Tutorial: OCL

Purpose

Let you practise reading and writing OCL constraints.

Here are a couple more useful OCL operations on collections that were not explained in the reading or slides. (There are more: if you want full details, see section 11.7 of the OCL spec.)

Suppose `c` is a Collection of elements of type T, and `t : T`. Then we can write:

- `c->includes(t)`

  a Boolean expression that will be true iff the element `t` is equal to an element of the collection (exercise: write this in terms of `exists` instead: yet another example of the non-parsimony of the UML/OCL language!)

- `c->including(t)`

  an expression that evaluates to a collection which is the same as `c` except that `c` has been added to the collection. (If `c` is a sequence, `t` is added as the last element of the new collection; if it is a bag or a set, the obvious thing happens.)

These questions refer to the following diagram extracted from the OCL specification.

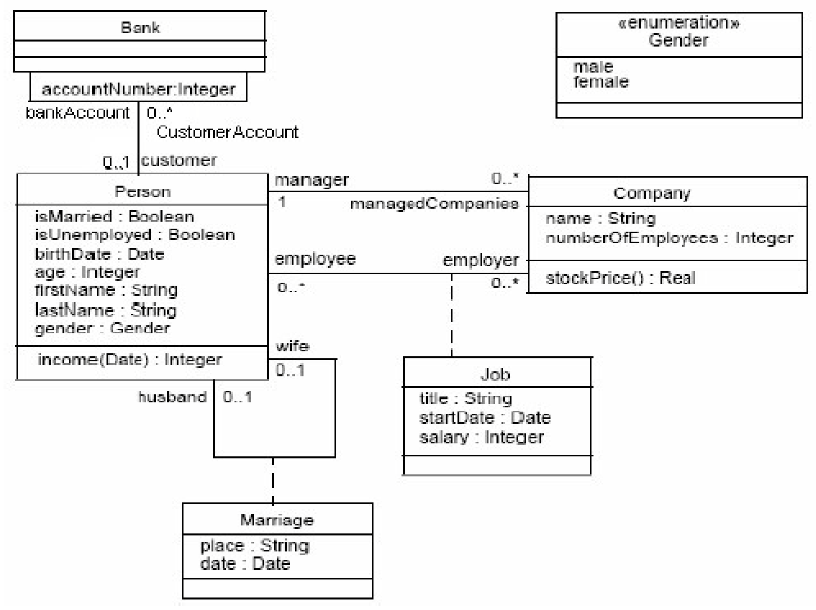

Figure 7.1 - Class Diagram Example

1 Question 1

Translate into English:

1. In the context of a Person:

   ```
   isMarried implies age > 15
   ```

2.
   ```
   context Company inv:
   numberOfEmployees = employee->size()
   ```

3.
   ```
   context Person::income(d:Date) : Integer
   pre: d.laterThan(self.birthDate)
   post: if age < 18
           then result < 100
           else result < 200
         endif
   ```

4. In the context of `bigBank : Bank`:

   ```
   bigBank.customer -> collect(p : Person | p.managedcompanies)
   -> asSet() -> size() >= 3
   ```

   What is the difference between this and

   ```
   bigBank.customer -> collect(p : Person | p.managedcompanies)
   -> size() >= 3
   ```

   ?

2 Question 2

Translate into OCL:

1. The length of a person's first name is always less than 20 characters, and so is the length of their last name.
2. Anyone who manages a company is an employee of that company. (You could write this in context Person – making it an invariant of Person – or in context Company – making it an invariant of Company. Try it both ways.)
3. Every company has a male employee.
4. It is a class invariant of Person that nobody can have more than 5 bank accounts.
5. Nobody can have two employments with companies that have identical names.

3 Extension exercises

for if you have time now, or for revision later.

Look at the UML Superstructure document, 11-08-06. You will see that most of it is organised by giving a class diagram (in which the classes are actually metaclasses, i.e. they represent concepts in the domain "modelling software systems", e.g., Association, Property, Class, Generalization etc.), and then, for each (meta)class, giving explanations of its meaning, attributes, associations etc. - and usually, some contraints in OCL. For example, you'll find the constraints for class Association on p53 of the PDF (p37 in the document's own numbering) and to interpret them you'll need to look at the diagram containing (meta)class Association which is Fig.7.12 on p45 of the PDF (p29 in the document's numbering).

Until you are bored/out of time:

1. Pick an OCL constraint from the document, find the diagram that gives its context, and check that you understand exactly what the OCL means and why it means it. Can you understand why this constraint is placed on this metaclass? Does the English version of the constraint capture precisely what the constraint means, or is it ambiguous?

2. Pick a section of the document that contains OCL constraints, preferably relating to one of the UML concepts that you know something about (e.g. Pseudostate on document page 585, but there are lots of other possibilities). Print out the relevant page(s) from the document but *do not look at the OCL yet.* Cover up the constraints section and gradually reveal so that you see the English explanation of a constraint but not the OCL. Consulting the relevant diagram which you'll typically find a few pages back in the document, try to write an OCL constraint that means what the English says. Compare what you write with what's in the UML spec.

Do not be surprised if in this process you find either

- a mistake in the OCL which is in the UML spec – most of it was written without tool support, and a few years ago there was a paper in the MODELS conference that systematically looked for mistakes using a recently developed OCL tool, and found many;
- a place where I have (very likely) severely simplified the version of UML that I taught in this course, or (less likely I hope) told you something that isn't strictly true according to the spec.

In either case, if you want my comments on what you've found, I'll be happy to give them: post on the Forum or email me.