Abstract
The Groovy Plug-in for the Eclipse IDE features a number of automated refactorings, that were realized in a bachelor thesis at HSR University of Applied Sciences, Rapperswil, Switzerland. Although Groovy and Java code can be used mutually in projects, the already implemented refactorings are limited to Groovy code.

In a subsequent bachelor thesis the Groovy Eclipse Plug-in was extended with Cross-language Refactorings by the first two authors: If either a Groovy or a Java element is renamed, both languages are respected throughout the complete refactoring process. The achieved goal was to support all possible rename actions, no matter in which language the element to be refactored was defined.

As a result, the Groovy Eclipse Plug-in now offers a higher level of integration with the Java Development Tools and more productivity for Groovy developers.

Due to automated testing over complete development process, the solution is ready for production use and will be submitted to the official plug-in maintainers.

Categories and Subject Descriptors D[2]: 6—Programming Environments

General Terms Design, Languages, Experimentation

Keywords Refactoring, Eclipse, Groovy, Java, Cross-language Refactoring

1. Introduction
The goal for every good software developer should be, to write not only functional code, it should also be readable. To achieve this, it is required to change and improve the code a few times during development. It is important, that this task, called refactoring - changing the structure without changing its functionality - does not need too much time from the developer. So nowadays most of the Integrated Development Environments (IDEs) have automated features for the most common refactorings included.

Groovy is a modern, dynamically typed programming language, based on the Java Virtual Machine (JVM). It is closely related to Java and projects can be mixed with both, Groovy and Java code. Given Java components can be accessed in Groovy and additional functionality can be added. Or, of course the other way around. For a Java developer, it’s really easy to start programming Groovy and he can use his already in Java written classes.

In a former Bachelor thesis, a team from our university added the refactoring functionality to the Groovy plug-in in Eclipse under Prof. Sommerlad’s supervision. Unfortunately this plug-in was just able to alter Groovy code. In a mixed project this is not really handy for a Groovy developer.

2. Approach
We (S. Sidler and S. Reinhard) are both Eclipse users since a long time, but none of us ever saw behind the scenes. So first we both had to dig ourselves into the Eclipse framework and into the already existing Groovy plug-in. In the Bachelor thesis of our predecessors [KKK08] the new functionality was theoretically analyzed. Based on this, we split the cross-language feature into four different scenarios.

- Local Java refactoring: refactor a Java element, started from the Java editor
- Remote Java refactoring: refactor a Java element, started from the Groovy editor
Local Groovy refactoring: refactor a Groovy element, started from the Groovy editor

Remote Groovy refactoring: refactor a Groovy element, started from the Java editor

For the last two scenarios, there were some unsolved problems on how to detect a Groovy element in Java, and rename Groovy elements in Java. We started with the other two scenarios, and tried find a solution to solve these problems in the meantime. This plan worked quite well, but it took us much more time to understand the framework as expected.

3. Results

At the end of this Bachelor thesis, we were able to expand the current plug-in with all used functions to rename a Groovy or Java element in both languages. Our private goal, to push the Groovy language is achieved as well, and we look forward to integrate our code into the official Groovy-Eclipse Plug-in. Hopefully, this will help a lot of Groovy developers, to improve their code.

During this thesis we came more common with dynamically typed languages, as Groovy. We learned the power it offers, as well as the threats it opens.

Both of us never worked in such a big Framework, as Eclipse before. It was a good experience, and showed us once more, how important it is to write easy readable code.

4. Outlook

At the end of the Bachelor thesis, the first two authors reached a level where the code can be integrated into the official Groovy-Eclipse repository. When this task is done, all Groovy developers can benefit from our work, and easily rename their elements. But even with this improvement, there are still a lot of open issues in the plug-in itself which have to be closed.

At the moment, the plug-in is not capable of working with dependencies between different projects. So as our refactorings. If this will change, the refactorings will have to be extended too.

At the IFS Institute for Software refactoring extensions for Eclipse and a variety of programming languages have been developed over the course of the last years. We support C++ refactoring in the Eclipse CDT project and plan to extend its C++ refactoring support further, especially with higher-level source file reorganization. At the time of the workshop work should be underway to provide C++ support for the upcoming C++ standard as well.

Acknowledgments

Many thanks to all students and assistants who contributed to IFS’ refactoring tool support.

References