ECCrawler
Visualizations for Eclipse

Informatikprojekt
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Manuel Friedli

Oktober 2010

Leiter der Arbeit
Prof. Dr. Oscar Nierstrasz
Adrian Kuhn

Institut für Informatik und angewandte Mathematik

Further information about this work may be found under the following addresses:

Manuel Friedli
manuel.friedli@students.unibe.ch
https://www.iam.unibe.ch/scg/svn_repos/Students/friedli

Software Composition Group
University of Bern
Institute of Computer Science and Applied Mathematics
Neubrückstrasse 10
CH-3012 Bern
http://scg.unibe.ch/

Abstract

Software visualization tools can help us to better understand the structure of a software system. Integrating a software visualization in an IDE like Eclipse allows us to see the visualization from within the IDE and thus makes switching between different programs unnecessary. ECCrawler is an interactive software visualization plug-in for Eclipse that serves as an alternative to the traditional Package Explorer by enabling mouse-interaction in order to navigate through a software project.

Chapter 1

Introduction

This paper presents ECCrawler, a new navigation view for the Eclipse IDE that uses software visualizations to present the project. The idea of using a system complexity view for navigation was taken from RBCrawler, a software visualization based navigation view for VisualWorks written by Adrian Kuhn[1].

ECCrawler aims to close the gap between software visualization and software editing. It is directly integrated in the IDE and allows us to navigate through the source code of a software project using an intuitive graphical representation. It displays a system-complexity view, i.e., the classes are represented as boxes, inheritance is marked with arrows from one box to another.

ECCrawler uses FAMIX, a language-independent meta-model that describes the static structure of object-oriented software systems[2], for analysis of the software, and ZEST, a graph-drawing framework, for drawing.

In Eclipse, ECCrawler is activated via a context-menu entry. It analyses the selected resource and displays the visualization. Selecting a class in the view will bring it up in the editor, thus allowing direct navigation.

As a plug-in for Eclipse it can easily be installed through the standard Eclipse plug-in manager.

The remainder of this work is structured as follows:

- section 2.1 presents related work.
- section 2.2 discusses the problem with current visualization tools and IDEs
- section 2.3 proposes a solution: ECCrawler
- chapter 3 presents the implementational details of ECCrawler

[1] `http://scg.unibe.ch/wiki/projects/rbcrawler`
[2] `http://www.moosetechnology.org/docs/famix`

- chapter 4 gives a short introduction to the functionality and how to use it
- Appendix A is an overview of the prerequisites needed to install and run ECCrawler
- Appendix B details the installation in a step-by-step installation guide

Chapter 2

Software Visualization

The goal of software visualization is to better understand a software system, to find defects or to analyze dependencies. Visualizations can come in many different flavours and can show information about a wide variety of metrics. Such metrics may include structure, size, history or dynamic behavior[1].

2.1 Related work

Depending on the task we can choose among many different tools that will visualize a method, a single class, a component or a whole software project.

Some tools intended for object-oriented languages and systems are:

- ArgoUML[2]: Create UML diagrams from Java source code.
- CrocoCosmos[3]: Analysis and visualization of structural and metrics information of large object-oriented programs for comprehension and quality assessment.
- Sotograph[4]: Draw various types of visualizations, e.g. visualize coupling between subsystems.
- CodeCrawler[5]: Language independent reverse engineering tool which combines metrics and software visualization [1].
- Structure101[6]: Visualize the structure of a software system.

[1] http://en.wikipedia.org/wiki/Software_visualization
[2] http://argouml.tigris.org/
[3] http://www-sst.informatik.tu-cottbus.de/CrocoCosmos/
[4] http://www.hello2morrow.com/products/sotograph/visualize
[5] http://www.inf.usi.ch/faculty/lanza/codecrawler.html
[6] http://www.headwaysoftware.com/products/structure101/index.php

An exhaustive list of software visualization tools can be found at this[7] URL.

2.2 Problem

The aforementioned tools are all quite powerful. They allow us to create informative views, nice UML diagrams or beautiful graphs. Yet most of them are quite complex. In some cases they may provide information which may be useful, but which is not of immediate interest to us. Also, they are stand-alone applications that perform a static analysis of the source code and produce a view of the state of the system of that point in time. What that means is that if we want information about a software project we're working on, we will have to leave our IDE, import the project in the visualization tool and generate the visualization. With the information we get from this we can turn back to the IDE and continue development and improvement.

The switch of applications interrupts the work flow. A first step to fix that would be to integrate the visualization in the IDE. That would make the switch between two independent applications unnecessary. Yet, a static, image-like visualization is not very appealing. That leads to the conclusion that the most important point is actually not displaying a visualization, but using it interactively for navigation through the source-code.

2.3 Solution

The immediate benefit of an interactive visualization is quite obvious: When we are able to quickly open with one click a class which we identified as being in need of refactoring, it is likely that we will use that feature much more often, given the fact that we are able to identify such classes in the first place. An interactive visualization is a comfortable means of identifying badly designed classes and directly starting work on them.

ECCrawler aims to be such an interactive visualization inside the IDE. It provides a System Complexity View and the navigation. As of now it lacks the ability of real-time updates. Still it can be of great assistance. Figure 2.1 shows a typical situation with the visualization view below the editor. We can see at one glance how classes are related to each other through inheritance. Also, the size of the boxes representing the number of methods and number of attributes gives us a rough impression of the size of the classes.

Implementing other types of views than System Complexity Views, e.g. method calls between classes, would be possible. See section 3.1.

[7] `http://ruthmalan.com/ArchitectureResourcesLinks/VisualizationInSoftware.htm`

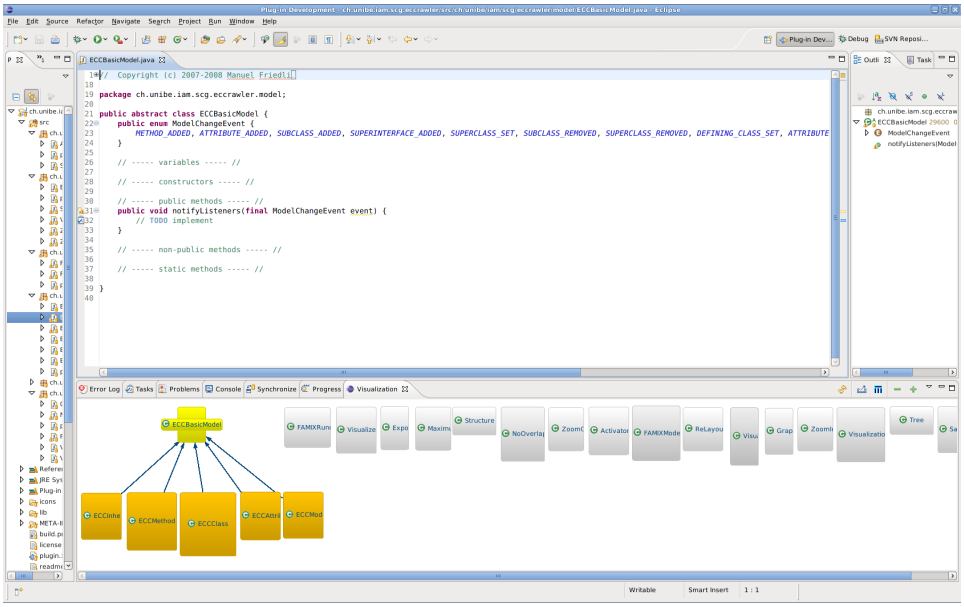


Figure 2.1: Screenshot of ECCrawler, showing a typical System Complexity View

Chapter 3

Implementation

ECCrawler is implemented as a plug-in for the Eclipse IDE. As such, it uses the standard means of obtaining input from the user and drawing on the screen that are given by Eclipse. Drawing the graphs is performed using ZEST, a powerful graph-drawing framework.

Figure 3.1 shows the architectural diagram of ECCrawler. Red arrows represent data flow, the blue arrows indicate the interaction between components that happens when the user selects to visualize an element.

As in any Eclipse plug-in, the `Activator` class serves as a hook for starting and stopping the plug-in, as well as obtaining a reference to the single instance of the plug-in that exists within an Eclipse environment. Work needed for the plug-in to behave correctly (e.g. registering a listener for changes on the source-files, a feature that is not yet implemented) can be done in the `start()` method, clean-up happens in the `stop()` method. Registering actions and menu entries in Eclipse is done automatically at startup by Eclipse itself, we just need to provide XML descriptions. For further information on writing plug-ins for the Eclipse IDE, refer to the Eclipse documentation[1] and read the chapter "Platform Plug-in Developer Guide".

The main action of analyzing a Java project (or parts thereof) is hooked up with an entry in the context menu of the package explorer view which is a standard procedure provided by eclipse. It triggers the `run()`-method in the `VisualizeAction` class (represented by the leftmost blue arrow in Figure 3.1), which implements the `IObjectActionDelegate` interface provided by Eclipse. The current selection is taken, and if it consists of a parsable Java element (either the whole project, a Java package or a single class; called `IJavaElement` from now on), it is handed over to the `FAMIXModel` for parsing (represented by the rightmost red arrow in Figure 3.1). After that, if no visualization view exists, one is created and displayed, otherwise the existing view gets updated with the parsed data and is then displayed.

[1] `http://help.eclipse.org/helios/index.jsp`

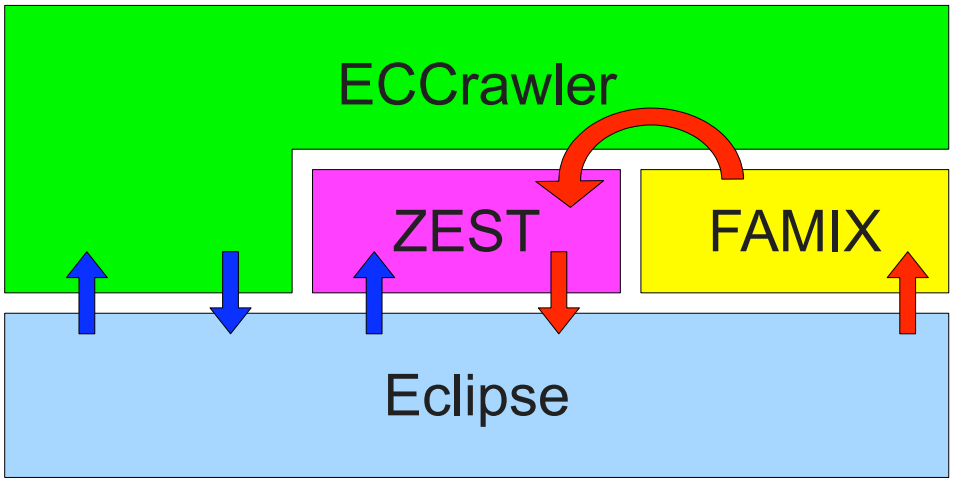


Figure 3.1: Architectural diagram showing the structure of ECCrawler

Parsing of an `IJavaElement` happens in the `FAMIXBuilder` provided by Sandro De Zanet. It takes an `IJavaElement` as input and returns a `Repository`, which is part of the FAMIX world and contains all relevant information, and even much more information that is not currently used by ECCrawler. In order to access that information, a conversion has to be made, so that the displaying part of the plug-in will understand and be able to handle it. The `Repository` gets transformed into the ECCrawler-model consisting of `ECCClasses`, `ECCInheritances`, `ECCMethods` etc. (represented by the semi-circular red arrow in Figure 3.1). All these objects are stored in `FAMIXModel` as a tree structure with the `ECCModelRoot` as the root.

The `VisualizationView` is responsible for displaying the System Complexity View. It extends the `ViewPart` class, which serves as the common base class for all Views in Eclipse. Upon creation, the `createPartControl()` method is invoked, which sets up a selection listener and the viewer that serves as the panel where the content is drawn. As this viewer, we use a `VisualizationTreeViewer` which extends `GraphViewer`, a class provided by ZEST[2], a powerful framework for drawing trees and graphs.

When creating or refreshing the visualization view, it gathers the `ECCModelRoot` from the `FAMIXModel` and hands it over to ZEST as the model root (represented by the leftmost red arrow in Figure 3.1). ZEST then performs the layout according to the chosen layout algorithm, in the case of ECCrawler a `TreeLayoutAlgorithm`, suitable for the purpose of drawing a System Complexity View.

[2] `http://www.eclipse.org/gef/zest/`

3.1 Other View Types

Since ECCrawler gathers its information from FAMIX, it could display all the information that FAMIX provides us with. As mentioned in the introduction, FAMIX describes the static structure of an object-oriented software system. It is therefore not possible to display e.g. the runtime behaviour of an application with ECCrawler without changing the whole architecture of the plug-in.

Another type of view would probably have a different underlying data model, so firstly, the `FAMIXModel` and the `FAMIXRunner` classes would have to be adapted to be able to deal with other data types than just inheritance.

Next, the `VisualizationView` would have to be changed in order to display graph types different from tree structured graphs. ZEST, the framework used for drawing the graph, supports various graph layouts. The following list is taken from [2]:

`TreeLayoutAlgorithm`: Graph is displayed in the form of a vertical tree

`HorizontalTreeLayoutAlgorithm`: Similar to TreeLayoutAlgorithm but layout is horizontal

`RadialLayoutAlgorithm`: Root is in the center, the other nodes are placed around this node

`GridLayoutAlgorithm`

`SpringLayoutAlgorithm`: Layout the graph so that all connections should have approx. the same length and that the edges overlap minimal

`HorizontalShift`: Moves overlapping nodes to the right

`CompositeLayoutAlgorithm`: Combines other layout algorithms, for example HorizontalShift can be the second layout algorithm to move nodes which were still overlapping if another algorithm is used

If another layout were desired, we could implement our own `LayoutAlgorithm` class.

Chapter 4

User Manual

The plugin adds a new entry named "Visualize element" to the context menu of the Eclipse Package Explorer. Selecting it generates the visualization and opens it in the visualization view. At the top there are icons for performing various actions, described hereafter:

 **Re-layout the graph:** Performs a fresh layout of the visualized classes. This can be handy if the view has gotten cluttered from moving lots of boxes around by hand or when the view has been resized.

 **Save the current visualization as a PNG file:** Exports the current visualization and saves it as a PNG file. A file dialog will open that lets you select the location of where you want to store the image.

 **Export current visualization as MSE file:** Exports the visualization as MSE file, ready for import in MOOSE[1].

 **Zoom out of the view:** Zooms out in order to get a better overview of the visualization.

 Zoom in on the view: Zooms in to show more details.

When the visualization view is active, the main Eclipse status bar will show information on the currently selected class (see Figure 4.1): firstly the class name, next the number of methods (NOM), then the number of attributes (NOA) and finally the lines of code (LOC).

A double click on a box will open the corresponding class in the editor. Should a class be openend in the editor from the Package Explorer, the corresponding box in the visualization view will be selected.

[1] `http://www.moosetechnology.org/`

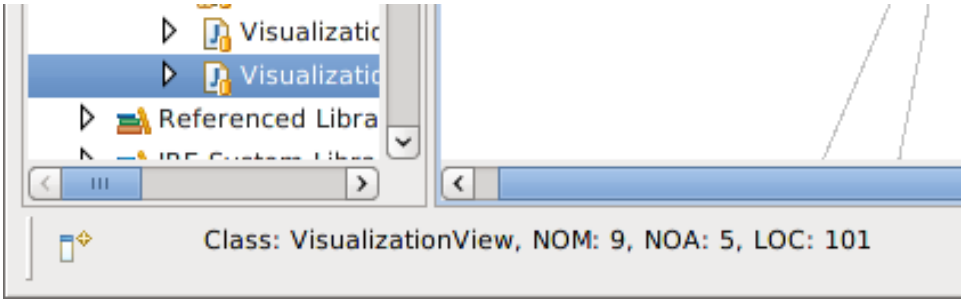


Figure 4.1: The toolbar shows the class name, the number of methods, the number of attributes and the number of lines of code of the selected class

As mentioned, boxes can be dragged around with the mouse. The selected class will be highlighted in yellow. Superclasses and subclasses will be highlighted in orange, with the connections in between highlighted in blue.

The width of a box represents the number of attributes, the height represents the number of methods. The color represents the lines of code: the darker the box, the more lines of code it has. A white box represents an empty class.

Chapter 5

Concluding Remarks

ECCrawler provides a visualization inside an IDE, thus closing the gap between two separate tools and merging them into one, for the developer's convenience. It allows us to navigate through a software project by selecting and opening classes, thus serving as an alternative to the traditional Package Explorer.

As mentioned before, no real-time updates take place, i.e. when a class is changed, added or deleted, the visualization view will not be updated automatically. This feature would make the tool complete. It has not been implemented due to the way ECCrawler is architectured: It is uses FAMIX for gathering information about classes and the project's structure, and FAMIX does not easiliy support partial updates.

Also, MSE export seems not to be working due to a bug in FAMIX; the export aborts with an error.

Appendix A

Installation Notes

In order to install and use ECCrawler, your must have the following software installed:

- Java Runtime Environment Version 6 or later
- Eclipse SDK 3.3 (Europa) or later

ECCrawler depends on the following Eclipse plug-ins. Some of them are part of the Eclipse core, others are optional plug-ins. However, any plug-in that is not installed already will be installed automatically by the Eclipse Plug-in Manager.

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.jdt.core
- org.eclipse.core.resources
- org.eclipse.rcp
- org.eclipse.draw2d
- org.eclipse.jface.text
- org.eclipse.jdt.ui
- org.eclipse.jst.jsp.core
- org.eclipse.jst.j2ee
- org.eclipse.wst.sse.core
- org.eclipse.wst.xml.core
- org.eclipse.jst.j2ee.core

- org.eclipse.gef

Download and install ECCrawler via Eclipse's built-in plug-in manager. Add the following new update site location: `https://www.iam.unibe.ch/scg/svn_repos/Students/friedli/eccrawler-update`

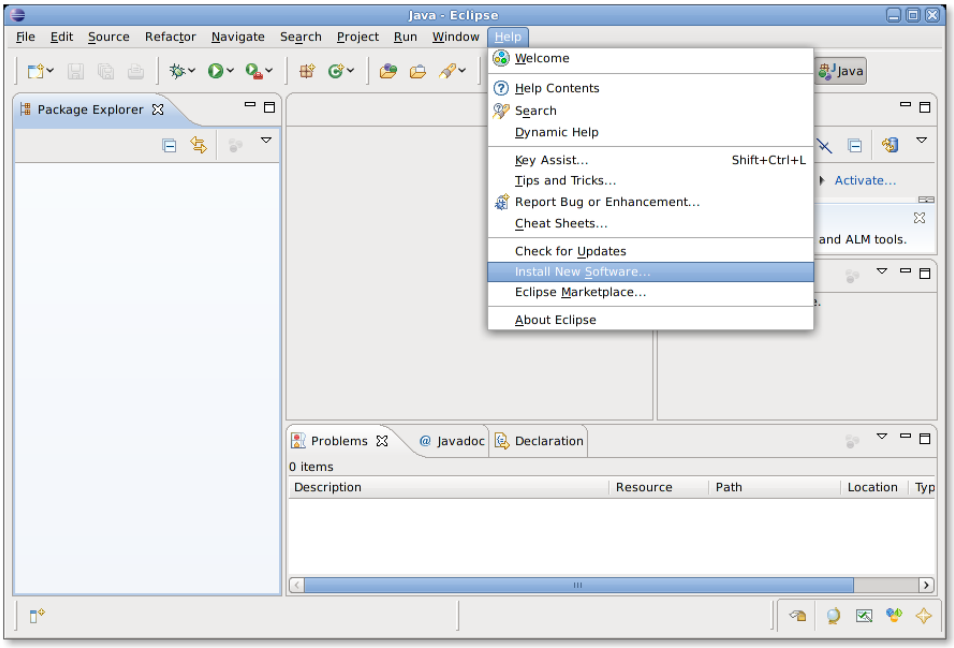
After restarting Eclipse, ECCrawler should now be available and usable through the context menu entry in the Package Explorer.

Appendix B

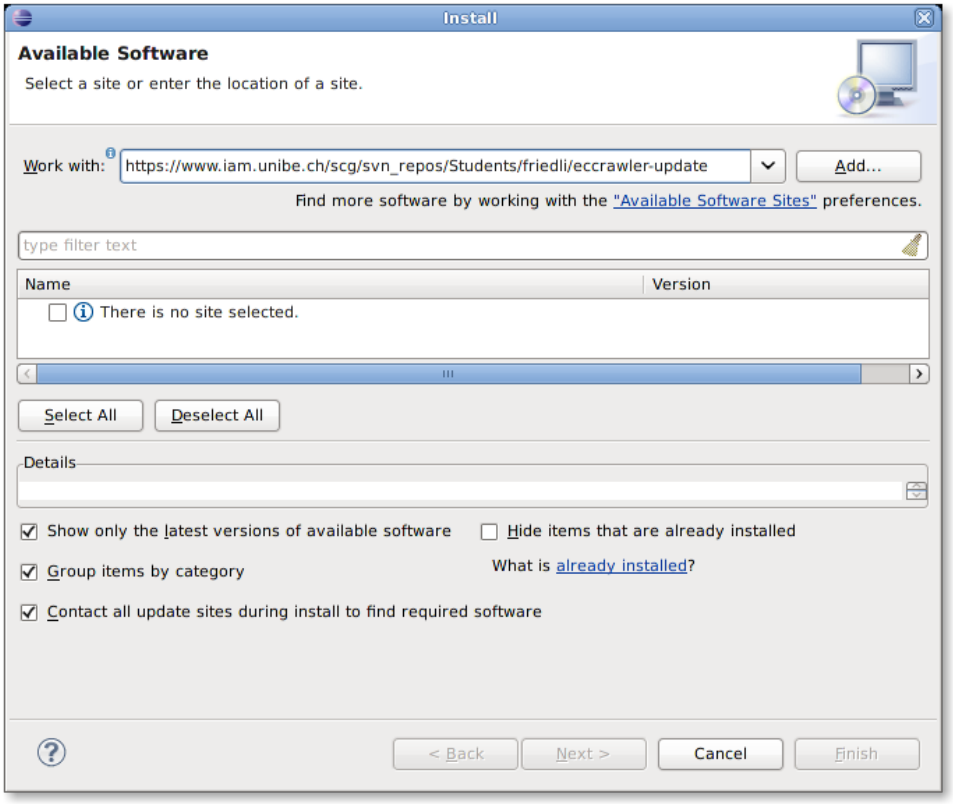
Step-by-step installation guide

If you are unfamiliar with Eclipse you may find this step-by-step installation guide useful. We will walk through the installation with a clean installation of Eclipse 3.6 (Helios).

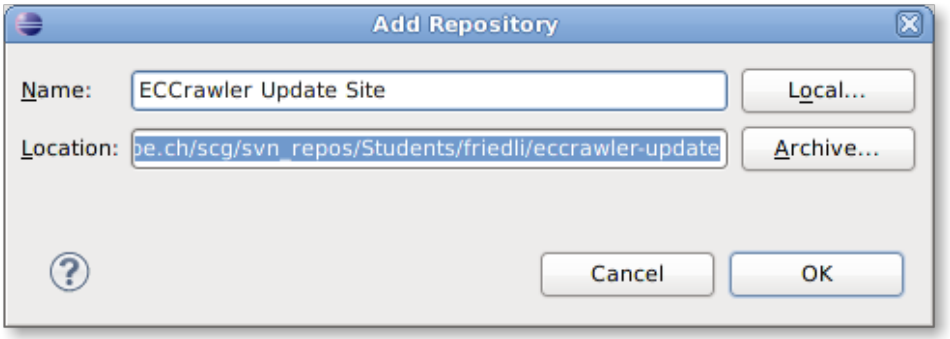
From the main Eclipse window, open the "Help" menu and select "Install New Software...":



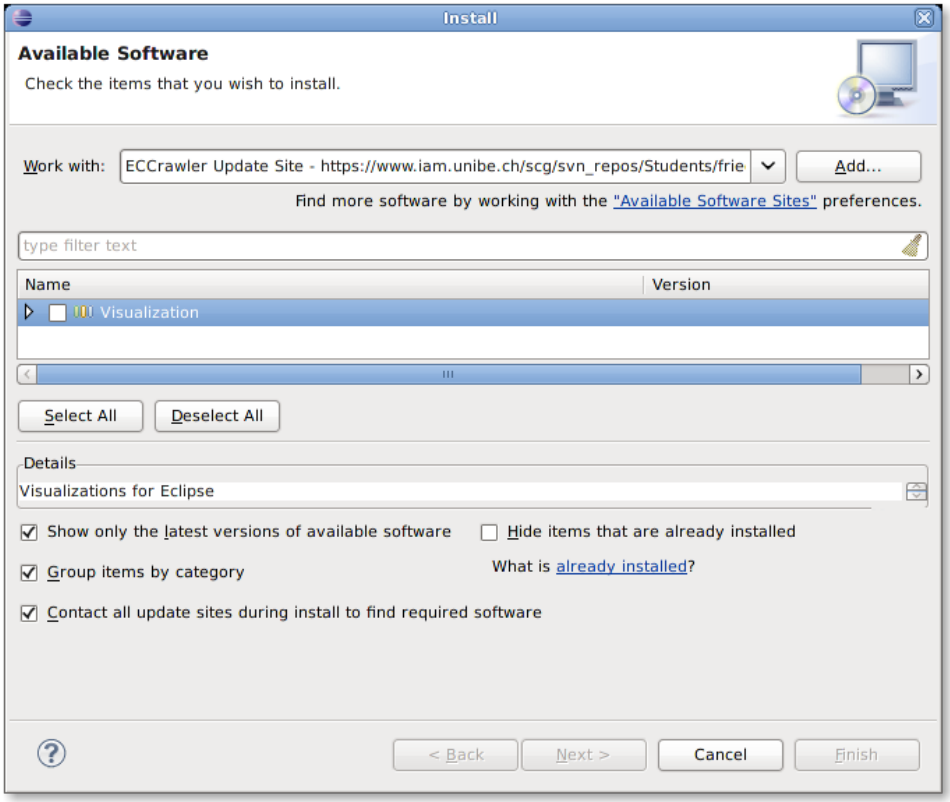
In the dialog that opens, enter the URL of the ECCrawler update site (`https://www.iam.unibe.ch/scg/svn_repos/Students/friedli/eccrawler-update`) and click "Add...":



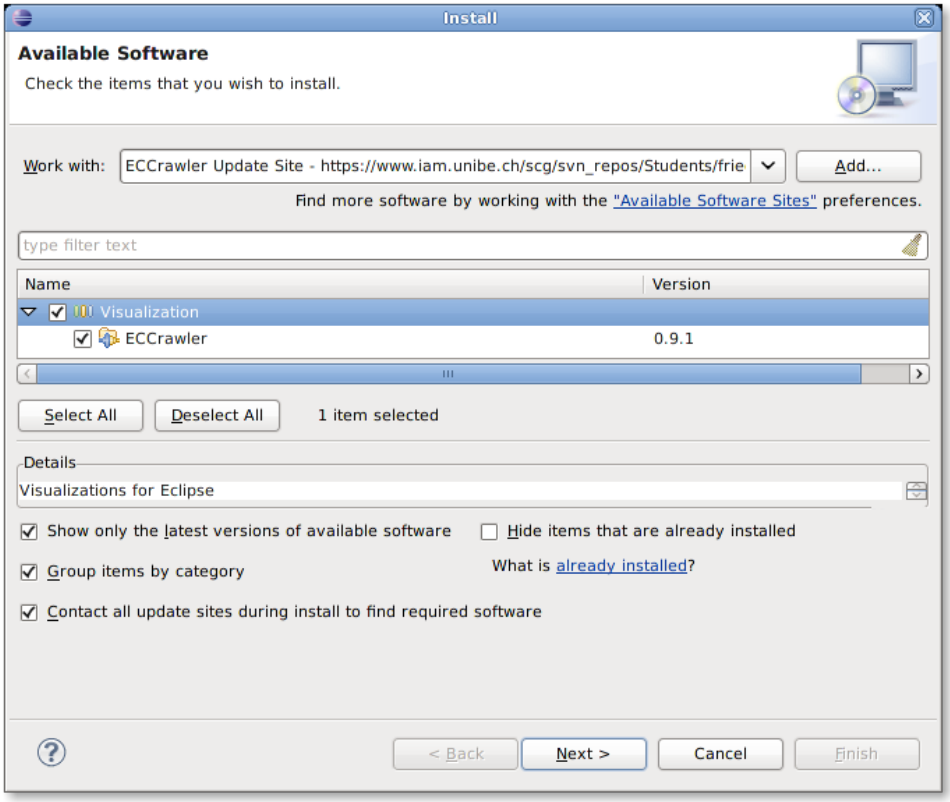
In the dialog that opens, enter a name for the update site, e.g. "ECCrawler Update Site" and click "OK":



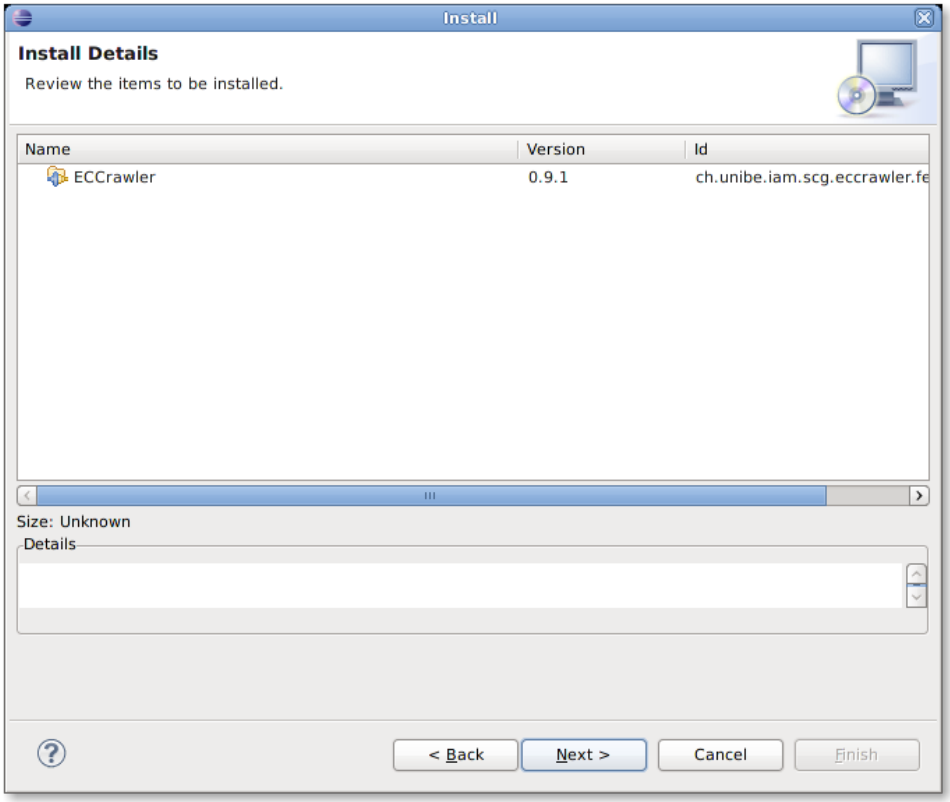
Eclipse will now gather all information about the new update site. This may take a while. Once it has finished downloading the meta information, you will see a new item called “Visualization” in the software installation dialog:



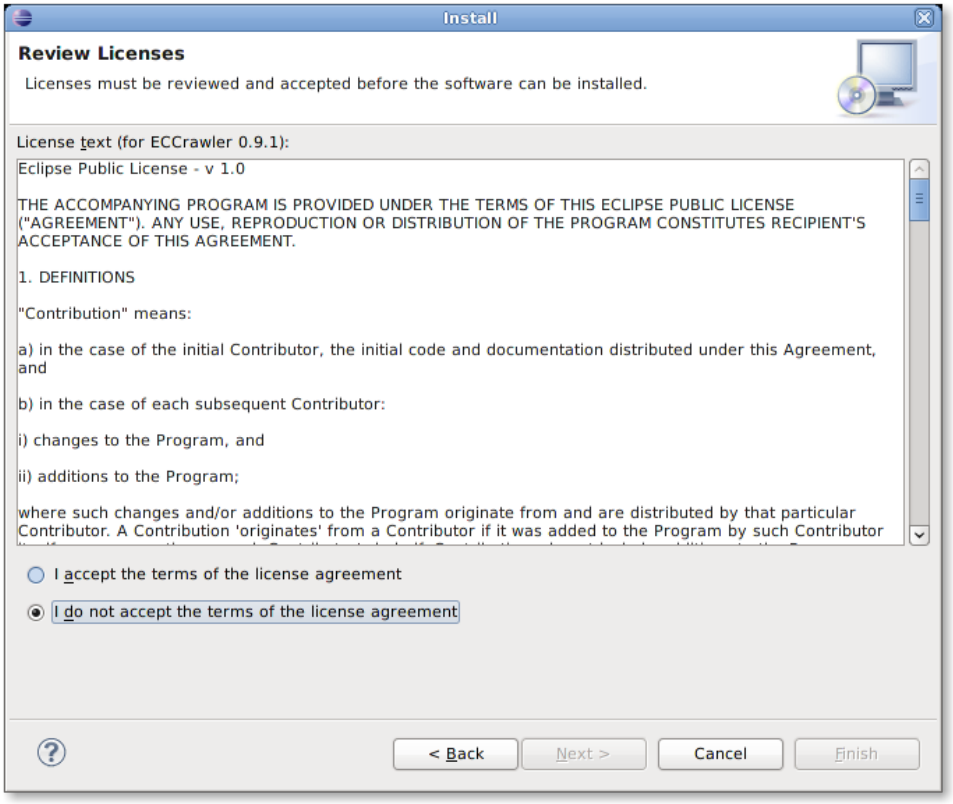
Select the item's check box and click "Next >":



In the “Install Details” dialog, click “Next >”:



In the “Review Licenses” dialog, select “I accept the terms of the license agreement”. Then click “Finish”.



When prompted with a security warning, click “OK”:

After the installation process has finished, you need to restart Eclipse in order to use ECCrawler. Do so by selecting "Restart Now":

Bibliography

[1] Michele Lanza and Stéphane Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, September 2003.

[2] Lars Vogel. Eclipse zest — tutorial. `http://www.vogella.de/articles/EclipseZest/article.html#layoutmanager`.