# Navigation in OCL
## (Single-Valued versus Multi-Valued OCL Expressions)

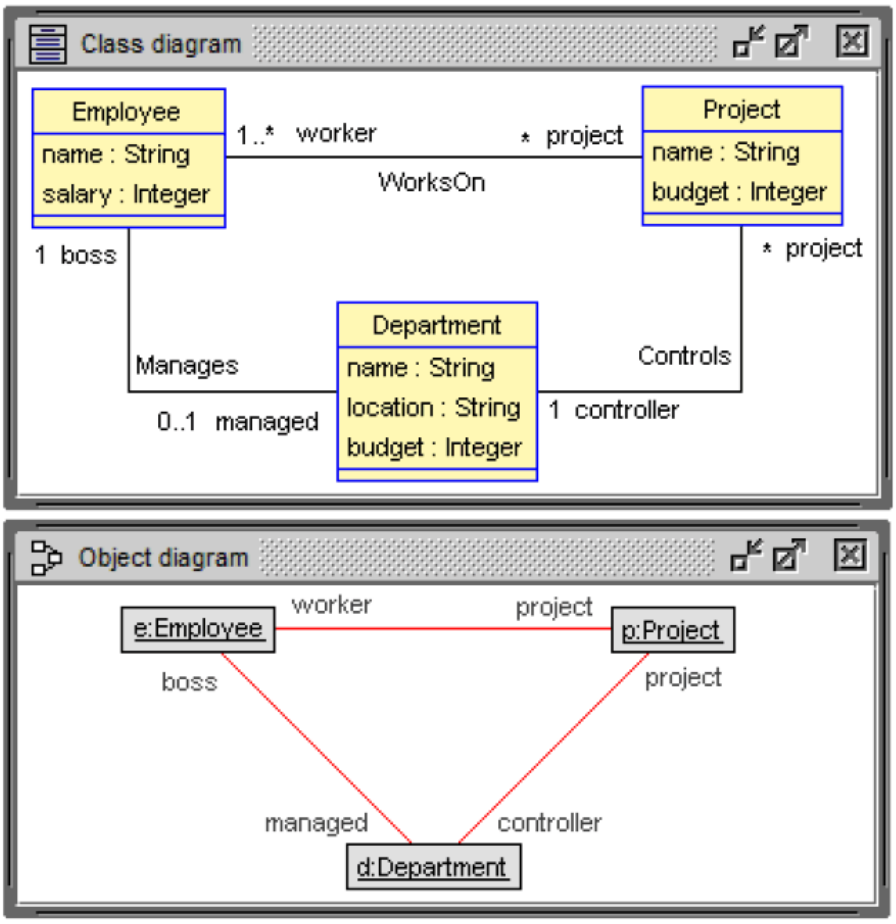
Martin Gogolla

2014-06-26

**The good news: One can easily navigate with OCL in a class diagram by employing role names. Starting from a single object or a single class variable, one can build OCL expressions by appending role names.**

**The bad news: It is not so easy to fully understand the type of the resulting OCL expression.**

**Example**




**In the example, the most common single- and multi-valued multiplicities are present: 0..1, 1, 0..*, 1..*.**

**dot shortcut & collect VS collectNested:**

**- if 'role' is single-valued (0..1 or 1): 'multiValuedExpr.role' ~~>**
**'multivaluedExpr->collectNested(x|x.role)'**

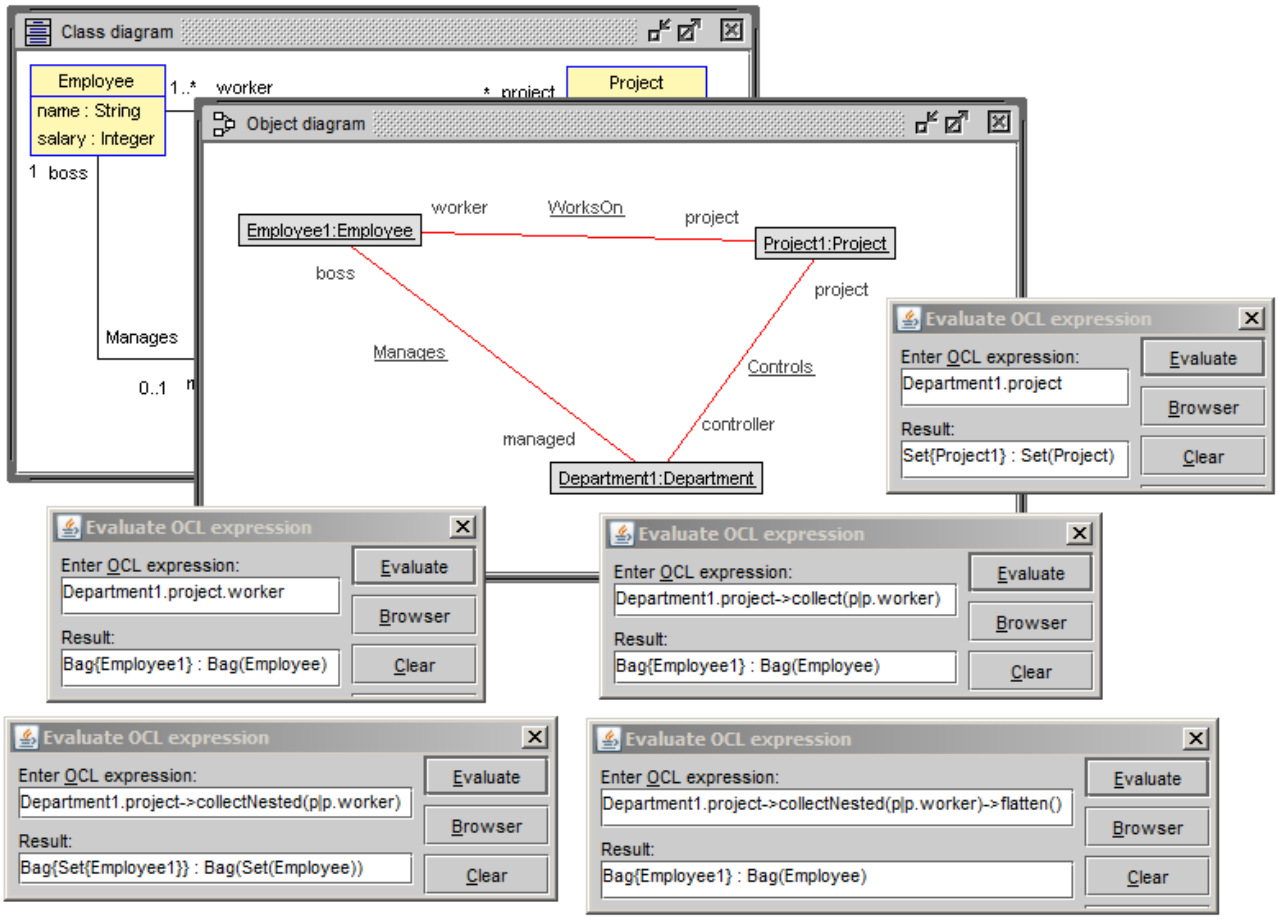
**- if 'role' is multi-valued (upper bound > 1): 'multiValuedExpr.role' ~~>**
**'multivaluedExpr->collectNested(x|x.role)->flatten()'**

**- 'expr1->collect(x|x.expr2)' ~~>**
**'expr1->collectNested(x|x.expr2)->flatten()'**

**The following OCL expressions employing a single role name or two role names can be built. There are no other expressions with one or two role names. The type of the expression can be single-valued, set-valued or bagvalued. For ease of identification, the single-valued role names are underlined.**

```
use> ?e.project
     Set{p}: Set(Project)
use> ?e.managed
     d: Department
use> ?p.worker
     Set{e}: Set(Employee)
use> ?p.controller
     d: Department
use> ?d.boss
     e: Employee
use> ?d.project
     Set{p}: Set(Project)

use> ?e.project.controller   dot shortcut
     Bag{d}: Bag(Department) e.project->collectNested(p|p.controller)
use> ?e.managed.project
     Set{p}: Set(Project)
use> ?p.worker.managed       dot shortcut
     Bag{d}: Bag(Department) p.worker->collectNested(e|e.managed)
use> ?p.controller.boss
     e: Employee
use> ?d.boss.project
     Set{p}: Set(Project)
use> ?d.project.worker     dot shortcut and flatten
     Bag{e}: Bag(Employee) d.project->collectNested(p|p.worker)->flatten

use> ?e.project.worker     dot shortcut and flatten
     Bag{e}: Bag(Employee) e.project->collectNested(p|p.worker)->flatten
use> ?e.managed.boss
     e: Employee
use> ?p.worker.project    dot shortcut and flatten
     Bag{p}: Bag(Project) p.worker->collectNested(e|e.project)->flatten
use> ?p.controller.project
     Set{p}: Set(Project)
use> ?d.boss.managed
     d: Department
use> ?d.project.controller   dot shortcut
     Bag{d}: Bag(Department) d.project->collectNested(p|p.controller)
```

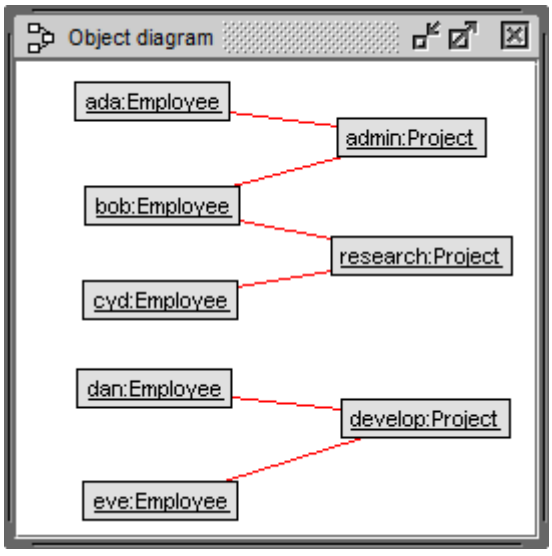
```
use> ?Set{-4,-2,2,4}->collectNested(i|Set{i*i,i*i*i*i})
     Bag{Set{4,16},Set{4,16},Set{16,256},Set{16,256}} : Bag(Set(Integer))

use> ?Set{-4,-2,2,4}->collectNested(i|Set{i*i,i*i*i*i})->flatten()
     Bag{4,4,16,16,16,16,256,256} : Bag(Integer)

use> ?Set{-4,-2,2,4}->collect(i|Set{i*i,i*i*i*i})
     Bag{4,4,16,16,16,16,256,256} : Bag(Integer)

use> ?Seq{-4,-2,2,4}->collectNested(i|Set{i*i,i*i*i*i})
     Seq{Set{16,256},Set{4,16},Set{4,16},Set{16,256}} : Seq(Set(Integer))
```



```
model EDP

class Employee
attributes
  name : String
  salary : Integer
end

class Department
attributes
  name : String
  location : String
  budget : Integer
end

class Project
attributes
  name : String
  budget : Integer
end

association Manages between
  Employee[1] role boss
  Department[0..1] role managed
end

association WorksOn between
  Employee[1..*] role worker
  Project[*] role project
end

association Controls between
  Department[1] role controller
  Project[*] role project
end
```

```
collection kinds
- Project::worker : Set(Employee)
- Project::substitutionPriority : OrderedSet(Employee)
- Department::workshopSpeaker : Sequence(Employee)
- Department::formerProjectWorker : Bag(Employee)

collection operations
- forAll, exists
- select, collect
- isEmpty, notEmpty, size
- iterate
- closure

Employee.allInstances->forAll(e|e.salary>=0)

Employee.allInstances->exists(e|e.project->size>=2)

Employee.allInstances->select(e|e.managed->isEmpty)

Employee.allInstances->select(e|e.managed->notEmpty)->
  collect(e|Tuple{NAME:e.name,DEPT:e.managed.name})

Department.allInstances->collect(d|
  Tuple{DEPT:d.name,BUD:d.budget,
        SUMBUD:d.project->iterate(p;s:Integer=0|s+p.budget)})

Employee.allInstances->collect(e|
  Tuple{EMP:e,CLIQUE:Set{e}->closure(e|e.project.worker)})
```

```
!new Employee('ada')
!new Employee('bob')
!new Employee('cyd')
!new Employee('dan')
!new Employee('eve')
!new Project('admin')
!new Project('research')
!new Project('develop')
!insert (ada,admin) into WorksOn
!insert (bob,admin) into WorksOn
!insert (bob,research) into WorksOn
!insert (cyd,research) into WorksOn
!insert (dan,develop) into WorksOn
!insert (eve,develop) into WorksOn
```




```
Bag{Tuple{EMP=dan,CLIQUE=Set{dan,eve}},
    Tuple{EMP=eve,CLIQUE=Set{dan,eve}},
    Tuple{EMP=ada,CLIQUE=Set{ada,bob,cyd}},
    Tuple{EMP=bob,CLIQUE=Set{ada,bob,cyd}},
    Tuple{EMP=cyd,CLIQUE=Set{ada,bob,cyd}}} :
  Bag(Tuple(EMP:Employee,CLIQUE:Set(Employee)))
```