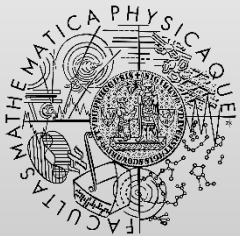# OCL: Object Constraint Language

Department of
Distributed and
Dependable
Systems

D3S

Pavel Parízek

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Motivation

- UML models (class diagrams, …)
    - Main limitations: incomplete, ambiguous
    - Some domain knowledge is not captured

- How to specify additional constraints
    - Natural language (plain text)
    - Formal languages (some logic)

Constraints in UML diagrams

- Precise exact statement (sentence)
  - Captures some condition or restriction

- Attached to elements (classes, fields, …)
  - Context: entity in diagram for which the constraint is evaluated and time of evaluation

- Graphical notation
  - Textbox connected to entity with a dashed line

What is OCL

- Formal specification language
- Extension for UML

- Main features
  - Declarative and very strongly typed
  - Constraints written as precise text
  - Supports object query expressions

Official information

- Maintainers
  - Object Management Group (OMG)

- Resources
  - Specification: http://www.omg.org/spec/OCL/
  - https://en.wikipedia.org/wiki/Object_Constraint_Language

Department of
Distributed and
Dependable
Systems

What can be specified in OCL

- Initial values of properties (object fields)
- Derivation rules (constraints for values)
- Operation preconditions and postconditions
- Operation bodies (side-effects)
- Invariants for objects (classes)

Initial values

- Syntax
  - **context** TypeName::PropertyName : Type
  - **init** *<expression representing the initial value>*

- Example
  - **context** Thesis::state
  - **init**: ThesisStatus::assigned

Derivation rules

- Purpose
  - Restricts value of some property (object field)

- Syntax
  - **context** TypeName::PropertyName : Type
  - **derive:** *<expression representing the derivation rule>*

- Example
  - **context** Lecturer::courses
  - **derive** self.teaching->size()

Operation pre/post-conditions

- Syntax
  - **context** `TypeName::OperName (p1 : Type1, ..., pN : TypeN): ReturnType`
  - **pre:** *<precondition expression>*
  - **post:** *<postcondition expression>*

- Example
  - **context** `Student::enrollToCourse(c:Course): Boolean`
  - **pre:** `c.enrolledStudents < c.limit`
    `        and self.enrolled->excludes(c)`
  - **post:** `c.enrolledStudents = c.enrolledStudents@pre + 1`
    `        and self.enrolled->includes(c)`
    `        and result = c.students->includes(self)`

Operation bodies

- Purpose
  - Capturing side-effects
    - How the operation changes values of properties

- Syntax
  - **context** `TypeName::OperName (p1 : Type1, ..., pN : TypeN): ReturnType`
  - **body:** *<expression>*

Invariants

- Purpose
  - Constraint for every instance of the class (type)

- Syntax
  - **context** TypeName
  - **inv:** *<invariant expression>*

- Example
  - **context** Student
  - **inv:** self.yearOfStudy > 5 implies self.payingFee

OCL features

- Type system

- Collections

Type system

- Generic types: `OclAny`, `OclInvalid`

- Basic types: `Boolean`, `Integer`, `String`, …
  - Common operators and functions

- Collection types: `Set`, `Bag`, `OrderedSet`, `Sequence`
  - Instances created through navigation over associations in UML class diagrams

- User-defined types
  - Elements of UML diagrams

Collections

- How they are created
  - Navigation via properties (association ends or attributes) produces a new collection object
  - Chain `a.p1.p2.[...].pN` of properties p1, ..., pN from variable a

- Collection constants
  - Syntax: TypeName`{ value1, value2, ..., valueN }`

- Operations
  - Filtering by predicate: `select`, `reject`
  - Quantifiers: `forAll`, `exists`
  - Loop with accumulator: `iterate`
  - Transitive closure by recursive application of an expression: `closure`
  - Some other: `count`, `includes`, `excludes`, `isEmpty`, `size`
  - Iterating over all instances of a given type: `<Type>.allInstances()`

Collections – examples

- **context** Course
- **inv**: self.passed->reject(s|self.enrolled ->includes(s))->size()=0

- **context** Lecturer
- **inv**: self.courses->forAll(c|c.guaranteedBy ->includes(self))

- **context** Course
- **inv**: self.enrolled->iterate(s : Student ; somePassed : Boolean = false | somePassed or s.pointsFor(self) >= 50)

Remarks

- Likely, OCL is not used that much in practice

- Take-away message (knowledge)
  - General concepts, transferable to some other specification languages and frameworks