Kevin Nguyen, Jacqueline Wong

Common Coupling Detection at File-Level and Method-Level Granularity: An ASTVisitor Implementation: Final Report

**Introduction**

Our project detects common coupling, which is defined as “when two modules share the same global data.” Specifically, our program tracks common coupling at the file-level and method-level granularity. We achieve this by using Eclipse ASTVisitor to perform static analysis on java code. This allows us to identify in said source code when global variables are declared as well as used in other java files within the system.

**Project Implementation Details (Source Code on GitHub)**

File Granularity

We detect common coupling in file-level granularity by keeping track of non-private static variables. When we find these variables being declared during our first ASTVisit, we record what java file the variable was declared in as well as the variable itself in an ArrayList. After constructing a list of “global” variables found in the system, we then revisit the system, this time checking for all accesses/ uses of “global” variables by checking against the previously created static variable ArrayList mentioned earlier. Specifically, we looked at FieldDeclarations, MethodDeclarations (and method bodies), Assignments, and MethodInvocations. This allows us to record how many external java files accessed and used a non-private static variable that was declared in another java file. Additionally, we output the top 10 most widely used “global” variable in terms of number of external files utilizing said variable.

We also analyze external file usage at file-level granularity. We examine all of the external files included in the open source system, and add the file names into an ArrayList. Then, we check the java source files and check usages of the external file names through similar methods (FieldDeclarations, MethodDeclarations/bodies, Assignments, and MethodInvocations), then we print out those usages along with what java file the usage occurred in.

Method Granularity

For method-level granularity, we repeat the approach above except we look at the instance variables in a java file instead. We record all the methods in a file that use said instance variables by looking at MethodBodies, and then we display the usages in our output.

**Results**

Our results are posted on our github in the FINAL PROJECT DELIVERABLES COMMON COUPLING folder. The file-level granularity .txt files contain how many global variables were found, how many we determined to be used in at least one external files, a top 10 list of most used global variables in terms of number of external files that mention said variable, and a comprehensive list of the static variables themselves. Our Method-Level granularity results for CMS were too big for github, so we linked a google doc here: <https://drive.google.com/open?id=0B0HfBS_aeJTbZklJZ21tUlhOQ0k>

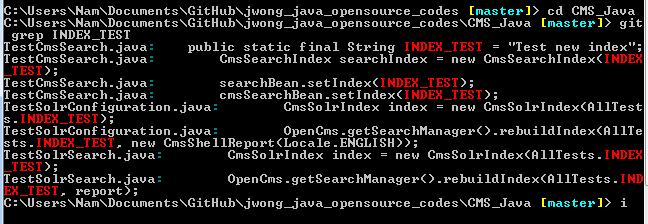
We also have an External File Usage .txt file for both Freemind and CMS that lists all of the external file names (All non-.java files found via looking in the system directories) that are mentioned and used within the system’s Java files. This was significant to our project because common coupling derived from external files can be just as important as common coupling of “global” variables within Java files. For example, changing the content in .xml files, or changing .jpeg images that are used by multiple Java files will effect every Java file that shares and uses said external files.

**Gold Set**

As discussed during our presentation, our personal methods of creating a gold set did not yield any valid measurements toward our results created by our common coupling detection program. Though our methods used a very common sense approach (git grep to find static variables and their usages), we were essentially emulating our program, so our gold set did not carry any weight/ validity. Because of this, instead of comparing our new results with the gold set again, we focused on comparing our results to the other group’s implementation of common coupling on the same systems.

**Result Anomalies**

In our Freemind File-Level granularity, we found 457 static variables and 253 global variables used in at least one external file. And in our CMS File-Level granularity, we found 11550 static variables and 1477 global variables used in at least one external file. In order to explain the anomalies in our results, we looked at many of our non-private static variables that our program determined as not being used in external files. When examining these cases in Freemind Java, we found that many of the variables that our program stated were not used in other files contained the “final” Java keyword. Because of this, we believe that the variable was most likely intended to be used as a constant within the class, and was not meant to be utilized outside of the file it was declared in. Additionally, we also found another edge case that we found messed with our data, found in CMS.



In the picture above, we see that the non-private static variable INDEX\_TEST was declared in the file TestCmsSearch.java. However, INDEX\_TEST is only ever called outside of TestCmsSearch.java in the context AllTests.INDEX\_TEST. Since the global variable is being used in such a way that there is a different class calling it, that use is not being recorded as we would be looking for TestCmsSearch.INDEX\_TEST in our program, thus contributing to our lopsided numbers.

**Conclusion**

After testing on the open source systems Freemind Java and Open CMS, we realized that there were even more edge cases (in addition to the ones that we found doing our interim report and presentation) that we needed to take into account for. These edge case primarily resulted in us having many “global” variables that our file-level granularity program determined as not being used in external files, throwing off our data. Our next step in our future work would be to address the edge cases mentioned earlier in our Result Anomalies as well as address differences between our results and the other groups’ implementation of common coupling detection.