Trying Out the Simple Language of Communicating Objects

Luc Engelen

March 26, 2013

1 Introduction

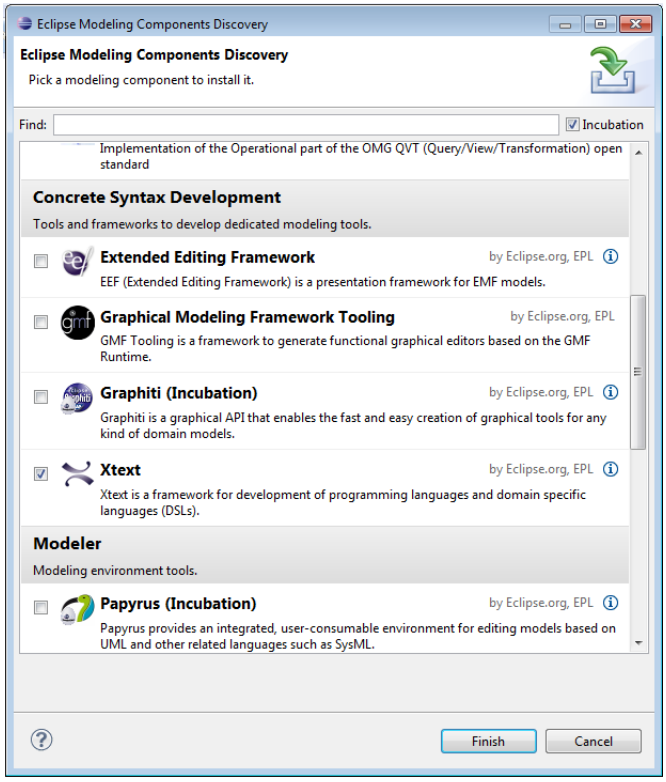
This document describes how to use the Simple Language Of Communicating Objects (SLCO) with Eclipse. SLCO is a domain-specific modeling language for describing systems consisting of objects that operate concurrently and communicate with each other by passing signals over channels. The behavior of these objects is defined by state machines. SLCO has a graphical and a textual syntax for models. By applying (sequences of) model transformations, SLCO models can be transformed to implementations as well as models for formal modeling languages that offer simulation and verification.

Chapter 2 describes how to install Eclipse and the modeling components that are needed to edit SLCO models and execute the transformations related to SLCO. The archive that accompanies this document contains two workspaces for Eclipse, which are described in Chapters 3 and 4. One of these contains the grammars and metamodels related to SLCO. This workspace is described in Chapter 3. The other workspace contains the model transformations related to SLCO and a number of example models. This workspace is described in Chapter 4.

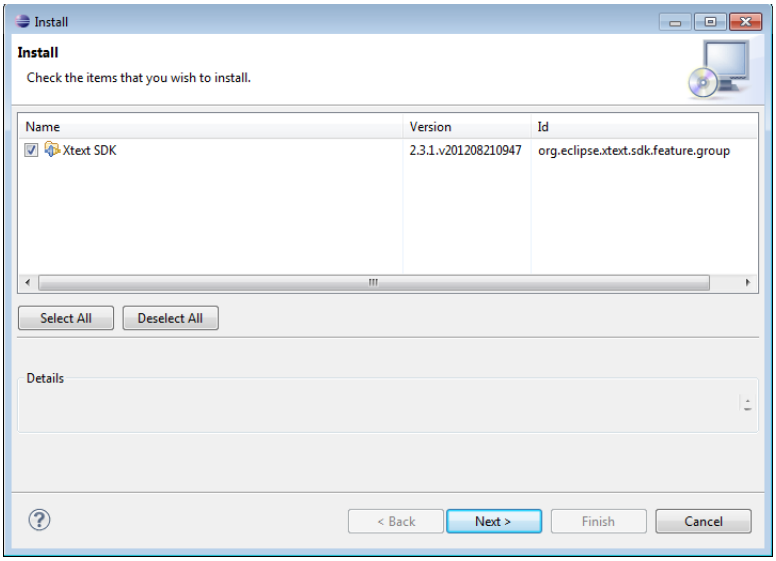
2 Installation

First, unpack the archive that accompanies this document to an appropriate folder. Afterwards, install Eclipse by following the steps below. Eclipse is available for Windows, Mac, and Linux. We explain how to install Eclipse and the required components for Windows, but the procedure for Mac and Linux is similar.

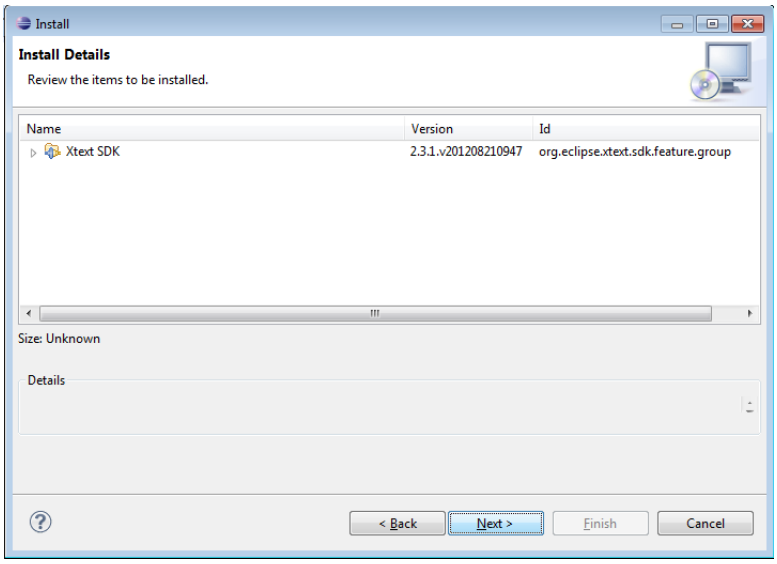
1. Navigate to `http://www.eclipse.org/downloads/` and download the version of Eclipse called Eclipse Modeling Tools.
2. Unpack the downloaded file.
3. Start Eclipse by running `eclipse.exe` from the eclipse folder.
4. Eclipse will ask you which workbench you want to use. Select the workspace called `grammars-and-metamodels` that you unpacked before installing Eclipse.
5. In the Help menu, select Install Modeling Components.
6. Select Xtext, as shown in the screenshot below, and click Finish.



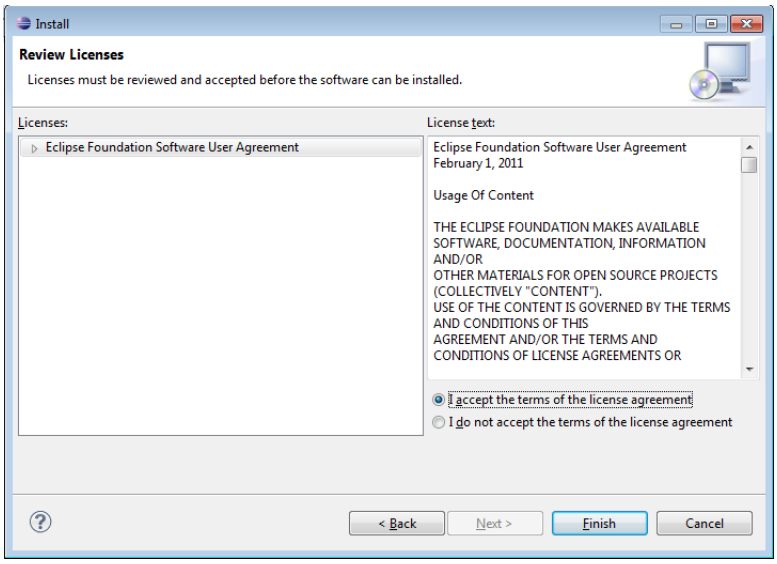
7. Ensure that Xtext SDK is selected, as shown in the screenshot below, and click Next.



8. Click Next again.

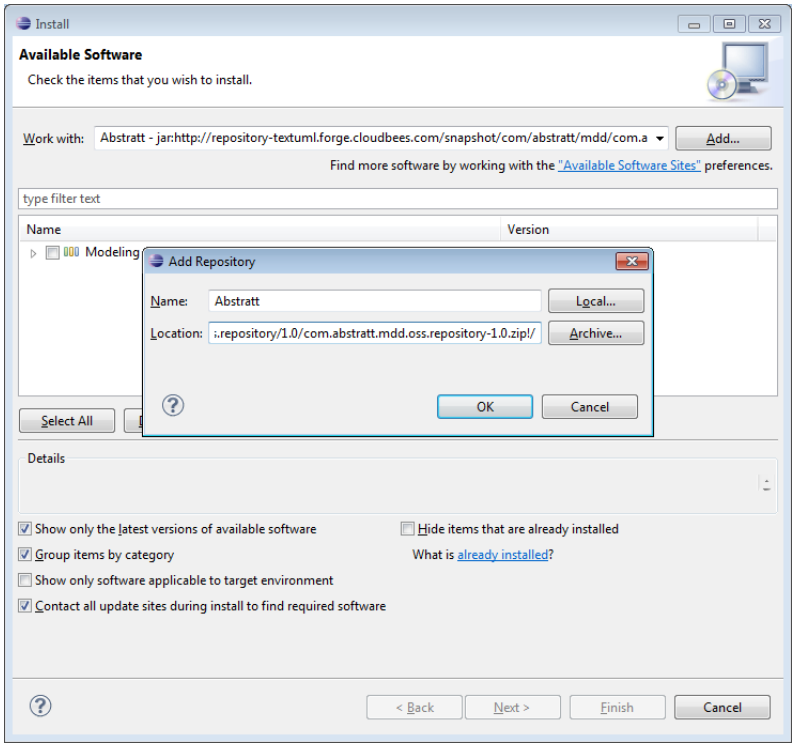


9. Indicate that you agree to the terms of the license agreement, as shown below, and click Finish.

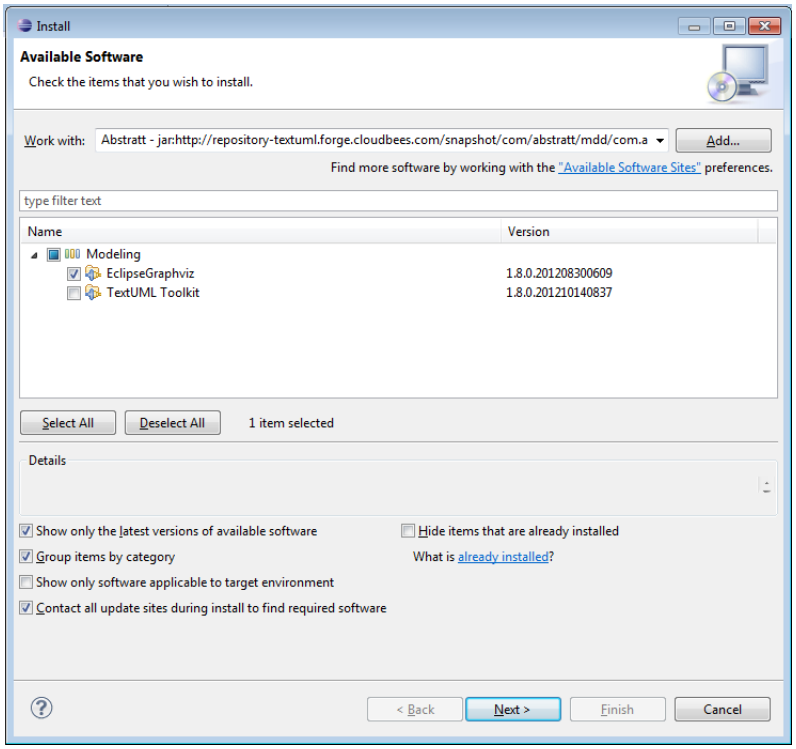


10. Restart Eclipse if needed.
11. Install Xpand and ATL following the same procedure.
12. Install Graphviz as described on `http://www.graphviz.org/download.php`.
13. In the Help menu of Eclipse, select Install New Software....
14. Click Add, and enter `Abstratt` as name and `jar:http://repository-textuml.forge.cloudbees.com/snapshot/com/abstratt/mdd/com.abstratt.mdd.oss.repository/`

`1.0/com.abstratt.mdd.oss.repository-1.0.zip!/` as location, as illustrated in the screenshot below. The link to the location of the update site can also be copied from the installation instructions on the TextUML website (`http://sourceforge.net/apps/mediawiki/textuml/`).



15. Select EclipseGraphviz, as shown below, and click Next.

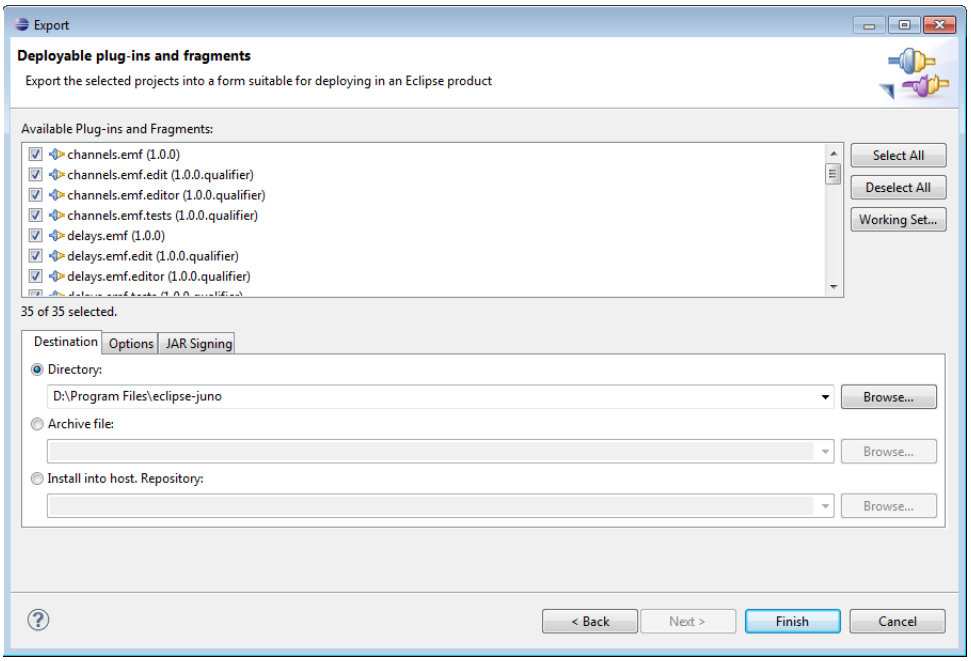


16. Complete the installation procedure of EclipseGraphviz as you did when installing Xtext, Xpand, and ATL.

3 Grammars and Metamodels

Once you have completed the installation of Eclipse and all the required components, you can deploy the projects in the `grammars-and-metamodels` workspace as plug-ins. The projects in this workspace implement the editors for all models related to SLCO. Deploy the projects as plug-ins by following the steps below, to be able to create, edit, and transform models.

1. In the File menu, select Export....
2. In the pop-up window, select Plug-in Development → Deployable plug-ins and fragments and click Next.
3. In the next pop-up window, click Select All and set the directory to your Eclipse folder. Finally, click Finish.



4. Restart Eclipse by switching to the `models-and-transformations` workspace.

4 Models and Transformations

The `models-and-transformations` workspace contains a number of models and transformations to illustrate the capabilities of SLCO. The content of this workspace is shown in Figure 1. The remainder of this chapter describes the projects in this workspace.

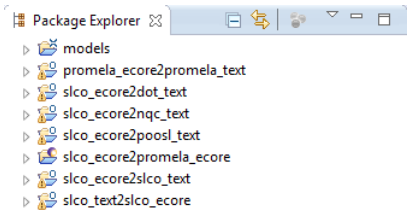


Figure 1: Projects of the `models-and-transformations` workspace

4.1 The `models` project

The `models` project contains a number of example models. The folder `models/slco_text`, for example, contains two textual SLCO models.

The EMF equivalents of the textual models in `models/slco_text` can be found in the folder `models/slco_ecore`. A number of dot files that illustrate the graphical versions of these models can be found in the folder `models/slco_ecore/dot`. You can view these dot files using

the EclipseGraphviz component you installed. Follow the steps below to add the Image Viewer of EclipseGraphviz.

1. Select Other... in the Windows → Show Menu submenu.
2. Select Image Viewer in the EclipseGraphviz folder.

Afterwards, you can display dot files in the Image Viewer after opening them.

The folder `models/slco_ecore/poosl_ecore` contains a number of POOSL models in EMF format, which are generated from the SLCO models in `models/slco_ecore`. The files that form the textual equivalents of these models can be found in `models/slco_ecore/poosl_text`.

Similar folders, named `models/slco_ecore/promela_ecore` and `models/slco_ecore/promela_text`, exist for Promela models. The transformation to Promela can only handle SLCO models containing unidirectional channels. The folder `models/slco_ecore/promela_slco_ecore` contains an SLCO model that is transformed to meet this requirement.

The folder `models/TransformationExamples` contains a number of models that illustrate the effect of each of the refining transformation for SLCO. The folder named `models/TransformationExamples/dot` contains the graphical versions of the models, `models/TransformationExamples/slco_ecore` contains the EMF versions of the models, and `models/TransformationExamples/slco_text` contains the textual versions. Most of the refining transformations for SLCO require an SLCO model and some model containing additional information as input. The models containing this additional information can also be found in the folder `models/TransformationExamples/slco_ecore`. For example, when adding delay statements to certain transitions in an SLCO model, the user must indicate the length of the delays and the transitions to which these statements must be added. The file named `models/TransformationExamples/slco_ecore/AddDelays_time.delays` contains this information. Given this file and the SLCO model `models/TransformationExamples/slco_ecore/AddDelays.slco`, the refined model `models/TransformationExamples/slco_ecore/AddDelays_time.slco` is created.

The folders `models/LegoCase` and `models/Simple` contain more elaborate examples of SLCO models and the resulting POOSL models, Promela models, and NQC implementations. The folders named `channels`, `delays`, `objects`, and `slco2nqc` contain the models with the additional information used to transform SLCO models to NQC implementations. The intermediate SLCO models of the sequences of transformations used to transform SLCO models to NQC implementations are stored in the folders named `nqc_slco_ecore`.

4.2 The `promela_ecore2promela_text` project

The `promela_ecore2promela_text` project contains an Xpand template that generates a textual representation of a model from an EMF representation of this model. This transformations can be found in the folder `promela_ecore2promela_text/src/Transformations`. The folder `promela_ecore2promela_text/src/Workflows` contains a number of mwe workflows that can be executed to generate such textual representations for specific models. Such a workflow can be executed by right-clicking and then running them as MWE Workflows.

4.3 The `slco_ecore2dot_text` project

The `slco_ecore2dot_text` project contains an Xpand template that generates Dot files from an SLCO model and a number of workflows to execute this template.

4.4 The `slco_ecore2nqc_text` project

The `slco_ecore2nqc_text` project contains a number of transformations that refine SLCO models, a transformation that transforms SLCO models to NQC models, a template that generates the textual representations of these NQC models, and a number of workflows that can be used to execute these transformations.

4.5 The `slco_ecore2poosl_text` project

The `slco_ecore2poosl_text` project contains an Xtend transformation to transform SLCO models to POOSL models and an Xpand template to generate textual representations of such POOSL models.

4.6 The `slco_ecore2promela_ecore` project

The `slco_ecore2promela_ecore` project contains an ATL transformation to transform SLCO models to Promela models and a number of Ant tasks to execute this transformation on specific models. These Ant tasks can be executed by right-clicking them and running them as ANT Builds.

4.7 The `slco_ecore2slco_text` project

The `slco_ecore2slco_text` project contains an Xpand template to generate textual representations of SLCO models from their EMF representations and a number of workflows to apply this template to specific models.

4.8 The `slco_text2slco_ecore` project

The `slco_text2slco_ecore` project contains an Xtend transformation to transform textual SLCO models to their EMF representations and a number of workflows to apply this transformation to specific models.