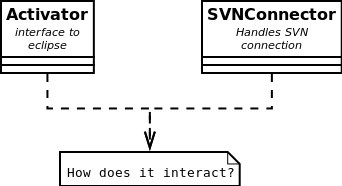
Speculate about design

We need to modify the current system to support multiple source control systems (SCM's). Our previous analysis found the current SVN functionality to be in the distiller package. As such we will speculate about its design.

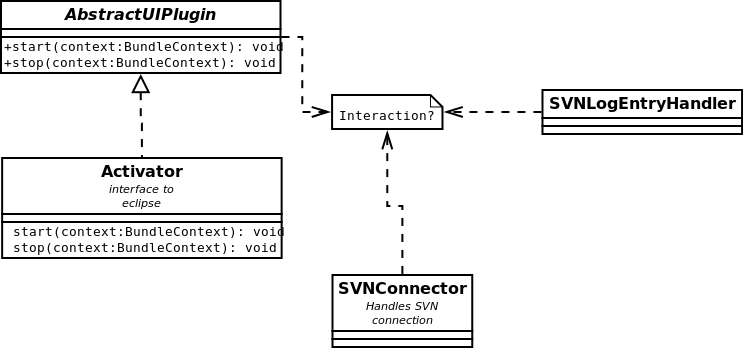
From our previous analysis we can derive the following diagram:



We now there is an activator that is an interface to eclipse, and an svnconnector that handles the svn connection. We do not know how those two entities interact with each other. As the Activator is an interface to eclipse (it extends AbstractUIPlugin) it should have at least a start(...) and stop(...) method.

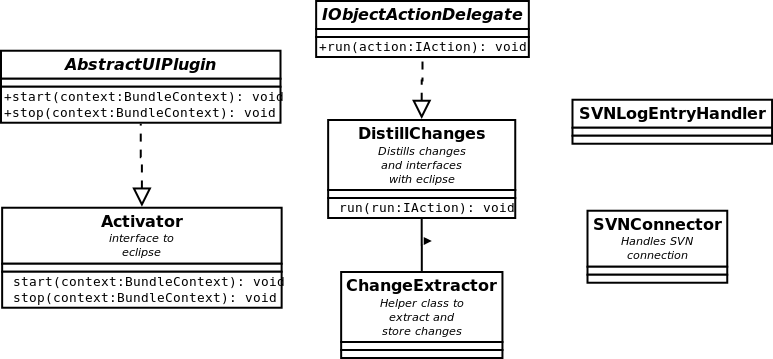
The svnconnector somehow extracts changes from an svn repository it should interface somehow with the changerecorders package and the model package to display the results. From viewing the files in the package there is also a SVNLogEntryHandler wich as the name suggests probably handles one entry.

From these observations we can construct a second diagram:

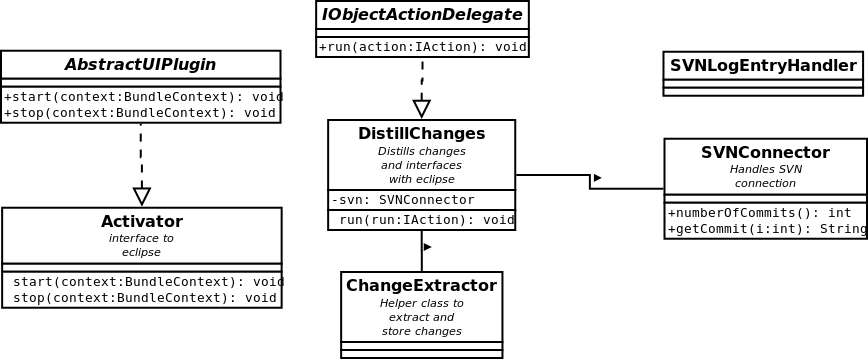


We still don't know the major interactions between these components. If we look to the actual implementation of the Activator class it quickly becomes clear that it does not interact with other components. This means the interaction between the SVNConnector and eclipse itself is via another way. In our previous analysis there was a subpackage popup.actions that implemented some interfaces from eclipse.

From the name of the popup.actions package it seems that some actions are provided that interface with eclipse. One action could be to distill all changes. This could be presented by a DistillChanges class that interfaces with the DistillChanges package. Furthermore because there are multiple actions available a common helper class could be present to abstract some common code. This helper could have some methods to help to store and extract additions and removals. A quick inspection of the source code reveals there is indeed such a class named ChangeExtractor. This class is used by the DistillChanges class that interfaces with eclipse. Upon inspecting DistillChanges we found that it implements IObjectActionDelegate from the eclipse SDK. The main entrypoint for eclipse is public void run(IAction action). This gives us the following class diagram:



We still don't know how the DistillChanges interacts with the SVNConnector. And how the SVNConnector interacts with the SVNLogEntryHandler. The DistillChanges probably has a field with type SVNConnector. The SVNConnector probably has some methods to get each commit, CheopsJ wants to record all changes so each commit will have to be processed for addidions/removals. This gives us the following diagram:

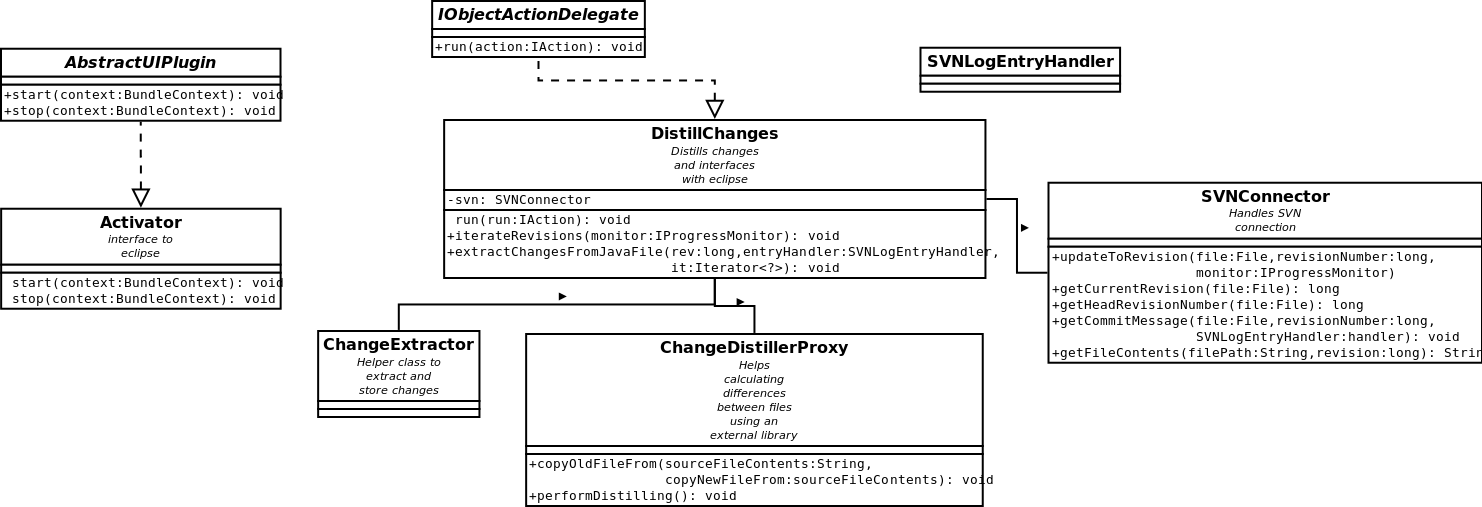


Upon inspection of the SVNConnector we did not find those exact methods as defined in the paragraph. Most of the methods we found operator on files with a given revision number. It seems that the SVN library needs a file and a revision number to get the actual contents of that file at that revision. There are methods available for getting the current revision of a file, the newest (head) revision of a file. There are also methods to update a file to a given revision. And to get the file contents of a file given a revision number. There is also a method to get the commit message of a revision.

The distill changes will probably have a method to iterate over all the revisions and extract the changes. This introduces an additional problem, the DistillChanges does not operate on files and it seems to only help to store changes, not to actually calculate them. This functionality seems to be provided elsewhere. Upon inspection of the package we found a ChangeDistillerProxy class that is used by DistillChanges to operate on files. The ChangeDistillerProxy uses an external library to calculate the actual changes. The two revision are copied to a file and the library is used to find the modifications.

The methods in DistillChanges to iterate the revisions are split up in several methods. First the iterateRevision() that oversees the whole process. It calculates the amount of revisions. It seems to always take into account all the revisions (so it always starts from scratch). In SVN the revision numbers are incremented by one each revision, so it is easy to iterate over all the revision numbers. In git however hashes are used and some methods will have to be supplied to get the next revision and the amount of revisions. The method extractChangesFromJavaFiles() extract the changes from the files. Using the proxy where necessary.

This gives us the following diagram:



There is still one important class in the popup.actions package that we haven't looked at. This is the DistillAddidionsFromCurrentVersion class. This gets all additions from the current version. It does not use the SVNConnector, just the local files.

From our analysis it is clear that DistillChanges is the main component that uses the SVNConnector and the SVNLogEntryHandler. DistillChanges uses the SVNConnector to iterate over all revisions, it does this by creating files for a specific revision and comparing it against the previous one. It then looks for additions, removals or modifications. It does this via the SVNLogEntryHandler. Most version control systems indiciate if a file was added, removed or modified. In the first two cases no differences must be computed. When a file is modified however it must. To calculate the differences it uses ChangeDistillerProxy. The proxy uses an external library to calculate the differences.

All the changes are stored using the ChangeExtractor wich uses the ChangeRecorder package to store differences.

To introduce support for multiple scm's an interface that offers similar functionality to SVNConnector and SVNLogEntryHandler will have to be made. This will require some substantial changes in the DistillChanges packages. For example, it assumes the revisionNumbers are numerical, but git uses hashes. So additional methods and modifications will have to be made to accommodate this.