Journal of Engineering and Applied Sciences 13 (6): 1320-1325, 2018
ISSN: 1816-949X

Using Object to Slice Java Program

Amir Ngah and Siti Aminah Selamat
School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu,
Kuala Terengganu, 21030 Terengganu, Malaysia

Abstract: Decomposition slicing is a slicing technique that captures all computation on a given variable regardless of the line number. This technique was originally proposed for the C language which is a structured programming language. Thus, this study proposes the slicing technique to slice the Java program by using the idea of decomposition slicing. The technique is called the Object Decom position Slicing (ODS) technique which consists of three steps. This study also presents the implementation of the ODS into a simple Java program.

Key words: Program slicing, Java, Object Decomposition Slicing (ODS), techniques, variable, language

INTRODUCTION

Program slicing is a process to decompose a large program into smaller components called slice. Weiser (1981) is the first to introduce program slicing that slices a program based on a slicing criterion (s, v) where s is the location that consist of the variable of interest and v is the variable of interest. The slicing technique proposed by Weiser is called static slicing. After that, Korel and Laski (1988) proposed a dynamic slicing technique that slices the program using three criteria with an addition of input into the two criteria used by Weiser. The slice produced in the dynamic slicing is much smaller than the static slicing but the slice produced can be varied because there are many possible inputs for one single program. Later, Gallagher and Lyle (1991) proposed decomposition slicing that would output only one slice for a variable. Decomposition slicing technique only has a variable as the slicing criterion without the location of interest and the technique only focuses on C program. Therefore, this study will try to implement the idea of decomposition slicing into the Java program with a little modification on the slicing criterion. The slicing criterion that will be used in this technique is an object instead of a variable.

Java program has been popular and widely used in many applications, especially in mobile applications. This might be because Java is a platform independent language that is easier to maintain. However, the object oriented features such as inheritance, polymorphism and dynamic binding must be taken into account while slicing the program. The technique proposed in this study will include all statements in the Java program related to the slicing criterion.

Literature review: Java is an object oriented programming that was released by Sun Microsystem in 1995. A year after that, Kovacs *et al*. (1996) proposed an improved static slicing that can be used to slice inter-procedural Java program while the tools for slicing sequential Java program were introduced later in 2007. Kovacs's static slicing is designed to handle Java features such as static variables, multiple package and interface. Kovacs also proposed a new dependency representation that can be used for polymorphic calls.

Mohapatrae *et al*. (2006) proposed dynamic slicing for the distributed Java program. In recent years, Zhang *et al*. (2014) used dynamic slicing for fault localization in the Java program and Xi *et al*. (2011) proposed coarse-grained dynamic slicing for Java program to encounter the drawback of the original dynamic slicing that use a lot of CPU time and memory space.

Java is a platform independent programming language that enables Java to be developed and run in any device as long as there is a Java Virtual Machine (JVM). In order to execute a Java program, the compiler first translates a high level language Java program into Java bytecode which is a machine language for JVM and the interpreter will execute the program. By using the bytecode, Zhao (2000) proposed a dependence analysis in the Java bytecode in order to understand the dependency on the Java program, Zhao also proposed a bytecode slicing by using a bytecode analysis. Other related research on Java bytecode as discussed by Szegedi and Gyimothy (2005) who proposed a dynamic slicing of the Java bytecode while Wang and Roychoudhury (2008) proposed the dynamic slicing using Java bytecode traces.

Corresponding Author: Amir Ngah, School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Kuala Terengganu, 21030 Terengganu, Malaysia

Java is a programming language that supports concurrent programming. Concurrent programming is a programming which the execution is computed concurrently instead of sequentially. This makes the execution of the program unpredictable resulting in many problems in the slicing process. As a solution for the problem, several study were proposed (Chen and Xu, 2001; Ranganath and Hatcliff, 2007) on slicing concurrent Java program.

Program slicing has been applied in many of the software engineering areas such as debugging, testing and maintenance. Panda and Mohapatra (2015) proposed a new slicing method for Java program using hierarchical characteristics in the Java program for detecting regression error in the modified program. Panda and Mohapatra (2015) also proposed the same technique to measure cohesion in Java programs while Jiang *et al.* (2010) proposed a debugging approach for Java Runtime Exception.

Program slicing is also proposed for the JavaScript, which is a programming language for web application developments. By combining dynamic and static analyses, Ye *et al.* (2016) proposed slicing the JavaScript by utilizing data dependency, control dependency and DOM dependency in the program.

MATERIALS AND METHODS

A proposed slicing technique for Java: In this study, the idea of decomposition slicing will be used to slice the Java program using object as the slicing criterion. The technique is called the Object Decomposition Slicing (ODS). ODS has three steps which are step 1: check object definition; step 2: get information and step 3: combine. The steps of ODS illustrated in Fig. 1 shows that the ODS accepts the Java program as an input and produces the object decomposition slice with respect to the object slicing criterion as an output. The Java program is assumed formatted and compilable. This study shows the example of the formatted Java program. Meanwhile, the number of slice depends on the number of object. The slice produced from this technique is an executable program. Thus, the ODS will only accept a program that can be compiled and run correctly as an input.

Step 1 of the ODS is to check the object definition. This step accepts the formatted java program and the output list of object from the Java program. Object is an instance of the class, a process of creating an instance is referred to as instantiation. Instantiation statement includes the type of the reference variable, the name of reference variable, the keyword new and the constructor of the instantiated class. Reference variable is

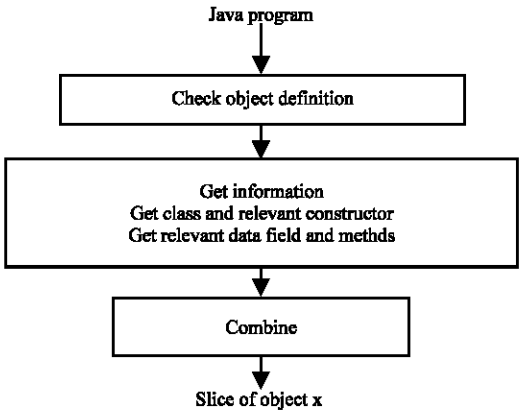

Fig. 1: Object decomposition slicing technique

used to store the location of the object of the instantiated class and the object then can be used to access the member of that class. Object reference variable and object are two different items but always used interchangeably. This is because the usage can help in the understanding of the object. In this study, object reference variable will be referred to as the object to make the terminology easier.

The class instantiation can be divided into two parts, declaration and initiation of object. These two parts can be written in two separate statements or in a single statement that combines both parts. The following statement shows how the instantiation statement can be written:

- Written in two separate statements
  <typeOfObject objectName;>
  <objectName= new constructorOfInstantiatedClass;>
- Written in one combined statement
  <typeOfObject objectName = new constructor of InstantiatedClass;>

For class instantiation, the type of object is referring to the name of the class. The object must be of java primitive data type which is boolean, byte, int, short, long, float, double or char. This is because object is a generic type that stores reference instead of the value of primitive type. Algorithm 1 and 2 show the syntax of the class instantiation. The class in Algorithm 1 instantiates and defines the object for the called class in Algorithm 2.

Algorithm 1; class instantiation and object definition:

```
class className{
    returnType methodName{
        class instantiation and defination
        object usage
    }//end method
}//end class
```

Algorithm 2; called class:

```
Class className{
data field
default constructor
parameterized constructor
method
}//end class
```

After the object is defined, it can be used in the class where the object is defined. The statement of object usage is the statement that contains the object accessing the class member of the called class.

The object definition and usage will be included in the slice. In addition, the declaration of class in Algorithm 1 that consists of method and class header along with the open and closed braces of the respective class and method will also be included in the slice to produce an executable slice.

Step 2 of ODS has two parts. The first part is to get the class name and the relevant constructor of the called class. The class name of the object can be obtained from the statement where the object is created. The statement consists of the constructor's name where the constructor has the same name as the class. By identifying the constructor, the class name is simultaneously captured. The class header of the called class will be included in the slice because the instantiation statement has dependency on the object definition. The class header that will be included in the slice is as follows:

<modifier class className{}>

Constructor is a special type of method that are invoked to create the object using the new operator. Different from normal method, the constructor has its own characteristics for it to be a constructor; otherwise, it is considered as a normal Java method. These three characteristics differentiate the constructor from the method in Java (Liang, 2013):

- Constructor must have the same name as the class where it resides
- Constructor cannot have any return type
- Constructor is invoked during the initialization of an object. The invocation operator is a new keyword

Constructor can be divided into two types, default constructor and parameterized constructor. Default constructor is a constructor without argument and the parameterized constructor is a constructor with argument. The program can have one default constructor and many parameterized constructor but each parameterized constructor must have different parameter. The Java compiler differentiates the constructors based on the number and the type of arguments had by the constructor.

It is possible for the Java class to not have any constructors declared explicitly. If the called class in the program is instantiated but the called class does not have a constructor, the system will generate the default constructor implicitly when the program is run. To invoke a constructor, the keyword new is used along with the type of constructor invoked:

- Invoke default constructor:

<new constructorOfInstantiatedClass();>

- Invoke parameterized constructor:

<new constructorOfInstantiatedClass(parameter);>

The second part of step 2 is to get the relevant data field and method from the called class. In this part, the output from step 1 will be used as an input to get the relevant data field and method. When the object is created, its members which are the data field and method can be accessed and invoked. In order to access the data members, the dot operator (.) is used. The following statement demonstrates how the object members can be accessed:

- Accessing object data field:

<object.data_field;>

- Invoking object method:

<object.method();>

Data field is defined by using identifier variable. Variable needs to be declared before it can be used and it can have a public, protected, default or private modifier. Variables are best set private and accessed only through method, so that, the data is encapsulated. However in this research, the default modifier will be applied to reduce complexity in the Java program. The program in this study will assume that the members are the default type, whereby the data can only be accessed by the class under the same package. Access includes calling, manipulating and changing the value of the data field. When the object accesses the data field, the data field in the called class is considered to have an influence on the object. Therefore, the declaration of the data field is included in the slice. The declaration of the data field is as follows:

<modifier dataType dataFieldName;>

Method in the called class can be accessed through the object such as the data field. Method is almost similar to the constructor except for the three characteristics that

Table 1: Input and output for steps 1-3

Steps	Input	Output
1	Java program	Object
		Class header, definition and usage of object
2	Object	Class header
		Relevant constructor
		Declaration of data field
		Relevant method
3	Information from steps 1 and 2	Complete slice

differentiate the two of them. One of the characteristics is that the method can have a return type while the constructor cannot. Method with the same name but different argument is called overloading method. Java runtime system differentiates this type of method based on the difference in the argument type. When the object accesses the method, the method is considered to have an influence on the object. The accessed method in called class then included in the slice. The method must have the following statement.

<modifier returnType methodName(argument){statement(s)}>

Not all statements of object members that are accessed are considered to have object dependency and taken into the slice. Any object members that are accessed inside the double quotation mark will be ignored and considered to have no dependency on the object.

Step 3 will combine the output from steps 1 and 2 into one whole complete slice. Step 1 output produces the component of slice from the class that consists of object definition while step 2 produces the component of slice from the called class. This two components will be combined to produce one complete slice. The input and output for steps 1 and 2 are simplified in Table 1.

RESULTS AND DISCUSSION

An example: Simple Java program is used to demonstrate the ODS technique. The example Java program consists of two classes which are the class that consists a definition of the object and the called class. The name of the class that consists the definition of object is TestShape as shown in Algorithm 3, the name of the called class is rectangle as shown in Algorithm 4.

Algorithm 3; class test shape:

```
M1    public class TestShape
M2    {
M3          public static void main (string [] arg)
M4          {
M5                Rectangle recl = new Rectangle ()
M6                recl. getArea ()
M7                recl. getPrimeter ()
M8          }
M9    }
```

Algorithm 4; class rectangle:

```
C1 1    public class rectangle
C1 2    {
C1 3          double width
C1 4          double length
C1 5          Rectangle ( )
C1 6          {
C1 7                                Width = 1
C1 8                                Length = 2
C1 9          }
C1 10   Rectangle ( int width, int length)
C1 11   {
C1 12   this.width = width
C1 13   this.length = length
C1 14   }
C1 15   double getArea ( )
C1 16   {
C1 17         return width*length
C1 19   double getPerimeter ( )
C1 20   {
C1 21         return 2*(width+length)
C1 22   }
C1 23 }
```

First step is to check the object definition in the TestShape class. In Algorithm 3, there is one definition for the object of the rectangle class and the object is stored in the reference variable rec1 in line number M5. Object rec1 is used in line numbers M6 and M7. Thus, the statement in line numbers M5, M6 and M7 is included in the rec1 slice of TestShape class. In order to produce an executable slice, the class and method header where the object is defined in line numbers M1 and M3 and all related curly braces in line numbers M2, M4, M8 and M9 are also included in the slice. Therefore, all the statements in the TestShape class are included in the slice and the output of step 1 is object rec1 as follows:

ObjectList = {rec1}
Slice-rec1$_{TestShape}$ = {M1, M2, M3, M4, M5, M6, M7, M8, M9}

The first part of step 2 is to get the name of the class that is used to create an object and all relevant constructors. The line number M5 in Algorithm 3 shows that the rectangle class is used to define the object named rec1. Therefore, the class header and the relevant constructor of the rectangle class will be included in the slice of the ODS. In Algorithm 4, the class header for the rectangle class is on line number $C_1$1 and the curly braces are on line number $C_1$2 and $C_1$19. Thus, the statement on line numbers $C_1$1, $C_1$2, and $C_1$19 will be included in the slice. The object rec1 is initialized using the default constructor of the rectangle class. The constructor invocation can be found written after the keyword new in the object definition statement. The constructor that is invoked to initialize the object, therefore, will be included in the slice. For object rec1, the default constructor that is included starts on line number $C_1$5 until line number $C_1$9 of the rectangle class.

Second part of step 2 is to get the relevant data field and the method of an object member. The statement where

the object rec1 accesses the object member is on line numbers M6 and M7 of the TestShape class. The accessing statement is called object usage where the object is used to access the object members. Members of the class that are accessed using the object will be included in the slice.

The rectangle class has two data fields, width and length that were initialized in the constructor. The initial value of the data field can be initialized directly or defined during constructor invocation. For object rec1, the data field is initialized during the constructor invocation where the value is set in the constructor. If the data field is not initialized, the Java Runtime System will set the data field with a default value. In the rectangle class, the definition of data field is included in the slice because it is used in the constructor. The following statement is the declaration of the data field on line numbers $C_1$3 and $C_1$4 of the rectangle class.

Object member method is invoked when the object accesses the method in rectangle. The method that will be included in the slice is only the user-defined method and not the pre-defined method. There are two user-defined methods in the rectangle class which are getArea and getPerimeter of type double. All statements in the invoked method will be included in the slice. Object rec1 accesses the method on line number M6 and M7 of the TestShape class. Both accessing statements invoke the method getArea and getPerimeter in the rectangle class.

At the end of step 2, ODS has chosen all the statements in the rectangle class related to the object rec1. The statement that will be included in the slice is the all statements related to the definition and usage of object rec1. All statements of the rectangle class that will be included in the slice are as follows:

$$\text{Slice-rec1}_{\text{Rectangle}} = \{C_1 1, C_1 2, C_1 3, C_1 4, C_1 5, C_1 6, C_1 7, C_1 8, C_1 9, C_1 10, C_1 11, C_1 12, C_1 13, C_1 14\ C_1 15, C_1 16, C_1 17, C_1 18 \text{ and } C_1 19\}$$

Step 3 is last step in the ODS technique. This step will combine all the information gathered from steps 1 and 2 into complete slice based on an object. The rectangle class has one object which is rec1, thus, the ODS will produce one slice as an output. The slice will consist of two classes, the TestShape class and the rectangle class where all the statements that are not related to the object are sliced away from the program:

$$\text{ODS}_{\text{rec1}} = \{\text{Slice-rec1}_{\text{TestShape}} \cup \text{Slice-rec1}_{\text{Rectangle}}\}$$

For the slice of object rec1, the slice of the TestShape class will include all 9 lines of codes. As for the rectangle class, the slice will include 19 lines from 23 lines of codes. The class rectangle is reduced by 5 lines of codes equivalents to 15.63% of the whole program. The 5 lines removed from the program are the statements that do not have dependency on the object slicing criterion.

CONCLUSION

Object decomposition slicing is a new slicing technique proposed for Java program. The technique consists of three steps and the slice consists of all statement that have dependency on slicing criterion object. The result from the example shows that the slice produced is smaller than the original program with a reduction of 15.63% from the whole original program. The ODS technique is expected to be applied in more complex programs in the future.

ACKNOWLEDGEMENT

This research is sponsor by Ministry of Education, Malaysia Government under Research Acculturation Collaborative Grant (RACE), Vot No. 56032.

REFERENCES

Chen, Z. and B. Xu, 2001. Slicing concurrent Java programs. ACM. Sigplan Not., 36: 41-47.

Gallagher, K.B. and J.R. Lyle, 1991. Using program slicing in software maintenance. IEEE Trans. Software Eng., 17: 751-761.

Jiang, S., H. Zhang, Q. Wang and Y. Zhang, 2010. A debugging approach for java runtime exceptions based on program slicing and stack traces. Proceedings of the 2010 10th International Conference on Quality Software (QSIC), July 14-15, 2010, IEEE, Xuzhou, China, ISBN:978-1-4244-8078-4, pp: 393-398.

Korel, B. and J. Laski, 1988. Dynamic program slicing. Inform. Processing Lett., 29: 155-163.

Kovacs, G., F. Magyar and T. Gyimothy, 1996. Static slicing of Java programs. Master Thesis, University of Szeged, Szeged, Hungary.

Liang, D.Y., 2013. Introduction to Java Programming: Comprehensive Version. 10th Edn., Pearson, Upper Saddle River, New Jersey, USA.,.

Mohapatra, D.P., R. Kumar, R. Mall, D.S. Kumar and M. Bhasin, 2006. Distributed dynamic slicing of java programs. J. Syst. Software, 79: 1661-1678.

Panda, S. and D.P. Mohapatra, 2015. ACCo: A novel approach to measure cohesion using hierarchical slicing of Java programs. Innovations Syst. Software Eng., 11: 243-260.

Ranganath, V.P. and J. Hatcliff, 2007. Slicing concurrent Java programs using indus and Kaveri. Intl. J. Software Tools Technol. Transfer, 9: 489-504.

Szegedi, A. and T. Gyimothy, 2005. Dynamic slicing of Java bytecode programs. Proceedings of the IEEE 5th International Workshop on Source Code Analysis and Manipulation, September 30-October 1, 2005, IEEE, Szeged, Hungary, ISBN:0-7695-2292-0, pp: 35-44.

Wang, T. and A. Roychoudhury, 2008. Dynamic slicing on Java bytecode traces. ACM. Trans. Program. Lang. Syst., 30: 1-10.

Weiser, M., 1981. Program slicing. Proceedings of the 5th International Conference on Software Engineering, March 09-12, 1981, IEEE Press, New York, USA., ISBN:0-89791-146-6, pp: 439-449.

Xi, L., M. Li, Z. Dan and L. Wei, 2011. An approach of coarse-grained dynamic slice for Java program. Proceedings of the 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN), May 27-29, 2011, IEEE, Hunan, China, ISBN:978-1-61284-486-2, pp: 670-674.

Ye, J., C. Zhang, L. Ma, H. Yu and J. Zhao, 2016. Efficient and precise dynamic slicing for client-side Javascript programs. Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering, Vol. 1, March 14-18, 2016, IEEE, Shanghai, China, ISBN:978-1-5090-1855-0, pp: 449-459.

Zhang, P., X. Mao, Y. Lei and Z. Zhang, 2014. Fault localization based on dynamic slicing via JSlice for Java programs. Proceedings of the 5th IEEE International Conference on Software Engineering and Service Science (ICSESS), June, 27-29, 2014, IEEE, Changsha, China, ISBN: 978-1-4799-3279-5, pp: 565-568.

Zhao, J., 2000. Dependence analysis of Java bytecode. Proceedings of the 24th Annual International Conference on Computer Software and Applications, October 25-27, 2000, IEEE, Fukuoka, Japan, ISBN:0-7695-0792-1, pp: 486-491.