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

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

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

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

- mathematical concept of real
- $a + b$, $a - b$, $-a$ : Real
- $a \times b$ : Real
- $a \div b$ : Real
- $a < b$, $a > b$ : Boolean
- $a \leq b$, $a \geq b$ : Boolean
- $a.\text{abs}()$ : Real
- $a.\text{floor}()$, $a.\text{round}()$ : Integer
- $a.\text{min}(b : \text{Real})$ : Real
- $a.\text{max}(b : \text{Real})$ : 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
- \( a + b : \text{String} \)
- \( a\cdot\text{size}() : \text{Integer} \)
- \( a\cdot\text{concat}(b) : \text{String} \)
- \( a\cdot\text{substring}(s, e: \text{Integer}) : \text{String} \)
  - 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 three 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}`
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
  
  \[ \text{col} \to \text{op}(\text{expression}) \]
  
  \[ \text{or} \]
  
  \[ \text{col} \to \text{op}(v \mid \text{expression-with-v}) \]
  
  \[ \text{or} \]
  
  \[ \text{col} \to \text{op}(v: \text{Type} \mid \text{expression-with-v}) \]

- 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

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

\[
\text{context Person} \quad \text{inv: self.project} \\
\to \text{op}(p \mid p.\text{startDate} \>	ext{self.startDate})
\]
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

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

- 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.

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

\[
\sim
\]

\[
\text{collection} \rightarrow \text{select} (\text{not(boolean-expression)})
\]
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
    
    \[
    \text{collection} \rightarrow \text{collect(} \text{expression} \text{)}
    \]
  - for each element of the original collection, evaluate the expression and put the result into the new collection
context Person

inv: self.authoredDoc->collect(d|d.project)
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
    \[
    \text{collection} \rightarrow \text{forAll}(v1,v2 \ | \ expression) \\
    \sim \\
    \text{collection} \rightarrow \text{forAll}(v1 \ | \ collection) \\
    \text{collection} \rightarrow \text{forAll}(v2 \ | \ expression)
    \]
context Person
definition 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

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

- 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

  \[ \text{collection}\rightarrow\text{iterate}(\]
  \[ \quad \text{element} : \text{Type1} ; \]
  \[ \quad \text{result} : \text{Type2} = \langle \text{initial-value-expression} \rangle \]
  \[ \quad \mid \langle \text{expression-with-element-and-result} \rangle \rangle \]

- **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

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)

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