International Journal of Soft Computing 3 (1): 44-49, 2008
ISSN: 1816-9503
© Medwell Journals, 2008

Data Flow Testing of Inheritance Property in JAVA Software

[1]M.E. Geetha, [2]V. Palanisamy and [3]K. Duraiswamy
[1]Department of Computer Sciences and Engineering, K.S.R. College of Technology, Tiruchengode-637215, Namakkal District, Tamilnadu, India
[2]Government College of Technology, Coimbatore, India
[3]K.S.R College of Technology, Tiruchengode, India

Abstract: The overall goal of testing is to provide confidence in the correctness of a program. Testing of object oriented software presents challenges due to property bugs in well-written programs. Which will give a run time behavior of a program and property checking of a language especially object oriented programs like JAVA, C++, Small talk, Objective-C and precision to the developer about a software. Object oriented software development different from traditional development products. In this polymorphism, inheritance ,dynamic binding are the important features. An inheritance property is the main feature. Data flow testing is an appropriate testing method. This test analyzes structure of the software and gives the flow of property. This study is designed to detect the hidden errors with reference to the inheritance property. Set of classes and packages are analyzed and output are hierarchies of the classes, methods, attributes and a set of inheritance related bugs like naked access, spaghetti inheritance bugs are automatically detected by using this testing. The testing by 4 major analysis such function model, class model, variable model and interface model. It is a Static testing. In this study tool is created for JAVA programs.

Key words: Software testing, data flow analysis, testing tools, inheritance property, inheritance related bugs, adjacency matrix

INTRODUCTION

Software is the execution of the software with actual test data. On average even well- written programs have one to three bugs for every statements. It is estimated that testing to find those bugs consumes half the labor involved in producing a working program. Traditionally there are two main approaches of testing software: Black-box (or functional) testing and white-box testing (structural Testing) (Alexender, 2001).

In black-box testing , software is exercised over a full range of inputs and the outputs are observed for correctness.

In white-box testing include every line of source code is executed at least once or requiring every function to be individually tested. White-box tests can be done without modifying the program, changing values to force different execution paths, or to generate a full range of inputs to tests a particular function.

DATA FLOW ANALYSIS TESTING

Control structure testing is a white box testing. This improves the testing coverage. Data flow testing is a one of the control structure testing. Data flow analysis extracts semantic information which can be used for code optimization, program slicing, semantic change analysis, program restructuring and code testing.

Data flow testing selects test paths of a program based on a definition and use of a variable in the program (Binder, 1993). Define (D) actions occur in a statement that changes the concrete state of an instance variable. Use (U) actions occur in a statement changes the value of an instance variable without changing it.

Inter procedural data flow analysis is needed to obtain information about program properties that depend on the interaction between different procedures. Inter procedural data-flow analysis designed to analyze whole programs (Rountev, 2002). Define in one part of the program and it may be used in another part. But it is different in traditional approach and object oriented approach.

WHOLE PROGRAM ANALYSIS AND FRAGMENT ANALYSIS TESTING

Traditional inter procedural data flow analysis is performed on whole programs. That is testing of very large programs with hundreds of thousands lines of code. Inter procedural whole program analysis takes as input an

Corresponding Author: M.E. Geetha, Department of Computer Sciences and Engineering K.S.R. College of Technology, Tiruchengode-637215, Namakkal District, Tamilnadu, India

entire program and produces information about the possible run-time behavior of that program. This analysis model is that the source code for the whole program is available for analysis. Such analysis topically treats the entire program as a homogeneous entity and does not take into account the program's modular structure. Whole program analysis is based on the implicit assumption that it is appropriate and desirable to analyze the source code of the entire program as a single unit.

Whole program analysis is not feasible for large or incomplete programs. The time required to build a whole program representation and the space need to store it is prohibitive. Finding bugs are very difficult because of side effects when we changing the variables or value of a variable.

So, Fragment data flow analysis as an alternative approach which compute data flow information for a specific program fragment. Fragments are set of procedures, methods, libraries, components. In some cases the analysis results are not need for the whole program, but only for a relatively small part of it. Instead of addressing the problem of computing data flow information for the whole program, addressing the problem of computing data flow information for a specific program fragment (Rountev, 2002). Output of the fragment analysis is used by software development team for further enhancement or correcting on the current version. It will reduce the time and memory spaces occupied by unnecessary variables, unwanted flow between data definition and usage.

Fragment analysis computes information about only the portion of the program, which can be smaller than the program itself. Fragment analysis extracts several kinds of information like data flow framework, entry nodes of procedures called from outside of fragment, call nodes to procedures outside of fragment, return nodes from procedures outside of fragment, summary information about at boundary entry nodes, summary information at boundary call nodes.

The following points are advantages by using fragment analysis testing:

- Precision is easy by using fragment static analysis in coverage tools. By using fragment analysis specific language property is tested before deliver to the customer.
- This is very useful in object oriented software for testing the properties like polymorphism, inheritance, remote procedure calls etc.
- Component is a set of related procedures or classes. The interaction between components is through calls to methods and procedures and through access to shared variables. A component level analysis processes the source code of a single program component given some information about the environment of this component. The analysis input is the source code of the procedure together with summary information about the rest of the components. From the point of fragment analysis running time and memory usage is desirable instead to reanalyze a component every time this component is used as part of a new system.

Object oriented software is different from traditional software development. Object oriented development is a way to develop software by building self contained modules or objects that can be easily replaced, modified and reused. Each object had attributes and methods. Objects are grouped into classes. Basic concepts of object oriented programs are data abstraction and encapsulation which is wrapping up of data and methods in to a single unit. Inheritance is the process by which objects of one Class acquire the properties of objects of another class. It supports the concepts of hierarchical classification. Polymorphism means ability to take more than on form. Testing this properties in particular fragment is different from traditional software.

There are a number of testing tools perform white-box testing on executables, without modifying the source and without incurring the overhead of an interactive debugger. These tools speed testing and debugging because there is no need to wait for support code to be inserted in to the program. Most white-box testing tools change the executable in one way or another, or check for certain classes of failure. Tools provide a high level programming interface for writing code that patches the executable and performs a specific function or obtains a particular type of information, other features include patches that executes as part of the application.

Static analysis is concerned with algorithms for analyzing the possible behavior of software. By examine the software source code, static analysis can determine various software properties. These properties are fundamental importance for many software engineering tasks and tools.

Tools for software understanding use static analysis to identify interactions and dependencies between different parts of complex programs. This information is valuable for software development team may be used for next version development.

Unwanted code, variables, assignments, unnecessary call and returning between one place to another place are avoided. Coverage tools use static analysis to evaluate and improve the quality of software testing.

Dynamic analysis that is performed by observing software behavior while the program is executing.

Previous work was Polymorphism properties are analyzed and dynamic binding of concepts are verified (Rountev *et al*., 2004).

Inheritance property is an important feature in object oriented program. A testing tool developed for C++ language in previous work. Here level to level relationship found up to 'n' levels and matrix formed for their hierarchy.

This study focus on design of data flow testing of inheritance property in JAVA software. There is difference between JAVA and C++ inheritance property implementation. This is a static analysis tool.

DESIGN OF DATA FLOW TESTING OF INHERITANCE TOOL

The idea behind inheritance is creation of new classes that are built on existing classes. When inherit from an existing class (or inherit) methods and fields are reused and possible to add new methods and fields to adapt to the new situations. This new technique is essential in JAVA programming.

The object oriented , inheritance implementation may appear similar to those in other object oriented languages like C++, small talk etc, but there are many differences have to consider .

The inheritance in JAVA is basically carried by using the keyword "extends". Here is a simple example showing the use of the keyword "extends".

E.g.: Class Manager extends Employee {added methods and fields}

The existing class is called super class and the new class is called subclass. JAVA does not support multiple inheritances as is supported by C++. To implement multiple inheritances JAVA uses interfaces.

A method in a JAVA program may be private, public, protected or it may not have any scope defined. A method can also be static, final or abstract.

To develop a software testing tool for inheritance for JAVA programs, first need to analyze the given source code. The various steps followed for the tool design are

- Accept the JAVA filename as input from the user and then locate the file. In case of a missing file, an error message is displayed.
- In any JAVA source code, the source code may work perfectly even if there are multiple spaces or new lines between any executable instructions.

E.g.: int a=10; // a valid statement to assign 10 to an integer named 'a'
int a=10; // works fine though having numerous spaces
int a = 10 ; //this will perform the same function as the above statements like int a=10; or the second statement with numerous spaces.

These kinds of statements may hinder the proper execution of scanning and parsing. To avoid such statements a method named format is called. This method "format" prepares the source code file for testing by indenting the file and removing multiple spaces and multiple new lines. Here also separate the identifiers, literals, keywords and the separators.

Separators play an extremely important role for analyzing the source code for any program. The parentheses '('and')' is used to contain the list of parentheses in method definition and invocation. It is also used for precedence in expressions, containing expressions in control statements and surrounding cast type. In this tool parentheses are used to search for the method definitions in a class or an interface. The semicolon and the opening brace '{'separators are used to mark the end of a statement.

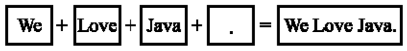
- The scanning of the formatted source code is performed character by character, this leads to creation of the various tokens used in the program. The characters join together to form strings which may either be a variable, a function name, a keyword, a separator etc. This is analogous to building words in the English language.

For example the word "JAVA" is built by the union of alphabets 'J', 'A', 'V' and 'A' . The word built has a meaning but it does not communicate anything. Similarly the tokens created here are just standalone tokens and they do not express any meaning.

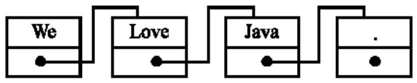
$$\boxed{\text{J}}+\boxed{\text{A}}+\boxed{\text{V}}+\boxed{\text{A}}=\boxed{\text{JAVA}}$$

- Move on to the joining of the various tokens to decipher the information related to every token. The placing of various tokens in respect to each other is observed, this placement provides the basis for building up the various information which is used for testing the authenticity and the security of the given JAVA program. This is one of the most import parts in architecture framework of the JAVA tool created. The amount of data mined in this step is directly proportional to the quality of testing and security analysis. This is analogous to creating sentences in English language. In this language, to understand the meaning of any complete sentence has understanding the relationship between the words and the position of the various parts of speech used in the sentence.

For example the sentence "We love JAVA." is composed of four tokens- "we" , "love", "JAVA" and the period "." .



- Next linked list is created for every executable statement, each node contains the various tokens forming the executable statement.



- This linked list is analyzed based on the different probabilities of the placement of tokens to form meaningful statements, this in turn leads to the creation of the various objects of the models defined above. The information extracted from the statements is used to fill the instance variable details of the objects created.

Eg: The creation of any class A can be done as any one of the following ways:

- class A{
- class A extends B{}
- private class A extends C implements I {}
- class A extends B implements I {}
  ....

The creation of any variable A may be done as any one of the following ways:

- int a;
- int a=1;
- private int a=2;
- protected int a;
- static int a=0;
- static int a=(2*3/4%5);...

These are just a few of the numerous ways by which a class or a variable may be created. Similarly possible to show numerous ways for functions and interface creation also.

- After this create different classes
  class _model
  variable_model
  Function_ model
  interface_model

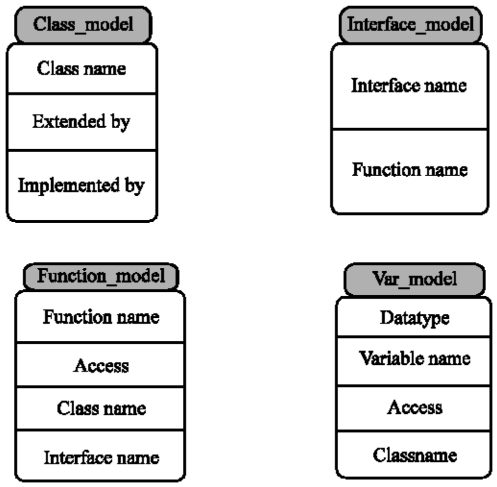
These objects of classes are used to store the various details of the classes, variables, functions and interfaces. The object of class_model has instance variables which store detail like the name of the class, the procedure followed during defining the class for implementation, i.e. it is either extended or implemented. The interface or the package in which the class is defined is also recorded for future references. The objects of the class_model are used to store detail of the various classes which are encountered during the execution of the program.

The objects of variable_model class are used to store the details of the various attributes related to a variable, like the name of the variable, the type of the variable, the access granted for the usage of the variable which might be public, private or protected and the last attribute is the class in which the variable is declared. The objects of this class are created when we encounter any variable declaration in the program.

The function_model forms an essential part of the program by keeping track of the details of the various functions which are created either in a class or an interface. The function name is recorded. This model stores the access granted for each function and it also stores if the statement scanned is just a function prototype or the complete function definition. One of the most attribute of any method is its signature, this model stores the signature of any function using a linked list, this linked list containing the signature is used in determining inheritance.

The interface_model objects are used to store the details of the interface name and the various function prototype which should be defined once a class implements the interface. The objects of this model are created once possible to scan any interface definition. The interfaces which are already defined in the JAVA development kit are also included in the array with the method names which are defined abstract e.g. When implement the

KeyListener interface, there should give the method definition for the methods keyPressed, keyReleased and keyTyped. Hence, the interface name KeyListener and the 3 mentioned methods are stored in an object of this class.



- Next, create four object arrays, one for storing the variable_model objects, one for class_model objects, third stack for storing function_model objects and the fourth for interface_model objects. These array objects serve as data banks by storing the objects containing the details. These are referred for checking the different kinds of errors.
- Once all the object arrays are made its ready for testing the various kinds of errors like an inheritance sensitive parser should connote. The following are errors

Class not defined error: This is one of the most common errors that people often do, while dealing with inheritance. This error can occur in 2 cases:

Missing override: Here a subclass specific implementation of a super class method is omitted. As a result, that super class method be incorrectly bound to subclass object and a state could result that was valid for the super class but invalid for the subclass owing to a stronger subclass invariant. This is also possible when a class extends a class defined in JAVA development kit but does not override the method necessary to inherit it. For example if a class extending the class thread does not override the run method then it may be called a missing override error.

If consider the following code snippet, we will see the missing override error.

```
class cls_object extends Thread{
    int i;
    cls_object(){
            i=5;
            }}
```

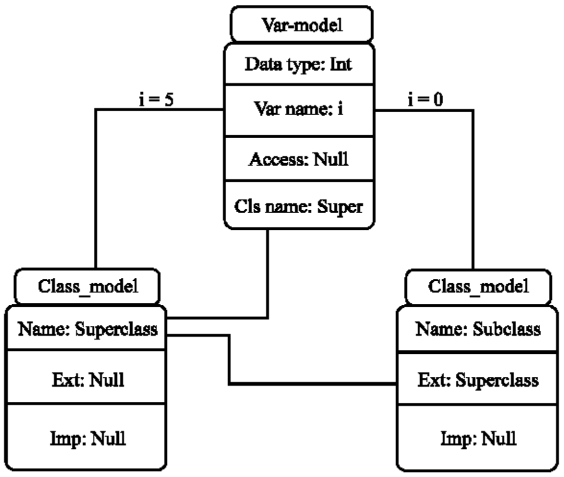
When we have a super class method defined private and some where down the inheritance hierarchy we try to override the method, then the function will not show polymorphism. In that case we should suggest the user to either remove the private modifier from the super class or provide his own method definition with the same signature as that of the super class method.

Incorrect initialization: Super class initialization is omitted or incorrect. Deep hierarchies may lead to initialization bugs. Determining how initialize is used in a subclass requires examination of the super class that defines new. The initialize method must be sent to super, not self. Suppose that new is refined and does not send initialize to self. Super's initialize will not be executed.

Naked access: A super class instance variable is visible in a subclass and subclass methods update these variables directly. This can be done in C++, JAVA, or objective C, where any public or protected base class data member will allow naked access. Naked access creates the same problems as unrestricted access to global data. Changes to the super class implementation can easily induce subclass bugs or side effects. Subclass bugs or side effects, in turn, can cause failures in super class methods. If a super class is changed, we should retest both the super class and its subclasses. If we change the subclass, we should retest both the subclass and the super class feature used in the sub class. If we consider the following code snippet

```
    class superclass{
      static int i=5;
      void display( ){
      System.out.println ("i value is " + i ); }}
      class subclass extends superclass { int j; subclass(
            i=0;
            j=1; }
            class prog{
    public static void main( String args[]){
superclass sp=new superclass( );
            subclass sb=new subclass( );
            sp.display( );
            }}
```

Here output may expect 5 but actually it will be 0 since the value will be changed during the constructor call while sub class object creation.



Possible to check for numerous inheritance related errors because of the implementation of graph theory in inheritance. The adjacency matrix provides us with an excellent understanding of inheritance till any depth. The rows and columns of the matrix represent the classes used in the program, be it user defined or the inbuilt classes, like the Thread class, etc. If a class subclass extends the class superclass then the matrix element (r,c) is 1 provided r is index of subclass in class array and c is index of superclass in the class array.

E.g.: If we consider the following code snippet :

```
class a{}
class b extends a{}
class c extends b {}
class d extends a {}
```

Then the adjacency matrix is of the form.

	a	b	c	d
a	0	0	0	0
b	1	0	0	0
c	0	1	0	0
d	1	0	0	0

CONCLUSION AND FUTURE WORK

In this researech whole JAVA program is an input and program is divided in to fragments example Classes. Every line of codes is analyzed by using compiler techniques and attributes, Methods are found. All gathered information about fragments is stored in knowledge base. From that inheritance property is checked. If there is inheritance, parents and children are traced. By using this information possible to developed a data flow analysis testing tool which diagnose the inheritance related bugs like incorrect initialization, missing override, naked access, naughty children, worm holes, spaghetti inheritance, fat interface.

The polymorphism related errors may be stated by obtaining the signatures of the various functions defined in a class. Whenever a function definition is encountered the signature linked list entity of the function_model array is analyzed and checked with the new encountered function's signature. On obtaining an object which shows the same function name but different signature as compared to the obtained function, occurrence of polymorphism is identified. In case we find two methods with the same signature and name but with different return type an error is reported. Finding property bugs in JAVA software for example Exception handling, multiple threading, testing generic classes in JAVA 1.5 are possible in future. This work focus on particular structure of the program. Testing for common structure is a g general difficulty.

Dataflow testing will useful during Software development and software maintenance.

REFERENCES

Alexander, R., 2001. Testing the Polmorphic Relationships of Object-Oriented Programs. Ph.D dissertation, George Mason University.

Binder, R., 1996. Testing Object-Oriented Software: Survey. J. Software Testing, Verifivation and Reliability, 6: 125-252.

Binder, R., 1999. Testing Object-Oriented Systems: Models, Patterns and Tools, Addison-Wesely.

Chun-Chia Wang and Wen C. Pai, An Automated Object-Oriented Testing for C++ Inheritance Hierarchy. Department of Information Management Kuang Wu Institute of Technology and Commerce.

Rountev, A., 2002. Dataflow Analysis of Software Fragments. PhD Dissertation Rutgers University, Available as Technical Report DCS-TR-501.