AOC: An Aspect — Oriented Approach for Comment to Reduce the Redundant Behavior between Source Code and Compilation Code

Siwadol Sateanpattanakul*
Department of Computer Engineering, Faculty of Engineering at Kamphaeng Sean, Kasetsart University, Kamphaeng Sean Campus, Nakhon Patom, 73140, Thailand.

* Corresponding author. Tel.: +6689-5556-956; email: sidol.sat@gmail.com
Manuscript submitted May 22, 2015; accepted August 30, 2015.
doi: 10.17706/jsw.10.12.1336-1342

Abstract: Compiler has the duty to transform the source code to machine code. After the compilation process, there are two types of file was be appear as the former source code and the byte code. The program behaviors exist in two places, source code and machine code. They make repetition of the software engineering process. It means every piece of knowledge must have a single within a system.

For resolve duplication of knowledge, this paper proposes the methodology to extract and rebuilds the comments. Then, using the aspect-oriented approach to manage and advise their comments in the "AOC" form. The extension compiler can take all comments from the source code. At ratio 1, this methodology induces to reduce redundant knowledge between source code and byte code to 34.26 percent.

Key words: Software engineering, code redundant, compiler, reverse engineering.

1. Introduction

Comments in programming languages are also known as the guideline of the source code. Many programmers use the comments for various reasons and conditions. Almost of the comments are used to remind or explain what the program does. It not only provides a summarization of the idea of the code [1]. It also represent as a document expansion that contains a significant term of program behavior. Although they are important for the programmer, but all comments have abandoned by the compiler. The byte code that contains the program behavior is available. At the same time, the comments are gone. From a compilation process, the program behavior appears in two files. First, it appears in the source code from. Second, it appears in the byte code from. There are duplicating data between two files. To avoid repeat yourself mechanism, the program behavior and comments have to exist only one place [2].

Software reverse engineering (SRE) is the process to extract and recover the knowledge of software system to comprehensible data [3], [4]. This process has supposed to disassembling current data into earlier form. It's also known as many terms depending on the level of information such as Model Driven Reverse Engineering (MDRE) in term of analysis and design level. The lowest level of SRE is transforming the machine code by reverse engineering process. In Java, it has been called "Java bytecode reverse engineering". Definitely, it uses to transform byte code to source code. It is a decompilation process in bytecode term. Decompilation has many uses in the real world, such as the recovery of lost source code for a crucial application [5], [6], therefore if the quality of Java decompilers increased they might be of more use

commercially. There are many approaches to do reverse engineering in Java, for example, Dava [7] and JD Project [8]. Both of them are decompile tools which have been used to lowest reverse engineering in Java.

The software maintenance and reuse are software processes that can take the advantages from SRE. Because, they can adjust some knowledge of the source code directly if the former system cannot be fit for some problems [9]. Although the SRE can transform all existing data from the byte code to former form, but it cannot bring all parts of the data back. Since, some original data have been removed at compilation time. The comment is a kind of data that not allow to be existed in the byte code form. This is a reason why the SRE cannot transform all original source code.

Obviously, the existing of the comments is reasonable when they can support program behavior. Because of the duties of comments make a readable and comprehensible to the source code when reverse from byte code. Therefore, the important thing is the way of keeping their comments because it will be removed after compilation process. Absolutely, the comments can be extracted by many approaches, but it should be handled during the compilation process.

This paper purposes the compiler which could be refined and gathered comments into "AOC" file format. The "AOC" file take Aspect-oriented programming (AOP) [10] to provide the necessary information. These essential data should be used to set the comment to the appropriate position after through the compilation process. All process would be described in each part of this paper as follows; the Overview of the extractor would be described in Section 2. In Section 3 the paper demonstrates an Implementation of our compiler. Section 4 represents a preliminary experiment of the extension compiler. Finally, the future work and concludes are explained in Section 5.

2. System Overview

According to the compiler construction theory, there are three main modules in the compiler front-end as follows.

- Lexical analysis (Lexer or Scanner)
- Syntactic analysis (Parser)
- Semantic analysis

And the core module of compiler back-end is bytecode generator. They are the key of the problem and the solution. Because, all comments are vanishing at the first step and cannot recover them back from the remaining parts. Therefore, the problems should be started to fix from Lexical analysis section. To extract and gather all comments, there are three necessary parts of this system as follows Extractor, Adding information and Pivot maker.

2.1. Extractor Module

The first part, extractor, it is the embedded part in the lexical analysis to extract all comments including single line, multi-line and Java document. The duty of this part is separating all kinds of comments in Java from source code. Then, all extracted comments are stored in the temporarily readable format. The Java script object notation (JSON) has been selected to do this interchange format. It is an intermediate data that will be transformed to proper form in the last section.

Because of all comments are texts or string. They have no associated between each other. It is a big problem for the track where the original sources of the comment are from. So, The JSON is the specific format for describing and handles three kinds of comments in Java as a single line, multi-line comment, and Javadoc. To build this format, the stored procedure has to involve the comment which abandons from the scanner. And then the store procedure built it into JSON format. The file format was designed as lightweight and readable information. It can be described the format as follows

- S (Single line comment)
- M (Multiline comment)

- E (Extra comment)

The comment extractor transforms comments to JSON by adding some necessary information with original data. For example, there is some comment as “sample comment” at line 1. This part encases the data by adding a tag (see Fig. 1).

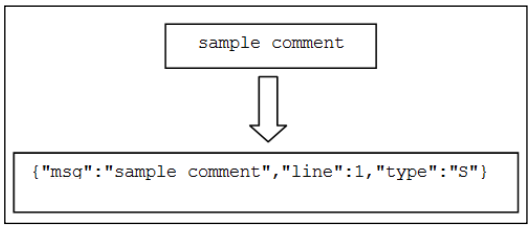

Fig. 1. Transforming the data to JSON format.

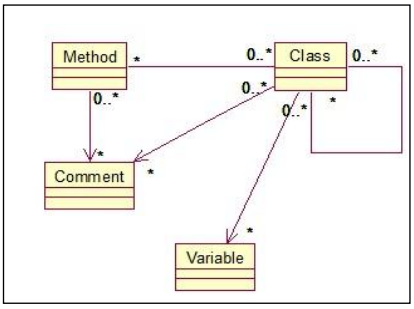

Fig. 2. The Association between among of entities.

Additionally, there are some relative between entities inside Java class. But with pure data from this section cannot enough the make it happen. The relation and association among the entities can be represented as association between Plain Old Java Object (POJO) (see Fig. 2). Accordingly, the next section will take advantage of association among all entities to assist JSON to build and manage the hierarchy of all data inside it.

2.2. Insert Information Module

In the second part, there is some crucial information about Syntactic analysis part as method header, variable, line and class name. Because of there is no relation between the data from previous. So, the system cannot know the origin comments are from. With this reason, all data have to be added the information and work with original JSON data from previous for mark the relation and association between among of data. To do this process, the data must be got from JSON by a single entity that has no relative like a comment from Fig. 1. And insert the new data from the parser for creating the relation and hierarchy between all data. Therefore, after this part the JSON would contain all data from the extractor and insert information module (see Fig. 3).

```
{"name":"Sample","line":1
,"Comments":
[
 {"pc":-1,"msg":"sample comment 1","line":3,"type":null},
 {"pc":-1,"msg":"sample comment 2","line":7,"type":null},
 {"pc":-1,"msg":"sample comment 4","line":14,"type":null}
]
,"Class":[]
,"Methods":
[
{"Class":[],"Comments":
[
{"pc":0,"msg":"sample comment 3","line":9,"type":null}
],"line":8,"name":"getValue()","classes":[]}
]
,"Variables":
[
{"name":"i","line":4},
{"name":"j","line":15}
]
}
```

Fig. 3. Full relationship in JSON format.

```
Class <<name>>{
   bfs : <<comment>>
   afs : <<comment>>

   <<variable-name>>{
                bfs : <<comment>>
                afs : <<comment>>
   }
   <<method-name>>{
                bfs : <<comment>>
                afs : <<comment>>
                ibfs : <<pc>> <<comment>>
                iafs : <<pc>> <<comment>>
   }
}
```

Fig. 4. AOC file format.

2.3. Pivot Assembler

Finally part, the pivot assembler, this part uses all contain data from JSON and rearrange them to a

suitable position. The key of reorganizing from former structure is the data from this section of the compiler. All data from JSON format have to be advised by some essential data from the bytecode generator module. The bytecode generator can provide various important data such as method's line, variable's line, program counter (PC) etc. With JSON and bytecode generator information, all comments are reconstructed and transform to the final format. It has been called "Aspect oriented Comment Format (AOC)". This format is a declarative programming paradigm to declare the point for weave the comment to the bytecode. These AOP approaches have been used in many kinds of format such as AspectJ [11] and JastAdd [12].

This format is used to explain all points of comment in a class and method (see Fig. 4). The concept of AOP is used to design the point that will be used in the reverse engineering process [10]. Hence, The AOC is the essential format that contains all data of the comments from JSON in appropriate point.

Table 1. Aoc Tags and Meaning

Keyword / Tag	Meaning
<<name>>	Class name
<<variable-name>>	Variable name
<<method-name>>	Method name
<<comment>>	Comment message
bfs, afs	Before and after with single line comment at the outside.
bfm, afm	Before and after with multi-line comment on the outside
bfe, afe	Before and after with javadoc at the outside
ibfs, iafs	Before and after with single line comment at the inside
ibfm, iafm	Before and after with multi-line comment on the inside
ibfe,iafe	Before and after with javadoc comment on the inside

```
public class Sample {
 // sample comment 1
 private int i ;
// sample comment 2
public int getValue(){
  // sample comment 3
  return i+j ;
}
  // sample comment 4
 private int j ;
}
```

```
public class Sample {

  // Field descriptor #6 I
private int i;

  // Field descriptor #6 I
private int j;

  // Method descriptor #9 ()V
  // Stack: 1, Locals: 1
public Sample();
 0  aload_0 [this]
 1  invokespecial java.lang.Object() [11]
 4  return
   Line numbers:
   [pc: 0, line: 1]

  // Method descriptor #15 ()I
  // Stack: 2, Locals: 1
public int getValue();
 0  aload_0 [this]
 1  getfield Sample.i : int [16]
 4  aload_0 [this]
 5  getfield Sample.j : int [18]
 8  iadd
 9  ireturn
 Line numbers:
 [pc: 0, line: 7]
}
```

```
class Sample{
 i:{
  bfs : sample comment1
 }
 j:{
  bfs : sample comment 4
 }
 getValue(){
  bfs : sample comment 2
  ibfs : 0 sample comment 3
 }
}
```

Fig. 5. Transforming from source code to byte code; Fig. 5a. (left) represent Java source code; Fig. 5b. (middle) represent byte code; Fig. 5c. (right) represent AOC file format that got comment from 5a.

Usually, Java class consists of three necessary headers for drive content in the byte code as <<name>>, <<variable>> and <<method>>. This approach is focused on vital component for explaining the comments

location in the class. The table 1 can illustrate all types of format in the AOC.

The consequently as the previous part, this module is the embedded system to work together with the bytecode generator module. There are 2 types of file to use in the bytecode generator module as follows

- Java byte code (. class)
- Java Aspect oriented Comment form (. Jaoc)

Therefore, when the compilation process was done, the original source code is transformed to 2 types of file as Java bytecode and AOC (see Fig. 4). Let see the code In the Sample class, there are two variable declarations and a single method name "getValue()". This method returns the plus value of "i" and "j". And the comments have been defined above of the statements. Normally, The Java compiler (Javac) generates a byte code to the method header "public int getValue();" and all statements in the method. At the inside of the method, the compiler uses a program counter (PC) for describing the address of a statement. In the Fig. 5 represent the AOC can use this information to advise the proper pivot of each comment.

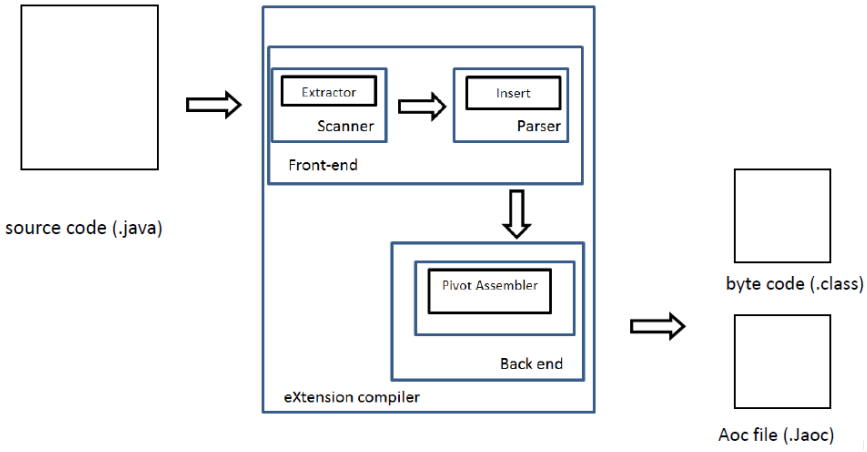

Fig. 6. The extension modules inside the compiler.

3. An Implementation

The compiler has been extended from Eclipse compiler [13] that allows developer to extend a new specification into itself [14], [15]. This compiler system consists of several sub-modules which work together. There are three extensive sub-modules in Eclipse compiler as lexical analyzer, syntactic and code generator. With the techniques developed and extended at previous section, the extension compiler is now available. Although the compiler was being modified but there is no disturb at the core of compiler directly. It means there is no severe effect of the compiler. It can compile the code with normally procedure. The overview of the system of extension compiler can be illustrated in the Fig. 6.

For these reasons, the meaning of the programming language has not changed. So, the semantic module isn't represented in this extension compiler.

4. Preliminary Experiment and Discussion

The test cases were designed for assessing the factors that be effected when adding this methodology into a former compiler. So, the system has to be evaluated by various test cases. All primary tests were carried out on an Intel core i7-4770 @ 3.40GHz with 8 Gbyte memory, running Microsoft Window 7 Ultimate service pack 1 and including JRE7.

The compiler in Eclipse, Java development tools (JDT) is a former compiler. The version of eclipse compiler is eclipse.jdt.core_ 4.5.0 and eclipse.jdt.compiler.apt_ 1.1.100.

Table 2. The Compilation Time from Different Ratio

Compilers	Compilation time (ms)						
	0.0	0.25	0.5	0.75	1.0	1.25	1.5
eclipse.jdt.compiler.apt_1.1.100	156.47	156.51	156.41	156.47	156.43	156.48	156.41
Extensible compiler	156.41	366.61	371.28	373.23	393.61	396.14	405.23

Table 3. File Size from Different Ratio

Compilers	File size (Bytes)						
	0.0	0.25	0.5	0.75	1.0	1.25	1.5
eclipse.jdt.compiler.apt_1.1.100	3040	4093	5158	6132	7425	8286	9601
Extensible compiler	0	1025	2433	3620	4881	5982	7223

This experiment is conducted to evaluate the compilation time between the former compiler and the adjustment compiler with similar programs. And then the adjust amount lines of comment with different ratio (line of comment per line of code). The result of the experiment is shown in Table 2.

The result of Table 1 represents compilation time at various comments. The lines of code are the program behavior that contains source and byte code. The extension compiler can separate all comments, but the compilation time of extension compiler is much than a former compiler. If focus the ratio at 0.25, there is a lot different compile time between the 2 compilers. While the source code has a comment, the extension has to make the header of variable and method. Moreover, it must spend a time to make the relation between each element of command and headers to do reverse engineering.

With the similar test case, there is another result that this experiment provides. The result of reducing the redundant comment has been shown in Table 3.

The result in Table 3 induces the differentiation between original code and only AOC file. At 1 ratio, the similar knowledge can be reduced around 34.26 percent. The difference at 50 percent is ideal because some information like header appeared in AOC file. They are essential info for AOP technic to induce proper comment positions. The similar knowledge in AOP file can be reduced with Natural Language Model [16].

5. Conclusion

This dissertation presents the methodology and implementation of the extension compilers. This approach uses an extended architecture for the design and implements the compiler. The results from the preliminary experiment represent all comments can be captured by an extensible compiler. And an extension compiler can be reconstructed them into the appropriate position on the AOC form. The AOC provides necessary capability to produce a former form when do reverse engineering process.

Now, the first implementation focuses on the purpose the possibility to extract the comments. And arrange them to the right position. This project has to be engaging on improving the performance of this compiler for assembling the comments to the right place from any possible style of code. And then this project will focus on the reverse engineering process for transforming the comment in the original source code.

Acknowledgment

I am grateful to my colleague reviewers for their constructive comments. And also appreciative to department of computer engineering, faculty of engineering at Kampheang Sean for providing many facilities.

References

[1] Sridhara, G., Hill, E., Muppaneni, D., Pollock, L., & Shanker, K. V. (2010). Towards automatically generating summary comments for java methods. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (pp. 43–52).

[2] Hunt, A., & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley Professional.

[3] Eric, J. B., & David, A. G. (1992). *A Software ReEngineering Process Model in COMPSAC ACM*.

[4] Baxter, I., & Mehlich, M. (2000). Reverse engineering is reverse forward engineering. *Science of Computer Programming*, *36*, 131147.

[5] Wu, X., Murray, A., Storey, M. A., & Lintern, R. (2004). A reverse engineering approach to support software maintenance: Version control knowledge extraction. *Proceedings of the Reverse Engineering 11th Working Conference on Engineering* (pp. 90-99).

[6] Emmerik, M., & Waddington, T. (2004). Using a decompiler for real-world source recovery. *Proceedings of the 11th Working Conference on Reverse Engineering (Washington, DC, USA, 2004), IEEE Computer Society*.

[7] Miecznikowski, J., & Hendren, L. (2002). Decompiling java bytecode: Problems, traps and pitfalls. *Proceedings of the 11th International Conference on Nigel Horspool. Compiler Construction*.

[8] JD Project. Retrieved 2015, from http://jd.benow.ca

[9] Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, *31(7)*, 529-536.

[10] Kiczales, G. *et al*. *Aspect-Oriented Programming*. Springer Verlag.

[11] Gregor, K., Erik, H., Jim, H., MikKersten, J. P., & William, G. G. (2001). An overview of aspect. *Proceedings of the15th European Conference Object-Oriented Programming*.

[12] Hedin, G. (1999). Reference attributed grammars. *Second Workshop on Attribute Grammars and Their Applications*.

[13] JDT Core Component – Eclipse. Retrieved 2015, from http://eclipse.org/jdt/core/

[14] Aumer, D. B., Gammar, E., & Kiezun, A. (2001). Integrating refactoring support into a Java development tool.

[15] Rivieres, J. D., & Wiegand, J. (2004). Eclipse: A platform for integrating development tools. *Journal IBM Systems Journal*, *43(2)*, 371-383.

[16] Movshovitz-Attias, D., & William, C. (2013). Natural language models for predicting programming comments. *Association for Computational Linguistics*.

Siwadol Sateanpattanakul was born in Lop Buri province, Thailand in 1981. He received his B. Eng. in computer engineering and the M. Eng from Suranaree University of Technology (SUT), Thailand in 2003, 2007 and D. Eng degree from King Mongkut's Institute of Technology Ladkrabang (KMITL) in 2012. His research interests are software engineering, java technology, compiler construction and computer programming language theory.