Object Constraint Language

Analyzing the formal Specifications

Recall the Employees, Departments and Projects Example

```
model Company

-- classes

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
```

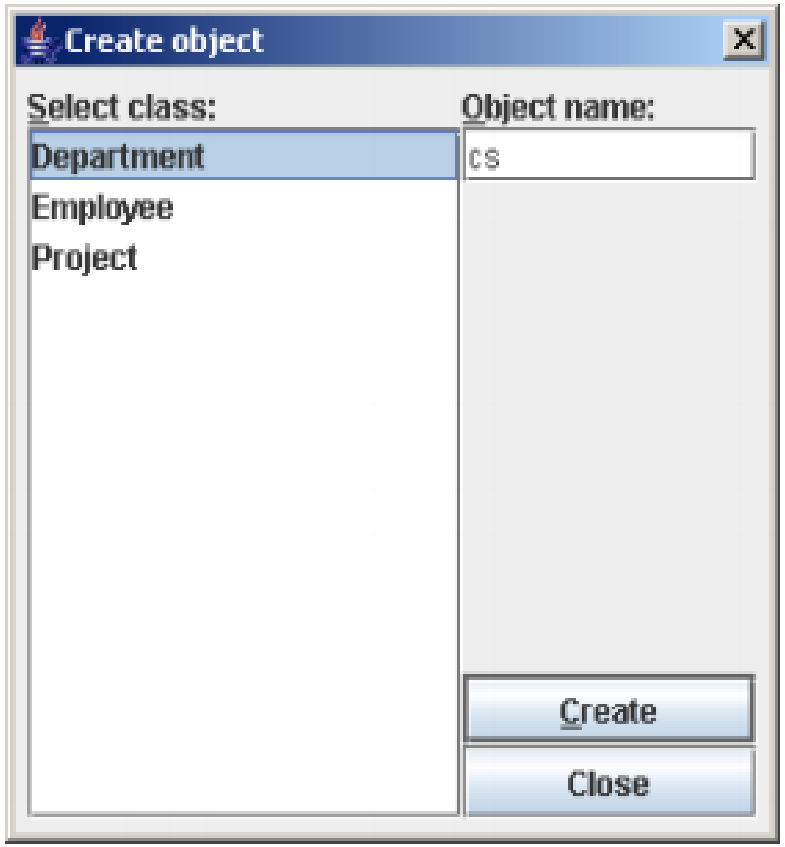
```
-- associations

association WorksIn between
  Employee[*]
  Department[1..*]
end

association WorksOn between
  Employee[*]
  Project[*]
end

association Controls between
  Department[1]
  Project[*]
end
```

Analyzing model Company in USE

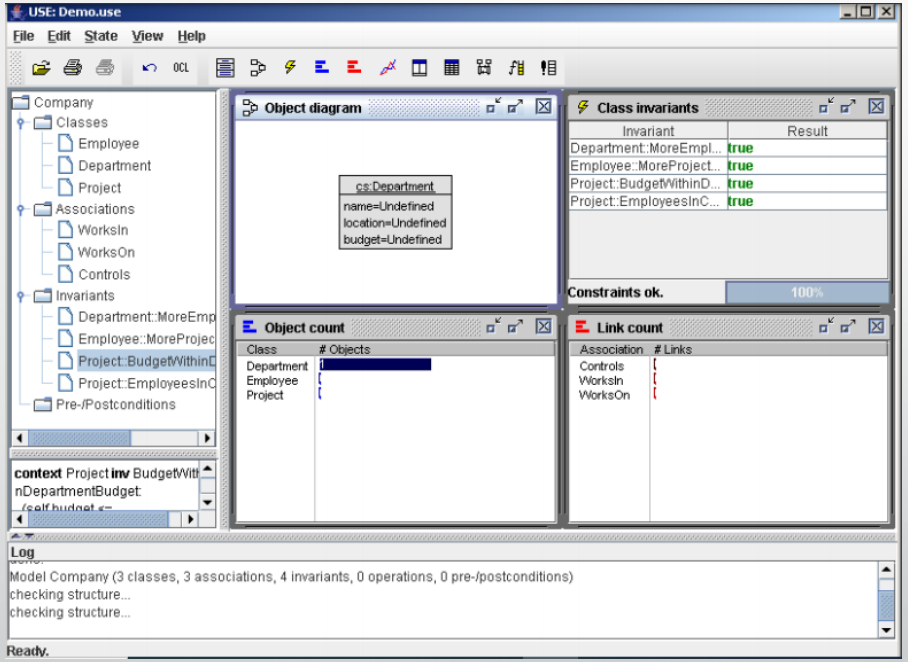
- After specifying a UML model within a .use file you can open it with USE. The USE system will parse and type check the file automatically.
- **Creating System States**
  - Objects can be created by selecting a class and specifying a name for the object. The menu command State|Create object opens a dialog where this information can be entered.
  - Alternatively, the following command can be used at the shell to achieve the same effect.

```
use> !create cs:Department
```

  - And, even simpler, an object can be created via drag & drop. Just select a class in the model browser and drag it to the object diagram



Main Window after Object Creation

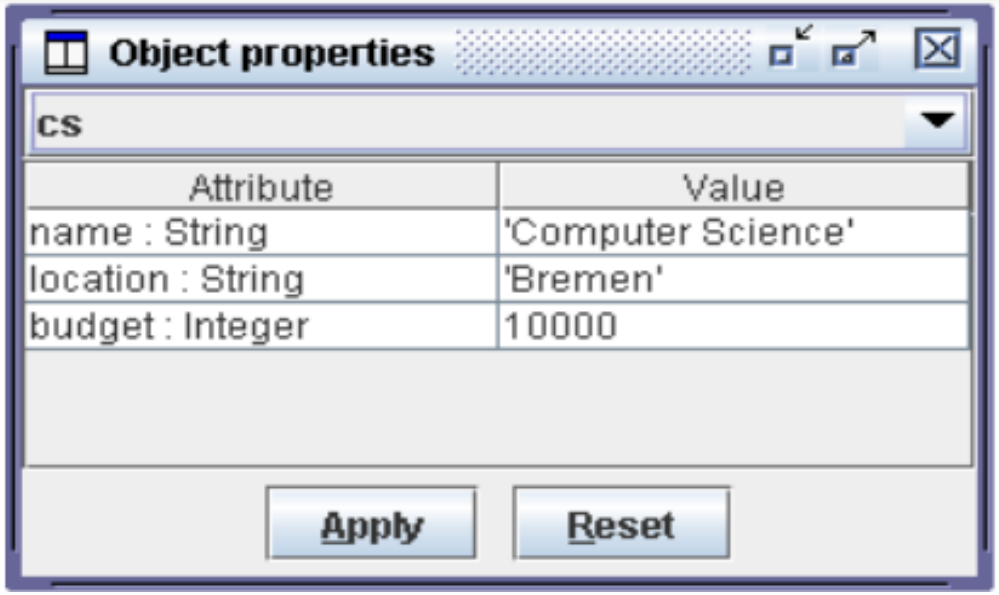


Setting System State

- You can see that the attribute values of the Department object are all undefined. For changing attribute values, we can use the set command:

```
use> !set cs.name := 'Computer Science'
use> !set cs.location := 'Bremen'
use> !set cs.budget := 10000
```

- Attributes can also be changed with an Object Properties View. If you choose View|Create| Object Properties from the View menu and select the cs object, you get the view shown in figure, where you can inspect and change attributes of the selected object.



Contd.

- We continue by adding two Employee objects and setting their attributes.

```
use> !create john : Employee
use> !set john.name := 'John'
use> !set john.salary := 4000
use> !create frank : Employee
use> !set frank.name := 'Frank'
use> !set frank.salary := 4500
```

- Now we have three objects, a department and two employees, but still no connections between them.

Model Inherent Constraints

- The invariant view indicates some problem with the new system state. The message says: Model inherent constraints violated.
- Model inherent constraints are constraints defined implicitly by a UML model (in contrast to explicit OCL constraints).
- The details about this message are shown in the log panel at the bottom of the screen.
- They are also available by issuing a check command at the prompt:

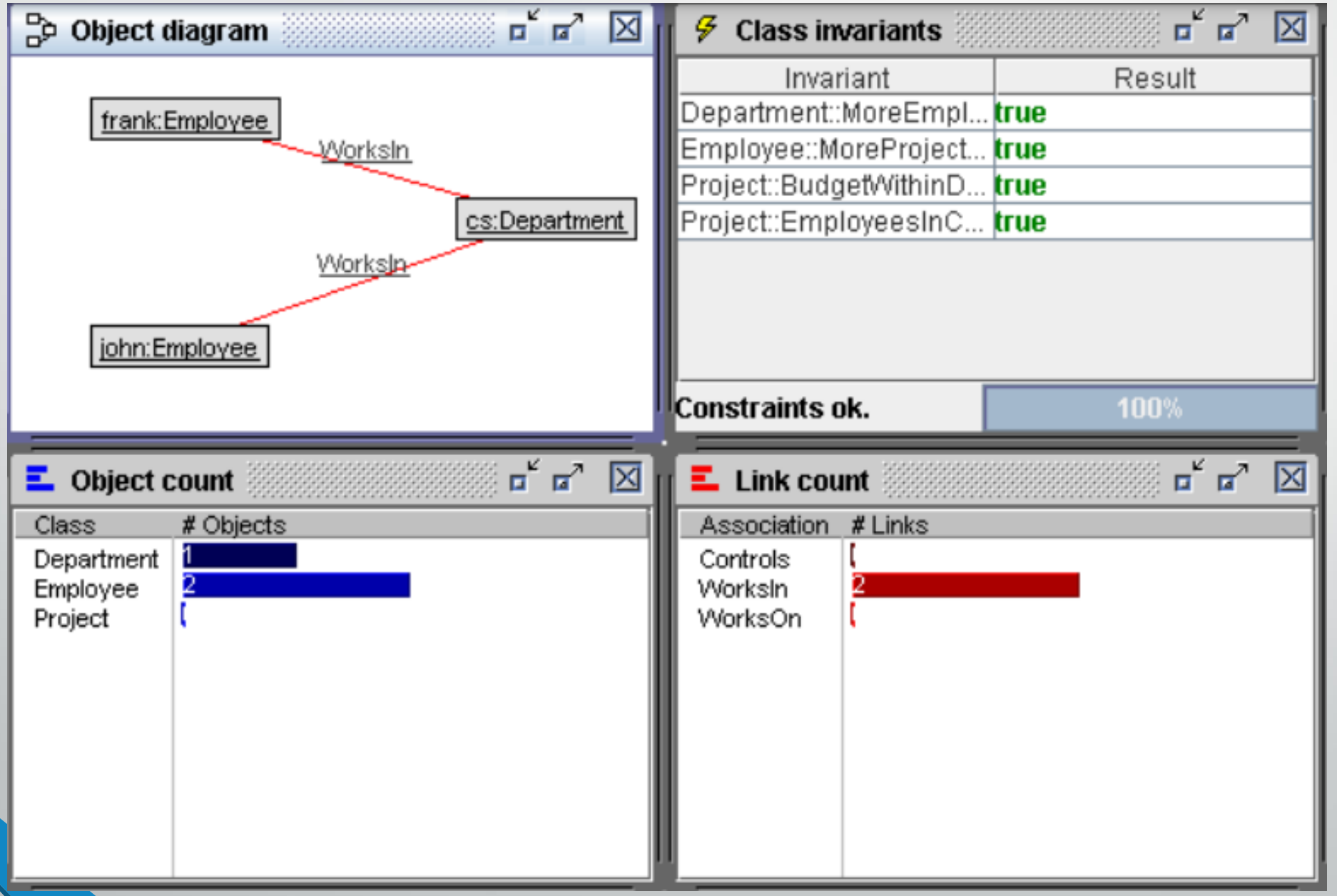
```
use> check
checking structure...
Multiplicity constraint violation in association 'WorksIn':
  Object 'frank' of class 'Employee' is connected to 0 objects of
    class 'Department' via role 'department'
  but the multiplicity is specified as '1..*'.
Multiplicity constraint violation in association 'WorksIn':
  Object 'john' of class 'Employee' is connected to 0 objects of
    class 'Department' via role 'department'
  but the multiplicity is specified as '1..*'.
...
```

Contd.

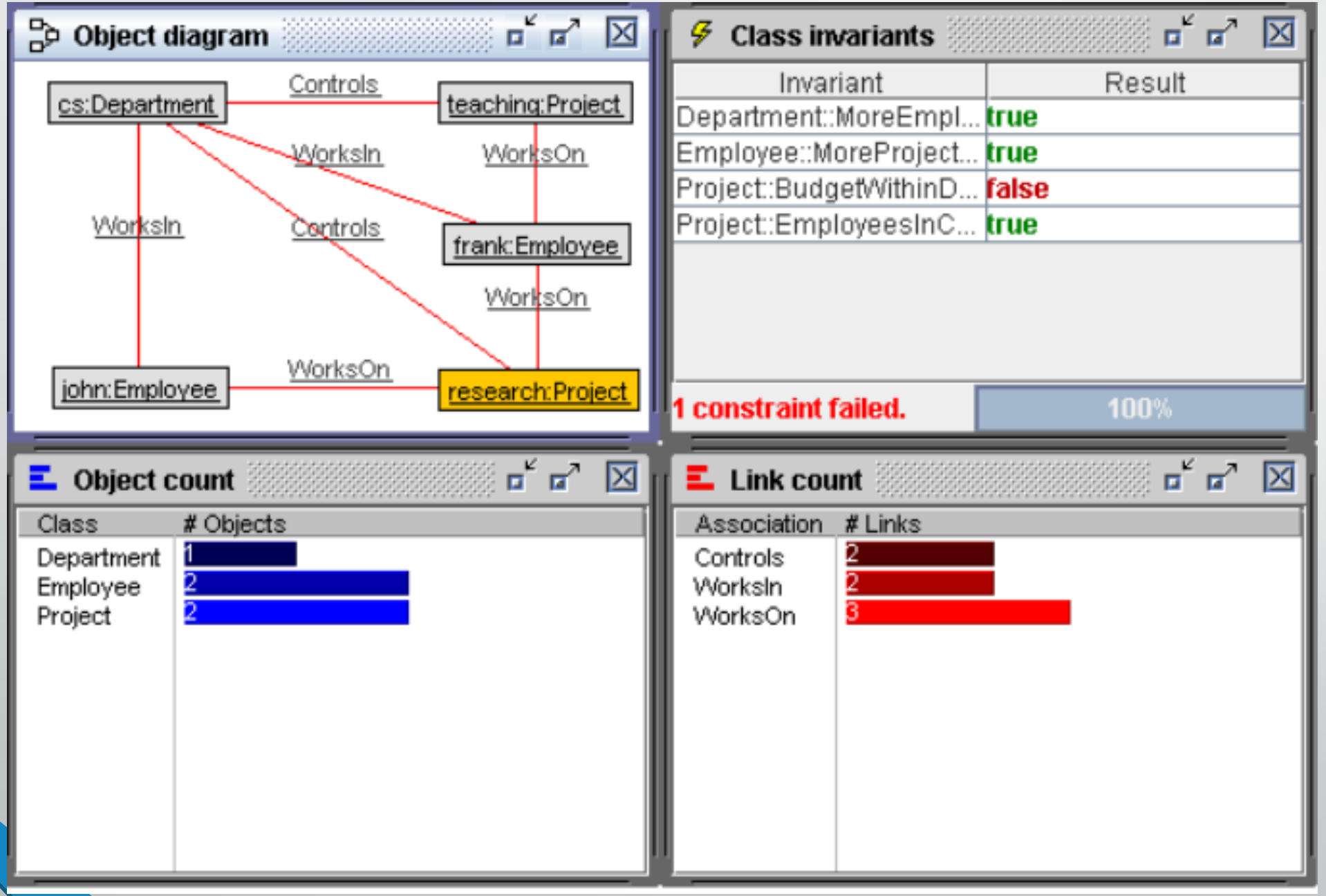
- The problem here is that we have specified in the model that each employee has to be related to at least one department object.
- In our current state, no employee has a link to a department.
- In order to fix this, we insert the missing links into the WorksIn association:

  use> !insert (john,cs) into WorksIn

  use> !insert (frank,cs) into WorksIn
- Links can also be inserted by selecting the objects to be connected in the object diagram and choosing the insert command from the context menu.
- The new state shows the links in the object diagram as red edges between the Employee objects and the Department object


Object diagram
frank:Employee
WorksIn
cs:Department
WorksIn
john:Employee
Class invariants
Invariant
Result
Department::MoreEmpl...
true
Employee::MoreProject...
true
Project::BudgetWithinD...
true
Project::EmployeesInC...
true
Constraints ok.
100%
Object count
Class
Objects
Department
1
Employee
2
Project
Link count
Association
Links
Controls
WorksIn
2
WorksOn

```
use> !create research : Project
use> !set research.name := 'Research'
use> !set research.budget := 12000
use>
use> !create teaching : Project
use> !set teaching.name := 'Validating UML'
use> !set teaching.budget := 3000
use>
use> !insert (cs,research) into Controls
use> !insert (cs,teaching) into Controls
use>
use> !insert (frank,research) into WorksOn
use> !insert (frank,teaching) into WorksOn
use> !insert (john,research) into WorksOn
```


Object diagram
cs:Department
Controls
teaching:Project
WorksIn
WorksOn
WorksIn
Controls
frank:Employee
WorksOn
john:Employee
WorksOn
research:Project
Class invariants
Invariant
Result
Department::MoreEmpl...
true
Employee::MoreProject...
true
Project::BudgetWithinD...
false
Project::EmployeesInC...
true
1 constraint failed.
100%
Object count
Class
Objects
Department
1
Employee
2
Project
2
Link count
Association
Links
Controls
2
WorksIn
2
WorksOn
3

Validating the pre and post conditions

- Recall the Person and Company Example

```
model Employee

-- classes

class Person
attributes
  name : String
  age : Integer
  salary : Real
operations
  raiseSalary(rate : Real) : Real
end

class Company
attributes
  name : String
  location : String
operations
  hire(p : Person)
  fire(p : Person)
end
```

```
-- associations

association WorksFor between
  Person[*] role employee
  Company[0..1] role employer
end

 -- constraints

 constraints

 context Company::hire(p : Person)

   pre  hirePre1: p.isDefined()
   pre  hirePre2: employee->excludes(p)
   post hirePost: employee->includes(p)

 context Company::fire(p : Person)
   pre  firePre:  employee->includes(p)
   post firePost: employee->excludes(p)
```

Creating objects for the model

- use> !create ibm : Company
- use> !create joe : Person
- use> !set joe.name := 'Joe'
- use> !set joe.age := 23

Calling Operations and Checking Preconditions

- We invoke the operation hire on the receiver object ibm and pass the object joe as parameter.
- use> !openter ibm hire(joe)
- precondition ‘hirePre1’ is true precondition ‘hirePre2’ is true

Operation Effects

- We can simulate the execution of an operation with the usual USE primitives for changing a system state.
- The postcondition of the hire operation requires that a WorksFor link between the person and the company has to be created. We also set the salary of the new employee.
- use> !insert (p, ibm) into WorksFor
- use> !set p.salary := 2000

Exiting Operations and Checking Postconditions

- After generating all side effects of an operation, we are ready to exit the operation and check its postconditions.
- The command opexit simulates a return from the currently active operation.
- The syntax is: !opexit ReturnValExpr
  - The optional ReturnValExpr is only required for operations with a result value.
- The operation hire specifies no result, so we can just issue:
- use> !opexit
- postcondition ‘hirePost’ is true

e - Example

```
context Person::raiseSalary(rate : Real) : Real
  post raiseSalaryPost:
    salary = salary@pre * (1.0 + rate)
  post resultPost:
    result = salary
```

- We call raiseSalary on the new employee joe. The rate 0.1 is given to raise the salary by 10%.
- use> !openter joe raiseSalary(0.1)
- The salary attribute is assigned a new value with the set command.
- use> !set self.salary := self.salary + self.salary * rate
- Since raiseSalary is an operation with a return value, we have to specify a result value on exit. This value is bound to the OCL result variable when the postconditions are evaluated.

use> !opexit 2200 postcondition ‘raiseSalaryPost’ is true postcondition ‘resultPost’ is true

Demonstration

C:\Users\Mariam\Desktop\18SW FMSE\OCL\Employee.use - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Employee.use

```
12      raiseSalary(rate : Real) : Real
13  end
14
15  class Company
16
17  attributes
18      name : String
19      location : String
20
21  operations
22      hire(p : Person)
23      fire(p : Person)
24  end
25
26  -- associations
27
28  association WorksFor between
29      Person[*] role employee
30      Company[0..1] role employer
31  end
32
33
34  -- constraints
35  constraints
36
37  context Person
38      inv: age > 18
39      inv: salary > 1000
40
41  context Company::hire(p : Person)
42
43      pre hirePre1: p.isDefined()
44      pre hirePre2: employee->excludes(p)
45
46      post hirePost: employee->includes(p)
```


Line 13, Column 4 Tab Size: 4 Plain Text

Type here to search 26°C Sunny 6:31 PM 2/22/2022

THANK YOU!