Model Engineering

Final Exam WS13

20.01.2014

Group B

1)

a) Describe graph transformations (in general, the steps, how it’s working) and give a small example.

b) 7 statements which need to be checked true or false

e.g.

- ATL supports multiple inheritance
- UML profiles are a hardweight-extension
- XPand is a template-based model-to-model transformation
- there can only be one NAC with a graph transformation

2)

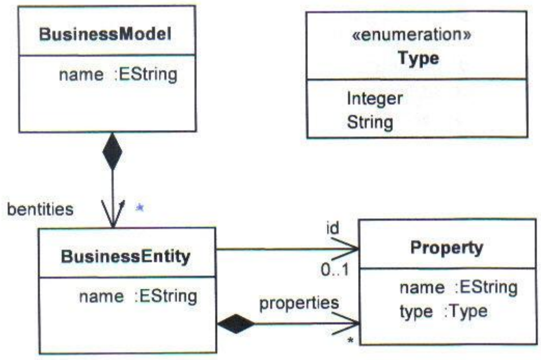
A Xtend file was given and some missing parts/lines had to be filled out. Metamodel and a model instance where given in the appendix as well as the targeted output code. In Xtend-file the doGenerate method was overwritten, a new output-file was created and the textual output was generated via templates (within this code some lines/words where missing).

3)

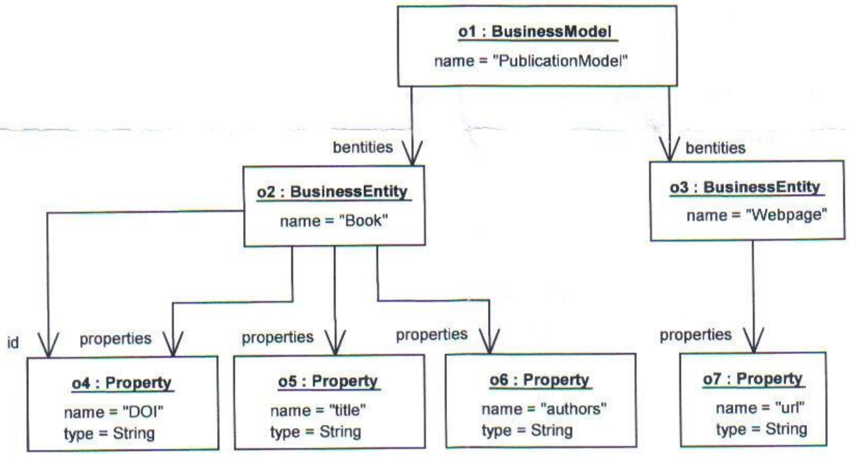
A metamodel as well as a model instance where given. The output model of the ATL transformation (see appendix) had to be drawn (input and output model have the same metamodel).

Appendix Task 2:

Metamodel "BusinessModel.ecore"



Model "BusinessModel.bmodel"



Target Code (filename "PublicationModel.db")

```
CREATE TABLE Book (
        DOI TEXT PRIMARY KEY,
        title TEXT,
        authors TEXT
)
CREATE TABLE Webpage (
        url TEXT,
        ID INTEGER PRIMARY KEY
)
```

Appendix Task 3: ATL Code

```
module trafo;
create OUT : classMM from IN : classMM;

helper context classMM!Class def : getAllSuperclassAttributes() :
                                   OrderedSet(classMM!Attribute) =
     if(not self.superclass.oclIsUndefined()) then
        OrderedSet{}->union(self.superclass.attributes)
                     ->union(self.superclass.getAllSuperclassAttributes())
     else    OrderedSet{}
     endif;

abstract rule NamedElement2NamedElement {
  from
     src: classMM!NamedElement
  to
     target: classMM!NamedElement (
        name <- src.name
  )
}
rule Package2Package extends NamedElement2NamedElement{
  from
     src: classMM!Package
  to
     target: classMM!Package (
        classes <- src.classes
  )
}
rule Class2Class extends NamedElement2NamedElement{
  from
     src: classMM!Class
  to
     target: classMM!Class (
        attributes <- src.attributes
  )
  do {
     for(attr in src.getAllSuperclassAttributes()) {
        if(attr.modifier <> #private) {
             target.attributes <- thisModule.NewAttribute(attr.name);
        }
     }
  }
}
rule Attribute2Attribute extends NamedElement2NamedElement{
  from
     src: classMM!Attribute
  to
     target: classMM!Attribute (
        modifier <- #private
  )
}
rule NewAttribute (name: String){
  to
     target: classMM!Attribute (
        name <- name,
        modifier <- #private
  )
  do {
     target;
  }
}
```