Model Driven Architecture
OCL

Prof. Dr. Peter Thiemann

Universität Freiburg

17.05.2006

Addendum: Classifiers and Instances

- Classifier diagrams may also contain instances
- Instance description may include
  - name (optional)
  - classification by zero or more classifiers
  - kind of instance
    - instance of class: object
    - instance of association: link
    - etc
  - optional specification of values

Notation for Instances

- Instances use the same notation as classifier
  - Box to indicate the instance
  - Name compartment contains

    <u>`name:classifier,classifier...`</u>

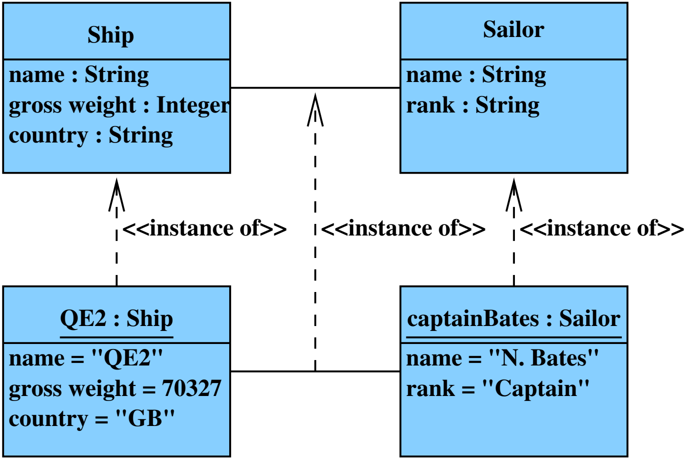
    <u>`name:classifier`</u>

    <u>`:classifier`</u> anonymous instance

    <u>`:`</u> unclassified, anonymous instance
  - Attribute in the classifier may give rise to like-named slot with optional value
  - Association with the classifier may give rise to link to other association end

    direction must coincide with navigability

Notation for Instances (Graphical)



Ship

name : String
gross weight : Integer
country : String

Sailor

name : String
rank : String

<<instance of>>

<<instance of>>

<<instance of>>

QE2 : Ship

name = "QE2"
gross weight = 70327
country = "GB"

captainBates : Sailor

name = "N. Bates"
rank = "Captain"

What is OCL?

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

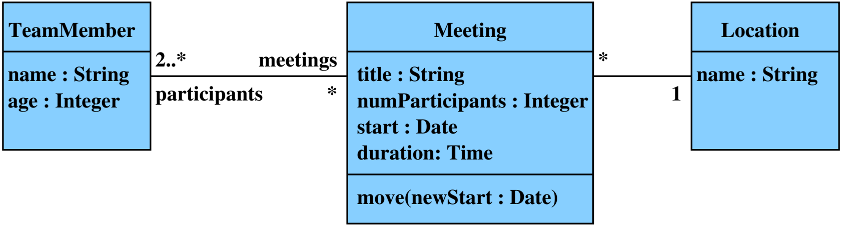
OCL/Expressions and Constraints

- Expressions
  - initial values, derived values
  - parameter values
  - body of operation (no side effects ⇒ limited to queries)
  - of type: **`Real`**, **`Integer`**, **`String`**, **`Boolean`**, or model type
- Constraints
  - 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

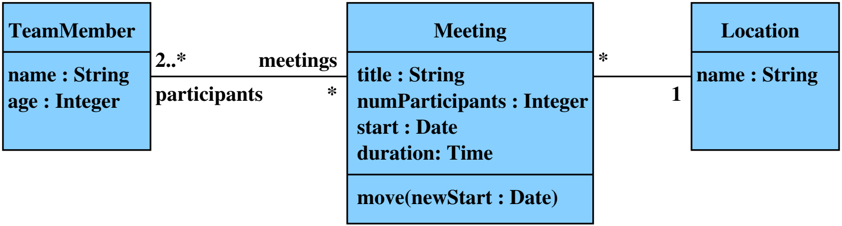
OCL/Context

- Each OCL expression is interpreted relative to a context
  - invariant wrt class, interface, datatype, component (a classifier)
  - precondition wrt operation
  - postcondition wrt operation
  - guard wrt transition
- Context is indicated
  - graphically by attachment as a note
  - textually using the **`context`** syntax
- Expression is evaluated with respect to a snapshot of the object graph described by the modeling artifact

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 > 0`
- `context Meeting inv: duration > 0`

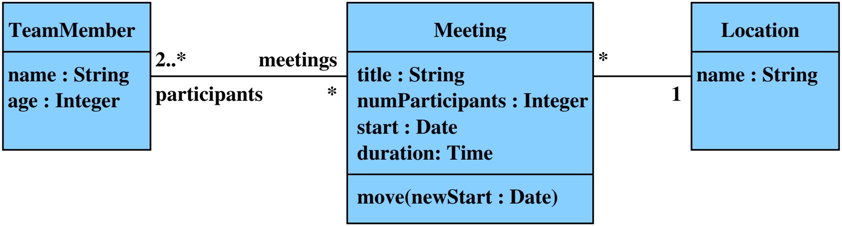
OCL/Invariants

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

OCL/Navigation

- Navigation leads 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
- Class name of other end if association end not named

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 use
  - attribute notation
  - navigation
  - operation calls

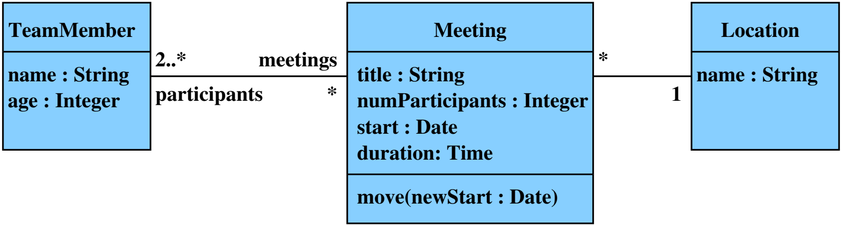
  to continue
- What if navigation yields a collection?

OCL/More Navigation

- If navigation yields object, then use
  - attribute notation
  - navigation
  - operation calls

  to continue
- What if navigation yields a collection?
- Collection operations:
  - notation *`collection`*`->`*`op`*`(`*`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* )`

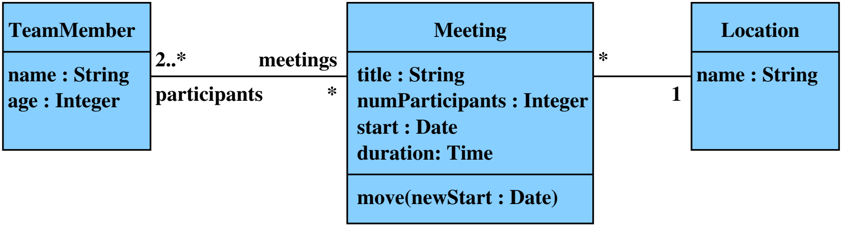
  evaluates *`expression`* for each element of *`collection`* (as context, optionally named)
- Result is bag (unordered with repeated elements); same size as original *`collection`*
- Change to a set using operation `->asSet()`

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 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

  ```
  source->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)
```

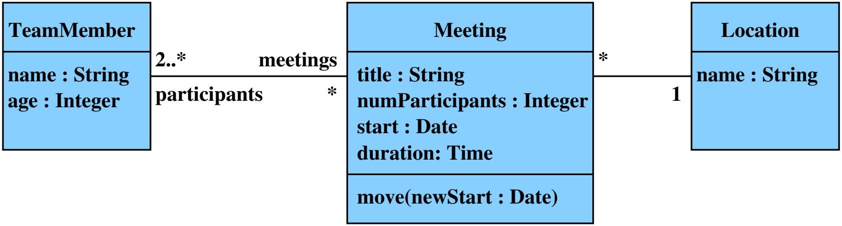
`select` subset where *body* is true

```
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

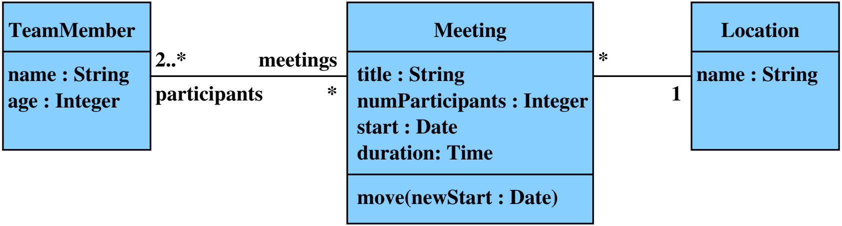
OCL/OclAny, OclVoid, Model Elements

- **`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



TeamMember

name : String
age : Integer

Meeting

title : String
numParticipants : Integer
start : Date
duration: Time

move(newStart : Date)

Location

name : String

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

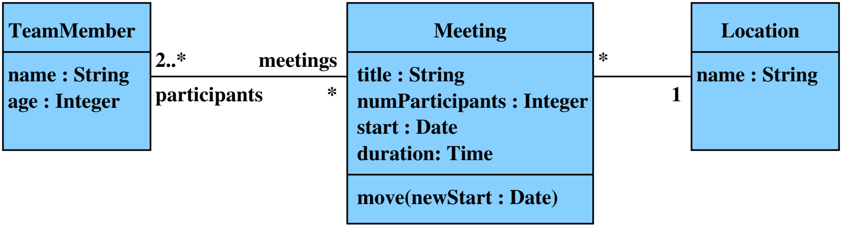
OCL/Pre- and Postconditions

Specification of operations by

```
context Type::operation(param1 : Type1, ... ): Retu
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)
```

Action Semantics

- An action is the fundamental unit of behavior specification.
- An action takes a set of inputs and converts them into a set of outputs [. . .].
- The most basic action provides for implementation-dependent semantics, [. . .].
- [. . .] primitive actions are defined [so] as to enable the maximum range of mappings.
- [. . .] they either carry out a computation or access object memory
- This approach enables clean mappings to a physical model, [. . .].
- In addition, any re-organization of the data structure will leave the specification of the computation unaffected.

From the UML 2 superstructure 11.1

Action Semantics/Why have it?

- build complete and precise models
- formal proofs of correctness of a problem specification
- high-fidelity model-based simulation and verification
- enables reuse of domain models
- stronger basis for model design and eventual coding
- support code generation to multiple software platforms.

From “Software-platform-independent, Precise Action Specifications for UML”, UML’99

Action Semantics/Idea

Basic idea: specify computation so that it is

- data driven and
- inherently parallel
- (sequential execution through data dependency or explicit control dependency)
- independent of concrete syntax

Action Semantics/Action Specification

Basic building blocks:

- Pins: input and output ports of an action; with type and multiplicity
- Variables: intermediate results
- Data flow: connects the output pin of one action to the input pin of another
- Control flow: explicit ordering constraint for action pairs
- Actions: for object manipulation, memory operations, arithmetic, message passing, etc.
- Procedures: packaging of actions with input and output pins

Action Semantics/Action Execution

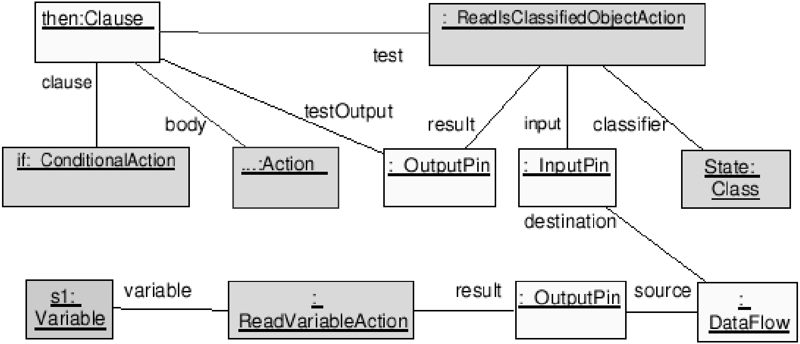
Life-cycle of an action

- Waiting. Initial state after creation of action execution.
- Ready. Action execution with all inputs available and all control dependencies in state **Complete**.
- Executing. Compute outputs from inputs.
- Complete. Values of output pins determined, signal to control-flow dependant actions.

Action Semantics/Types of Actions

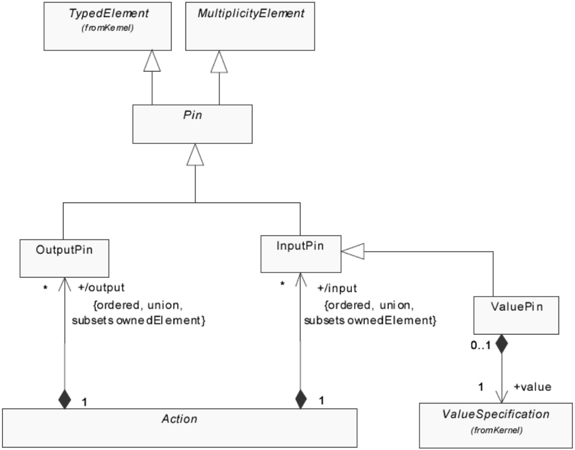
- Computation actions *e.g.* mathematical functions (left undefined by standard)
- Composite actions building blocks for control structures like loops and conditionals
- Read and write actions access, navigate, and modify model-level constructs (objects, links, attribute slots, and variables)
- Collection actions $\Rightarrow$ iterators for actions

Action Semantics/Example



From: UML Action Semantics for Model Transformation Systems, Varró and Pataricza (uses obsolete 1.5 metamodel)

Action Semantics/Basic Pins



Action Semantics/Object Actions