

1. ATL Transformation Example

1.1. Example: UML → Java

The UML to Java example describes the transformation of a UML model to a simplified Java model. The Java model holds the information for the creation of Java classes, especially what concerns the structure of these classes, namely the package reference, the attributes and the methods.

1.1.1. Metamodels

A possible target metamodel of Java (see Figure 1 Simplified Java) consists principally of JavaElements which all have a name. A JavaClass has Methods and Fields and belongs to a Package. Methods, Fields and JavaClasses are subclasses of Modifiers and therefore indicate whether they are public, static or final. JavaClasses and Methods declare with the isAbstract attribute whether they are abstract or not. PrimitiveTypes and JavaClasses are Types. A Method has a Type as return Type and parameters of certain Types. A Field has also a Type.

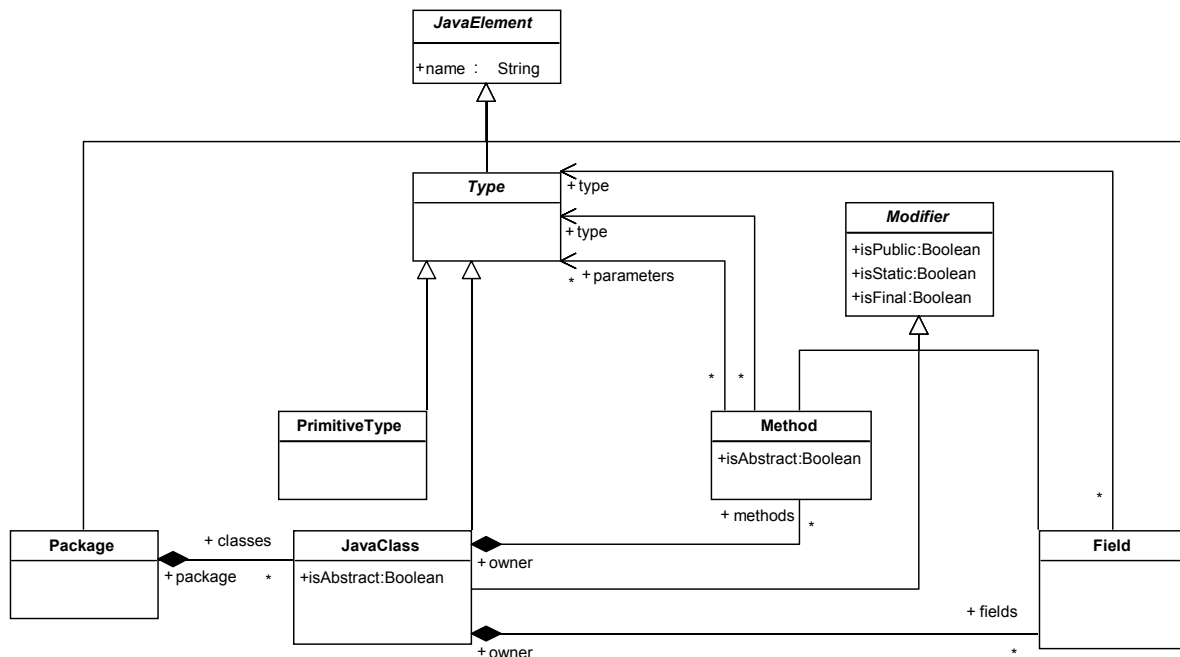
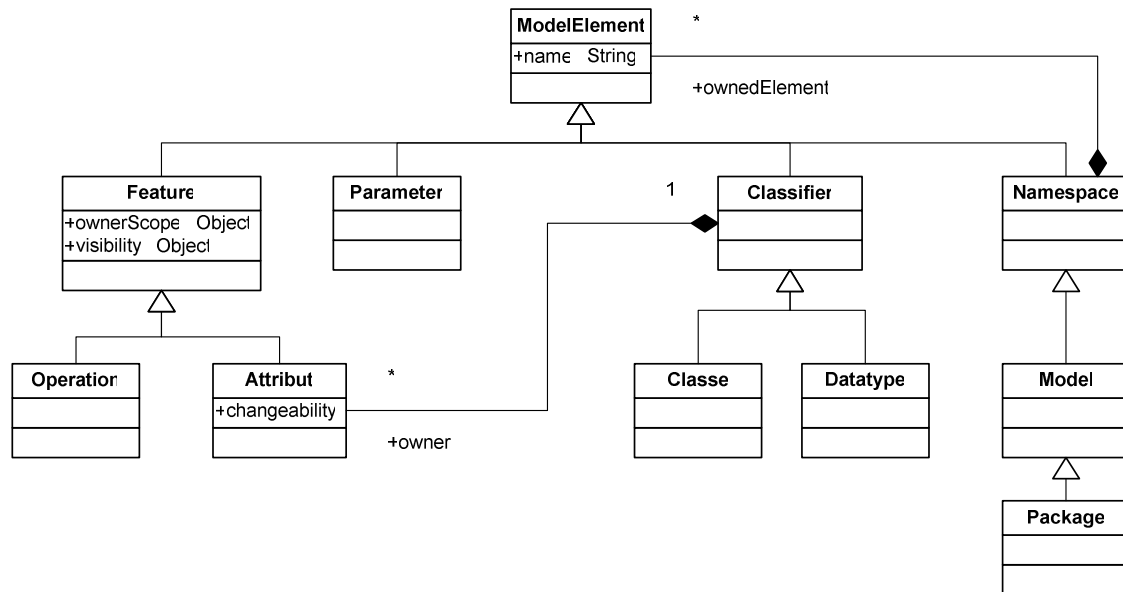


Figure 1 Simplified Java Metamodel

The source metamodel of UML is well described by the OMG [1]. It has been simplified for the purpose of this transformation example:


Figure 2 Simplified UML Metamodel

1.1.2. Rules Specification

These are the rules to transform a UML model to a Java model:

- For each UML Package instance, a Java Package instance has to be created.
 - Their names have to correspond. However, in contrast to UML Packages which hold simple names, the Java Package name contains the full path information. The path separation is a point “.”.
- For each UML Class instance, a JavaClass instance has to be created.
 - Their names have to correspond.
 - The Package reference has to correspond.
 - The Modifiers have to correspond.
- For each UML DataType instance, a Java PrimitiveType instance has to be created.
 - Their names have to correspond.
 - The Package reference has to correspond.
- For each UML Attribute instance, a Java Field instance has to be created.
 - Their names have to correspond.
 - Their Types have to correspond.
 - The Modifiers have to correspond.
 - The Classes have to correspond.
- For each UML Operation instance, a Java Method instance has to be created.

- Their names have to correspond.
- Their Types have to correspond.
- The Modifiers have to correspond.
- The Classes have to correspond.

1.1.3. ATL Code

This ATL code for the transformation of a UML to Java consists of several functions and rules. Among the functions, it is important to mention `getExtendedName` which recursively explores the namespace to concatenate a full path name. Concerning the rules, there are remarks necessary respective the rule O2M. This rule shows how to access sets via OCL expressions. For simplicity of implementation, the (return) type of a Java Method is the first parameter of an UML Operation. Some minor details, such as modifiers, are not yet fully implemented.

```
module UML2JAVA;
create OUT : JAVA from IN : UML;

helper context UML!ModelElement def: isPublic() : Boolean =
    self.visibility = #vk_public;

helper context UML!Feature def: isStatic() : Boolean =
    self.ownerScope = #sk_static;

helper context UML!Attribute def: isFinal() : Boolean =
    self.changeability = #ck_frozen;

helper context UML!Namespace def: getExtendedName() : String =
    if self.namespace.oclIsUndefined() then
        ''
    else if self.namespace.oclIsKindOf(UML!Model) then
        ''
    else
        self.namespace.getExtendedName() + '.'
    endif endif + self.name;

rule P2P {
    from e : UML!Package (e.oclIsTypeOf(UML!Package))
    to out : JAVA!Package (
        name <- e.getExtendedName()
    )
}

rule C2C {
    from e : UML!Class
    to out : JAVA!JavaClass (
        name <- e.name,
        isAbstract <- e.isAbstract,
        isPublic <- e.isPublic(),
        package <- e.namespace
    )
}


rule D2P {
```

```
    from e : UML!DataType
    to out : JAVA!PrimitiveType (
        name <- e.name,
        package <- e.namespace
    )
}

rule A2F {
    from e : UML!Attribute
    to out : JAVA!Field (
        name <- e.name,
        isStatic <- e.isStatic(),
        isPublic <- e.isPublic(),
        isFinal <- e.isFinal(),
        owner <- e.owner,
        type <- e.type
    )
}

rule O2M {
    from e : UML!Operation
    to out : JAVA!Method (
        name <- e.name,
        isStatic <- e.isStatic(),
        isPublic <- e.isPublic(),
        owner <- e.owner,
        type <- e.parameter->select(x|x.kind=#pdk_return)->
            asSequence()->first().type,
        parameters <- e.parameter->select(x|x.kind<>#pdk_return)->
            asSequence()
    )
}

rule P2F {
    from e : UML!Parameter (e.kind <> #pdk_return)
    to out : JAVA!FeatureParameter (
        name <- e.name,
        type <- e.type
    )
}
```

	ATL TRANSFORMATION EXAMPLE	
	UML to Java	Date 18/03/2005

References

- [1] OMG Unified Modeling Language (UML), version 1.4 (formal/03-03-01), 2002,
<http://www.omg.org/technology/documents/formal/uml.htm>