iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 1 **Issue:** V **Month of publication:** December 2013

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Vol. 1 Issue V, December 2013
ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Slicing of Software Components Using Object-Oriented Technique

Arvind Kumar[#1], Dr. Surender[*2]

[#] *Research Scholar*

Computer Science Engineering Department

JJT University, Jhunjhunu, Rajasthan, India

[*]*Department of Information Technology*

HCTM Technical Campus, Kaithal, Haryana, India

Abstract: Component-Based Software Engineering (CBSE) is an approach used to develop a large software system with the assembly of reusable software components according to the client-specific requirement. CBSE is based on reusable software component that can be replaced and updated easily. This paper extends the graph less technique proposed by Beszedes for slicing Object Oriented Programs. The proposed approach computes the dynamic slices of the OOPs especially in case of polymorphism. The proposed approach generate the defined-used chains of the objects and variables used in the program and then compute the slice using those defined-used information, then debug the program by detecting the various possible bugs and generating the suggestion messages which may remove the present bug. A GUI tool has been developed to compute and display the computed slice. Tool allows the user to browse the program and then generate the results. So using this technique, researcher selects the optimal components from component repository.

Keywords: Program slicing, Control Flow Graph, d-u chain

INTRODUCTION

The computation of slices of the program is called program slicing. Program slicing is a technique to extract program parts with respect to some special computation. In this area many research papers are presented, the idea of program slicing was proposed by Weiser in 1979 [1]. Program Slicing is to remove the irrelevant statements from the program code. Irrelevant statements are those statements to which the buggy statement is neither data dependent nor control dependent. Weiser found that the output of a particular execution of the program depends on the small portion of the source code and change in the rest of the program does not affect the output of the program. Weiser use Control Flow Graph (CFG) as the intermediate representation of the Program for computing Slices[1].

CFG can only represent the control dependency between the program statements. Ottenstein et. al. [11]. proposed a new graphical representation of the program called Program Dependency Graph(PDG) which represented both Control as well as Data dependency [11], which gave better results than CFG. One disadvantage of PDG was that it can only be applied on single-process procedures.

Horwitz introduced the System Dependence Graph (SDG) for computing Slice of the inter procedure program with procedure call [3]. Later Horwitz gave the better algorithm to compute

slices of inter procedure program with procedure call [12]. This algorithm used SDG for the computation of the slices.

Object oriented programs have different properties like class, objects, inheritance and polymorphism. These all features strengthen the use of object oriented programs but at the same time pose the challenge for slicing. These features cannot be represented in SDG. Larson and Harrold introduced a new kind of intermediate graph called as Class Dependence Graph (CLDG) for each class in an object-oriented program. This intermediate representation is able to represent object oriented features. Later Larson, B.A. Malloy presented OPDG (Object Oriented Program Dependence Graph) which was the extended version of the PDG with some more dependency edges and nodes so that it can represent object oriented features [4]. Since then many variation of the OPDG has been proposed by many researchers, Rothermel and Harrold proposed the class dependence graph (CDG) [5]. Larsen and Harrold proposed an extension of system dependence graph for object oriented software (ESDG)[2]. Liang and Harrold proposed extensions to ESDG for the purpose of object-slicing [6]. Harrold and Rothermel proposed a family of representations for object-oriented software: class hierarchy graph (CHG), class call graph (CCG), class control flow graph (CCFG), and framed graph[7].

Krishnaswamy [13] proposed a different approach to slicing object-oriented programs. He used another dependence-based representation called the object oriented program dependency graph (OPDG) to represent the object-oriented programs. The OPDG of an object oriented program represents control flow, data dependencies and control dependencies.

One important variation is proposed by Rajib Mall and Samanta, called Call Based Object Oriented System Dependence Graph (COSDG)[8]. This graph represented object oriented features, at the same time it also represented the structural features of the program. The graph was used for Object Slicing as well as for test coverage analysis.

Horwitz again introduced the new kind of slicing called call-stack sensitive slicing[9]. call stack is used for reducing the size of slice ,as call stack gives the procedure call which is active at the time of slicing. This helped in removing those procedure calls which are of no use in that particular execution of the program.

Although lots of graphical representation of the program has been proposed but generating the graph from program is in itself a very difficult task and then following dependencies to compute slice makes it more tedious. Arpad Beszedes and Tamas Gergely [10], proposed a new method to compute slices based on definition-use(d-u)chains. Since graphical representation of the program is not required so the method proposed is quite useful for computing slices.

In section 2 some background details are provided. Section 3 presents the proposed approach and algorithm for computation of slices. Section 4 presents an example to illustrate our approach in detail, the implementation and results are also presented in this section. Finally we draws some conclusion in section 5.

Background Information

When a statement assigns some value to some variable, that statement is defining that variable. And when it references some variable, it uses that variable. If some variable is defined by referencing to some other variable in the same statement, the definition of that variable is influenced by that use. a statement can uses and defines several variables, and it may be the case that a variable is defined by referencing to more than one variable, and on the other hand it may be that the use of a variable influences different definitions. For example in the following C language instruction [10],

$x = y + v; z = v;$

Use of variables y and v influence definition x, also the use of variable v influences definition z too in the same statement. A execution path from node n to node m is called to be definition-clear with respect to variable v, if none of the nodes of the execution path contain definition to v. The definition of variable v in node n reaches another node m, if there exists a definition-clear execution path with respect to variable v from n to m. When the definition of variable v at node n reaches a node m in which the same variable v is referenced, the definition of v in node n and the data-dependent node m is called a d-u pair for v and denoted by (n, v, m). The sequence of connected du pairs, in which each adjacent pair corresponds to a du pair, is referred to as a du chain.

Data flow analysis is used to collect information about the flow of data values across basic blocks. defined-used information is part of data flow analysis.

d-uchain = { S|S is < d,u,l > where d is object defined and u is object used, l is the line number }.

Objlist = {O|O is < n,c,t,l > where n is the name of the object, c is the class name, t is the type of object whether pointer or non pointer, l is the line number }.

Clslist = {C|C is < d,l > where d is the name of class or method defined in line number l}.

Methodlist = {M| M is < n,c,t,l > n is the name of the method , c is the class of the in which method is defined , t is the type of the object whether virtual or non virtual, l is the line number}.

IC = {node|node is < n,next > where n is the name of the class, and next is the pointer to the next node in IC (inheritance chain)}

Slicing Criteria: it includes the object name and the line number at which we want to compute Slice.

Proposed Algorithm

Algorithm proposed here uses local definition-use information for computing slices of object oriented programs. This algorithm mainly concentrated on the polymorphism case, it first identify the polymorphic call and then analyse the run time information of the object to identify the method which is to be inserted in the slice. In this proposed algorithm we use du-chain for data flow analysis.

The input to the algorithm are, P:a program and Slicing Criteria < O,l >where O is the name of the object and l is the line number, and output is S:slice of the program P.

Begin:

```
for ∀a ∈ duchain
  if (a(d) = O || a(u) = O)
    S = S ∪ a(l)
endfor
for ∀b ∈ objlist
  if (b = O)
    S = S ∪ a(l)
    Include_class_structure(b(c))
endfor
for ∀c ∈ duchain
 if(c(u) is method call && e(l) is in S)
   CP =check whether the object is Pointer
   CN = find class name of object used
 if(!CP || (CP && address assigned to the object is of its own class))
   for ∀f ∈ methodlist
    if(f(c) = CN && f(d) = c(u))
     S = S ∪ {all lines of this method}
   endfor

 else if(function is defined in superclass and is non virtual)
   S  = S ∪ {method in superclass}
 else if(sub class does not have definition of that method)
   S = S ∪ { method in superclass}
 else
   S = S ∪ { method in subclass}
endfor
```

```
Include_class_structure(str)
    for ∀ c1 ∈ clslist
      if(c1(d) = str)
       S = S ∪ {c1(l) line number of opening and closing brace of class c1(d)}
      if(c1(next) = NULL in IC)
          Include_class_structure(c1(next))
```

IMPLEMENTATION

The proposed algorithm is implemented in c# in .net framework 3.5 with the help of visual studio. The program is instrumented to take a program as an input, for every method call, program records the object on which the method is invoked. For each object program records the unique identifier representing that object. The pointer objects have given the different identity to differentiate them with non pointer objects.

The input program is given below and its computed slice with the help of Slicing tool is shown in figure 1. C++ language is used for the input program.

```
1 #include <iostream>
2 using namespace std;
3 class CPolygon {
4  protected:
5    int width, height;
6  public:
7    void set_values(int a,int b)
8     {
9       width=a;
10       height=b;
11     }
12    virtual int area()
13     {
14       return(0);
15     }
16  };
17 class CRectangle: public CPolygon
18 {
19  public:
20    int area()
21     {
22       int area;
23       area=width*height;
24       return(area);
25     }
26  };
27 class CTriangle: public CPolygon
28 {
29   public:
30    int area()
31     {
32       int area;
33       area=width*height/2;
34       return(area);
35     }
```

```
36  };
37 int main()
38 {
39   CRectangle rect;
40   CTriangle trgl;
41   CPolygon poly;
42   CPolygon *ppoly1;
43   ppoly1=&rect;
44   CPolygon *ppoly2;
45   ppoly2=&trgl;
46   CPolygon *ppoly3;
47   ppoly3=&poly;
48   ppoly1->set_values(4,5);
49   ppoly2->set_values(4,5);
50   ppoly3->set_values(4,5);
51   ppoly1->area();
52   ppoly2->area();
53   ppoly3->area();
54   return 0;
55 }
```

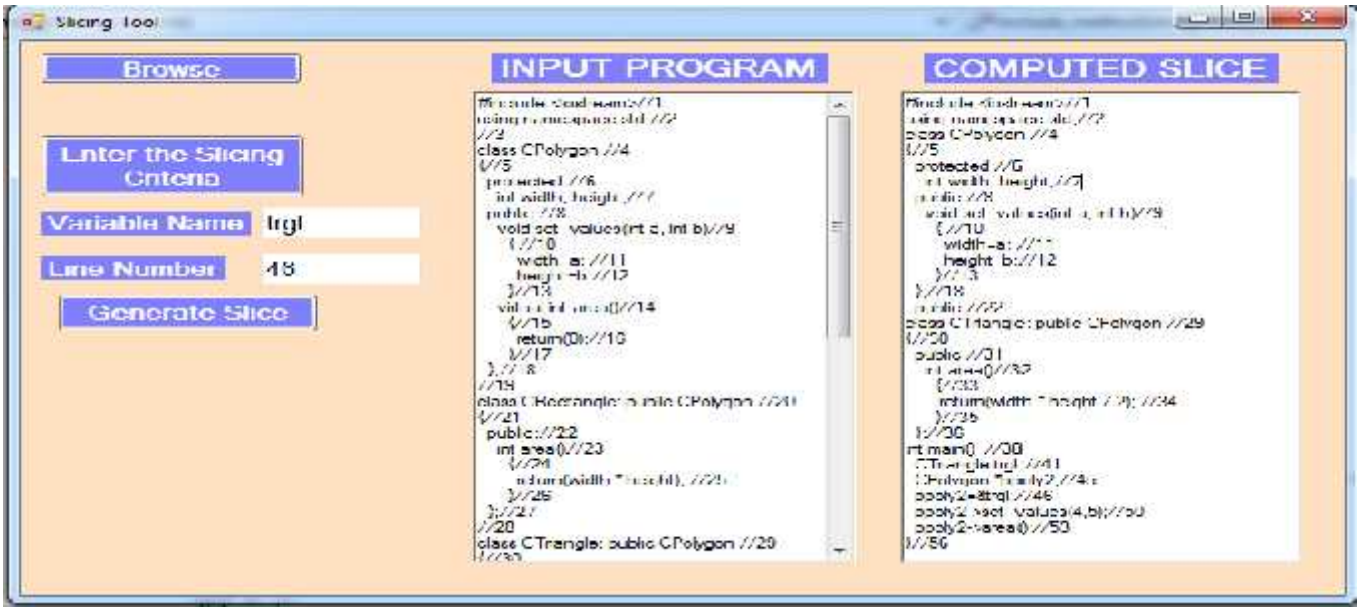


Figure 1. Example program and its computed slice

In fig 1. the slicing criteria entered is <trgl,46>. The computed slice is executable in itself and preserve the complete behavior of the object 'trgl'.

CONCLUSION

The past approaches use various dependence graphs for the intermediate representation of the program. These dependence graphs are very complex to generate. This paper proposed a new intermediate representation of the programs called d-u chains. An algorithm for the computation of slices of object oriented programs is also proposed, this algorithm is based on computing the dependencies between the statements by using definition-use information of the program statements i.e. d-u chains. Computing slice using d-u chains as the intermediate representation of the program makes it much easier, since the space complexity of this proposed data structure is much lesser. So using this approach, researcher can select the optimal component.

REFERENCES

[1] Weiser M.,: Program slicing, *IEEE Trans. Software Engineering, 1984,* Vol 16: pp 498-509

[2] Larsen L. and M. Jean Harrold.: Slicing object-oriented software. *In Proc. of the 18th International Conference on Software Engineering Berlin, Germany, Mar 1996*, pp 495–505

[3] Horwitz S., T.Reps, and D.Binkley, :Interprocedural slicing using dependence graphs, *ACM Transaction on Programming Languages and System Jan 1990*, Volume:12, Issue: 1, pp.26-60

[4] Malloy B. A., J. D. McGregor, A. Krishnaswamy, and M. Medikonda. :An extensible program representation for object-oriented software. *ACM SIGPLAN Notices Dec 1994*, vol 29, pp 38–47

[5] Rothermel G. and M. J. Harrold. :Selecting regression tests for objectoriented software. *In Proc. of the International Conference on Software Maintenance - 1994 Victoria, BC, Canada, Sep 1994*, pp 14–25

[6] Liang D. and M. J. Harrold. :Slicing objects using system dependence graphs. *In Proc. of the IEEE International Conference of Software Maintenance (ICSM '98) Bethesda, MD, USA, Nov 1998*, pp 358–367

[7] Harrold M. J. and G. Rothermel. :A coherent family of analyzable graphical representations for object-oriented software. *Tech. Report OSUCISRC- 11/96-TR60, Department of Computer and Information Science, The Ohio State University, Nov 1996*

[8] Najumudheen, E. S. F., R. Mall and D. Samanta, :A dependence representation of coverage testing, *Journal of Object Technology,2010*

[9] Horwitz S., B. Libit, and M. Polishchuk, :Better Debugging via Output Tracing and Callstack-Sensitive Slicing, *IEEE Transactions on Software Engineering, , January/February 2010*, VOL. 36, pp 1

[10] Beszedes A., T. Gergely and T. Gyimothy, :Graph-Less Dynamic Dependence Based Dynamic Slicing Algorithms, *In Proceedings of the Sixth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'06),2006*

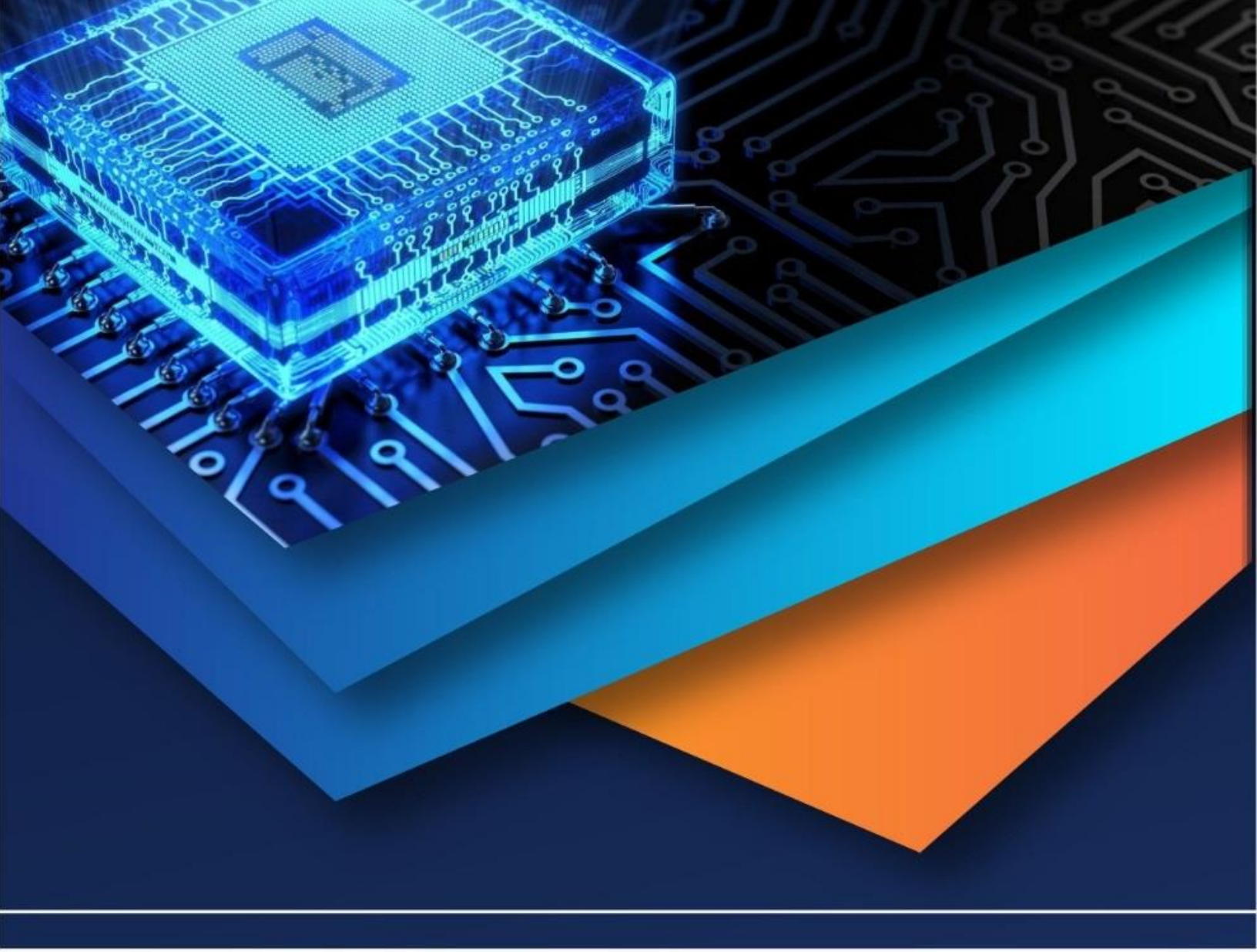
[11] Ottenstein K.J., L.M. Ottenstein :The program dependence graph in a software development environment, *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, April 1984*, pp.177-184

[12] Horwitz S., T.Reps, and D.Binkley, :Interprocedural slicing using dependence graphs, *ACM Transaction on Programming Languages and System, , Jan 1990*, Volume:12, Issue: 1 pp.26-60

[13] Krishnaswamy A.,: Program slicing An application of program dependency graphs. *Technical report,Department of Computer Science, Clemson University, August 1994*

[14] Duanzhi C., :Program slicing, *International Forum on Information Technology and Applications,2010*

[15] Prasad D., Mohapatra, R. Mall and R. Kumar, :An overview of slicing techniques for object-oriented programs, *Informatica 30,2006*



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY