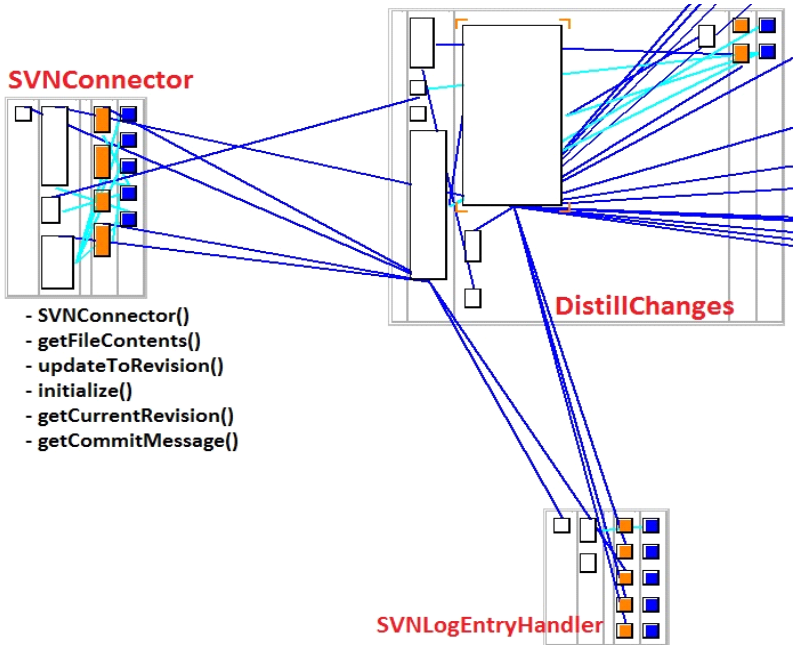
RFC: Refactory plan for ChEOPSJ

Introduction

After looking at the class blueprints and their interactions in codecity, we learned that the classes SVNConnector and SVNLogEntryHandler are only being used by the class DistillChanges (more precisely only in its methods: ExtractChangesFromJavaFiles(), iterateRevisions() and updateOneRev(). We also noticed that both SVN related classes don't really interact with eachother!



We can see that there are 11 functions being used (6 of the SVNConnector class) by the DistillChanges class, these will be provided in common interfaces.

For making our refactoring succesful and preventing risks, we won't use a big bang integration, we will first focus on refactoring the SVNConnector and afterwards refactor the SVNLogEntryHandler. It's very important to work in schematic well ordered steps!

Aspects

- RepositoryConnector Interface with shared functions that are currently in the SVNConnector Class
- For each kind of repository (Git, SVN, ...) a connector class (SVNConnector, GITConnector, ..) which implements the RepositoryConnector Interface.
- Factory pattern to create the connector of choice
- Singleton pattern, because there can be only one factory object, SVNConnector object, GITConnector object, ...
- The use of a property file instead of hardcoded configurations.

  Possible ways to handle this: ordered from worst to best:

  1) Hardcoded Configs (currently)
  2) Config XML/property file
  3) Own Plugin with preferences (however this goes out of the scope of our time frame and project description)

Package Structure

The package 'svnconnection' will be renamed to connections.

```
-- be.ac.ua.ansymo.cheopsj.distiller.connections
       ---- ConnectionFactory.java (factory)
       ---- RepositoryConnector.java (interface)
       ---- impl (package)

-- be.ac.ua.ansymo.cheopsj.distiller.connections.impl
       ---- SVNConnector.java
       ---- GITConnector.java
```

Properties file

Because the location of a file is rather hard to find at runtime when using plugins, the location to the file can be passed to the factory, this way the factory can read what kind of connection it should return, it will be passed furtheron to the Connector classes itself, so those can read the url, username and passwords out of it!

The structure would be something like:

```
type_in_use = SVN
SVN_username =  abc
SVN_password = def
SVN_url = http:...

GIT_username = xyz
GIT_password = klm
GIT_url = http:...
```

RepositoryConnector <interface>

```
interface RepositoryConnector{
       void initialize();
       void updateToRevision(File, long, IProgressMonitor);
       long getCurrentRevision(File);
       void getCommitMessage(File , long , SVNLogEntryHandler);
       String getFileContents(String, long );

}
```

SVNConnector & GITConnector <class>

```
class SVNConnector implements RepositoryConnector{
       SVNConnecter();
       //implementation of the 5 functions of the interface
       RepositoryConnector getConnector(File configFile);
       //will be explained below

}

class GITConnector implements RepositoryConnector{
       GITConnecter();
       //implementation of the 5 functions of the interface
       RepositoryConnector getConnector(File configFile);
       //will be explained below

}
```

Possible improvement for the future

In case we notice that there is a lot of shared common functionality between all Connector classes, we could add an abstract connector class which get's extended by the SVNConnector & GITConnector, this way common private functions can be used. However our first goal is making it work, then make it better!

RepositoryFactory

The factory will be provide the function:

```
RepositoryConnector RepositoryFactory.getConnector(File configFile);
```

which returns an object that implements the interface 'RepositoryConnector'

Singleton

The repositoryFactory (as well as the Connectors itself) are singletons, therefore only one object can be created out of it. This is done as follows:

```
public class RepositoryFactory{

	static private RepositoryFactory singleton;

	private RepositoryFactory();
	//this is made private so other classes can't create a new object by just
calling new RepositoryFactory();

	private static RepositoryFactory getFactory(){
		if(singleton == null){
			singleton = new RepositoryFactory();
		}
		return singleton;
	}

	public static RepositoryConnector getConnector(File configFile){
		RepositoryFactory factory = RepositoryFactory.getFactory();
		RepositoryConnection RC = factory.instantiateConnection(File
configFile);
		return RC;
	}

	private RepositoryConnection instantiateConnection(File configFile){
		String type = fetchFromPropertiesFile(File,type_in_use);

		if(type.equalsIgnoreCase("svn"){
			return SVNConnector.getConnector(File configFile);
		}else if(type.equalsIgnoreCase("git"){
			return GITConnector.getConnector(File configFile);
		}
	}

}
```

SVNConnector's getConnector function (same for GITConnector)

```
public static RepositoryConnector getConnector(File configFile){
	if(singleton == null){
		String username = fetchFromPropertiesFile(File,SVN_username);
		String password = fetchFromPropertiesFile(File,SVN_password);
		singleton = new SVNConnector(username,password);
```

```
        }
        return singleton;
}
```

Singleton

Like said above, the Connector classes itself are singletons too, therefor they should also have:

```
static private SVNConnector singleton;
private SVNConnector(username,password);
```

What's next?

Well the next step is doing almost the same process but then for the SVNLogEntryHandler.