Génie Logiciel Avancé

Part III : UML/OCL
Plan of the Chapter

- Syntax & Semantics of OCL Constraint Expressions
  - Logic
  - Connection to UML
  - Basic Data-Types
  - Collection Types

- Semantics & Semantics of OCL Constraints
  - Class Invariants
  - Pre- and Post-Conditions

- Ultimate Goal: Specify system components for test and verification
Motivation: What is **OCL**

- Acronym for **Object Constraint Language**
- Semantically: ... is a typed « Annotation language » (like JML, VCC) that constrains an underlying **state**
- *Can be used (in principle) in all diagram types, where annotations were used ...*

```text
Compteur
id:Integer

{ id must be larger 0 }
```

...
Motivation: What is OCL

- ... 
- ... exists since UML 1.4, and is heavily used to define the UML Meta-Model itself
- ... is most comprehensively described in J. Warmer, A. Kleppe: « The Object Constraint Language (Second edition) », Addison-Wesley, 2003
- ... has a formal semantics, at least in some versions (OMG document ptc/03-10-14, see also HOL-OCL)
- ... is usually not directly supported in CASE TOOLS, but quite a few PlugIns are available.
Why OCL?

Well, not everything can be expressed in diagrams ...

Informal annotations should be expressed formally, too.

In particular, for:

- Classes and Associations (invariants)
- Operations (pre/post-conditions)
- Transition conditions in State Models (guards)
Why OCL?

Well, not everything can be expressed in diagrams ...

Informal annotations can be expressed in OCL as invariants (supported by SOME UML-editors as annotations)
Why OCL?

Well, not everything can be expressed in diagrams ...

<table>
<thead>
<tr>
<th>Compteur</th>
<th>context Compteur: inv A: id &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>id:Integer</td>
<td></td>
</tr>
</tbody>
</table>

... or in a textual representation (expressing the link in by the context <class> notation) in an external file associated to the UML model.
A first Glance to an Example: Bank

Opening a bank account. Constraints:

- there is a blacklist
- there is a present of 15 euros in the initial account
- account numbers must be distinct.
A first Glance to an Example: Bank (2)

context Compte:
  inv unique: Compte.allInstances-&gt;isUnique(c | c.no)

class Banque::ouvrirCompte(nomC: String) : Integer

pre: Personne.allInstances-&gt;forall(p | p.nom &lt;&gt; nomC)
post: Personne.allInstances-&gt;isUnique(p | p.nom = nomC
       and p.oclIsNew()
       and )

and Compte.allInstances
   &gt;isUnique(c | c.titulaire.nom = nomC
       and c.oclIsNew()
       and c.solde = 15)

and Personne.allInstances
   &gt;includesAll(Personne.allInstances@pre)
and Compte.allInstances
   &gt;includesAll(Compte.allInstances@pre)
OCL: a specification language?

- In the following, we will discuss the OCL Language in more detail ...
OCL: a specification language?

- OCL is not an imperative language: no side-effects, no sequencing.

- However, OCL is similar to functional languages like Ocaml, SML, Haskell, ... and has a quite operational flavor: it is executable. (No unbounded quantifiers in standard OCL !!!)
  - let x1 : T1 = E1, ..., xn : Tn = En in E
  - fold-like operators (called iterators):
    Collection->iterate(elem; acc:Integer=0 | acc + 1)
OCL: a specification language?

- OCL has a special exceptional element in each type:
  
  OclUndefined or invalid (UML2.1)

  which can be result of illegal computations like
  
  $1 / 0$ or illegal accesses to objects in a state.
OCL: a specification language?

- All operations in OCL are strict, i.e. satisfy:

  \[ f(\text{invalid}) \neq \text{invalid} \]

  \[ g(x,\text{invalid}) = \text{invalid} \ldots \]

- except the operation \( \_\text{.isInvalid}() \):

  \[ \text{invalid.isInvalid}() = \text{true} \]

- Semantically, it is adequate to consider \( \text{invalid} \) as a raised exception and \( \text{isInvalid}() \) as catching it...
Syntax and Semantics of OCL

- **OCL is a typed language.** The underlying types of an OCL expression can be:
  - Basic Types: Boolean, Integer, Real, String
  - the hierarchy Collection with sub-types: Set, Ordered Set, Bag, Sequence
  - the class-types of an associated class model
  - the enumeration type of an associated class model
  - special types: OclAny, OclVoid, Tuples
Syntax and Semantics of OCL

- The logical core language: expressions of type Boolean:
  - not $E$, $E$ or $E'$, $E$ and $E'$, $E$ implies $E'$
  - $E = E'$, $E <> E'$,
  - if $C$ then $E$ else $E'$ endif

- Quantifiers are handled on the base of (finite) collections (see later):
  
  ```
  Collection->forall(x:Type| E(x))
  Collection->exists(x:Type| E(x))
  ```
Syntax and Semantics of OCL

Note that:

- \( E = E' \), \( E \not\sim E' \)

denote the « strict equality »!
For logical reasoning, we use the « logical equality »: \( E \equiv E' \)

\((E = invalid) \equiv invalid\) \equiv true

is a true logical statement!
The arithmetic core language. 
expressions of type \texttt{Integer} or \texttt{Real}:

- $E, E + E'$,
- $E * E', E / E'$,
- $E.abs(), E.div(E), ...$
Syntax and Semantics of OCL

- The expressions of type String:
  - $E$.concat($E$)
  - $E$.size()
  - $E$.substring($i$, $j$)
  - $E$.toInteger()
  - $E$.toReal()
  - 'Hello'
Syntax and Semantics of OCL Collections

- The power of the language comes from the expressions over Collection's, i.e. Set, Ordered Set, Bag, Sequence.

- Operations on Collections $C$ are:
  - $C \rightarrow$ iterate($\text{elem}$; acc:$T=\text{init} \mid \text{op}(\text{elem}, \text{acc})$)
  - ... and its special cases:
    - $C \rightarrow$ forall($x$:Type$\mid E(x)$)
    - $C \rightarrow$ exists($x$:Type$\mid E(x)$)
    - $C \rightarrow$ select($x$:Type$\mid P(x)$) -- «$\{x::C \mid P(x)\}$»
    - $C \rightarrow$ reject($x$:Type$\mid P(x)$) -- «$\{x::C \mid \neg P(x)\}$»
    - $C \rightarrow$ any($x$:Type$\mid P(x)$) -- «$\exists x::C \mid P(x)$» some
    - $C \rightarrow$ one($x$:Type$\mid P(x)$) -- «$|\{x::C \mid P(x)\}|=1$»
Syntax and Semantics of OCL Collections

- ... 
- `C->size()` 
- `C->count(e)` 
- `C->isUnique(e)`  -- e has different value for each element 
- `C=C', C<>C'`  -- different semantics according type 
- `C->isEmpty(), C->notEmpty()` 
- `C->sum()`  -- for a collection of numbers 
- `C->includes(e), `  -- «e∈C» 
- `C->includesAll(C')`  -- «C'⊆C» 
- `C->excludes(e), `  -- «e ∉ C» 
- `C->excludesAll(C')`  -- «¬(C'⊆C)» 
- ...
Syntax and Semantics of OCL Collections

- \( C \rightarrow \text{including}(e) \), \quad \text{--} \quad \{ C \cup \{x\} \}
- \( C \rightarrow \text{excluding}(e) \), \quad \text{--} \quad \{ C \setminus \{x\} \}
- \( C \rightarrow \text{union}(C') \), \quad \text{--} \quad \{ C \cup C' \}
- \( C \rightarrow \text{intersection}(C') \), \quad \text{--} \quad \{ C \cap C' \}
- \( C \rightarrow \text{flatten}() \) \quad \text{--} \text{flattens collections of collections}
  
  \text{ex. Set}\{\text{Set{}},\text{Set}\{1\}\}. \text{flatten}() = \text{Set}\{1\}

- \text{conversions:}
  - \( C \rightarrow \text{asSet}() \),
  - \( C \rightarrow \text{asBag}() \),
  - \( C \rightarrow \text{asSequence}() \),
  - \( C \rightarrow \text{asOrderedSet}() \)
Syntax and Semantics of OCL Sequences

Additionally to the Collection operations, Sequences $S$ have the following ops:

- $S$->first()
- $S$->last()
- $S$->at($i$) -- for $i$ between 1 et size()
- $S$->append($e$) -- append at the end
- $S$->prepend($e$) -- append at the beginning

- Finally, denotations of collections:
  - Set{1,2,3}, Sequence{1,3,2,7,1}, ...