Organization of custom software generators in the Xtext framework

Antonio Natali

Alma Mater Studiorum – University of Bologna
via Venezia 52, 47023 Cesena, Italy
{antonio.natali}@unibo.it

Abstract. Introduction to the organization of custom generator software in the Xtext framework

Keywords: Software factories, (meta)models, domain models, `ECore`, `Xtext`.

1 Introduction

In this work we make reference to the custom language `mydsl` taken from *Xtext 2.1 Documentation* [1] pg.14 (*15 Minutes Tutorial*). It allows us to model entities and properties; for example:

The mydsl meta-model

```
package m1 {
import java.util.*
datatype String
datatype Date
datatype BigInteger

 entity Blog {
        date : Date
        size : BigInteger
        title: String
        posts: Post *
 }
 entity HasAuthor {
        author: String
        age : BigInteger
 }
 entity Post extends HasAuthor {
        title: String
        content: String
        comments: Comment *
 }
 entity Comment extends HasAuthor {
        content: String
 }
}
```

The "code" above is a model, instance of the meta-model defined by the following grammar:

The mydsl meta-model (file Mydsl.xtxt)

```
grammar it.unibo.Mydsl with org.eclipse.xtext.common.Terminals
generate mydsl "http://www.unibo.it/Mydsl"

DomainModel:
        (elements+=AbstractElement)*;
AbstractElement:
        PackageDeclaration | Type | Import;
Import:
        'import' importedNamespace=QualifiedNameWithWildCard;
PackageDeclaration:
        'package' name=QualifiedName '{'
```

```
        (elements+=AbstractElement)*
        '}';
Type:
        Entity | DataType;
DataType:
        'datatype' name=ID;
Entity:
        'entity' name=ID ('extends' superType=[Entity])? '{'
        (features+=Feature)*
        '}';
Feature:
        name=ID ':' type=[Type] (multi?='*')?;
QualifiedName:
        ID ('.' ID)*;
QualifiedNameWithWildCard:
        QualifiedName '.*'?;
```

2 The generation process

In the following we will assume that the `mydsl` meta-model is defined in a `Xtext` project named *it.unibo.mydsl*. From the meta-model specification, we must generate the `Xtext` artefacts, that will include a file named `MydslJavaValidator.java` (in the package *it.unibo.validation*) and a file named `MydslGenerator.xtend` (in the package *it.unibo.generator*).

2.1 Model validation rules

The file `MydslJavaValidator.java` allows an user to define model-checking rules. For example, we could state that a user-model is correct only if:

1. the name of each entity starts with a capital letter;
2. the model defines a package.

MydslJavaValidator.java

```
package it.unibo.validation;
import it.unibo.mydsl.DomainModel;
import it.unibo.mydsl.Entity;
import it.unibo.mydsl.MydslPackage;
import it.unibo.mydsl.PackageDeclaration;
import org.eclipse.xtext.validation.Check;

public class MydslJavaValidator extends AbstractMydslJavaValidator {

 @Check
 public void startsWithCapital(Entity e) {
        if (!Character.isUpperCase(e.getName().charAt(0))) {
                error("Entity name should start with a capital", MydslPackage.Literals.TYPE__NAME );
        }
 }
 @Check
 public void hasPackage(DomainModel dm) {
        if ( ! ( dm.getElements().get(0) instanceof PackageDeclaration )  ) {
                error("A package is mandatory", MydslPackage.Literals.DOMAIN_MODEL__ELEMENTS );
        }
 }
}
```

2.2 The generator entry point

The file `MydslGenerator.xtend` constitutes the main entry for the user-defined generation process related to the custom language/metamodel. It takes the following form:

MydslGenerator.xtend

```
/*
 * generated by Xtext
 */
package it.unibo.generator
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess

class MydslGenerator implements IGenerator {
 override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    //TODO implement me
 }
}
```

For the sake of clearness and modularity, we use this class just as an entry point for our custom generation processes, that will be defined in another package (e.g. *it.unibo.mygenerator*).

2.3 The custom generator entry point

In the package named *it.unibo.mygenerator* we define our main generation class (named here `MydslMyGen`). The `doGenerate` method of `ExpGenerator` is modified as follows:

doGenerate: new version

```
/*
 * generated by Xtext and modified by AN
 */
package it.unibo.generator
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import com.google.inject.Inject
import it.unibo.mygenerator.GenUtils
import it.unibo.mygenerator.MydslMyGen

class MydslGenerator implements IGenerator {
@Inject GenUtils util
        override void doGenerate(Resource resource, IFileSystemAccess fsa) {
                //Implemented by AN
                println("MydslGenerator starts resource=" + resource)
                println("MydslGenerator starts fsa=" + fsa)
            util.setFsa(fsa)
            new MydslMyGen().main(resource,util)
        }
}
```

2.4 The `GenUtils` (custom) class

The `GenUtils` class is introduced to provide methods useful in the generation process. At the moment it can be defined as follows:

GenUtils (written in Xtend2)

```
package it.unibo.mygenerator
import org.eclipse.xtext.generator.IFileSystemAccess
```

```
public class GenUtils {
IFileSystemAccess curFsa
 def setFsa( IFileSystemAccess fsa ){
    curFsa = fsa
 }
 /*
 * Generate a file by using IFileSystemAccess
 */
  def genFile( String name, String suffix, CharSequence contents){
        var sysName = name.replace(".", "/")
        var fName = sysName + "." + suffix
            curFsa.generateFile( fName,contents )
 }
}
```

Other utilities can be defined also as conventional `Java` classes. For example:

GenUtilJava (written in Java)

```
package it.unibo.mygenerator;
import it.unibo.mydsl.DomainModel;
import it.unibo.mydsl.Entity;

public class GenUtilJava{
    public String cvtToString( DomainModel e ){ return ""+e; }
    public String getName( Entity e ){ return e.getName(); }

}
```

2.5 A user-defined generator

The task of `MydslMyGen` class is to run a set other user-defined generation tasks, each organized into one or more classes. For example

MydslMyGen.xtend

```
/*
 * User defined generator entry
 */
package it.unibo.mygenerator
import static extension org.eclipse.xtext.xtend2.lib.ResourceExtensions.*
import org.eclipse.emf.ecore.resource.Resource

class MydslMyGen{
 def main(Resource resource, GenUtils util){
        println("MydslMyGen starts")
        new DomainModelFirstGenerator().genJavaCode(resource,util)
 }
}
```

The *MydslMyGen* generator creates an instance of a generator (named `DomainModelFirstGenerator`) in order to translate a user-defined model from the `mydsl` language to `Java`. This further generator is just an example of a typical generation pattern, that defines the content of a new file created in the `src-gen` directory:

DomainModelFirstGenerator.xtend

```
/*
 * A first user-defined generator
 */
package it.unibo.mygenerator
import org.eclipse.emf.ecore.resource.Resource
import it.unibo.mydsl.AbstractElement
import it.unibo.mydsl.PackageDeclaration
import it.unibo.mydsl.Import
import it.unibo.mydsl.Type
import it.unibo.mydsl.DataType
import it.unibo.mydsl.Entity
import it.unibo.mydsl.Feature

class DomainModelFirstGenerator{
 String importStr=""
 GenUtils util
 GenUtilJava javaUtil

 def genJavaCode(Resource resource, GenUtils util){
 this.util = util
 javaUtil = new GenUtilJava()
        println("DomainModelJavaGenerator genJavaCode")
        for( e: resource.allContents.filter( typeof(AbstractElement) ).toIterable  ){
              genCode(e)
    }
 }
}
```

genCode (generation for packages).

genCode

```
def dispatch genCode(AbstractElement pack)'''//not here AbstractElement'''
def dispatch void genCode(PackageDeclaration pack){
var String outS = ""
  outS =  outS + "package " + pack.name +";" + "\n"
  for( elem: pack.elements ){
        outS =  outS + genCode( pack.name,  elem)
  }
}
```

genCode (generation for entity).

genCode

```
def dispatch void genCode( String packname, AbstractElement e  ){}
def dispatch genCode(String packname, Import el){
 importStr = importStr + ''' import «el.importedNamespace» ;
'''
}
def dispatch void genCode( String packname, Entity e  ){
        util.genFile( packname+"."+javaUtil.getName(e) , "java",  genBody(packname,e) )
}
```

```
def  genBody( String packname,Entity e  )'''
 package «packname» ;
 «importStr»
 public class «e.name» «IF(e.superType != null)» extends «e.superType.name» «ENDIF»{
  «genFeatures(e)»
 }
'''
```

genFeatures.

genFeatures

```
def genFeatures( Entity e ){
 var outS = ""
        for( Feature f : e.features ){
                outS = outS +  genFeature(f)
        }
        outS
}

def genFeature(Feature f){
if( f.multi )
'''protected java.util.List< «f.type.name»> «f.name»;//FEATURE (MANY)
'''
else
'''
protected «f.type.name» «f.name»; //FEATURE

public «f.type.name» get«f.name.toFirstUpper»() {
        return «f.name»;
}
public void set«f.name.toFirstUpper»(«f.type.name» «f.name») {
        this.«f.name» = «f.name»;
}
'''
}
```

2.6 Running the generation

To execute the user-defined generation process, it is sufficient to perform the following actions:

1. put in execution the *it.unibo.mydsl.ui* plugin;
2. create a conventional `Java` project;
3. write a sentence of the custom language, by creating a new file in the `src` directory with `mydsl` suffix.

   For example, let us write a model in the file `src/um0.mydsl`:

src/um0.mydsl user-model

```
package it.unibo.is.interfaces {
        datatype IBasicEnvAwt
}

package m0{
import it.unibo.is.interfaces.IBasicEnvAwt
datatype String
```

```
datatype Boolean

 entity Session {
        title: String
        isTutorial : Boolean
}
entity Conference {
        frame :  IBasicEnvAwt
        name : String
        attendees : Person*
        speakers : Speaker*
}
entity Person {
        name : Boolean
}
entity Speaker extends Person {
        sessions : Session*
}
}
```

As soon as the file is saved, the `Xtext` framework activates the user-defined code-generator; the result is that `Java` classes are created in the `src-gen` directory. For exampe:

The generated m0/Conference.java file

```
package m0 ;
import it.unibo.is.interfaces.IBasicEnvAwt ;
public class Conference {
 protected IBasicEnvAwt frame; //FEATURE

 public IBasicEnvAwt getFrame() {
                return frame;
 }
 public void setFrame(IBasicEnvAwt frame) {
                this.frame = frame;
 }
 protected String name; //FEATURE

 public String getName() {
                return name;
 }
 public void setName(String name) {
                this.name = name;
 }
 protected java.util.List<Person>  attendees;//FEATURE (MANY)
 protected java.util.List<Speaker>  speakers;//FEATURE (MANY)
}
```

References

1. Xtext. Xtext 2.1 documentation.
   http://www.eclipse.org/Xtext/.