Coccinelle4J: Automated Program Transformations for Java

Hong Jin Kang
School of Information Systems, Singapore Management University, Singapore
hjkang.2018@phdis.smu.edu.sg

Ferdian Thung
School of Information Systems, Singapore Management University, Singapore
ferdiant.2013@phdis.smu.edu.sg

Julia Lawall
Sorbonne Université/Inria/LIP6, France
Julia.Lawall@lip6.fr,

Gilles Muller
Sorbonne Université/Inria/LIP6, France
Gilles.Muller@lip6.fr

Lingxiao Jiang
School of Information Systems, Singapore Management University, Singapore
lxjiang@smu.edu.sg

David Lo
School of Information Systems, Singapore Management University, Singapore
davidlo@smu.edu.sg

Abstract

The program transformation tool Coccinelle is designed for making changes that is required in many locations within a software project. It has been shown to be useful for C code and has been been adopted for use in the Linux kernel by many developers. Over 6000 commits mentioning the use of Coccinelle have been made in the Linux kernel.

Our artifact, Coccinelle4J, is an extension to Coccinelle in order for it to apply program transformations to Java source code. This artifact accompanies our experience report "Semantic Patches for Java Program Transformation", in which we show a case study of applying code transformations to upgrade usage of deprecated Android API methods to replacement API methods.

2012 ACM Subject Classification Software and its engineering → Software notations and tools

Keywords and phrases Java, semantic patches, automatic program transformation,

Digital Object Identifier 10.4230/DARTS.VOL.ISS.ART

1 Scope

In this document, instructions to set up Coccinelle4J are provided. Furthermore, we provide a selection of semantic patches that can be applied by Coccinelle4J to source code extracted from real-world Java projects. These semantic patches are written in SmPL, a scripting language provided by Coccinelle.

2 Content

The artifact package includes:

- a Dockerfile to build the Docker image `coccinelle4j/coccinelle4j`
- this document that provides instructions on how to run Coccinelle4J

- Coccinelle4J's source code
- The examples described in the experience report. For each example, we include
  - semantic patch specified in SmPL
  - some .java source files extracted from real-world Java projects
  - output of each semantic patch after applying it with Coccinelle4J

3 Getting the artifact

There are two methods to set up Coccinelle4J. To minimize setup problems, it is preferable to use the first method using Docker.

3.1 Docker

A Docker image is similar to a virtual machine image, simplifying the set up of a project's environment. However, unlike a virtual machine, Docker containers are lightweight, sharing the operating system's kernel with the host machine.

We use Docker to run Coccinelle4J in a container so that the dependencies of Coccinelle4J can be installed in an environment isolated from the rest of the machine. We provide a Docker image `coccinelle4j/coccinelle4j` to easily set up containers that already have Coccinelle4J installed. This image also contains the examples described in the experience report.

The instructions to install Docker varies between operating systems and can be found on the official Docker document at `https://docs.docker.com/install/overview/`.

With Docker installed, the following commands can be executed to create a container based on our Docker image. We have uploaded the image at DockerHub and Docker will automatically fetch the coccinelle4j image from DockerHub. This image is approximately 2.44GB.

```
docker pull coccinelle4j/coccinelle4j
docker run -it coccinelle4j/coccinelle4j /bin/bash
```

The command will start a new container of the coccinelle4j image and run `bash` on it. On some machines, executing the above commands as root may be required. Next, the instructions to run the examples are in Section 3.3.

3.2 Make

If Docker is unavailable, an alternative to set up Coccinelle4J is to build the Coccinelle4J executable using make. OCaml (with a version >4.04), git, autoconf, make should be installed first.

```
git clone https://github.com/kanghj/coccinelle
cd coccinelle
git checkout java
./autogen && ./configure
make && sudo make install
```

3.3 Instructions for running examples

In our paper, we provided 7 examples of deprecated Android API methods, and show semantic patches that can be used to migrate them to their corresponding replacement API methods. We provide these examples both in the docker image and in the git repository hosted at `https://github.com/kanghj/coccinelle`.

Each example is contained in sub-directory in the `ecoop_example_patches` directory. Every sub-directory consists of the semantic patch (a .cocci file) and examples of source code (.java files) that the semantic patch will be matched on. Some sub-directories may contain a file with isomorphisms that we extracted from the Java projects the source files were taken from. More details about each example are described in the experience report.

To apply the semantic patches on the source file, the `spatch` command can be executed within each sub-directory. For the first example of replacing `sendStickyBroadcasts`, corresponding to Listing 11 in the experience report, the following command can be run.

```
cd ecoop_example_patches/sticky_broadcasts
spatch --sp-file sticky_broadcasts.cocci FileDownloader.java
```

The first example can be found in the `sticky_broadcasts` directory (under `ecoop_example_patches/`). `spatch` takes a semantic patch (in this case, sticky_broadcasts.cocci, specified with the `-sp-file` argument) and source files (in this case, FileDownloader.java) as input. By default, the patch generated by Coccinelle4J is printed to standard output. If it is desirable for Coccinelle4J to modify the source file directly, the `-in_place` flag can be passed to `spatch`.

For the example of `sticky_broadcasts.cocci`, Coccinelle4J produces the following output (printed to standard output) after the command above is executed.

```
HANDLING: FileDownloader.java
 diff =
--- FileDownloader.java
+++ /tmp/cocci-output-4196-ae14a8-FileDownloader.java
@@ -677,7 +677,7 @@ public class FileDownloader extends Serv
                 end.putExtra(EXTRA_LINKED_TO_PATH, unlinkedFromRemotePath);
             }
             end.setPackage(getPackageName());
-            sendStickyBroadcast(end);
+            sendBroadcast(end);
         }


@@ -695,7 +695,7 @@ public class FileDownloader extends Serv
             added.putExtra(EXTRA_FILE_PATH, download.getSavePath());
             added.putExtra(EXTRA_LINKED_TO_PATH, linkedToRemotePath);
             added.setPackage(getPackageName());
-            sendStickyBroadcast(added);
+            sendBroadcast(added);
         }

         /**
```

The instructions to apply the semantic patches for the other examples are as follows. From the set_text_size directory, run the following command. This applies the semantic patch described in Listing 13 in the experience report. The `-iso` option passes in a file containing isomorphisms for the project.

```
spatch --sp-file set_text_size.cocci \
--iso lucid_browser.iso CustomWebView.java
```

From the get_color directory, run the following command. This applies the semantic patch described in Listing 17 in the experience report.

```
spatch --sp-file get_color.cocci PushNotifications.java
```

From the should_vibrate directory, run the following command. This applies the semantic patch described in Listing 19 in the experience report.

```
spatch --sp-file should_vibrate.cocci IncomingRinger.java
```

From the get_height directory, run the following command. This applies the semantic patch described in Listing 22 in the experience report.

```
spatch --sp-file get_height.cocci TouchUtils.java
```

From the on_console_message directory, run the following command. This applies the semantic patch described in Listing 25 in the experience report.

```
spatch --sp-file on_console_message.cocci ViewFileFragment.java
```

From the get_drawable directory, run the following command. This applies the semantic patch described in Listing 26 in the experience report.

```
spatch --sp-file get_drawable.cocci \
DisplayUtils.java SimpleListItemDividerDecoration.java
```

In total, 7 examples are provided. A script `run_examples.sh` that runs all the examples is included in the `ecoop_example_patches/` directory. Instead of manually running each example one by one, executing `run_examples.sh` will apply all the semantic patches mentioned above on the examples and write them into `output.patch` in each directory.

3.4 Running Coccinelle4J on an entire project

We provide examples to run Coccinelle4J on entire projects. Each directory contains a script (download_project.sh) to download a project that a patch can be run on, and checks out the version of the project used in our experience report.

Run the following command in the root of the ecoop_example_patches directory to clone the projects. This command will clone the relevant projects into the directories containing the example patches.

```
./download_all_projects.sh
```

Run the following command in the root of the ecoop_example_patches directory to run the semantic patches on the projects. This produces a `project.patch` file in every directory under ecoop_example_patches, excluding `get_drawable`. We omit running `get_drawable.cocci` since it was used an example only to demonstrate the limitations of Coccinelle4J in our experience report. The `project.patch` files contains all the additions and deletions Coccinelle4J generated for each project.

```
./run_examples_on_entire_projects.sh
```

To count the number of lines in a semantic path, `grep` is used. For example, the following command can be executed (in the sticky_broadcasts directory) to count the number of non-empty lines in the sticky_broadcasts.cocci.

```
grep -cve "^\s*$" sticky_broadcasts.cocci
```

To count the number of additions and deletions in the diff generated by Coccinelle4J, the following command can be executed. The command looks for lines starting with "-" or "+" in the diff. This command should be run in any of the directories (e.g. sticky_broadcasts)

```
grep -ce "^[-+]\s" project.patch
```

A similar command can be used to count the number of files modified by Coccinelle4J.

```
grep -ce "diff" project.patch
```

4 Tested platforms

In general, Coccinelle4J is supported on any Unix-like platform. The Docker image we have provided should work on any platform supporting Docker.

5 License

The artifact is available under GNU GPL version 2.