Object Constraint Language 2

Martin Nečaský
Dept. of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague
Type System

- predefined types in OCL standard library
  - generic types
    - OclAny, OclInvalid
  - basic types
    - similar to those in other known languages
    - Boolean, Real, Integer, String
  - collection types
    - results of navigation through associations in class diagrams
    - Collection
    - Set, Bag, OrderedSet, Sequence

- user-defined types
  - defined by a user in UML diagrams
  - every instantiable model element in UML diagrams is automatically a type in OCL expressions
OclInvalid

- OclInvalid = \{invalid\}
- conforms to all other types
  - i.e. invalid can be an instance of any type in OCL
- any property call applied on invalid results in invalid
  - except for oclIsUndefined() and oclIsInvalid()
OclAny

- behaves as a supertype for all OCL types
- \( a = b, a \neq b : \text{Boolean} \)
  - equals, not equals
- \( a.oclIsNew() : \text{Boolean} \)
  - true if \( a \) is created during performing an operation
  - can only be used in the operation postcondition
- \( a.oclIsUndefined() : \text{Boolean} \)
  - true if \( a \) is equal to invalid or equal to null
- \( a.oclIsInvalid() : \text{Boolean} \)
  - true if \( a \) is equal to invalid
- \( a.oclIsTypeOf(t : \text{Classifier}) : \text{Boolean} \)
  - true if \( a \) is of the type \( t \) but not a subtype of \( t \)
- \( a.oclIsKindOf(t : \text{Classifier}) : \text{Boolean} \)
  - true if \( a \) is of the type \( t \) or a subtype of \( t \)
- \( a.oclType() : \text{Classifier} \)
  - evaluates to type of \( a \)
Boolean

- represents common true/false values
- $a \text{ or } b$, $a \text{ xor } b$ : Boolean
- $a \text{ and } b$ : Boolean
- $a \text{ implies } b$ : Boolean
- $\text{not } (a)$ : Boolean
Real

- mathematical concept of real
- \( a + b, a - b, -a \) : Real
- \( a \times b \) : Real
- \( a / b \) : Real
- \( a < b, a > b \) : Boolean
- \( a \leq b, a \geq b \) : Boolean
- \( a.abs() \) : Real
- \( a.floor(), a.round() \) : Integer
- \( a.min(b : Real) \) : Real
- \( a.max(b : Real) \) : Real
**Integer**

- mathematical concept of integer
- subclass of `Real`
- $a + b$, $a - b$, $-a$ : Integer
- $a * b$ : Integer
- $a / b$ : Real
- $a.abs()$ : Integer
- $a.div(b : Integer)$ : Integer
- $a.mod(b : Integer)$ : Integer
- $a.min(b : Integer)$ : Integer
- $a.max(b : Integer)$ : Integer
String

- string is a sequence of characters in some suitable character set used to display information about the model
- a + b : String
- a.size() : Integer
- a.concat(b) : String
- a.substring(s, e: Integer) : String
  - substring starting and ending at positions between 1 and a.size()
- a.toInteger() : Integer, ...
- a.toUpperCase() : String
- a.toString() : String
- ...
Collections

- navigation via properties (association ends or attributes) results in a **Collection**

- **Collection** is an abstract type with 4 concrete sub-types:
  - Set
  - OrderedSet
  - Bag
  - Sequence
Navigation

- navigation via property \( p \) from \( a \)
  \[ a.p \]
- \( a \) is self or a variable with an instance
- results to
  - single instance (object or value) or an empty \( Set \) when the max multiplicity of \( p \) equals to 1
  - a \( Set \) when the max multiplicity of \( p \) is greater than 1
  - an \( OrderedSet \) when the max multiplicity of \( p \) is greater than 1 and \( p \) is modified by \{ordered\}

Formal Foundations of Software Engineering
Navigation

- navigation via a chain of properties $p_1 \ldots p_n$ from $a$

  $a.p_1.\ldots.p_n$

- results to $\text{Bag}$
Collection Constants

Set\{1,2,5,88\}
Sequence\{'apple', 'orange', 'pear'\}
Sequence\{1..(6+4)\}
Collection Iterator Operations

- Different operations which iterate a collection and create a new collection from the existing one
- **select** and **reject** – specify a selection from a collection
- **collect**
- **forAll**
- **exists**
- **closure**
- **iterate**
- ... and more
Collection Iterator Operations

- general syntax of iterator operations is
  
  \[
  \text{col->op(expression)}
  \]
  
  or
  
  \[
  \text{col->op(v | expression-with-v)}
  \]
  
  or
  
  \[
  \text{col->op(v: Type | expression-with-v)}
  \]

- sub-expressions of `expression` and `expression-with-v` implicitly start with the iteration variable when it is not present
  
  - contextual instance is referred by `self` reserved word but `self` is not implicit

context Person
inv: project
->op(startDate > self.startDate)

context Person
inv: self.project
->op(p | p.startDate > self.startDate)
Select and Reject Operations

- **select** specifies a subset of a collection containing all elements of the original collection for which a given expression evaluates to true

\[
\text{collection} \rightarrow \text{select(boolean-expression)}
\]
Select and Reject Operations

- **reject** specifies a subset of a collection containing all elements of the original collection for which a given expression evaluates to false.

\[
\text{collection} \rightarrow \text{reject} (\text{boolean-expression})
\]

\[
\sim
\]

\[
\text{collection} \rightarrow \text{select} (\text{not} (\text{boolean-expression}))
\]
context Person
inv: self.authoredDoc
    ->reject(d|self.project.output->includes(d))
    ->size()=0
Collect Operation

- `collect` specifies a collection that is computed from other collection
  - the new collection is not a sub-collection but contains elements computed/derived from the elements of the original collection
    - `collection->collect(expression)`
  - for each element of the original collection, evaluate the expression and put the result into the new collection
context Person

inv: self.authoredDoc->collect(project)->size()>0
Collect Operation

collection->collect(PropertyName)

~

collection.PropertyName
ForAll Operation

- **forAll** specifies a Boolean expression which must hold for all objects in a collection
  \[
  \text{collection} \rightarrow \text{forAll}(\text{expression})
  \]

- extended variant with more than one iterators of the same collection
  - iterator variables must be used in this case
    \[
    \text{collection} \rightarrow \text{forAll}(v1, v2 \mid \text{expression})
    \]
    \[
    \sim
    \]
    \[
    \text{collection} \rightarrow \text{forAll}(v1 \mid \text{expression})
    \]
    \[
    \text{collection} \rightarrow \text{forAll}(v2 \mid \text{expression})
    \]
ForAll Operation

context Person
inv: self.authoredDoc
    ->forall(d|self.project.output->includes(d))
context Project

inv: self.output -> forAll (d1, d2 | d1 <> d2 implies d1.serialNumber <> d2.serialNumber)
Exists Operation

- **exists** specifies a Boolean expression which must hold for at least one object in a collection
  
  \[ \text{collection} \to \text{exists}(\text{expression}) \]

- extended variant with more than one iterators is also possible
Closure Operation

- `closure` specifies a new collection created by a recursive application of an expression

  \[ \text{collection} \rightarrow \text{closure(\text{expression})} \]

- allows for expressing a transitive closure
  - the expressive power of OCL exceeds the power of relationally complete languages
Closure Operation

context Person
inv: self->asSet()->closure(boss)->size() <= 3
Iterate Operation

- **iterate** is a general loop operation
  
  \[
  \text{collection} \rightarrow \text{iterate}(
  \quad \text{element} : \text{Type1};
  \quad \text{result} : \text{Type2} = \langle \text{initial-value-expression} \rangle
  \quad \mid \langle \text{expression-with-element-and-result} \rangle
  \)
  
- **element** is iterator
- **result** accumulates the resulting value
  - it is also called accumulator
- for each element in **collection**, the **expression-with-element-and-result** is calculated using the previous value of **result**
Iterate Operation

source->forAll(v | body ) =
source->iterate(
    v; result : Boolean = true
    | result and body(v))

source->exists(v | body ) =
source->iterate(
    v; result : Boolean = false
    | result or body(v))
Other Operations on Collections

- `collection->count(object) : Integer`
  - the number of occurrences of the object in the collection

- `collection->includes(object) : Boolean`
  - true if the collection contains the object

- `collection->isEmpty() : Boolean`
  - true if the collection is empty

- `collection->size() : Integer`
  - the number of elements in the collection

- ... and more
Other Operations on Collections

- difference to the iterator operations is that the default context variable is `self`, not the iteration variable

```java
context Person
inv: self.authoredDoc-> excludesAll(reviewedDoc)
```

```java
context Person
inv: self.authoredDoc-> excludesAll(self.reviewedDoc)
```