Object Constraint Language

Object-Oriented Analysis and Design

CSCI 6448 - Fall 1998

Kenneth M. Anderson

Goals of the Lecture

- Present the Object Constraint Language
  - As best as possible, with the limited information available from UML in a Nutshell and the Rational Website
- The official reference for the OCL will be the forthcoming Addison-Wesley book:
  - The Unified Modeling Language Reference Manual (expected mid-December)

Additional Reference

- The Object Constraint Language
  - Precise Modeling with UML

Jos Warmer

Anneke Kleppe

Addison-Wesley: ISBN 0-201-37940-6

Just published!

Constraints: A review

- A constraint is a restriction on one or more values of (part of) an object-oriented model or system
  - Supports design by contract
    - Operations can be provided pre- and post-conditions
    - These conditions can be specified using constraints

Constraints Review continued

- A precondition must be true at the moment that an operation is executed
- A postcondition must be true at the moment the operation finishes executing
- A different type of constraint is an invariant
  - An invariant specifies a condition that must always be true of its associated elements
  - This contrasts with pre- and post-conditions that only need to hold before and after the execution of a single operation

Advantages of Constraints

- Better Documentation
  - The semantics are kept close to the model they constrain
- Improved Precision
  - Constraints cannot be interpreted differently by different people
- Communication without Misunderstanding

Declarative or Operational

- Two types of interpretations
- Declarative
  - States what must be true, not what must be done
- Operational
  - Breaking a constraint triggers an operation
- For example, in Eiffel, if a constraint is broken an exception is thrown
- UML adopts a declarative style of constraints

Advantages of Declarative style

- Constraints have no side-effects
  - The state of a system does not change when a constraint is evaluated
- It separates the specification of a constraint from the response required if a constraint is broken
- Constraints should be stable; actions may change over time
- Atomicity requirements can be avoided

Object Constraint Language

- OCL is a pure expression language
  - It can not modify the state of a model
  - It can however specify a state that is required by a pre- or post-condition
  - An OCL expression simply states a requirement that must be met in order to consider an instantiation of the model to be valid

OCL, continued.

- OCL is not a programming language
  - It can not be used to invoke a process
  - It can not be used to code program logic
  - It can not be used to specify flow of control
- OCL is a typed-language
  - Each expression has a type
  - Any UML Classifier can function as an OCL type: class, use case, actor, etc.

OCL, continued

- OCL was developed at IBM in 1995
- It was submitted into the Object Management Group's standards efforts in 1996
- It was merged into the UML standard in 1997 and became an official OMG standard in November 1997 (UML 1.1)
- Note: the UML metamodel's semantics are specified using OCL

Uses for the OCL

- To specify invariants on classes
- To specify type invariants for stereotypes.
- To describe pre- and post-conditions on operations and methods
- To describe guards
- To navigate associations
- To specify constraints on operations

OCL Building Blocks

- The basic building blocks of OCL are
  - Objects
  - Object properties
- Each object can have a type
  - Predefined Types
    - Basic types
    - Collections
  - User-defined model types

OCL Types

- Basic Types
  - Integer, Real, String, and Boolean
- Collection Types
  - Collection, Set, Bag, and Sequence
- Model Types
  - Customer, Perspective, Drafter, etc.
  - Also enumeration types

Expressions and Constraints

- Not all expressions are constraints
- For instance
  - "1+3" is a valid expression
  - Its result is "4" and its return type is "Integer"
- OCL constraint
  - An OCL Expression that returns a Boolean
    - e.g. {person.age > 1 + 3}

Context of an Expression

- Each OCL expression must have a context
  - The context is the model element to which the constraint is attached
  - This element is referred to as the "contextual element"

<u>Perspective</u>

objects->notEmpty()

Context of an Invariant

- The context of an invariant is always a type (class, interface, attribute, etc.)

<u>Customer</u>

Name = “Edward”

- Specifies that all Customer objects must have the name of Edward
- Not a very useful invariant!

Context of Pre- and Post-Conditions

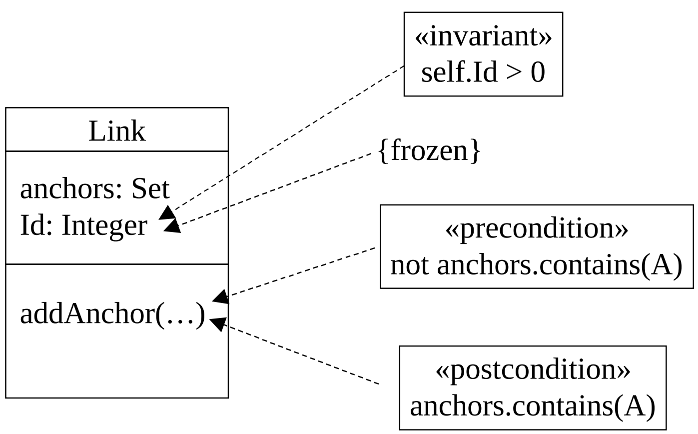
- The context of pre- and post-conditions is always an operation or method

<u>AddAnchor(L : Link, A : Anchor)</u>

pre: not L.anchors->Includes(A)

post: L.anchors->Includes(A)

Example in a Class Diagram



Keywords for Post-Conditions

- @pre
  - Attach this to an element to refer to the value of that element in the precondition
  - {age = age@pre + 1}
- result
  - Use this keyword to place a constraint on the return value of an operation
  - This does not define the value, it just constrains it
  - {result < 100 * input@pre} or {result = self.age}

The self keyword

- The self keyword is used to refer to the contextual element of an OCL constraint
- It can be omitted when the context is clear
- An example where its needed

<u>Container</u>

not Container->contains(self)

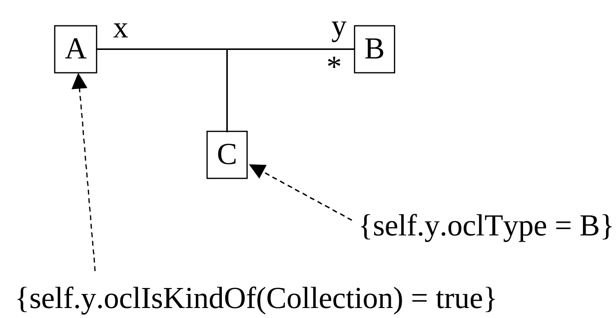
Navigating associations

- If A is related to B and A plays the role of x and B plays the role of y
  - Then, a constraint on A can refer to B
    - {self.y = …} or {self.b = …}
- If the multiplicity of B is greater than one
  - Then the type of self.y or self.b is a collection

Navigating Association Classes

- An association class can reference either end of its association via the same rules described on the previous slide
- However, the type of a reference is always an instance, never a collection, regardless of the multiplicities of the associated association
- This is because an instance of an association class is associated with only a single instance of an association

Example



Boolean Operations

- or, and, xor, not, =, <>, implies
  if a then b else b' endif
- Examples

  <u>Person</u>

  Self.age > 17 implies self.canVote() = true

  <u>Customer</u>

  title = (if isMale = true then "Mr." else "Ms." endif)

Semantics of Implies

- a implies b
- Returns true
  - If a is false (a falsity can imply anything)
  - Or a is true and b is true
- Returns false
  - If a is true and b is false

Integer and Real Operations

- =, <>
- <, >, <=, >=
- +, -, *, /
- mod, div, abs
- max, min
- round, floor

String Operations

- concat()
- size
- toLower()
- toUpper()
- substring()
- =, <>
- 'Ken'.size = 3

Enumerations

- Enumerated Types can be defined
  - – enum { blue, red, green, yellow }
  - – operations: =, <>
- In order to avoid name conflicts
  - – enumerated types are referenced like this:
    - #blue

<u>Customer</u>

(gender = #male) implies (title = “Mr.”)

Collections

- The OCL has one collection type and three subclasses
  - – Set - unordered set of items, no duplicates
  - – Bag - set of items, duplicates allowed
  - – Sequence - ordered Bag
- There are an amazing number of operations defined for these four types
  - – http://www.rational.com/uml/html/ocl/ocl_spe7.html

Collection Operations

- size
- count(object)
- includes(object)
- includesAll(Col.)
- isEmpty
- iterate(expression)
- eum()
- exists(expression)
- forAll(expression)
- notEmpty

A collection operation is always invoked using the arrow operator: anchors->isEmpty()

Other Operations

- Set Operations
  - – equals, minus
  - – symmetricDifference
  - – union()
  - – intersection
  - – asSequence()
  - – asBag()
  - – ...
- Sequence Ops
  - – append()
  - – prepend()
  - – subSequence()
  - – at()
  - – first
  - – last
  - – ...

Iteration functions

- select(expression) - returns collection

<u>Hyperweb</u>

link->select(L: Link | L.anchors->size() = 2)

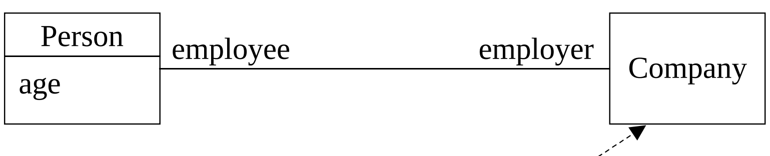
(Finds all links with exactly two anchors)

- reject(expression)
  - Opposite of select finds all members where the expression evaluates to false

Iteration Functions continued

- collect(expression)
  - The expression computes a value for each element of a collection. The values are returned in a collection
- forAll(expression)
  - Specifies that the expression holds for each member of the collection
- Exists(expression)
  - The expression is true of at least one element of the collection

Example



{employee->forAll(p: Person | p.age >= 18 and p.age <= 65) }

Meaning: A company's employees all must be between the ages of 18 and 65.

Syntax for forAll:
    forAll(element: Type | <expression>)

Note: Element and Type can be omitted

Instance Operations

- =, <>
- oclType
- oclIsKindOf()
- oclIsTypeOf()
- oclAsType()

Operator Precedence

- Dot (“.”) and arrow (“->”)
- Unary “not” and Unary minus (“-”)
- Multiplication (“*”) and Division (“/”)
- Addition (“+”) and Subtraction (“-”)
- Logical “and”, “or”, and “xor”
- Logical “implies”
- Logical “if then else endif”
- Logical comparison operators “<“, “>”, etc.

Type Casting

- A type can be cast to one of its subtypes using the following syntax
  object.oclAsType(Type2)
  -- object is now considered of type Type2
- Notes
  - If Type2 is not a valid type for object the result of the expression is undefined
  - Comments are started with two dashes

Type Conformance

- In constructing an expression, all sub-expressions and operators must “fit” properly → conformance
- Definition of Conformance
  - Type1 conforms to Type2 if an instance of Type1 can be substituted at each place where an instance of Type2 is expected.

Type Conformance Rules

- Type1 conforms to Type2 when they are identical
- Type1 conforms to Type2 when Type1 is a subtype of Type2
- Each type is a subtype of OclAny
- Type conformance is transitive
- Integer is a subtype of Real
- There are separate rules for collections on the next slide...

Type Conformance for Collections

- Every type Collection(T) is a subtype of OclAny
- Set(T), Bag(T), and Sequence(T) are subtypes of Collection(T)
- Collection(Type1) conforms to Collection(Type2) if Type1 conforms to Type2
- Set(T) does not conform to Bag(T) or Sequence(T)
- Bag(T) does not conform to Set(T) or Sequence(T)
- Sequence(T) does not conform to Set(T) or Bag(T)

Undefined

- An OCL expression may result in Undefined
  - Example: referencing a non-existent attribute on an object
- Any expression that has a sub-expression that is undefined evaluates to undefined
  - With two exceptions…
    - True or Undefined = True
    - False and Undefined = False

The OCL Book is good!

- Chapter 4 talks about modeling with constraints
  - Lots of examples
- Chapter 5 talks about extending OCL
- Appendix A documents all operations
- Appendix B provides a formal grammar for the Object Constraint Language