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
- \( a = b, \ a \not\approx b : \text{Boolean} \)
  - 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
- \( a \ or \ b, \ a \ xor \ b : \ Boolean \)
- \( a \ and \ b : \ Boolean \)
- \( a \ implies \ b : \ Boolean \)
- \( \text{not} \ (a) : \ Boolean \)
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**
- \( a + b , \ a - b , \ -a : \ Integer \)
- \( a \times b : \ Integer \)
- \( a \div b : \ Real \)
- \( a.abs() : \ Integer \)
- \( a.div(b : \ Integer) : \ Integer \)
- \( a.mod(b : \ Integer) : \ Integer \)
- \( a.min(b : \ Integer) : \ Integer \)
- \( a.max(b : \ Integer) : \ Integer \)
String

- string is a sequence of characters in some suitable character set used to display information about the model
- \( a + b : \text{String} \)
- \( a \cdot \text{size}() : \text{Integer} \)
- \( a \cdot \text{concat}(b) : \text{String} \)
- \( a \cdot \text{substring}(s, e : \text{Integer}) : \text{String} \)
  - substring starting and ending at positions between 1 and \( a \cdot \text{size}() \)
- \( a \cdot \text{toInteger}() : \text{Integer}, \ldots \)
- \( a \cdot \text{toUpperCase}() : \text{String} \)
- \( a \cdot \text{toLowerCase}() : \text{String} \)
- \( \ldots \)
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 $\text{self}$ or a variable with an instance

results to
  
  - single instance (object or value) or an empty $\text{Set}$ when the max multiplicity of $p$ equals to 1
  - a $\text{Set}$ when the max multiplicity of $p$ is greater than 1
  - an $\text{OrderedSet}$ when the max multiplicity of $p$ is greater than 1 and $p$ is modified by $\{\text{ordered}\}$
Navigation

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

  $a.p_1.\ldots.p_n$

- results to $Bag$
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 \textit{expression} and \textit{expression-with-v} implicitly start with the iteration variable when it is not present
  - contextual instance is referred by \textit{self} reserved word but \textit{self} is not implicit

context Person
inv: project
\[ \text{->op(startDate > self.startDate)} \]

context Person
inv: self.project
\[ \text{->op(p|p.startDate > self.startDate)} \]
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

```
collection -> select(boolean-expression)
```

- for each element of the original collection, evaluate the expression and put the result into the new collection
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.

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

\[
\sim
\]

\[
\text{collection} \rightarrow \text{select} (\text{not(} \text{boolean-expression})\text{)}
\]
Select and Reject Operations

```
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
    \[ \text{collection} \rightarrow \text{collect}(\text{expression}) \]
  - for each element of the original collection, evaluate the expression and put the result into the new collection
context Person

inv: (self->collect(authoredDoc)->size()) +
    self->collect(reviewedDoc)->size()) > 0
Collect Operation

collection->\texttt{collect}(PropertyName)

\[\sim\]

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:
    \[ \text{collection} \rightarrow \text{forAll} (v1,v2 \mid \text{expression}) \]
    \[ \sim \]
    \[ \text{collection} \rightarrow \text{forAll} (v1 \mid \text{expression}) \]
    \[ \text{collection} \rightarrow \text{forAll} (v2 \mid \text{expression}) \]
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

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

- 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
context Person

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

- **iterate** is a general loop operation

```
iterate(collection->iterate(
   element : Type1;
   result : Type2 = <initial-value-expression>
   | <expression-with-element-and-result>)
```

- **element** is iterator
- **result** accumulates the resulting value
  - it is also called accumulator
- for each element in **collection**, the **expression** is calculated using the previous value of **result**
iterate Operation

\[
\text{source->forall}(v \mid \text{body}) =
\text{source->iterate}(v; \text{result} : \text{Boolean} = \text{true} \mid \text{result} \text{ and body}(v))
\]

\[
\text{source->exists}(v \mid \text{body}) =
\text{source->iterate}(v; \text{result} : \text{Boolean} = \text{false} \mid \text{result} \text{ or body}(v))
\]
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