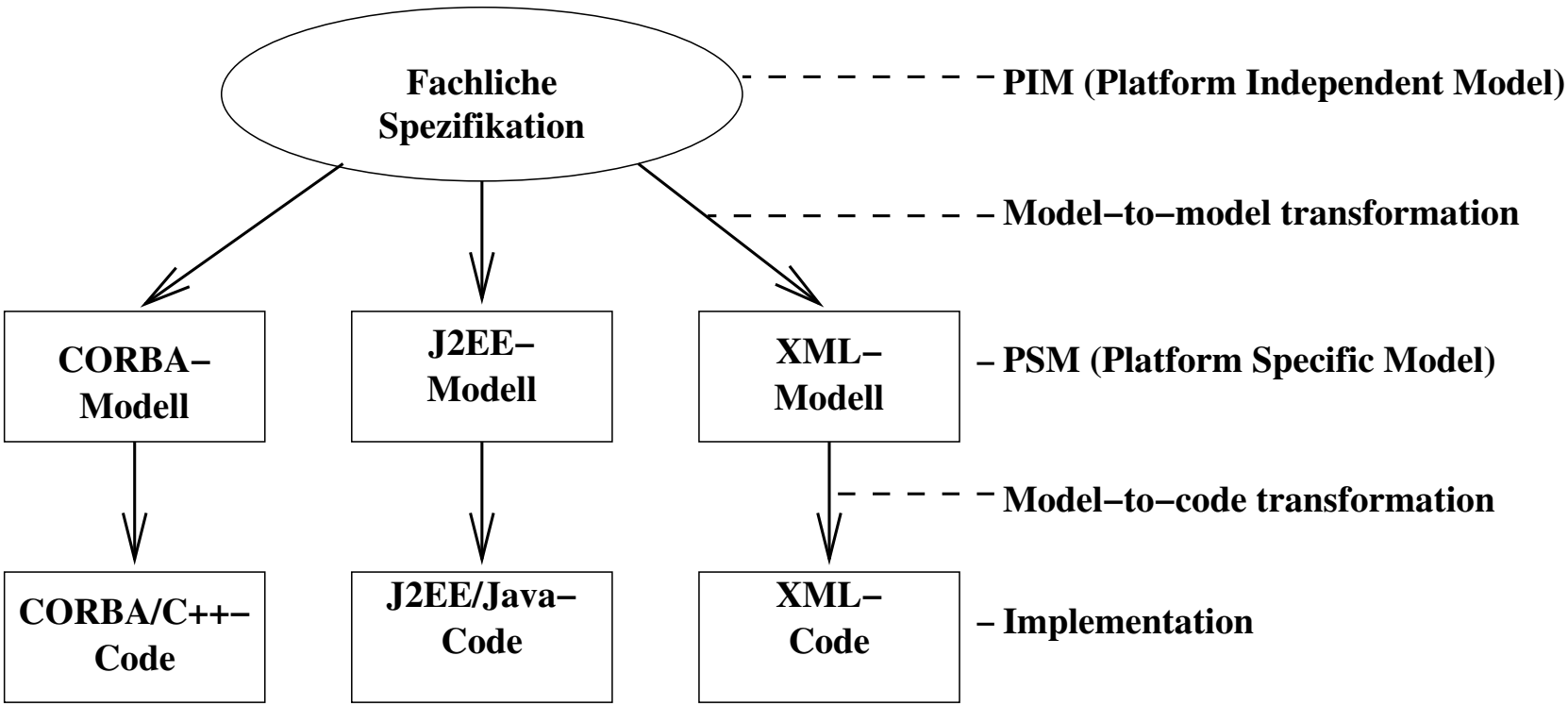
Models authored in a formal language

- Textual: defined by grammar, BNF, etc
- Grafical: defined by *Metamodel*
  - Which modeling elements?
  - Which combinations?
  - Which modifications?

Models with a formal semantics

- Example: logical formula $\Rightarrow$ truth value
- Example: context-free grammar $\Rightarrow$ language
- Example: program $\Rightarrow$ programm execution

Models in MDA



Models in MDA/2

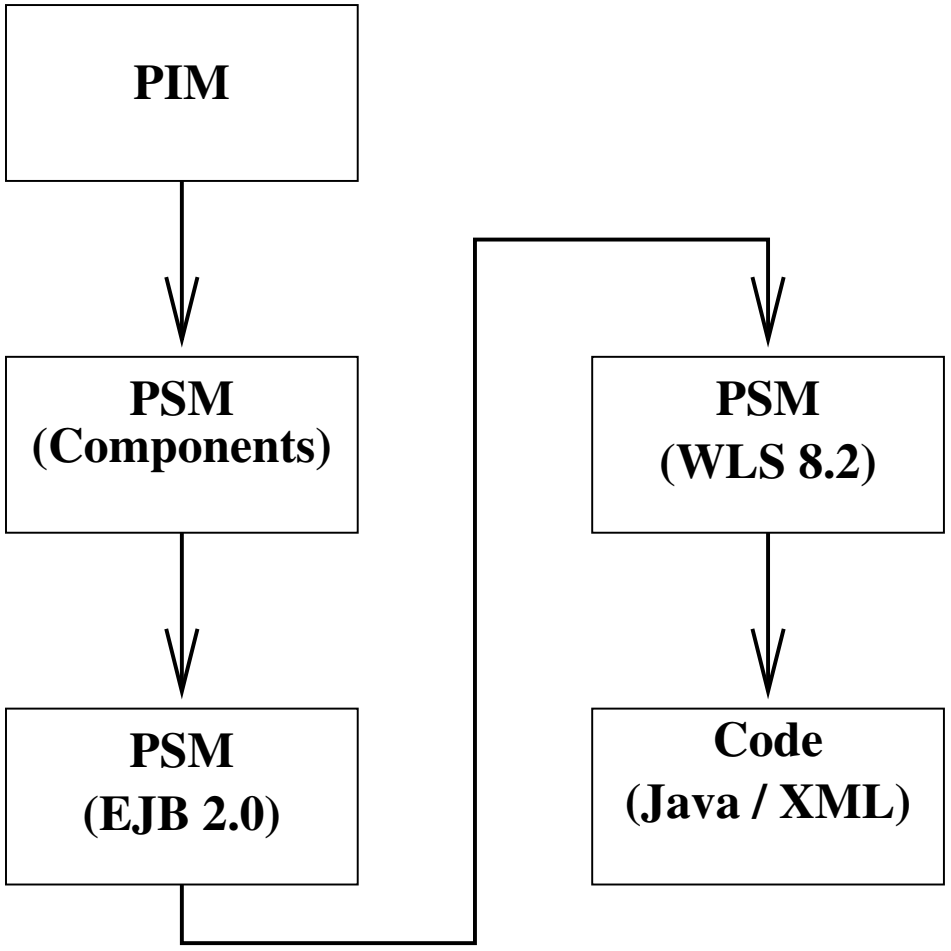
PIM vs PSM

- Relative concepts
- Smooth transition
- Several levels of model and transformation steps possible
- Inverse transformation PSM $\Rightarrow$ PIM unlikely

Transformation

- Code is the ultimate model (PSM)
- Model-to-code is a special case

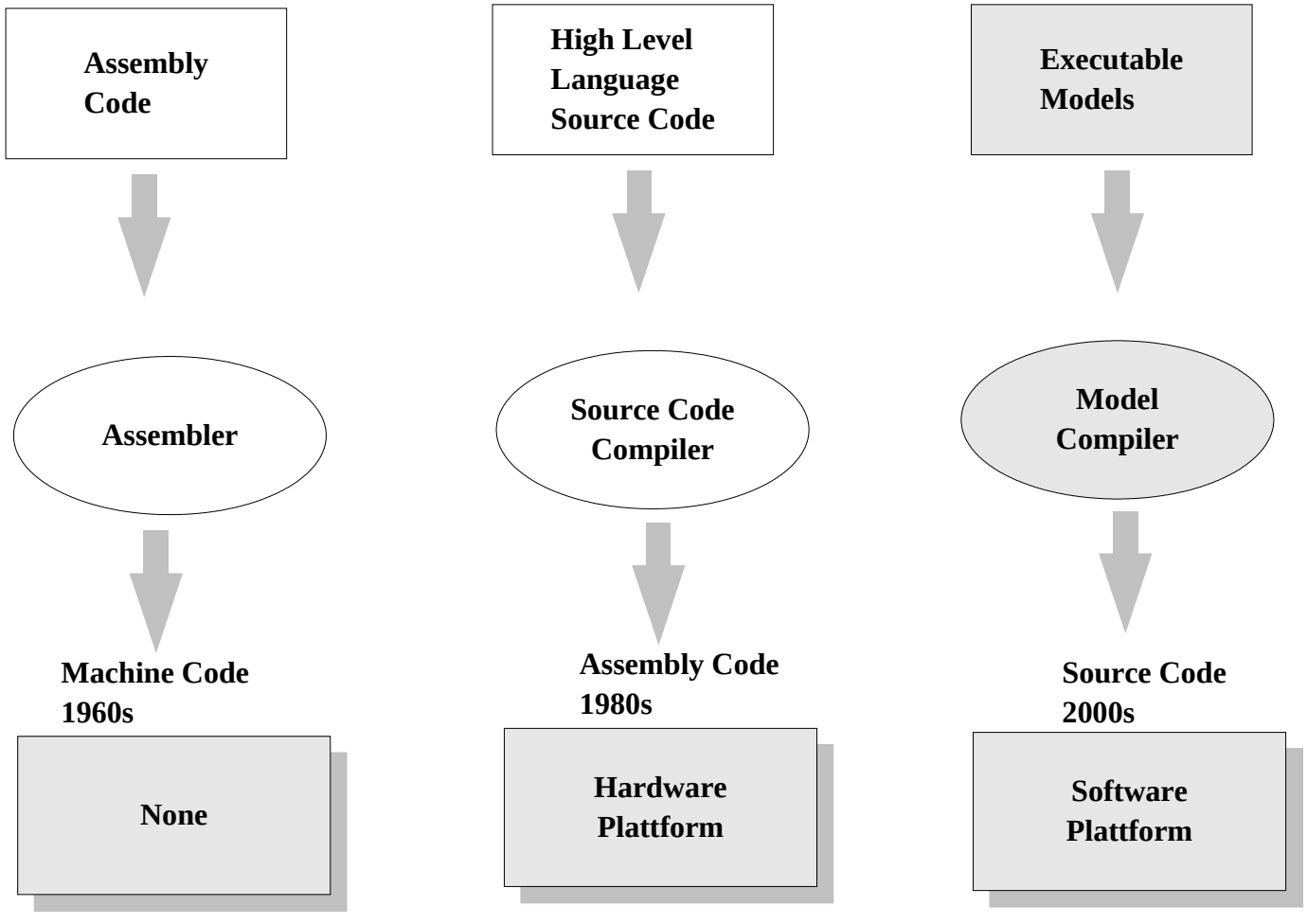
Models and Transformations



Platform

- API
- Virtual machine
- Provides several services
- Examples
  - Different processors $\Rightarrow$ hardware platform
  - Operating system $\Rightarrow$ software platform
  - Java VM $\Rightarrow$ software platform
  - EJB $\Rightarrow$ component platform
  - CORBA, Webservices, ...
  - Application architecture, DSL (Domain Specific Language)

Examples for Platforms



OCL

What is OCL?

- OCL = object constraint language
- standard query language of UML 2
- expressions and constraints in object modeling artifacts

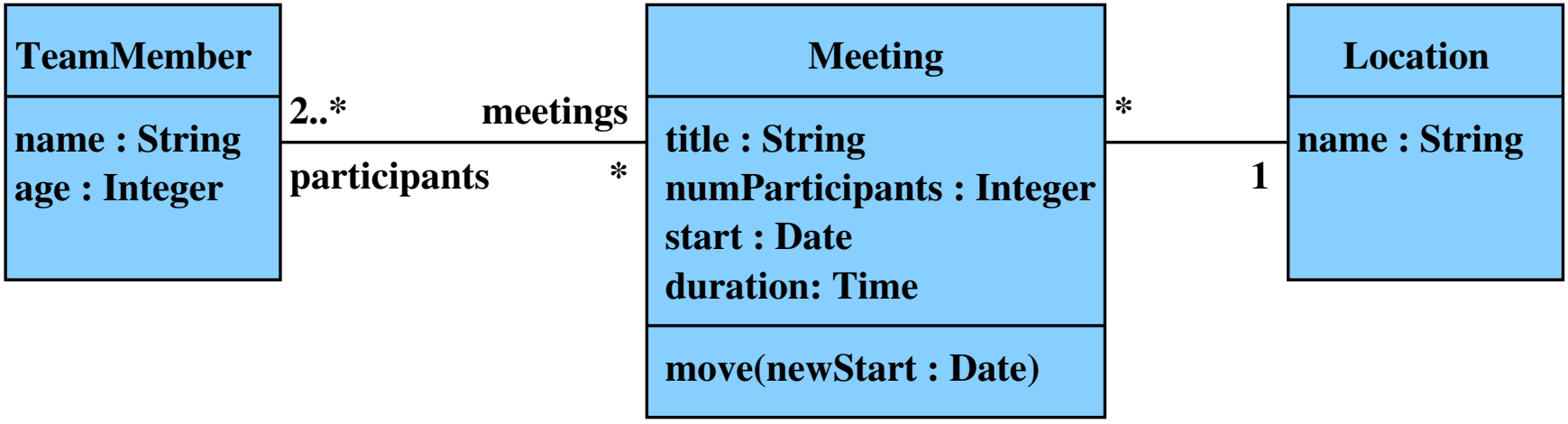
OCL/Expressions and Constraints

- Expressions
  - initial values, derived values
  - parameter values
  - body of operation (no side effects $\Rightarrow$ limited to queries)
  - of type: `Real`, `Integer`, `String`, `Boolean`, or model type
- Constraints restrict the set of admissible instances
  - invariant (class): condition on the state of the class's objects which is always true
  - precondition (operation): indicates applicability
  - postcondition (operation): must hold after operation if precondition was met
  - guard (transition): indicates applicability
- Evaluation with respect to a snapshot of the instance graph

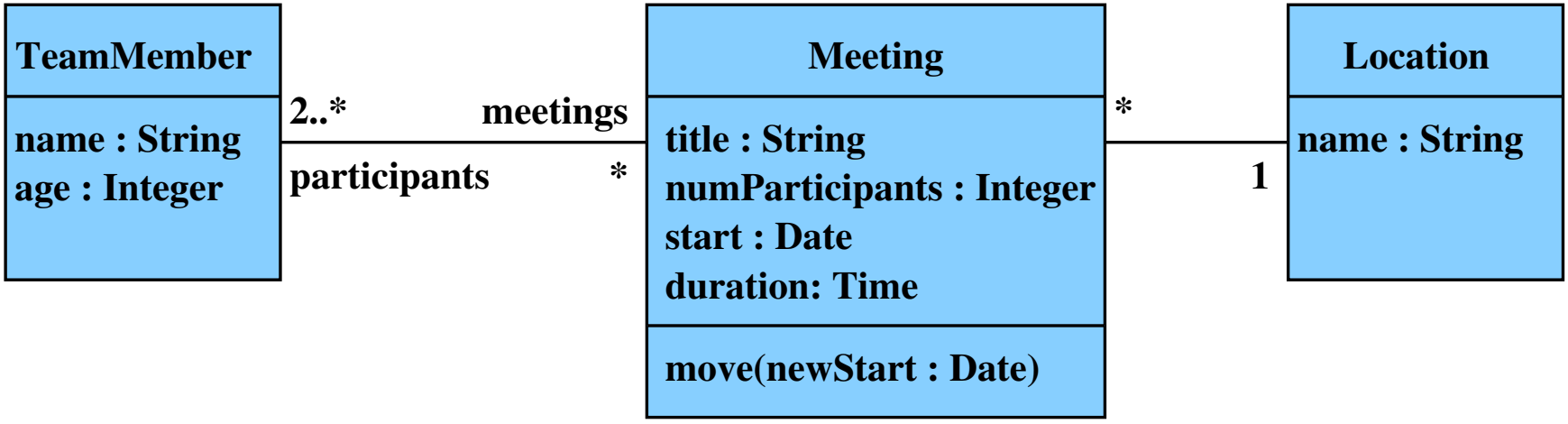
OCL/Context

- Each OCL expression is interpreted relative to a context
  - invariant: class, interface, datatype, component (a classifier)
  - precondition, postcondition: operation
  - guard: transition
- Context is indicated
  - graphically by attachment as a note
  - textually using the `context` syntax

OCL/Example


TeamMember
name : String
age : Integer
2..*
meetings
participants
*
Meeting
title : String
numParticipants : Integer
start : Date
duration: Time
move(newStart : Date)
*
1
Location
name : String

OCL/Example



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

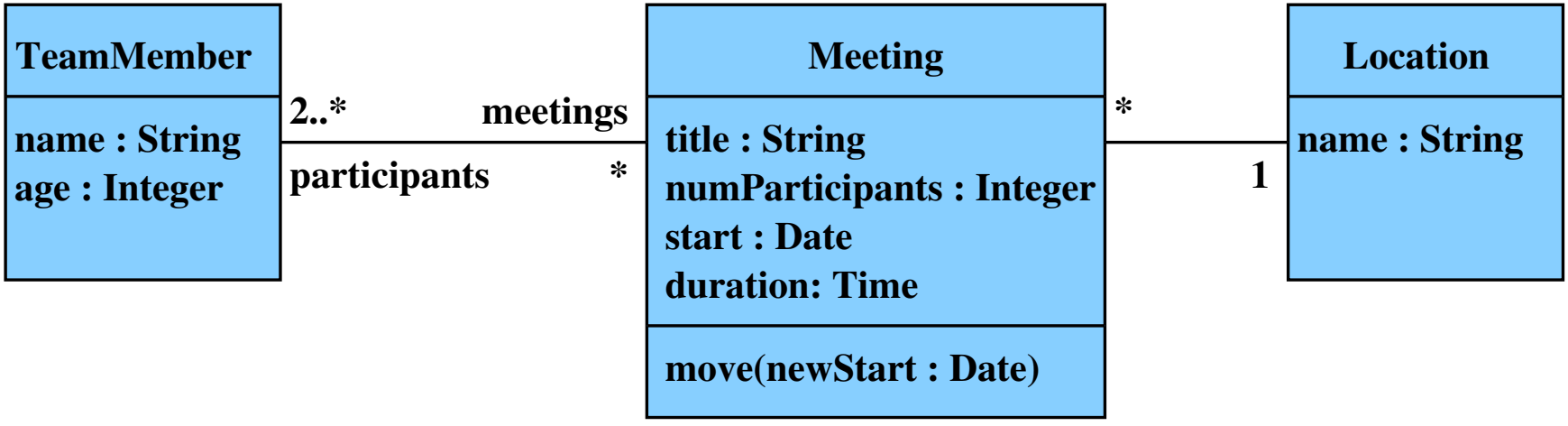
OCL/Invariants

- Expressions of type **`Boolean`**
- Interpreted in 3-valued logic (**`true`**, **`false`**, undefined)
- Arithmetic/ logic expressions with usual operators
- Attributes of the context object directly accessible
- Alternatively through **`self.`**⟨***`attributeName`***⟩
- Other values available through navigation

OCL/Navigation

- Navigation traverses associations from one classifier to another
- Dot notation ⟨***object***⟩.⟨***associationEnd***⟩ yields
  - associated object (or undefined), if upper bound of multiplicity ≤ 1
  - the ordered set of associated objects, if association is **{ordered}**
  - the set of associated objects, otherwise
- If association end not named, use ⟨***object***⟩.⟨***classNameOfOtherEnd***⟩

OCL/Navigation/Examples



- `context Meeting`
  - `self.location` yields the associated object
  - `self.participants` yields set of participants

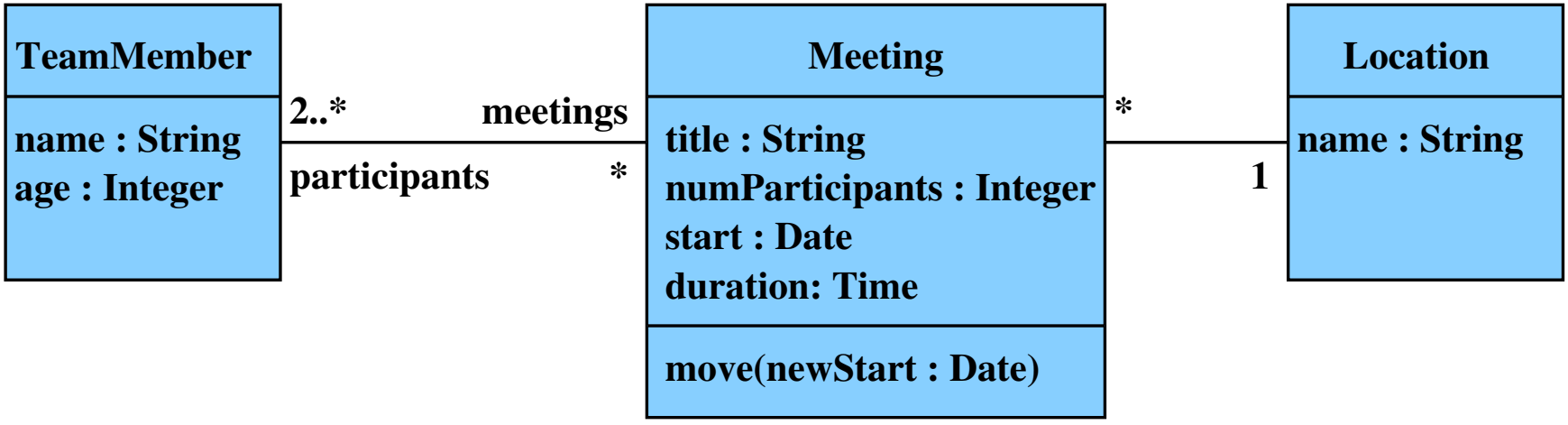
OCL/More Navigation

- If navigation yields object, then continue with
  - attribute notation
  - navigation
  - operation calls

OCL/More Navigation

- If navigation yields object, then continue with
  - attribute notation
  - navigation
  - operation calls
- If navigation yields a collection, then continue with a collection operation ⟨*collOp*⟩:
  - notation ⟨`collection`⟩`->`⟨`collOp`⟩`(`⟨`args`⟩`)`
  - examples: `size()`, `isEmpty()`, `notEmpty()`, ...
- Single objects may also be used as collections
- Attributes, operations, and navigation of elements not directly accessible

OCL/More Navigation/Examples



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

OCL/Accessing Collection Elements

- Task: Continue navigation from a collection
- The **collect** operation
  - ⟨*collection*⟩->collect(⟨*expression*⟩)
  - ⟨*collection*⟩->collect(*v* | ⟨*expression*⟩)
  - ⟨*collection*⟩->collect(*v* : ⟨*Type*⟩ | ⟨*expression*⟩)

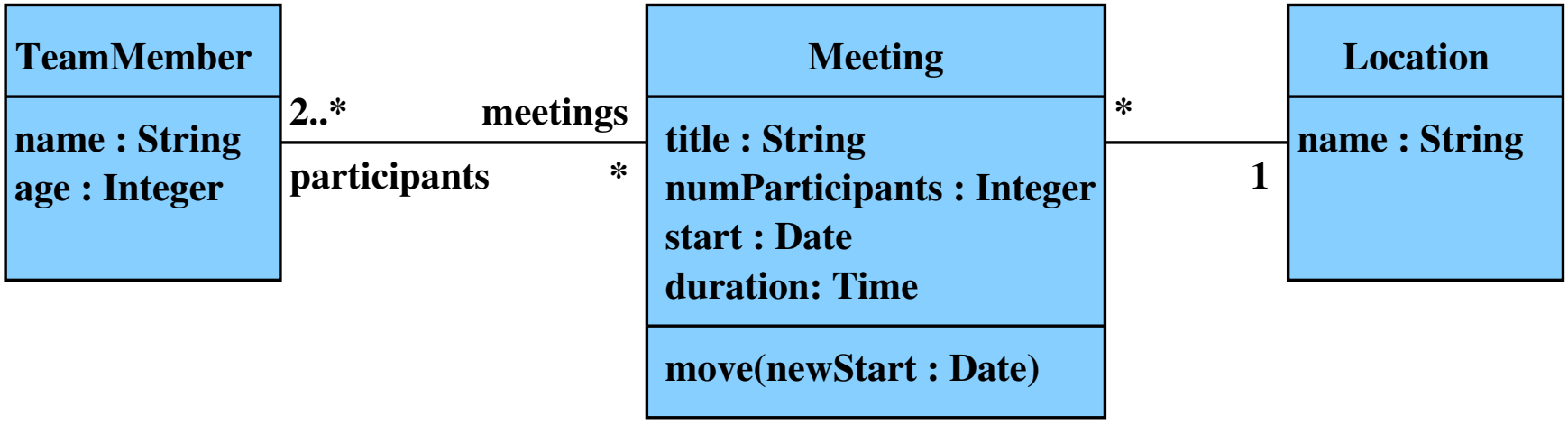
  evaluates ⟨*expression*⟩ for each element of ⟨*collection*⟩ (as context, optionally named)
- Result is a bag (unordered with repeated elements); same size as original ⟨*collection*⟩
- Change to a set using operation **->asSet()**

OCL/Accessing Collection Elements

- Task: Continue navigation from a collection
- The **collect** operation
  - ⟨*collection*⟩->collect(⟨*expression*⟩)
  - ⟨*collection*⟩->collect(*v* | ⟨*expression*⟩)
  - ⟨*collection*⟩->collect(*v* : ⟨*Type*⟩ | ⟨*expression*⟩)

  evaluates ⟨*expression*⟩ for each element of ⟨*collection*⟩ (as context, optionally named)
- Result is a bag (unordered with repeated elements); same size as original ⟨*collection*⟩
- Change to a set using operation **->asSet()**
- Shorthands
  - ⟨*col*⟩.⟨*attribute*⟩ for ⟨*col*⟩->collect(⟨*attribute*⟩)
  - ⟨*col*⟩.⟨*op*⟩ (⟨*args*⟩) for ⟨*col*⟩->collect(⟨*op*⟩ (⟨*args*⟩))

OCL/Accessing Collection Elements



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

OCL/Iterator Expressions

- Task:
  - Examine a collection
  - Define a subcollection
- Tool: the **iterate** expression

  ⟨*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 Expressions/Predefined

exists there is one element that makes ⟨***body***⟩ true

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

forAll all elements make ⟨***body***⟩ true

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

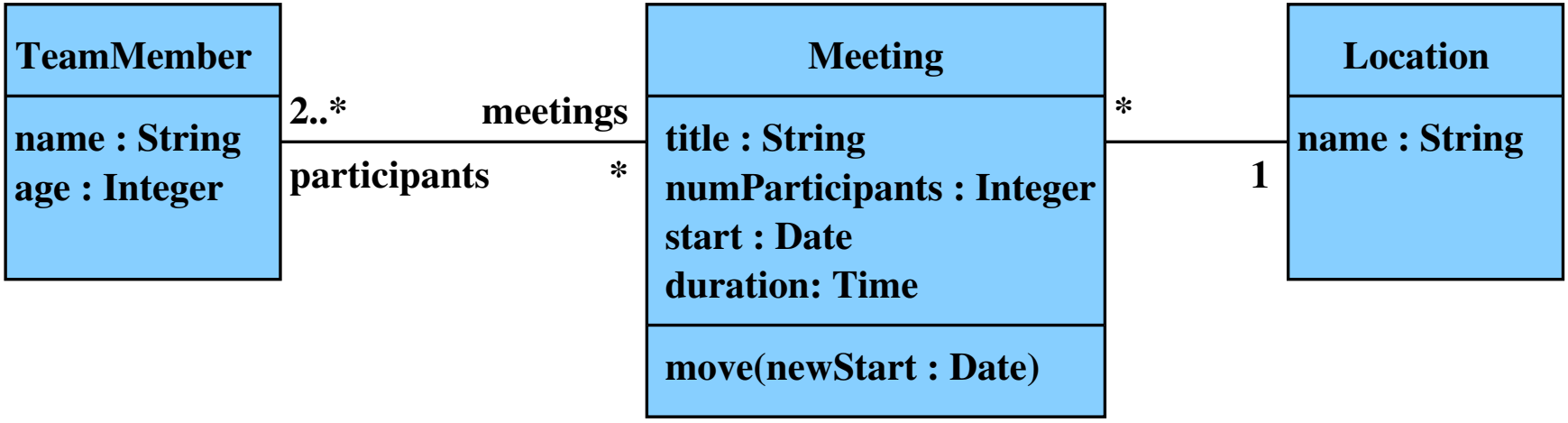
select subset where ⟨***body***⟩ is true

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

OCL/Iterator Expressions/Predefined/2

- Shorthand with implicit variable binding
  ⟨*source*⟩`->select(`⟨*body*⟩`)`
- Further iterator expressions
  - On Collection: `exists`, `forAll`, `isUnique`, `any`, `one`, `collect`
  - On Set, Bag, Sequence: `select`, `reject`, `collectNested`, `sortedBy`

OCL/Iterator Expressions/Examples




```
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

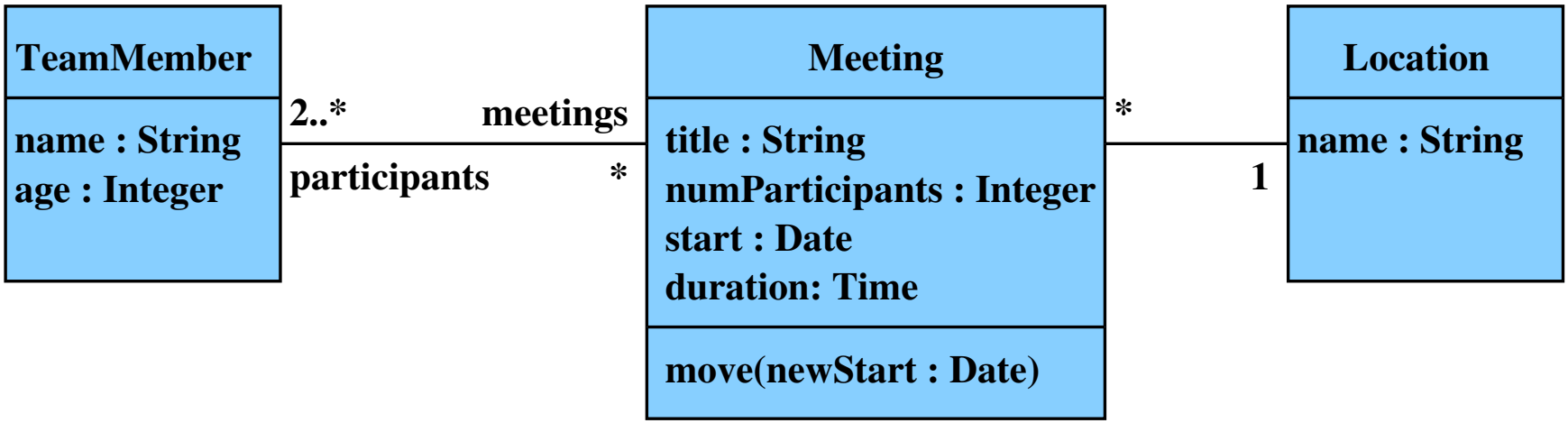
OCL/OclAny, OclVoid, Model Elements

- **`OclAny`** is supertype of types from the UML model and all primitive types (not of collection types)
- **`OclVoid`** is subtype of every type
  - single instance **`OclUndefined`**
  - any operation applied to **`OclUndefined`** yields **`OclUndefined`** (except **`oclIsUndefined()`**)
- **`OclModelElement`** enumeration with a literal for each element in the UML model
- **`OclType`** enumeration with a literal for each classifier in the UML model
- **`OclState`** enumeration with a literal for each state in the UML model

OCL/Operations on 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)` must be applied to a classifier with finitely many instances
- `=` and `<>` also available on `OclModelElement`, `OclType`, and `OclState`

OCL/Operations on OclAny/Examples




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

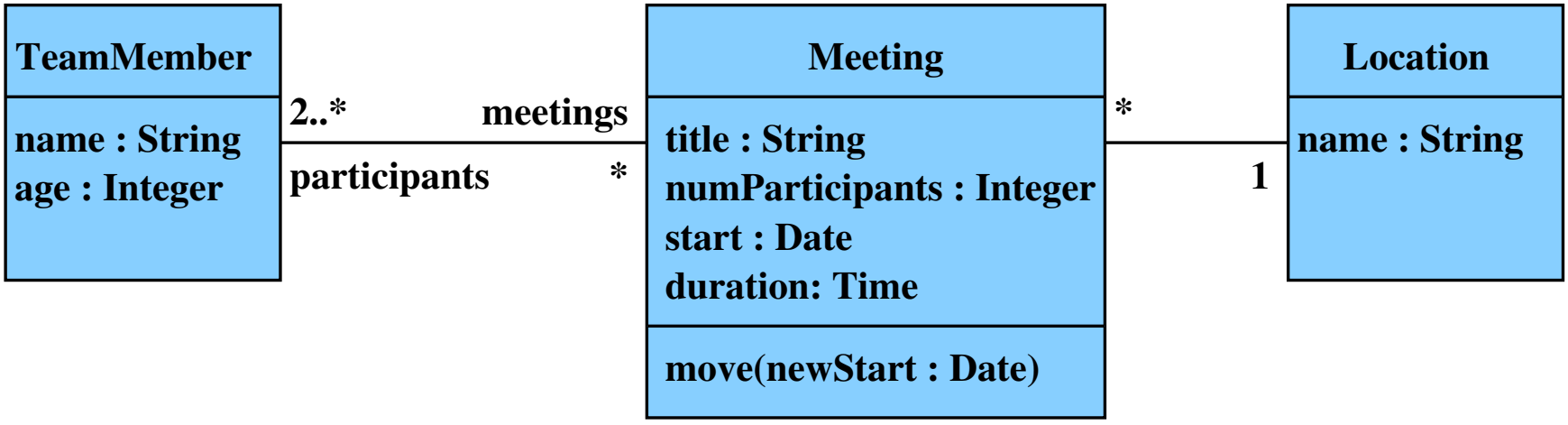
OCL/Pre- and Postconditions

Specification of operations by

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

- **pre** precondition with optional name ⟨***parameterOk***⟩
- **post** postcondition with optional name ⟨***resultOk***⟩
- **self** receiver object of the operation
- **result** return value of the operation
- **@pre** accesses the value before executing the operation
- **body:** ⟨***expression***⟩ defines the result value of the operation
- **pre**, **post**, **body** are optional

OCL/Pre- and Postconditions/Examples




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

OCL/Pre- and Postconditions/Examples/2

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