Tool Guide for DSL Design Course (Modelware part)

Introduction

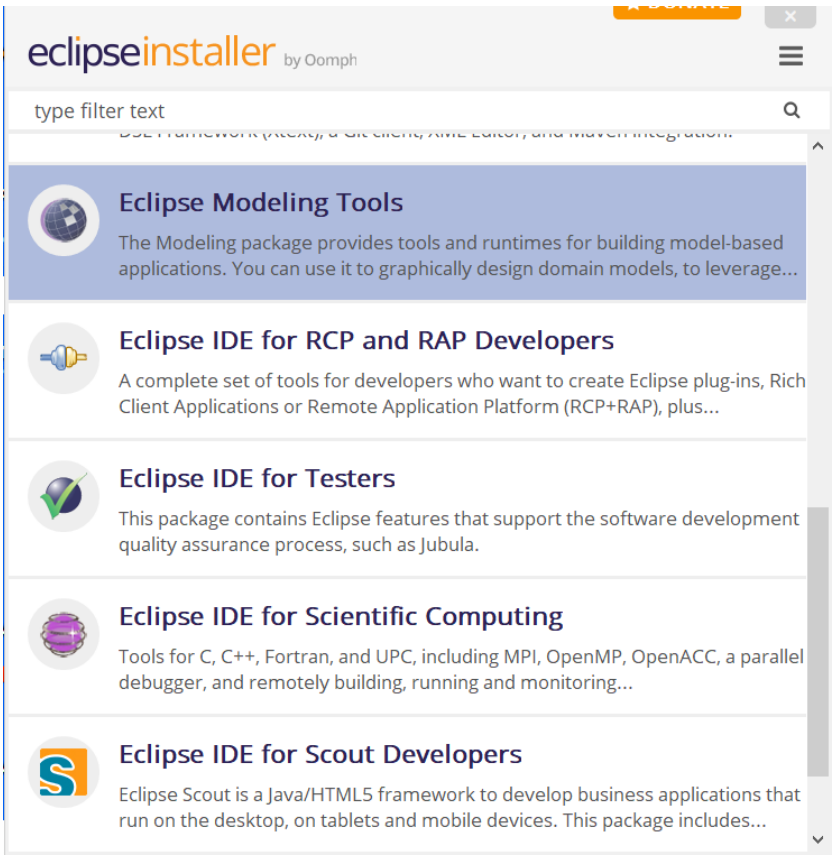
This document is a detailed guide for installing and working with the tools used in the Modelware part of the DSL Design course. The goal is to provide a single document that explains the installation steps and the most commonly performed tasks. For further information the documentation of the technologies should be used.

Eclipse Installation

The majority of the tools work within the Eclipse IDE (www.eclipse.org). Therefore, the first step is to install a proper version of Eclipse. This document will use Eclipse 2022-03 for Windows, 64 bits.

Eclipse 2022-03 Installation

- Navigate to https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2022-03/R/eclipse-inst-jre-win64.exe and select the button "Download". This will initiate the download of the Eclipse installer (an executable of around 100 MB). Save it in some folder. Be aware that this version of Eclipse includes Java Runtime Environment (JRE) for Windows, Linux and MAC OS X. You can also use an existing JRE as long as it is version Java SE 11 or greater.
- Start the installer. In the first screen, scroll down and select "Eclipse Modeling Tools":



In the next screen, give the desired installation folder and press Install. The installation is by default located in the subfolder 'eclipse' placed in the installation folder given by you. Follow the wizard by accepting the licenses. Once the installation is complete, choose to launch Eclipse.

- When an Eclipse installation starts for the first time, it asks for a workspace folder. This is the folder where the projects will be stored. Give a folder that you prefer (it can be changed later, switching between workspaces is also possible).
- If all previous steps are successful, Eclipse starts and shows its welcome screen.

Importing an Existing Project in Eclipse

Most of the lectures will be accompanied by code artefacts, models, etc. They will be provided as archived Eclipse projects (in zip format) and should be first imported into your Eclipse workspace before inspecting and trying them.

In order to import a project from an archive file:

- Select menu *File/Import…* Expand the folder *General* and select *Existing Projects into Workspace.* Press *Next*.
- In the next screen, select the radio button *Select archive file*. Open the browse window and navigate to the archived project. Press *Finish*.

Eclipse Modeling Framework (EMF). Metamodeling

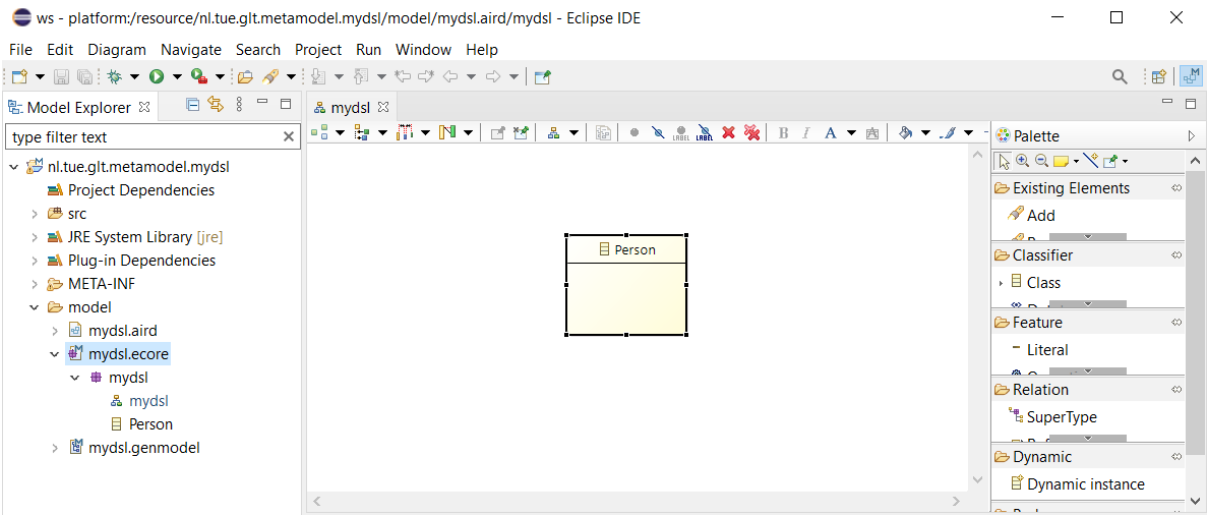
The majority of the tools used in the Modelware part of the course are built on top of EMF. EMF is included in the Eclipse version installed in the previous section (Eclipse Modeling Tools). This section explains how metamodels can be created with the language *Ecore*.

Creating a New Ecore Modeling Project

The following steps show how a new Ecore modeling project is created followed by the definition of the metamodel. The theory of metamodeling and the details of the Ecore language are given in the lectures.

- In a running Eclipse, select menu File/New/Project…. In the New Project wizard, expand folder *Eclipse Modeling Framework* and select *Ecore Modeling Project*
- Give a name of the project. Here we will use *nl.tue.glt.metamodel.mydsl* as an example. Press "Next"
- Give the name of the main package. It is Ok to accept the suggested default name. In the next screen, accept the default suggestions (sufficient for most tasks) and press "Finish"

- The new project is created and the visual editor for Ecore metamodels is shown. The metamodel will be stored in the file *mydsl.ecore* located in folder 'model'. The following shows a sample metamodel with a single class Person.



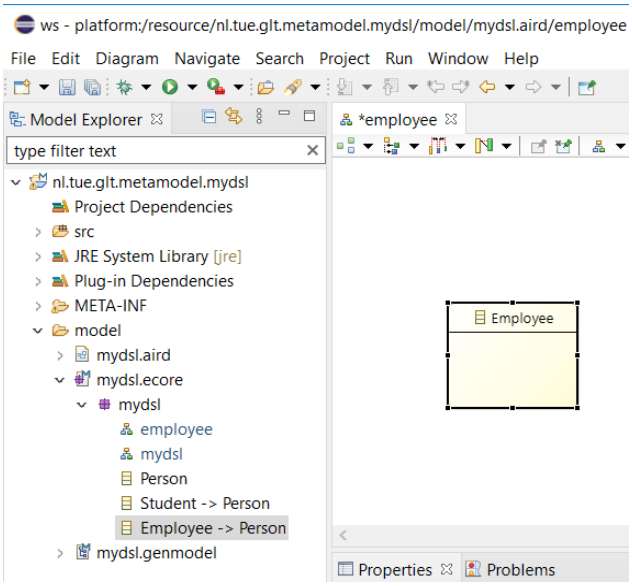
Working with Metamodels and Multiple Diagrams

A newly created Ecore project will contain one initial diagram shown in the Ecore visual editor. The presence of such diagram is indicated as an element in the package tree in the Model Explorer view:

> see the element surrounded with the oval in the previously shown image. If the diagram is closed, it can always be open by double click on the diagram element.

If a metamodel contains too many elements it may be impractical to show them on a single diagram. Multiple diagrams can be used that show different elements of the metamodel. The complete content of the metamodel is always available in the Model Explorer tree, the diagrams serve as views over this content.

In order to create a new diagram, do a right-button click over the package in the Model Explorer, select New Representation and then 'class diagram'. Give a name to the diagram. New elements can be created on the diagram and existing elements can be dragged and dropped from the Model Explorer tree to the canvas. The figure below shows a new diagram that contains only the class Employee. Observe that now there are two diagrams in the tree on the left-hand side:



Apart from using the diagrams and the Model Explorer tree, the content of the metamodel can be inspected and altered via a tree-based editor supplied in Eclipse. To do this, double-click on the ecore file shown in the Model Explorer. The tree-based editor will open.

> Consult the Eclipse Help, section *EcoreTools User Manual* for full details about working with the editors.

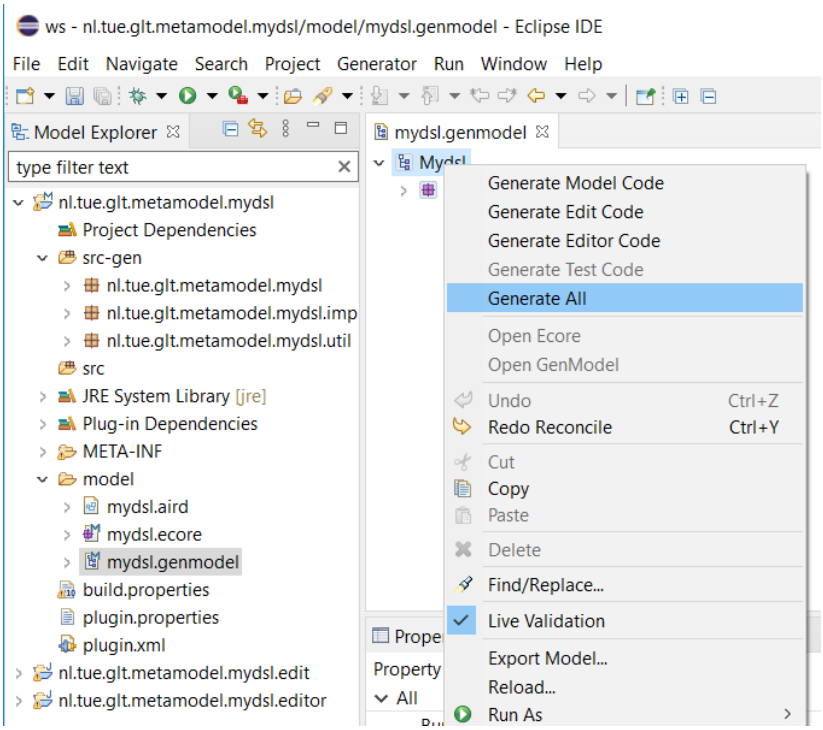
Generating Code from a Metamodel

A metamodel defines the abstract syntax of a language. Models expressed in this language conform to its metamodel. These models can be created, manipulated and stored in multiple ways. One of the main possibilities is by using a programming language to instantiate objects from the metamodel classes. In order to do this, first an implementation of the metamodel classes needs to be provided in some programming language. EMF supports automatic generation of Java classes from Ecore metamodels.

The process of generation of Java code from a metamodel is configured in the so called *generation model*. When a new Ecore modeling project is created, the generation model is automatically created along with the metamodel, stored in a file with the name of the ecore file and extension *genmodel*. For most of the tasks, the content of the file can be used without changes.

Java code generation steps:

- Open the *genmodel* by double clicking on it in the Model Explorer. The content is shown in a tree-like editor.
- Right-click on the root element and select *Generate All.*
- The Java implementation is placed in the *src-gen* folder. In addition, two new projects are created. They provide an implementation of a simple editor for creating models instances of the metamodel.



Important: If the metamodel is changed, the code needs to be regenerated.

If the generation of the default editor is not needed, then selecting option *Generate Model Code* is sufficient.

Creating Instance Models

We will show two ways for creating instances of the defined metamodel.

Creating a dynamic instance

- Select a class from the metamodel, usually one that plays the role of a container for model elements. You can use the Model Explorer view or the Ecore tree editor.
- Right click on the class will bring a menu that contains the option 'Create Dynamic Instance …'. Select the option.
- In the shown dialog window, give the location and the name of the file in which the instance will be stored. The default extension is *xmi*.
- Once the file is created, a tree-based editor will open. You can use it to create the content of the model.

Using the generated editor in an Eclipse Application

The generated code provides an editor for creating metamodel instances. It is rather simple but provides some mechanisms for customization (beyond the scope of this course). We will use the editor in an Eclipse application instance:

- Select menu Run/Run As/Eclipse Application
- When the new Eclipse instance is started, create a new project (a normal general project is sufficient)
- Select menu File/New/Other… In the wizard selection window, expand folder *Example EMF Model Creation Wizards*. Select "Mydsl Model" (recall that the root package was called *mydsl*).

- Provide a file name and a location (presumably in the newly created project), then select a class that will serve as root.
- The file will open in an editor.

Obviously, a third way to create an instance model is to write a Java program and use the generated model code (the API is explained in the lecture on EMF). The course covers other mechanisms for creating models in a DSL, for example, in a textual or visual editor.

> A good detailed tutorial on creating metamodels, model instances, code generation, and the Java API of EMF is available at https://www.vogella.com/tutorials/EclipseEMF/article.html

Other Tasks for Working with Ecore Metamodels

The following two tasks may be handy when working with Ecore metamodels. If you have created a metamodel following the procedure in the beginning of this section, then you may skip the next explanation.

Create a new metamodel

A new empty Ecore metamodel can also be created in an existing project. In order to do this:

- Select menu *File/New/Other…* Expand the folder *Eclipse Modeling Framework* and select *Ecore Model*
- In the next screen, give the folder where the model will be placed (typically a folder named *model* in one of the existing projects or just in the root of the project), and give the name of the ecore file. Hit *Next* and accept the defaults in the next screen. Press *Finish*
- The new ecore model will be opened in the tree editor. It is also possible to use the visual editor by creating a new diagram first, as explained earlier

Create a generation model

If a new Ecore model is created from scratch (like in the previous step), no genmodel is created by default. Recall that it is needed for generating Java model and editor code. In order to create a genmodel from an existing Ecore metamodel:

- Select menu *File/New/Other…* Expand the folder *Eclipse Modeling Framework* and select *EMF Generator Model*. Press *Next*.
- In the next screen, give the location and the name of the genmodel file. Press *Next*
- Select *Ecore Model* from the list of model importers. Press *Next.*
- In the next screen, navigate to the Ecore model (typically using the *Browse Workspace* functionality). Press *Load* and then *Next*
- Select the desired root packages (the examples in the course will typically have a single package). Press *Finish*.

Exercise

For each major topic we give a small exercise for self-practicing.

Create a Metamodel for Meeting Management

In this exercise you are asked to create a new EMF modeling project and a metamodel. This metamodel captures information about meetings:

- Each meeting has a date, time slot, and participants
- Participants are either individuals or whole teams (a group of individuals)

- Each meeting takes place in a room
- Rooms are identified by a unique number
- Individuals and teams have a name. In addition, the teams have individuals as members
- The instance models shall be capable of storing information about people, teams, rooms and meetings
- *Hint*: create a model container class that contains directly or indirectly all the information. For example, you may create class named MeetingManager

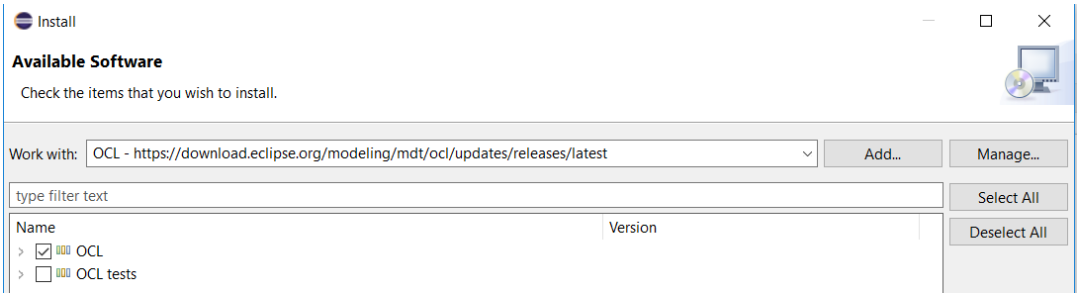
Once the metamodel is defined, generate model and editor code from it. Use dynamic instance or the generated editor to create an example instance model. Optionally, write a small Java program that loads the created instance model and prints out the names and the members of the teams. Use the example Java program shown in Lecture 1 as a template for the programming steps.

Object Constraints Language

Installing OCL in Eclipse via an Update Site

We will show how to install the latest version of the OCL tools via an Eclipse update site. This procedure is common to the Eclipse IDE and can be used for other tools as well. Here it will be explained once and used further in this document.

- Select menu *Help/Install New Software…*
- Press button *Add*. In the dialog box, fill the field *Name* (e.g. give OCL as a value). In the *Location* field give https://download.eclipse.org/modeling/mdt/ocl/updates/releases/latest as value. Press *Add*.
- In the next screen select the check box 'OCL':



- Complete the wizard by pressing *Next* and accepting the license agreements.

Creating OCL Constraints

In an Eclipse project, usually the one that contains the metamodel of a DSL, create a new text file with extension *ocl*. Open this file and write the desired OCL constraints. The syntax is explained in the lectures and an example OCL document is contained in a project available on Canvas (Files/code/XtextProjects.zip, project with the SLCO metamodel).

Validating Models

Once an OCL document is created, it can be applied on an instance model and this model can be validated.

- First make sure you have an instance model in XMI format
- Open this model by double clicking. Make sure that the file is not open as a text file, showing the XML representation. The file should be open in the Ecore editor (this is the default behavior for .xmi files). If this is not the case, perform right click on the XMI file, then select

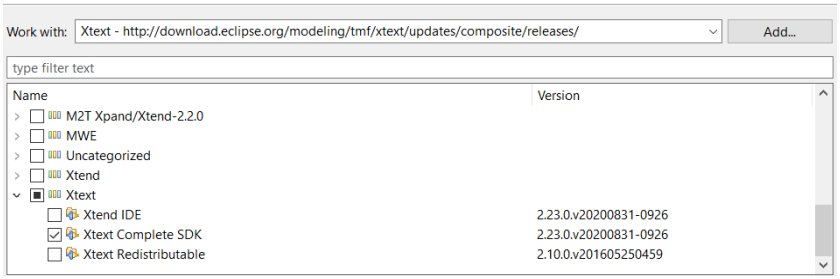
Open With menu. If menu item *Sample Reflective Ecore Model Editor* is visible, choose it. Otherwise select *Other…* and then find *Sample Reflective Ecore Model Editor* in the list

- Once the file is open in the tree editor, perform a right click on the root node and select *OCL*, then *Load Document*
- Use the button *Browse Workspace …* and locate the OCL file
- In order to validate the model, do again a right click on the root and select *Validate*. A message will report the validation result
- ***Important***: if you close the file and then open it again, you have to perform the steps above for loading the OCL document if you want to perform the validation

Xtext

Installing Xtext

- Install Xtext from the following update site address: http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/. If needed, consult the section on OCL for the details about installing software from an Eclipse update site.
- From the category *Xtext*, choose *Xtext Complete SDK*



Creating a Xtext Project

We will create a particular kind of Xtext project – one that uses already existing metamodel. Therefore, a *prerequisite* for this task is an *existing Ecore metamodel* with a *generator model* (.genmodel file) and *generated model code* (see the section on Eclipse Modeling Framework for details).

- From menu File/New/Project …, choose the wizard folder named *Xtext*. Select *Xtext Project from Existing Ecore Models*. Press *Next*
- In the next screen, press *Add...* button and select the existing .genmodel file (recall that it is required to already have a metamodel and its corresponding generator model). Press *Next*
- In the next screen, give the project name, the language name, and a file extension. The project name follows the conventions for Java packages. In this course, projects are typically prefixed with *nl.tue.glt*. The language name is typically formed by the project name concatenated with a simple name starting with a capital letter. The extension is the file extension that will be used for the model files expressed in your DSL. Once you give values of these three fields, the rest can be kept as suggested by default. Press *Next* and on the next screen press *Finish*
- After executing the steps above, Xtext will create 5 new projects: all start with the given language name, 4 projects having suffixes *ide*, *ui*, *test*, *ui,tests*. Furthermore, Eclipse will automatically open an .xtext file that contains a default grammar derived from the given

metamodel. This file is supposed to be modified in order to define the desired DSL grammar. Initially, the grammar will contain errors. Do a right-click on the Eclipse project that contains *the metamodel*, then select menu *Configure*, then *Convert to Xtext Project*. The errors in the grammar should disappear. If this is not the case, do some harmless change in the grammar (e.g. put a space) and save the file in order to refresh the grammar editor

- Define the grammar of your DSL (the grammar language is explained in the lecture). Remember that the first rule defines the starting symbol. Once you have a version that can be tried, you need first to generate the editor and parser infrastructure. Open the project with the language name, then the package with the same name. It should contain the .xtext file and another file with extension *mwe2*. Do a right-click on the .mwe2 file, then select *Run As,* and then *MWE2 Workflow*. Xtext will generate code, the result is reported in the console. If the grammar has problems like ambiguity or left recursion, the errors/warnings will be reported in the console (usually in green color)
- Inspect the generated code. Important packages are those that end with *validation*, *scoping* and *generator*. They contain files that can be modified by you as explained in the lecture. If errors are indicated, make sure that the model code is generated from the metamodel
- ***Important***: every time you change the grammar in the .xtext file, you have to repeat the execution of the MWE2 Workflow

Using the Generated Editor

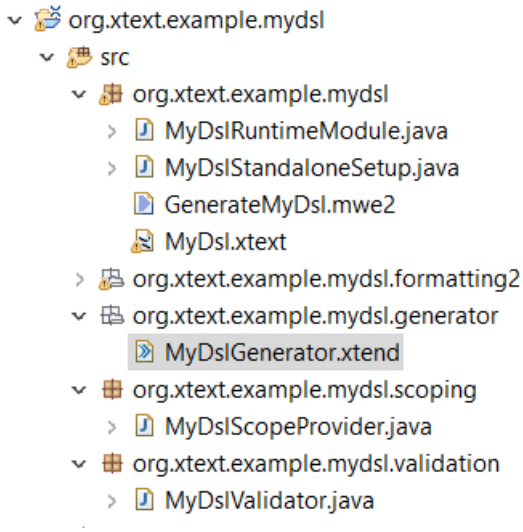
Once the grammar is defined and the editor is generated and customized, you can try it. The editor code and the grammar are contained in the so called *Eclipse development instance*. The editor is executed and used in another Eclipse instance, called *runtime Eclipse application*:

- Use the *first two steps* in section *Using the generated editor in an Eclipse Application* to start an Eclipse runtime application and to create a project
- In the newly created project, create a new text file and give as an extension the extension of your language
- Open the new file by double-clicking. It will be open in the generated Xtext editor with support for syntax highlighting, code completion, etc. If you are asked to agree to open an Xtext nature, accept it

Invoking the Code Generator

Every time you save a file from the generated editor, the code generator will be automatically invoked and the generated files will be placed in folder *src-gen* under the project that contains your model file. ***Important***: this happens in the *runtime Eclipse application* started as described in the previous section.

The code generator is located in your *Eclipse development instance*. It is placed by default in *class $LanguageName$Generator.xtend* in package *$ProjectName$.generator* in the Xtext project:



The class has one method named *doGenerate* that is initially empty. The content of this method has to be modified. Details about code generation with Xtext are given in the lecture on 8 January, 2021.

MyDslGenerator.xtend

```
2⊕ * generated by Xtext 2.23.0
4 package org.xtext.example.mydsl.generator
5
6⊕import org.eclipse.emf.ecore.resource.Resource
10
11⊖/**
12  * Generates code from your model files on save.
13  *
14  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
15  */
16⊖class MyDslGenerator extends AbstractGenerator {
17
18⊖    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
19 //      fsa.generateFile('greetings.txt', 'People to greet: ' +
20 //          resource.allContents
21 //              .filter(Greeting)
22 //              .map[name]
23 //              .join(', '))
24     }
25 }
```

More concretely, when the saving is performed, the Xtext framework automatically parses the model (in text) and if it is syntactically correct, creates an Ecore model instance of the DSL metamodel. The Ecore model is passed via the first parameter of the doGenerate method: the Ecore resource that contains the model.

QVTo

This section contains information about installing QVTo Eclipse plugins, creating a new transformation project and running a transformation.

Installing QVTo

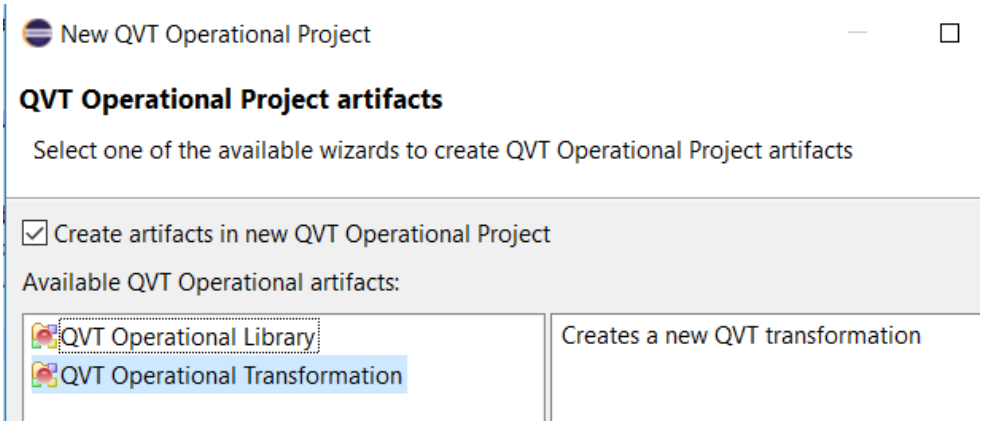
Install QVTo from the following update site:
http://download.eclipse.org/mmt/qvto/updates/releases/latest

If you are not familiar with the procedure of installing software via an Eclipse update site, please check the section about the OCL language where it is explained.

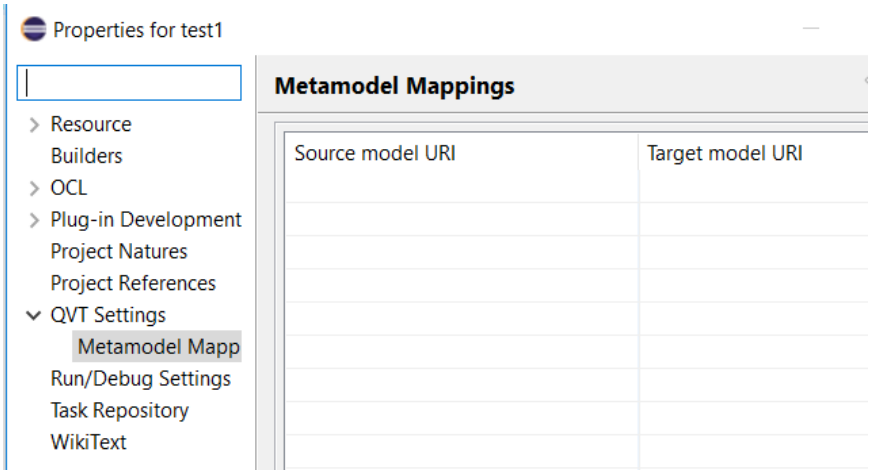
Creating a new QVTo Project

In order to create a new QVTo project, follow the steps:

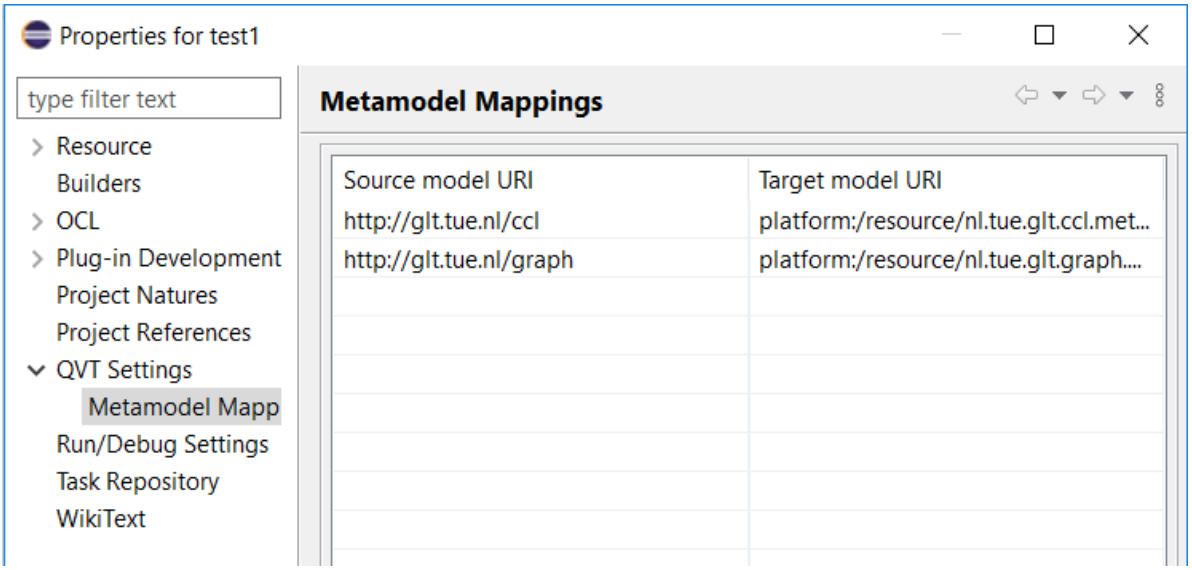
- Select menu *File/New/Project…* Expand the folder *Model to Model Transformation* and select item *QVT Operational Project*. Press *Next*
- In the next dialog box, give a name of the project. Keep the default suggested values for the other controls. Press *Next,* and again *Next* in the following dialog page
- Select the check box *Create artifacts in new QVT Operational Project*. Select *QVT Operational Transformation.* Press *Next*



- In the next dialog, give a name of the new transformation and press *Finish*
- The newly created project will contain a folder named by default as *transforms* with a file with extension *.qvto*. This file contains the model transformation to be developed. The last remaining step is to register the metamodels that will be used in the transformation
- Do a right-click on the QVTo project (in the Model or Project Explorer tree) and select *Properties*. In the dialog box, expand *QVT Settings* and then select *Metamodel Mappings*.



- Press the *Add* button. In the next dialog, a metamodel has to be selected. We assume that the Ecore file with the metamodel is located in some project in your workspace. Press button *Browse*. Locate the .ecore file and then press *OK*.
- Repeat the step above for the other metamodels that will be used. Be aware that the table columns named *Source model URI* and *Target model URI* are misleading and should be ignored. The table should contain all the used metamodels, one row per metamodel. In the following screen you can see an example that will use two metamodels:



- Once all needed metamodels are registered, press *Apply and Close*. Now you can write the transformation in QVTo, the syntax and semantics of this language is explained in the lecture. The URIs of the registered metamodels are known to the QVTo environment and can be used in the model type declarations.

Running a Transformation

A QVTo transformation is executed by taking one or more models as input and produces one or more models as a result. This information is provided in a *run configuration* (a common Eclipse concept)

- Select the .qvto file to be executed and do a right-button click. Select Run As/Run Configurations…
- From the list on the left hand side of the dialog, select QVT Operational Transformation. Press the button *New launch configuration* (in the top left corner, the left-most button) or just double click on *QVT Operational Transformation* item
- Give a name of the launch configuration. The field *Transformation module* should contain the qvto file selected in the first step
- The content of the dialog box reflects the transformation signature in the .qvto file. For each **in** (input) and **out** (result) model in the signature, there will be a text field. For each *input* model use the button *Browse* to navigate to an existing model (typically in XMI format). For each *result* model, give the *location* and the *name* of the file that will be produced. Be aware that usually the result model does not exist yet so you cannot use the browse button. Enter manually the path and the name of the file
- Press *Run*. The transformation will be executed. The created run configuration will be saved and can be used multiple times and modified if needed