Scenario tutorial day two: Converting grammar first projects to Metamodel first projects

Table of Contents

In part tutorial_scenario_part1.pdf, the tutorial started from the grammar (xtext) and then produced an ecore metamodel. While this approach is correct and rather intuitive for an introduction to MDE, it may lead to poorly organized metamodels. For example, it is difficult to create abstract super classes using Xtext.

This is why is is often better to start by writing the ecore metamodel and then connect the Xtext grammar on it.

If you haven't wrote the ecore first, you can convert your projects in order to do so.

This document describes how to achieve that. It starts from the solution projects of the part1 and will produce the base projects for the part2.

As this is only code refactoring, this part is not really important for understanding the crash course content.

Most reader should jump directly to the next tutorial (tutorial_scenario_part2.pdf) and use this one only as a reference if she has to do this task someday on another language.

This document indicates the steps to follow in order to reproduce the tutorial.

The result of this tutorial is available in the folder part2-mmfirst-base in githug repo or as a download in the following zip.

1. Installation

If you haven't followed the previous tutorial tutorial_scenario_part1.pdf:

- Install a Java JDK (minimum 17)
- grab and unzip the latest release version of GEMOC Studio https://gemoc.org/download.html

If you have followed the previous tutorial tutorial_scenario_part1.pdf:

- Either use a brand new workspace or clean it by deleting the projects in it since the project we will create will have the same names.

Then

- download the file **https://dvojtise.github.io/mde-crashcourse-logo/zips/part1-grammarfirst-solution.zip**
- in Eclipse,
  - *File → Import... → General → Existing projects into Workspace → Next*
  - *Select archive file → Browse* and select the file *part2-grammarfirst-solution.zip* you've downloaded
  - Finish

2. Convert Grammar first to metamodel first

- *File → New → Ecore Modeling Project*
  - name: `fr.inria.sed.logo.model`
  - nsuri: `http://www.inria.fr/sed/logo`
  - package: `Logo`

replace the *fr.inria.sed.logo.model/model/logo.ecore* by the one in *fr.inria.sed.logo.xtext/model/generated/Logo.ecore*

Optionnaly, open the ecore file, on the root package change *http://www.inria.fr/sed/logo/xtext/Logo* to http://www.inria.fr/sed/logo/Logo_

right click on Logo.genmodel → Reload...

- open the genmodel file
- on the second node (Package "Logo"), in the "All" section, change the base package from *fr.inria.sed.logo.model* to *fr.inria.sed*
- right click on the root node → generate Model code ;
- in *fr.inria.sed.logo.xtext/src/fr/inria/sed/logo/xtext/Logo.xtext* replace

```
generate logo "http://www.inria.fr/sed/logo/xtext/Logo"
```

by

```
// import "http://www.inria.fr/sed/logo/Logo"
import "platform:/resource/fr.inria.sed.logo.model/model/Logo.ecore"
```

in xtext project

in the mwe2 file, add:

```
language = StandardLanguage {
            ...
            referencedResource =
"platform:/resource/fr.inria.sed.logo.model/model/Logo.genmodel"
```

add a dependency to the fr.inria.sed.logo.model project (by opening either plugin.xml or manifest.mf, then the Dependencies tab) Also make sure to reexport the dependency.

run the mwe2 generator

fix any import such as *import fr.inria.sed.logo.xtext.logo.LogoProgram* into *import fr.inria.sed.logo.LogoProgram*

Do some refactoring in the metamodel inorder to enhance it:

- add an abstract concept PrimitiveInstruction as super class for PenUp, Foward, clear, etc
- add an abstract concept controlStructureInstruction as super class for If, Repeat, while, etc