OCL : Object Constraint Language

http://www-adele.imag.fr/~jmfavre

Cette fiche résume de manière synthétique mais incomplète et très informelle les principales caractéristiques du langage OCL. Pour plus de précision, se reporter au document de référence UML.

Types des identificateurs utilisés ci-dessous

i : Integer	c : Collection(T)	os : OrderedSet(T)	cs: constant
r : Real	st: Set(T)	t : Tuple(…)	pd : predicat
b : Boolean	bg : Bag(T)	id: identificateur	e : expression
s : String	sq : Sequence(T)	pt: property	ns: namespace

Constructions syntaxiques

ct	e op e	ns**::** ... ns**::**id
id	e**.** id	**if** pd **then** e **else** e **endif**
self	e **.** pt (e, ... , e)	**let** id = e **:** T, id2 = e:T, ... **in** e2
	c **->** pt (e, ..., e)	

Types, valeurs et opérations dans la bibliothèque standard

Integer	1, -5, 34	i+i2, i-i2, i*i2, i.div(i2), i.mod(i), i.abs, i.max(i2), i.min(i2), <, >, <=, >=
Real	1.5, 1.34, ...	r+r2, r-r2, r*r2, r/r2, r.floor, r.round, r.max(r2), r.min(r2), <, >, <=, >=
Boolean	true, false	not b, b and b2, b or b2, b xor b2, b implies b2
String	'', 'une chaine'	s.size(), s.concat(s2), s.substring(i1,i2), s.toUpper(), s.toLower(), s.toInteger(), s.toReal()
Enumeration	Jour::Lundi, Jour::Mardi, …	=, <>
TupleType(x : T1, y : T2, z : T3)	Tuple { y : T2 = …, x = … , z = … }	t.x t.y t.z
Collection(T)		c->size(), c->includes(o), c->excludes(o), c->count(o), c->includesAll(c2) c->excludesAll(c2), c->isEmpty(), c->notEmpty(), c->sum() c->exists(p), c->forall(p), c->isUnique(e), c->sortedBy(e), c->iterate(e)
Set(T)	Set {1,5,10,3}, Set{}	st->union(st2), st->union(bg), st->intersection(st2), st->intersection(bg) st - st2, st->including(e), st->excluding(e), st->symmetricDifference(st2) st->select(e), st->reject(e), st->collect(e), st->count(e), st->flatten(), st->asSequence(), st->asBag()
Bag(T)	Bag {1,5,5} Bag {}	bg->union(bg2), bg->union(st), bg->intersection(bg2), bg->intersection(st) bg->including(e), bg->excluding(e), bg->count(e), bg->flatten() bg->select(e), bg->reject(e), bg->collect(e) bg->asSequence(), bg->asSet()
OrderedSet(T)	OrderedSet{10,4,3} OrderedSet{}	…
Sequence(T)	Sequence{5,3,5} Sequence{}	sq->count(e), sq->union(sq2), sq->append(e), sq->prepend(e), sq->insertAt(i,o) sq->subSequence(i1,i2), sq->at(i), sq->first(), sq->last(), sq->indexOf(o) sq->including(e), sq->excluding(e) sq->select(e), sq->reject(e), sq->collect(e), sq->iterate(e) sq->asBag, sq->asSet

Exemples :

```
context Personne
  inv :  enfants->forall( e | e.age < self.age - 7)
  inv :  enfants->forall( e : Personne | e.age < self.age - 7)
  inv i3 :  enfants->forall( e1,e2 : Personne |  e1 <> e2 implies e1.prénom <> e2.prénom)
  inv i4 :  self.enfants -> isUnique ( prénom )
  def cousins : Set(Personne) = parents.parents.enfants.enfants->excluding( parents.enfants )->asSet()
context Personne ::age : Integer
  init : 0
context Personne :: estMarrié : Boolean
  derive : conjoint->notEmpty()
context Personne::salaire() : integer
```

```
  post : return > 5000
context Compagnie::embaucheEmployé( p : Personne)
 pre pasPrésent : not (employés->includes(p))
 post embauché : employés->includes(p)
context Personne::grandsParents() : Set(Personne)
  body : parents.parents->asSet()

(age<40 implies salaire>1000) and (age>=40 implies salaire>2000)
if age<40 then salaire > 1000 else salaire > 2000 endif
salaire > (if age<40 then 1000 else 2000 endif)
nom= nom.substring(1,1).toUpper().concat(
       nom.substring(2,nom.size()).toLower())
épouse->notEmpty() implies épouse.sexe = Sexe::Feminin
Set { 3, 5, 2, 45, 5 }->size()
Sequence { 1, 2, 45, 9, 3, 9 } ->count(9)
Sequence { 1, 2, 45, 2, 3, 9 } ->includes(45)
Bag { 1, 9, 9, 1 } -> count(9)
c->asSet()->size() = c->size()
c->count(x) = 0
Bag { 1, 9, 0, 1, 2, 9, 1 } -> includesAll( Bag{ 9,1,9} )
self.enfants ->select( age>10 and sexe = Sexe::Masculin)
self.enfants ->reject(enfants->isEmpty())->notEmpty()
membres->any(titre='président')

self.employé->select(age > 50)
self.employé->select( p | p.age>50 )
self.employé->select( p : Personne | p.age>50)
self.enfants->forall(age<10)
self.enfants->exists(sexe=Sexe::Masculin)
self.enfants->one(age>=18)self.enfants->forall( age < self.age )
self.enfants->forall( e | e.age < self.age - 7)
self.enfants->forall( e : Personne | e.age < self.age - 7)
self.enfants->exists( e1,e2 | e1.age = e2.age )
self.enfants->forall( e1,e2 : Personne |
                        e1 <> e2 implies e1.prénom <> e2.prénom)
self.enfants -> isUnique ( prénom )
self.enfants->collect(age) = Bag{10,5,10,7}
self.employés->collect(salaire/10)->sum()
self.enfants.enfants.voitures
enfants.enfants.prénom = Bag{ 'pierre', 'paul', 'marie', 'paul' }

enfants->collectNested(enfants.prénom) =
       Bag { Bag{'pierre', 'paul'}, Bag{'marie','paul'}

Sequence{1..s->size()-1} -> forall(i | s.at(i) < s.at(i+1) )
enfants->sortedBy( age )
enfants->sortedBy( enfants->size() )->last()
let ages = enfants.age->sortedBy(a | a) in ages.last() - ages.first()
s.Emploi
p.Emploi
s.Emploi->collect(salaire)->sum()

s.Emploi.salaire->forall(x | x>500)
p.Evaluation[chefs]
p.Evaluation[employés]
p.Evaluation[chefs].note -> sum()s.Emploi-> select(salaire<1000).employé
p.enfants->select(oclIsKindOf(Femme)).asTypeOf(Set(Femme))
   ->select(nomDeJF <> nom)
Personne.allInstances->size() < 500
Personne.allInstances->forall(p1,p2 | p1<>p2 implies p1.numsecu <> p2.numsecu)
Personne.allInstances->isUnique(numsecu)
```