CS/SE 6301 Software Analysis and Comprehension
Spring 2016

FINAL REPORT

Common coupling detection in Java: Eclipse JDT/AST and srcML approach

Ronaldo Goncalves Junior
Sungsoo Ahn
Bennilyn Quek

May 03, 2016

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.1	Ronaldo	Draft Report	5/1/2016
0.2	Sung Soo, Bennilyn	Preliminary Definitions (AST/SrcML)	5/2/2016
1.0	Ronaldo, Sung Soo, Bennilyn	Final Report	5/3/2016

Table of Contents

1 Introduction

This document is final report for the project, Common Coupling Detection in Java. We present final results for the project goal, which is to detect common coupling in Java source codes, calculate top 10 most used coupling degree, and provide detailed common coupling information to developers and maintainers for high quality system.

1.1 Common coupling

Common coupling (also known as global coupling) occurs when two modules are associated with the same global data (e.g., usage of a global variable) [1]. In other words, any changes to this data could potentially imply changes to all the modules that uses it. One additional aspect worth assessing is the set of constraints that might limit how java files (for instance) could be coupled. According to [4], there are two distinct categories of common coupling: strong and weak. Strong common coupling occurs when two or more than two modules access the global variable from any files of the system. The last one, weak common coupling, occurs when the global variable is accessed within files of the same package.

1.2 Java context

Some languages, for example Java, do not have global variables [2]. In Java, all variables that are not declared as local variables are declared as fields of a class. That is, all variables are declared within the scope of either a class or a method. In Java, static fields (also known as class variables) exist independently of any instances of the class and one copy is shared among all instances; hence public static fields are used for many of the same purposes as global variables in other languages because of their similar "sharing" behavior [2].

1.3 Other types of coupling

Coupling is not limited to shared global data (i.e., common coupling). There are five other types of coupling [3]: data coupling, stamp coupling, control coupling, external coupling, content coupling. These types of coupling refer to other categories of source code connection, which include parameters, data structure, logic control, and so on and so forth. However, none of these coupling types are considered in this project.

2 Systems

For the final report contained in this document, two systems are considered: Freemind and OpenCMS. The rationale for this decision relies mostly in the fact that these systems are used by other solutions that comprise the baseline (see Section 2.1).

In this section there is a brief description of the systems, including detailed information which might be considered important to the understanding and to the analysis of the results. It is essential to mention that the common coupling detection is going to be performed according to two distinct approaches: Eclipse JDT/AST and SrcML. The results are compared in order to comprehend the differences between the approaches. In other words, this final report yields four results - Freemind system for the AST approach and for the SrcML approach; analogously for the OpenCMS system.

2.1 Baseline

In order to validate the results of the automatic approaches for common coupling detection, it is necessary to use other projects as starting points for comparison. Nonetheless, the same can be said for the approaches described in this document. First, both approaches must be executed for the aforementioned systems. Results shall contain class-level information about common coupling. Subsequently, this information is going to be stored as a baseline so that other solutions can use these results for comparison.

2.2 Freemind

a. URL:http://sourceforge.net/projects/freemind/files/freemind/1.0.1/freemind-src-1.0.1.tar.gz/download
b. No need to use Git.
c. analysis directory
- ~/opennlp-opennlp-1.6.0-rc6/opennlp-tools/src

2.3 OpenCMS

a. URL: https://github.com/alkacon/opencms-core
b. Git tag: ms_9_4_7
c. analysis directory
- ~/opencms-core-ms_9_4_7/src

3 Eclipse JDT/AST Approach

The first iteration of the implementation of the Eclipse JDT/AST approach for common coupling detection is represented by a prototype. This first step consists of running the prototype on a single class, the same class selected for the baseline, and compare both results. The goal is to validate the prototype implementation using ASTVisitor, and finally execute the next step, which is the execution of the fully-implemented solution for the systems.

For each java file within the system

visit(FieldDeclaration node)

a. Process fields with public static or protected static
b. Store some information for later comparison
- Declaring class
- Line number

visit (QualifiedName node)

a. Process qualified names with public static or protected static

Compare whether the fields and qualified names stored are same

a. If the comparison is equal, store it

Find the files where the global variables are used and get statistics

a. # of strong coupling (public static)
b. # of weak coupling (protected static)
c. Top 10 most used global variables

Store the final results in a text file

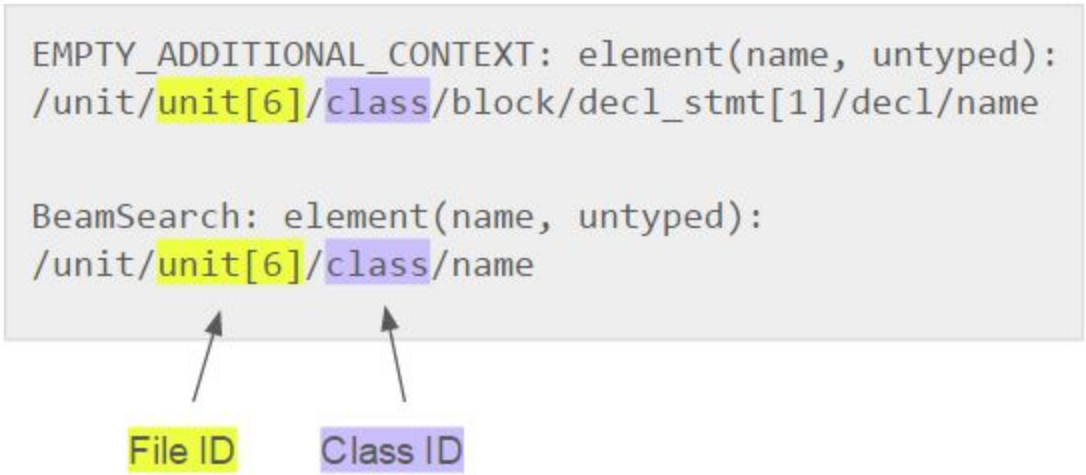
* The program used JDT / AST binding information

4 SrcML Approach

For the SrcML approach, first we have to format the system to the respective format. Then we extract the required information and detect common coupling within the system. In our case, we format the system by generating the XML with SrcML command. Then we extract the fileID and unitID with XPATH, and detect coupling by iterating over the system files.

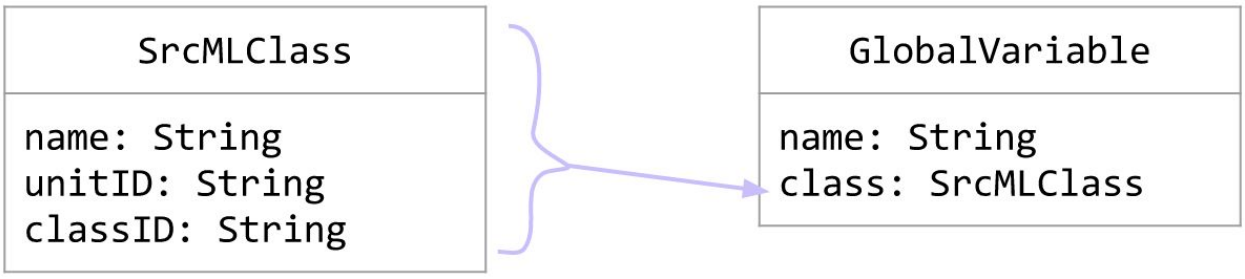
Figure 1 shows the results generated using XPATH, highlighting that unit corresponds to the fileID, and class to classID.

Figure 1



Then we identify usage in other system files using its qualified names. The structure is shown in figure 2.

Figure 2



One issue with SrcML is that it has trouble dealing with large projects, i.e., it takes a long period of time to generate xml format, execute the search and produce the final results.

5 Comparison

Similar results were produced by the approaches, but they are not equal. This section is going to explore the differences and present the analysis of the data. First, JDT/AST detects the global variables in a complete and independent manner. For the SrcML, however, there is an extra effort to generate similar results, because *srcml* commands must be tailored to specific goals, which are described by the software analyst using these commands. The problem with this approach is that the analyst might generate an incomplete input. In other words, whenever SrcML is used, the software analyst must predefine all possibilities and make sure all of them are addressed. This issue can be verified in Table 1.

Table 1: Overall results for the Freemind system.

Freemind	JDT/AST	SrcML
Global variables	261	232
Strong coupling variables (*public static*)	222	194
Weak coupling variables (*protected static*)	39	38
Not used	119	108

For instance, if we take the Freemind system, it is possible to assess that the difference in global variable detection is due to the fact that unconventional classes are used (e.g. inner classes), which is not the case in the OpenCMS system. For the results of the AST, inner classes were automatically considered because this approach uses the AST Visitor, which provides functionality that is not a responsibility of the analyst. On the other hand, for SrcML, inner classes were not captured. After the assessment, the analyst have to decide whether to consider it or not, because the corresponding XPath command would be too complex and inner classes represents a small portion of the system. In this project, it was decided not to consider.

Table 2: Overall results for the OpenCMS system.

OpenCMS	JDT/AST	SrcML
Global variables	7224	7224
Strong coupling variables (*public static*)	7063	7063
Weak coupling variables (*protected static*)	161	161
Not used	3217	3235

It is also important to highlight the global variables that are not used. If we take the AST approach for example, in the case of OpenCMS and Freemind, 3217 out of 7224 (44.5 %) and 119 out of 261 (45.6 %) respectively are not used. Since this is unexpectedly high, a further investigation was performed. As a result of this investigation, we could detect too important factors: (1) cases where a global variable is only used within the class it was declared is recurring (we do not consider this to be coupling, thus included in the “Not used”); (2) there are cases where a global variable is declared in a superclass and used throughout the system by some of its subclasses (there is no clear definition on how to handle coupling and inheritance, thus we also included these in the “Not used”).

(Freemind) AST - Top Ten Global Variables Used:

1. freemind.modes.EdgeAdapter.WIDTH_THIN were used in 7 classes
2. freemind.main.FreeMindCommon.FREEMIND_FILE_EXTENSION were used in 6 classes
3. freemind.modes.MindMapNode.STYLE_FORK were used in 6 classes
4. freemind.modes.MindIcon.LAST were used in 5 classes
5. freemind.modes.MindMapNode.STYLE_BUBBLE were used in 5 classes
6. freemind.modes.attributes.AttributeTableLayoutModel.SHOW_ALL were used in 4 classes
7. freemind.main.FreeMindCommon.RESOURCE_ANTIALIAS were used in 4 classes
8. freemind.main.FreeMindCommon.CHECK_SPELLING were used in 4 classes
9. freemind.main.FreeMindCommon.MINDMAP_LAST_STATE_MAP_STORAGE were used in 3 classes
10. freemind.common.OptionalDontShowMeAgainDialog.ONLY_OK_SELECTION_IS_STORED were used in 3 classes

(Freemind) SrcML- Top Ten Global Variables Used:

1. freemind.main.FreeMindCommon.FREEMIND_FILE_EXTENSION were used in 9 classes
2. freemind.modes.EdgeAdapter.WIDTH_THIN were used in 7 classes
3. freemind.modes.MindMapNode.STYLE_FORK were used in 6 classes
4. freemind.modes.MindMapNode.STYLE_BUBBLE were used in 6 classes
5. freemind.main.FreeMindCommon.FREEMIND_FILE_EXTENSION_WITHOUT_DOT were used in 6 classes
6. freemind.modes.MindIcon.LAST were used in 5 classes
7. freemind.main.FreeMindCommon.CHECK_SPELLING were used in 4 classes
8. freemind.modes.XMLElementAdapter.XML_NODE were used in 4 classes
9. freemind.main.FreeMindCommon.RESOURCE_ANTIALIAS were used in 4 classes
10. freemind.modes.attributes.AttributeTableLayoutModel.SHOW_ALL were used in 4 classes

For the freemind results, SrcML yields higher coupling than what is presented in AST results. This happens because the SrcML approach currently takes into account comments. That is, if a global variable is mentioned within a comment (e.g. //, /* */, etc), SrcML detects that as an occurrence of coupling, whereas AST does not. This project considers the AST approach to be closer to the expected behavior. However, for transparency purposes, the entire results were included, since there might cases where the SrcML approach is considered closer to the expected. The top ten list of global variables used represent the final overall output of the approaches[1]

(OpenCMS) AST - Top Ten Global Variables Used:

1. org.opencms.file.CmsResourceFilter.ALL were used in 109 classes
2. org.opencms.file.CmsResourceFilter.IGNORE_EXPIRATION were used in 72 classes
3. org.opencms.main.CmsLog.INIT were used in 49 classes
4. org.opencms.security.CmsPermissionSet.ACCESS_WRITE were used in 47 classes
5. org.opencms.file.CmsPropertyDefinition.PROPERTY_TITLE were used in 43 classes
6. org.opencms.file.CmsResourceFilter.DEFAULT were used in 35 classes
7. org.opencms.file.CmsProject.ONLINE_PROJECT_ID were used in 32 classes
8. org.opencms.i18n.CmsEncoder.ENCODING_UTF_8 were used in 31 classes
9. org.opencms.file.CmsResource.DATE_EXPIRED_DEFAULT were used in 30 classes
10. org.opencms.file.CmsResource.DATE_RELEASED_DEFAULT were used in 28 classes

(OpenCMS) SrcML - Top Ten Global Variables Used:

1. org.opencms.file.CmsResourceFilter.ALL were used in 110 classes
2. org.opencms.file.CmsResourceFilter.IGNORE_EXPIRATION were used in 72 classes
3. org.opencms.main.CmsLog.INIT were used in 49 classes
4. org.opencms.security.CmsPermissionSet.ACCESS_WRITE were used in 47 classes
5. org.opencms.file.CmsPropertyDefinition.PROPERTY_TITLE were used in 43 classes

[1] Complete output can be found on team's website: https://github.com/rpgoncalves/sw-comprehension/

6. org.opencms.file.CmsResourceFilter.DEFAULT were used in 42 classes
7. org.opencms.file.CmsProject.ONLINE_PROJECT_ID were used in 32 classes
8. org.opencms.i18n.CmsEncoder.ENCODING_UTF_8 were used in 31 classes
9. org.opencms.file.CmsResource.DATE_EXPIRED_DEFAULT were used in 30 classes
10. org.opencms.file.CmsResource.DATE_RELEASED_DEFAULT were used in 28 classes

Additional considerations:

1. The system are being developed. As the release schedule is near, some codes may be made in haste or may not be documented for the relevant use of global variables.
2. Although one developer creates the global variable and intends to be shared with other developers, it may take time to communicate and get to know the use of global variables.
3. Very knowledgeable, far-sighted, skilled and experienced developers will be careful not to have unused variables be placed in the source files. However, not all developers are like the first-level developers. Some developers start to develop software systems, others, although they know the importance of quality codes, do not get used with such development habits. A few developers may not know the concept of reducing common coupling or clean code. These practices or personal knowledge may contribute the high percentage of unused global variables.

6 References

[1] Wikipedia. **Coupling (computer programming).** Accessed on: March, 29. Available at: <https://en.wikipedia.org/wiki/Coupling_(computer_programming)>.

[2] Wikipedia. **Global variable.** Accessed on: March, 29. Available at: <https://en.wikipedia.org/wiki/Global_variable>.

[3] Boukari Souley and Baba Bata. **A Class Coupling Analyzer for Java Programs.** West African Journal of Industrial and Academic Research Vol.7 No. 1 June 2013.

[4] Michel Chaudron. **Design Heuristics and Architectural Styles (LL Chapter 9).** Accessed on: April, 5. Available at: <https://rickvanderzwet.nl/trac/personal/export/3/liacs/se/slides/09_Design_Heuristics_and_Styles_part2.pdf>.