TigersEye v. 0.0.1
User Manual

Leo Roos
leo_roos@rbg.informatik.tu-darmstadt.de

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Advisor: Tom Dinkelaker

Contents

1 Introduction

The TigersEye Eclipse Plug-in is an IDE that strives to support easy creation of EDSLs. This guide describes how to install TigersEye for development, the currently implemented features and shows on some examples how to create and install a new language. TigersEye has been previously known as `Popart`, which has been developed by Yevgen Fanshil, Leonid Melnyk, Thorsten Peter, David Marx, Kamil Erhard and Tom Dinkelaker. The used examples are adjusted versions of their prior work.

Plug-in name	Version	Description
de.tud.stg.tigerseye.eclipse.core	0.0.1	Core functionality.
de.tud.stg.tigerseye.eclipse.ui	0.0.1	User Interface functionality.
de.tud.stg.tigerseye	0.0.1	(Re)Exports commonly used libraries and plug-ins.
Groovy Eclipse Feature	2.1.1	Groovy Eclipse Plug-ins with version that is known to be compatible.
org.apache.commons.collections	-	Apache utility classes for collections.
org.apache.commons.io	-	Apache IO utility classes.
org.apache.commons.lang	-	Apache general language utility classes.
org.apache.log4j	[1.2 - 1.3)	Employed logging framework.
parlex	-	Preprocessor, performing the transformations.
slf4j-log4j12	-	Employed logging facade with log4j binding.

Table 1: Necessary Plug-ins

2 Installation for Development

The TigersEye plug-in itself consists of multiple separate plug-ins and has further dependencies to other plug-ins and libraries. It uses classes and extensions from the Groovy Plug-in and Eclpse's JDT plug-ins, makes some use of different `apache.commons` libraries and uses further libraries to process code. It's core dependency is the used preprocessor `parlex`. Listing 1 shows the necessary plug-ins and the required version if any.

Once all the necessary plug-ins have been installed the TigersEye plug-in can be started using the predefined `Tigerseye_IDE` launch, which is located in the core plug-in. It is possible that depending on the used operating system this launch configuration has to be adjusted. Alternatively one can start a new Eclipse Plug-in configuration with all available plug-ins active.

3 Features

The current version of TigersEye provides the fundamental functaionlities in order to be able to use it as a language workbench. New languages can easily be created and deployed. After a restart they can be used in projects that have the TigersEye nature. A TigersEye nature can easily be added and removed via a popup menu. The installed DSLs can be configured through the preference pages, as well as the employed transformations. The TigersEye editor is an extended version of the Groovy editor and provides keyword coloring for keywords of installed and activated DSLs.

3.1 Preferences

The prefrence pages are an important part of TigersEye , since they provide the configuration of registered DSLs. In the main prefrence page (Figure 1) the output source folder can be adjusted.

Registered languages can be configured on the languages preference page (Figure 2). DSLs have to be set into activated state in order to use them. The extensions, that indicate which concrete DSL class is responsible for which files of the according extension can be adjusted in the `Extensions` column. When a DSL is selected its keywords are shown below in the *Declared Keywords of Selected DSL* dialog.

The transformations preference page provides the adjustment of used transformations for the specified resource or DSL (Figure 3). For each resource and DSL different transformations might be provided. These can be configured using the `Select Transformations` dialog (Figure 4). The dialog also shows additional informations about the currently selected transformation in a tray window that can be opened clicking on the additinal information button (Figure 5). The TigersEye editor provides keyword coloring for active DSLs. The colors can be configured using the Tigerseye Editor preference page (see Figure 6), where every DSL can be configured seperately and the general keyword coloring can be activated or deactivated.

3.2 Add and Remove TigersEye Nature

TigersEye has additional requirements which will be imported when adding the TigersEye nature to a project. The project must have at least the Java nature otherwise the transformation to a TigersEye project is not possible. Figure 7 shows the available popup menu to add the TigersEye nature to a project. This will do two things. A seperate source folder will be created into which the translated DSL files will be output (here: `src-tigerseye`) and a new class path container will be added which contains the runtime libraries (`popartAnnotations.jar, popart.jar, edslNature.jar`) and the libraries of registered DSLs as well as their dependencies. For example `de.tud.stg.tigerseye.examples.LogoDSL`

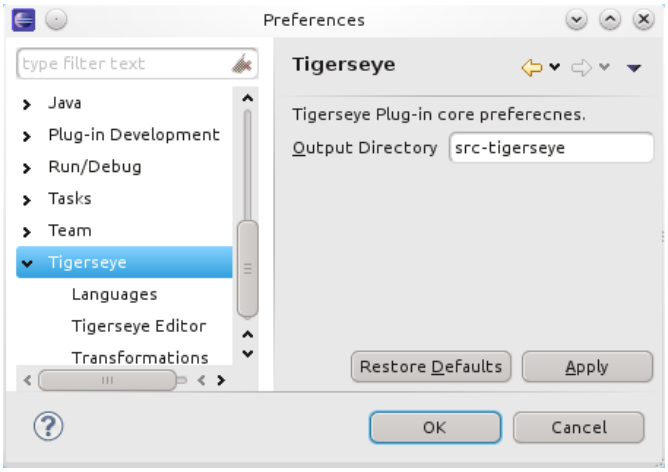

Figure 1: Tigerseye Main Preference Page

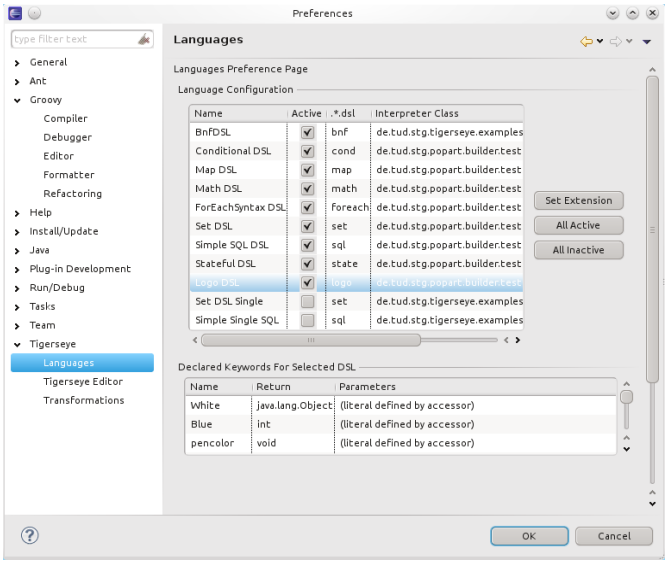

Figure 2: Tigerseye Language Configuration Page

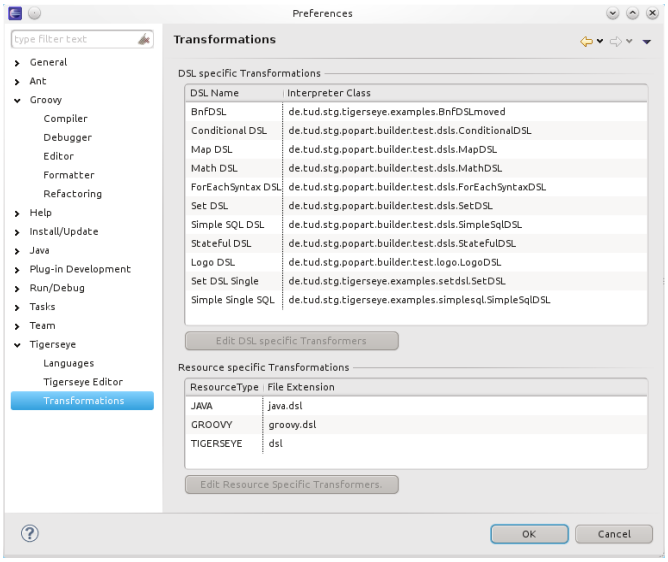

Figure 3: Tigerseye Transformations Preference Page

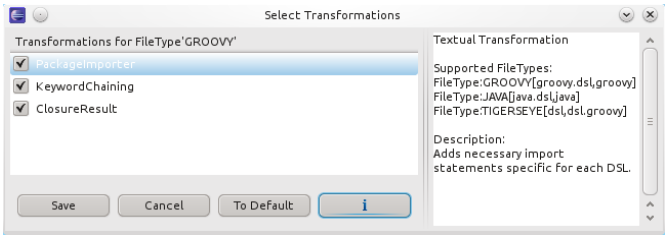

Figure 4: Tigerseye Transformations Configuration

Figure 5: Additional Information Button

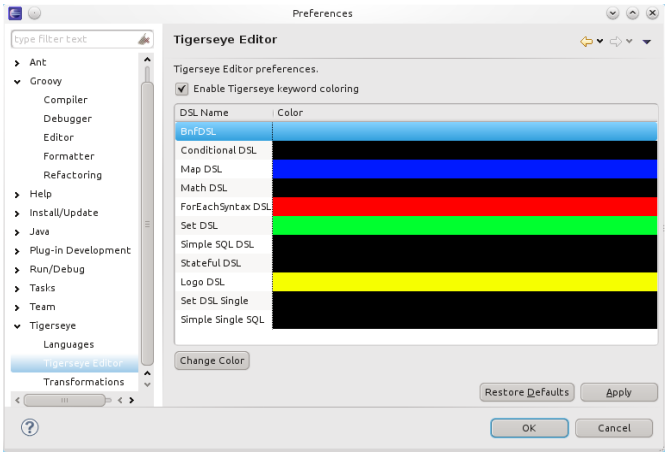

Figure 6: Tigerseye Editor Preference Page

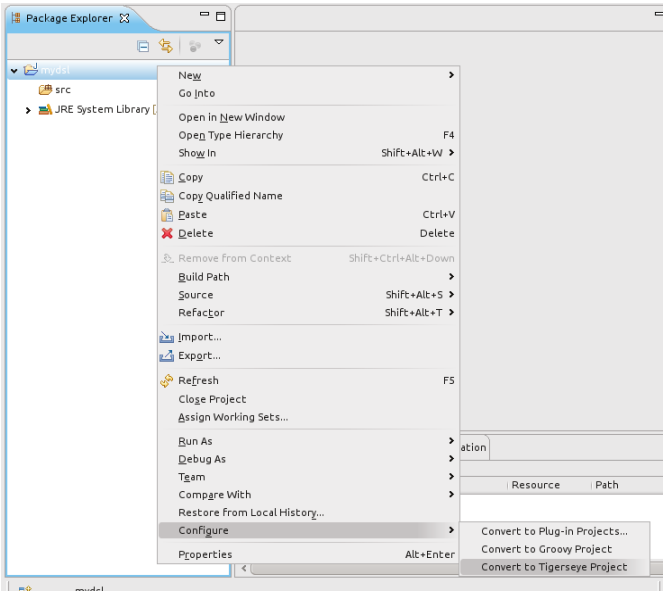

Figure 7: Add the TIGERSEYE Nature to a Java Project

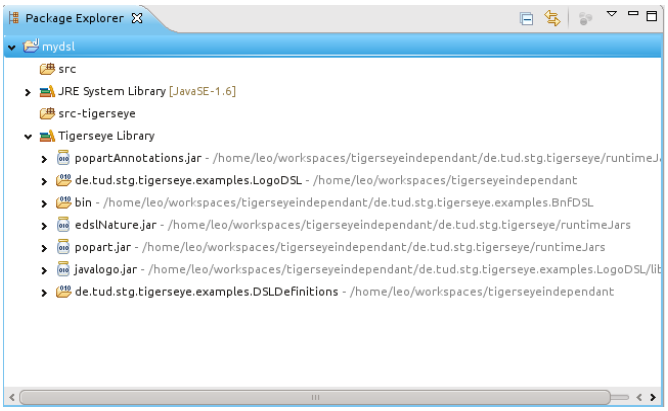

Figure 8: TIGERSEYE Dependencies

and `de.tud.stg.tigerseye.examples.DSLDefinitions`. Additionally the `GroovyNature` will be added if not already configured.

3.3 TIGERSEYE Language Definition Wizard

A new language can be created using the *New Language Wizard*. The Wizard can be accessed via File − > New − > Other (Figure 10). Figure 11 shows the first page of the Wizard. There the name of the main language class can be defined. As can be seen in the figure the default package is not a valid package for a language defintion since this will cause problems when trying to use the language within a Java class. Figure 12 shows the actual language definition page. There the different literals, operations and structured elements can be added. In Section 4 the usage of the wizard will be showcased.

3.4 New TIGERSEYE Class Wizard

The new TIGERSEYE *Class Wizard* enables easy creation of new DSL classes. In Figure 13 the Wizard is shown. It is basically an adjusted version of the new Java Class Wizard. Additionally to being able to define the typical class properties a DSL can be chosen which one wishes to use in the class.

3.5 Launch Tigerseye DSL

A TIGERSEYE 1 DSL can be launched using the launch shortcut or via the *Run Configurations* dialog. Figure 14 shows a launch using a launch shortcut. Figure 15 shows the launch via the *Run Configurations Dialog*. There a new launch

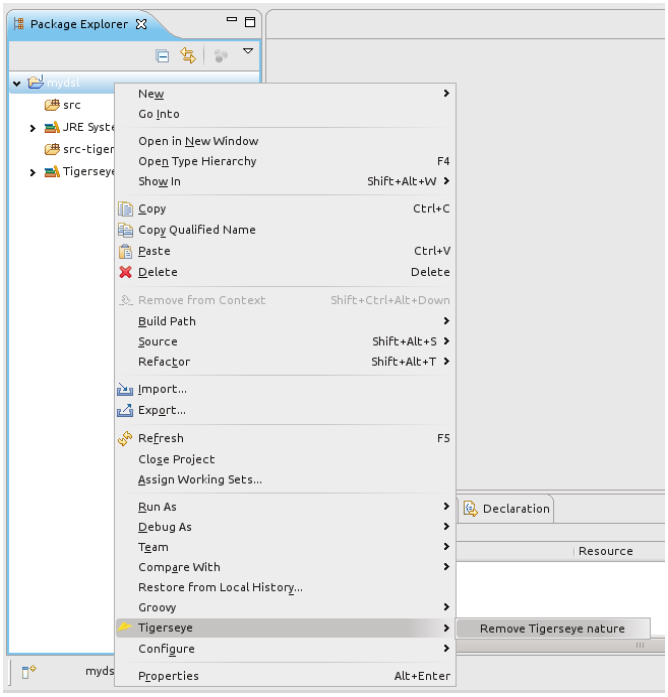

Figure 9: Remove the TIGERSEYE Nature from a Project

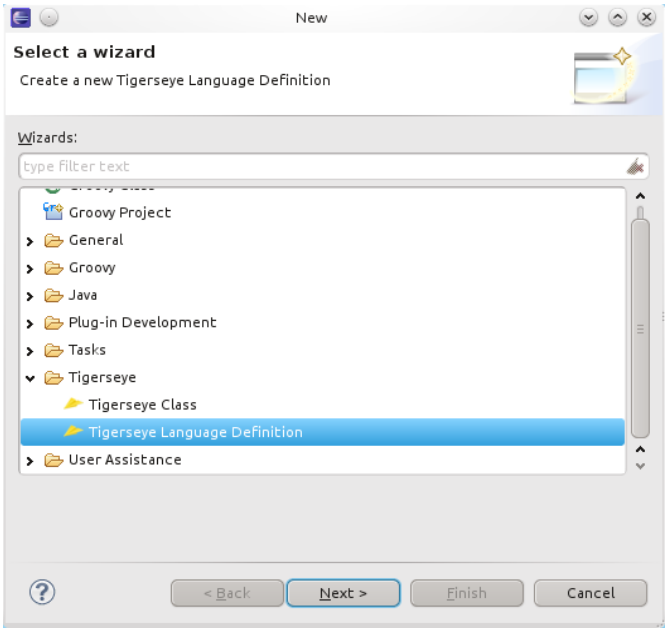

Figure 10: Choosing the New Tigerseye Language Wizard

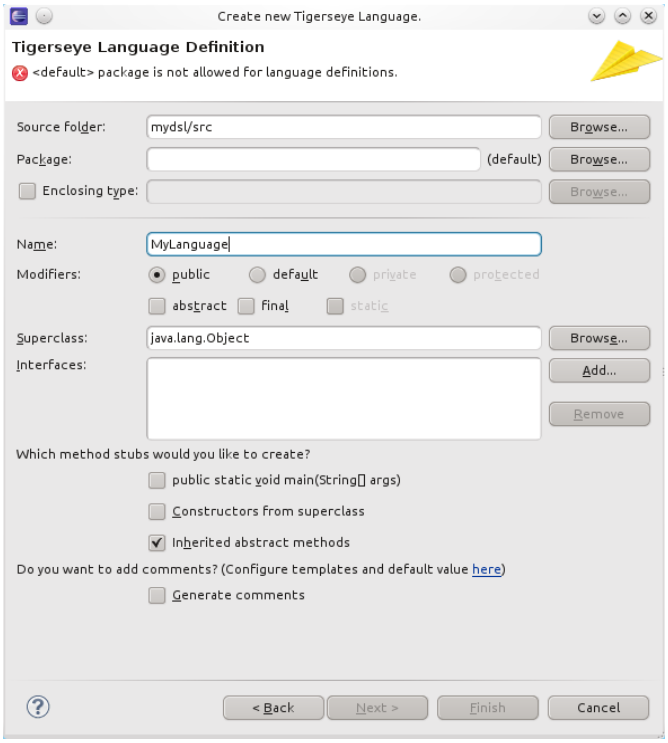

Figure 11: Tigerseye Language Definition Wizard Page 1

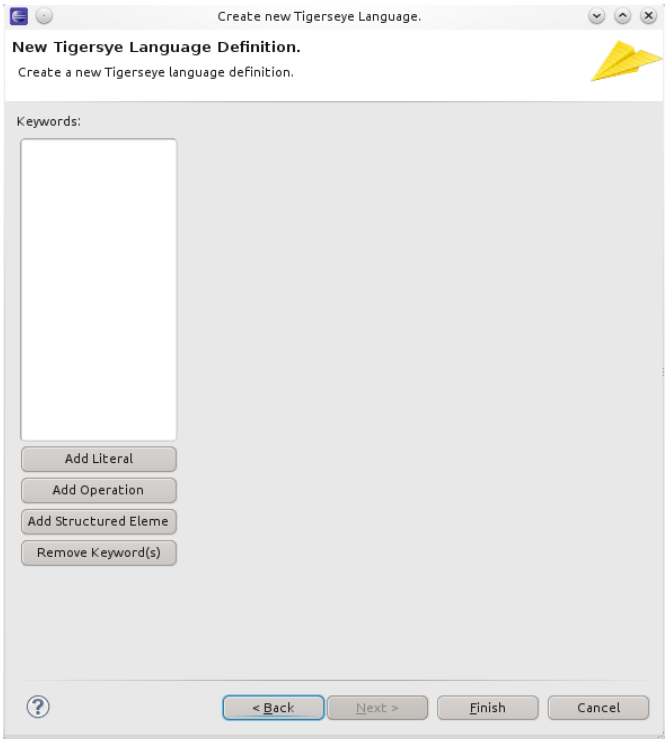

Figure 12: Tigerseye Language Definition Wizard Page 2

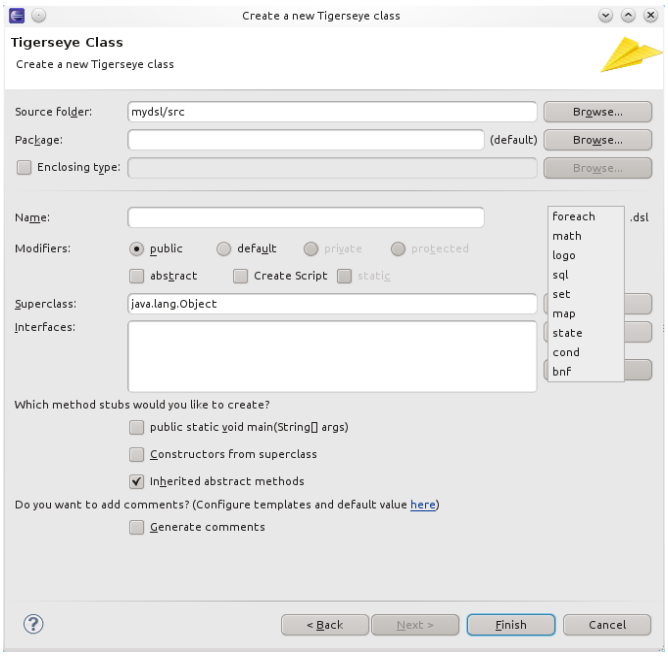

Figure 13: New Tigerseye Class Wizard

can be configured or a previous launch adjusted. On the *Tigerseye* tab the project from which a DSL will be launched as well as the dsl file to launch can be chosen. When using the launch shortcut the Groovy default launch configuration is assumed which will set additional classpath properties. Later these can be modified using this dialog.

4 Examples

This section showcases typical use cases for the TigersEye IDE.

4.1 Creating a new TigersEye Language Definition

The usage of the *New Tigerseye Language Definition* wizard will be explained by creating a *Trivalent-DSL*. A Trivalent logic DSL adds an `unknown (U)` value to the boolean values `true (T)` and `false (F)`, so `T&U` is U and `T|U` is T and so on.

- First create a new Java Project. This Project will contain the Trivalent language created by the wizard. In this example a project called `de.tud.stg.tigerseye.examples.trivalent` will be created
- Right Click on the Project and choose *New* > *Other*.
- Choose *Tigerseye Language Definition* in the *Tigerseye* folder (Figure 10).
- Each language definition consists of a Groovy class that defines all operations, literals and structured elements of the DSL. Type `TrivalentDSL` as the class name, `de.tud.stg.tigerseye.examples.trivalent` as the package and select *Next* (Figure 16).
- Now add three new literals: `T` (true), `F` (false) and `U` (unknown). They are all of type Trivalent (Figure 17). After finishing the wizard these three classes and the supertype Trivalent will be created. Notice that T, F and U extend Trivalent.
- The only operation will be an enhanced println which takes a String and a Trivalent expression and prints both, e.g. `puts("T|U: ",T|U)` will produce `T|U: T`. The return type should be `void` so we just let the `Return type:` field empty. For each operation you can choose if setting a breakpoint on a line containing this keyword should be possible (Figure 18). Also you can define all parameters and their types.
- Now add a repeat-statement. The following will simply print `T: T` ten times to stdout.

```
repeat(10) {
  puts("T:",T);
}
```

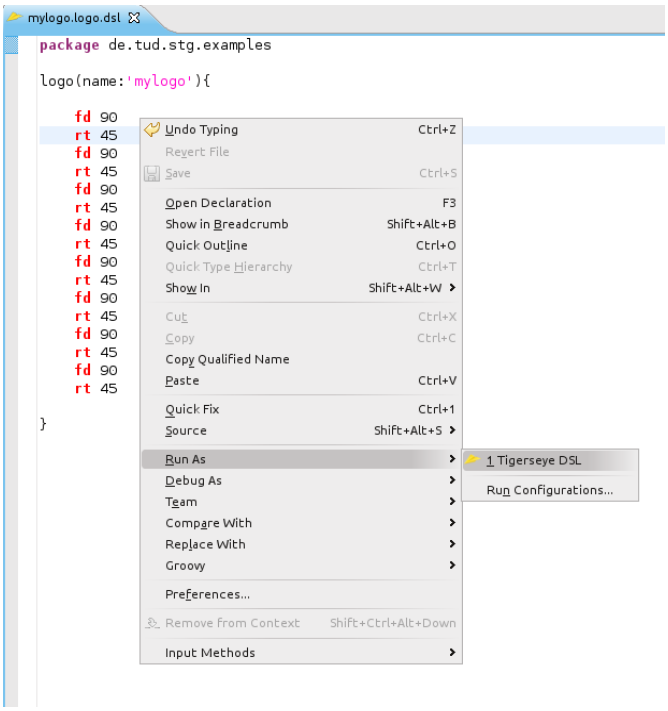

Figure 14: Launch via Context Menu

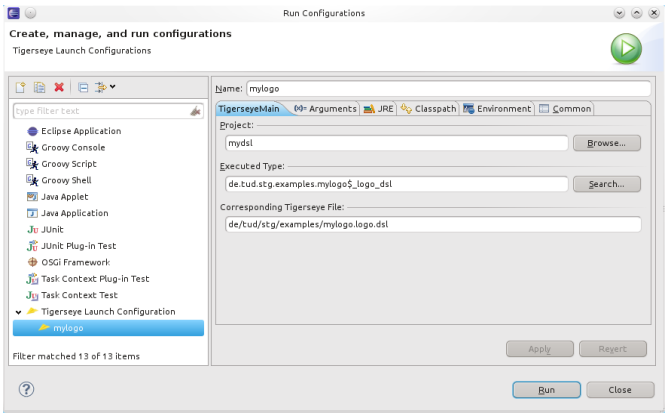

Figure 15: Launch via the Run Configurations Dialog

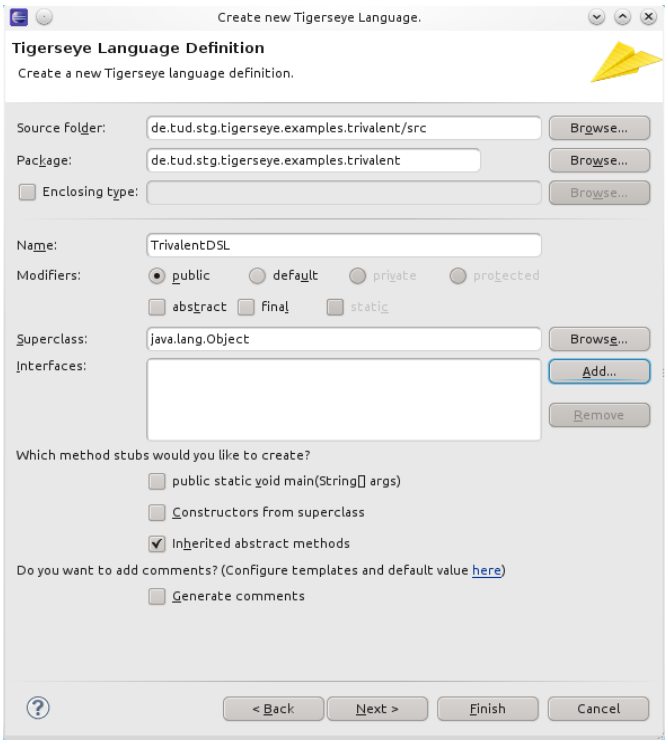

Figure 16: New TrivalentDSL Language Defintion

The return type will be `void` and there is one parameter named `n`. Select `explicit parameters` (Figure 19). Again, you can choose if it should be possible to set a breakpoint on a line containing this keyword.

- After you select `Finish` a dialog will pop up asking you if you want to add the TigersEye runtime libraries (Figure 20). Usually you should say yes, since language definitions have dependencies to the runtime libraries.
- The final result is shown in Figure 21. For the new type `Trivalent` as well as for the literals `T,U and F` a separate Groovy class is created. The language configuration is defined in the `TrivalentDSL` class.

4.2 Deployment of a TigersEye Language

To deploy a new language s.t., the user converts his designed language to a plug-in project. This plug-in project will declare its dependencies to two TigersEye plug-ins:

- `de.tud.stg.tigerseye`
- `de.tud.stg.tigerseye.eclipse.core`

The `de.tud.stg.tigerseye` plug-in provides the dependent on libraries and the `de.tud.stg.tigerseye.eclipse.core` plug-in the extension which declares that this plug-in project actually provides a new language.

The following steps have to be performed:

1. Convert the language definition project to a plug-in project. (Figure 22)
2. In the `MANIFEST.MF` file add the dependencies to the two Tigerseye plug-ins. (Figure 23)
3. Now open the `plugin.xml` and go the `Extensions` tab. Add the `de.tud.stg.tigerseye.dslDefinitions` extension point. On the extension point select `New > language`. There you can define the language class to be used. In this example this would be `de.tud.stg.tigerseye.examples.trivalent.Trivalent`. Additionally you should define a user friendly name of your new language, e.g. *Trivalent DSL*. Optionally you can define the default extension identifying your language, such as `tri`. Figure 24 shows the example configuration. The extension can also be configured using the TigersEye preference pages.
4. Currently only the deployment for development is supported. Language can either be copied or linked inside the eclipse instance in which the TigersEye plug-in is developed. The next time the TigersEye Eclipse instance is started the language will be visible in the preference pages and can be used.

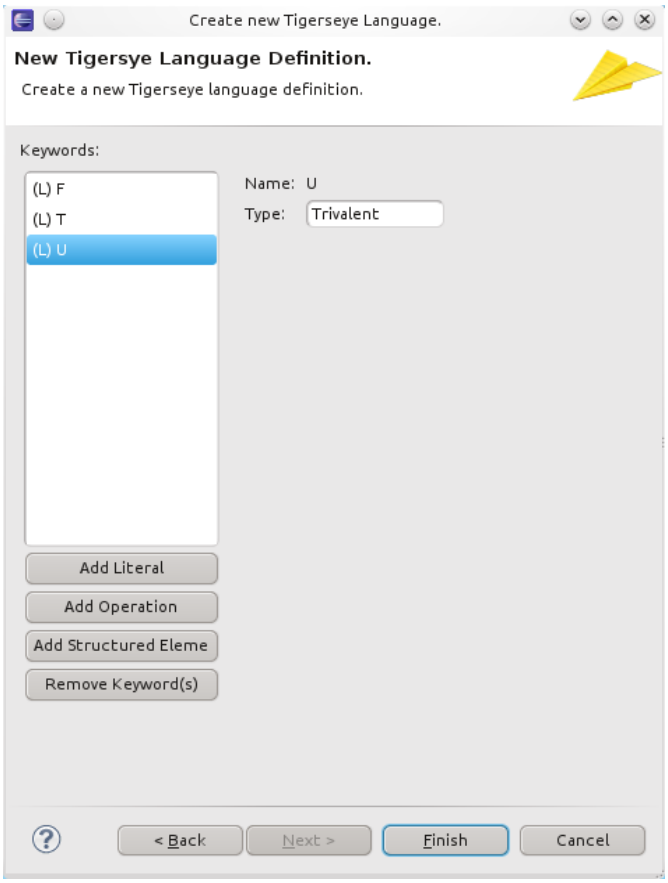

Figure 17: TrivalentDSL Language Definition with Literals added

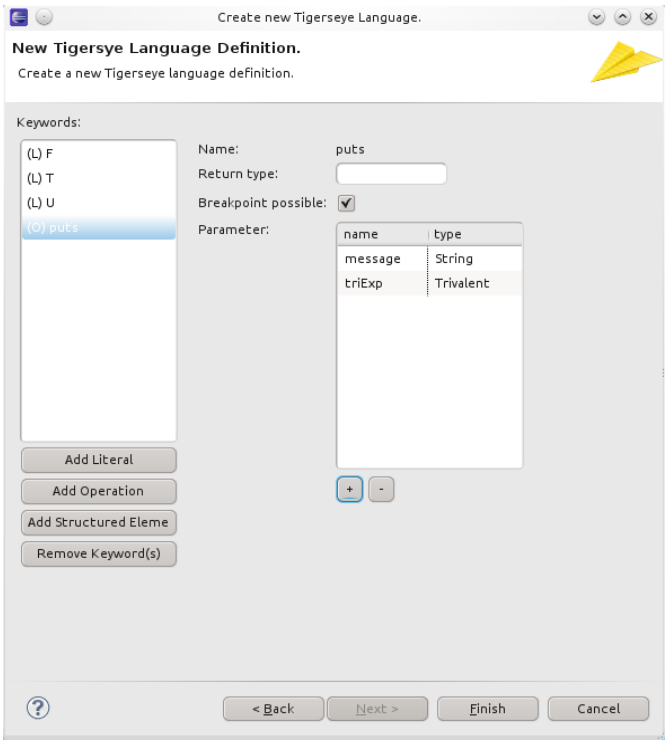

Figure 18: TrivalentDSL Language Definition Operation `puts` added

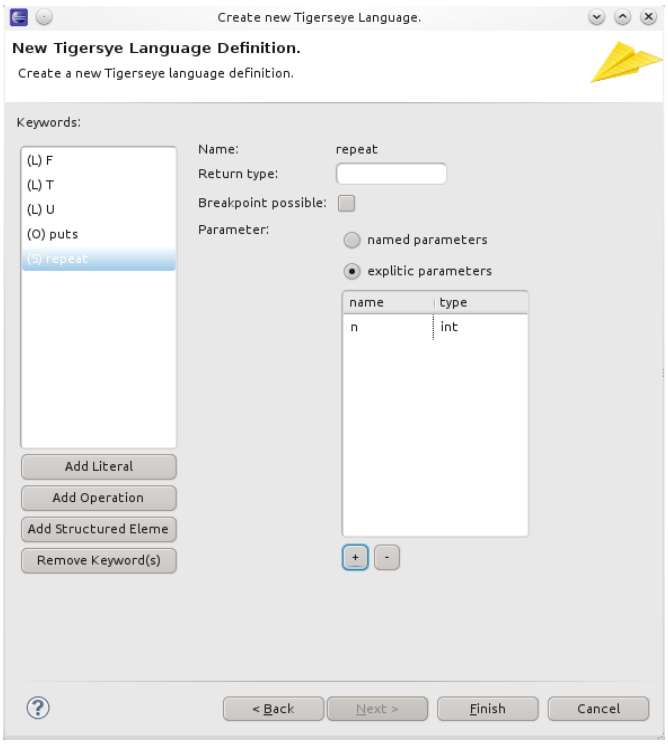

Figure 19: TrivalentDSL Language Definition Structured Element `repeat` added

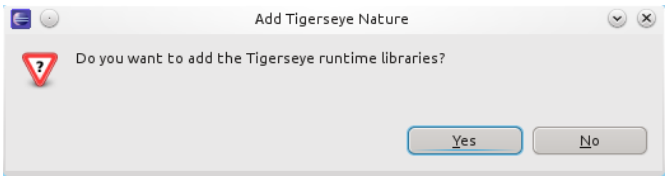

Figure 20: Question Dialog to add TigersEye Runtime Libraries

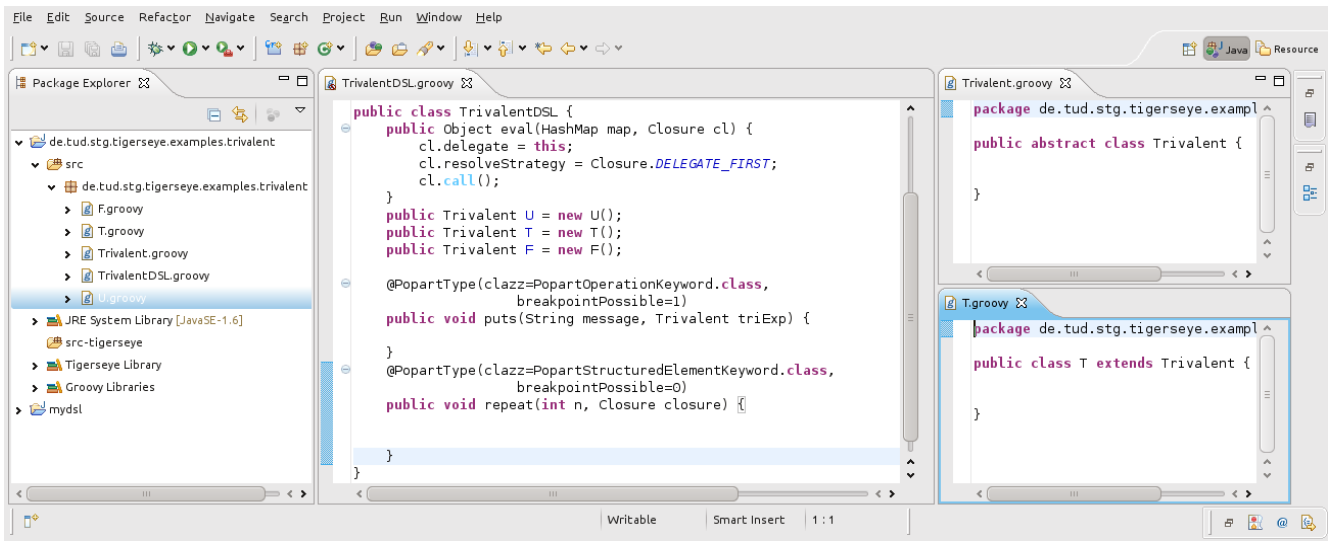

Figure 21: TrivalentDSL Generated Classes and Code

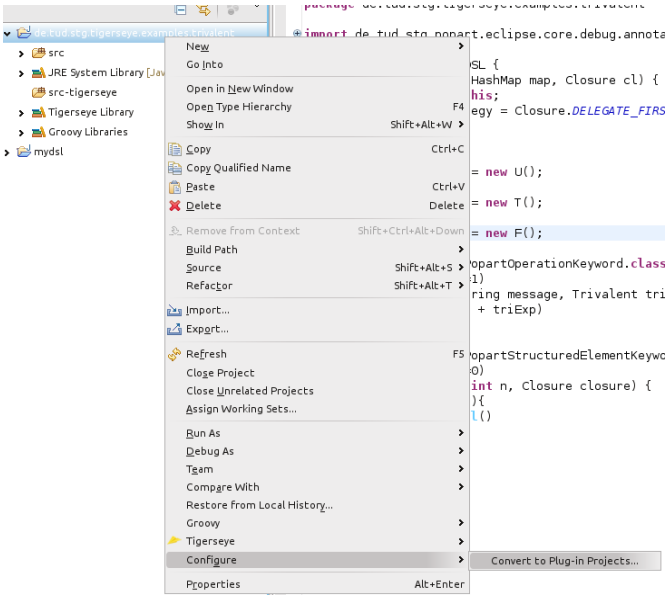

Figure 22: Add Plug-in Project Nature

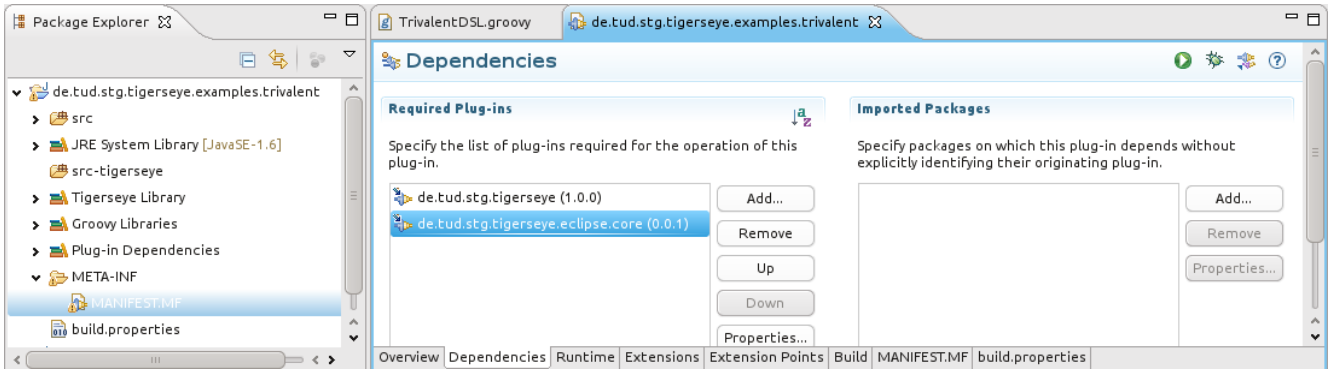

Figure 23: Add Plug-in Depedencies to TigersEye

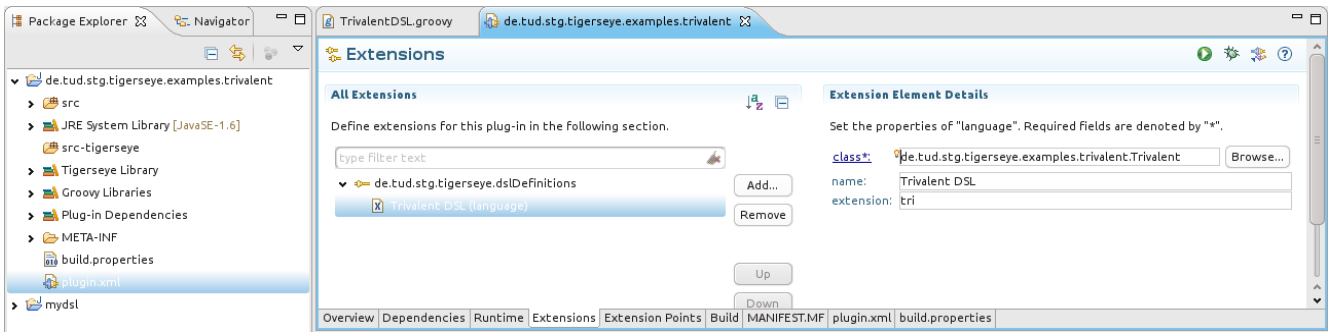

Figure 24: `dslDefinitions` Extension Point Configuration

List of Tables

List of Figures