Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Survey of Various Intermediate Representation in Software Testing

Manpreet Kaur*
Department of CSE
CGC, Gharuan(Mohali), India

Rupinder Singh
Assistant Proferssor, CSE
CGC, Gharuan(Mohali), India

Abstract—To ease the software visualization and to test the software product graphical notation is the key area. Graph tells the abstract view of software according to point of interest. Graphs tells the various object ,their attributes ,relations, intermediate dependencies with in a system or system to system .In this review we have discuss about various graphs and comparing them on set of features. It also helps to develop test cases at earlier phase of software development.

Keywords — Slicing, software testing, PDG, SDG, model dependency graph.

I. INTRODUCTION

Weiser was the first who introduce the concept of program slicing are now called static program slicing [2].Program slicing is a program analysis technique. It involves different activities like program understanding, debugging, testing, maintenance, optimization, complexity measurement etc. Weiser originally used a control-flow graph as an intermediate representation for slicing. It involves the extraction of statement of a program. Any line of a program that affect the values at some point, referred to as a slicing criterion. So, a slicing criterion consists of a pair < S, V >, where S is the statement number and V is a variable. The piece of a program that have a direct or indirect effect on the values computed at a slicing criterion < S, V > are called the program slice with respect to the slicing criterion < S, V > [1].
Alternatively, the slicing is a technique to identify program slice related to a slicing criterion. One can proposes that the slicing criteria is a key role in slice computation based on identifying dependence relations among the parts of the program. Various slightly different concepts of program slices have been proposed. There are many methods to compute the slices. The different applications require different properties of slices, thus different slicing techniques are produced. There are various types of graphs which are used as intermediate representation to produced slice.

II. GRAPHS

Weiser originally used a control-flow graph as an intermediate representation for slicing. There are different intermediate representations which can be used to compute the slice. Slice contains different features according to the intermediate representation. Following are different graphs which can be used as an intermediate representation:

A. Program Dependence Graph

The dependencies between the program statements are modeled by using PDG (Program dependence graph). PDG has two types of dependence edges: a data-dependence edge and a control-dependence edge. A data-dependence edge from (1) to (2) means that the computation performed in (2) depends on the value computed in (1).For Example

A=B*C (1)

D=A*E-l (2)

A control-dependence edge from (1) to (2) implies that (2) may or may not be executed depending on the result of (1), for instance, if-statement.

if(A) then (1)

B=C*D (2)

Endif

Program dependence graph have more than one edge between two vertices as shown in Fig 1. So, it is called multi-graph. When two vertices have more than one loop carried flow dependence edge then each is labeled by a different loop that convey the dependency. And when two vertices have more than one def-order edge then each is labeled by a vertex which is flow dependent on both definition that occur at the edge's source and edge's target.
Program Main ()


```
c =1;
while c<11 do
c =c+1;
end(c)
```

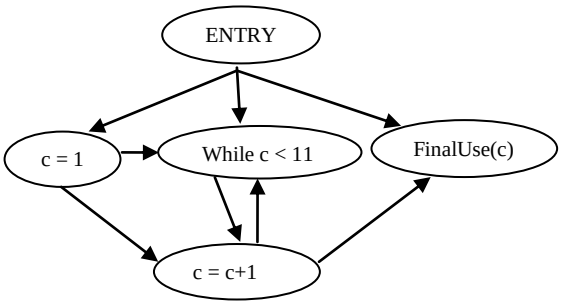


Fig 1: Program and its program dependence graph [3,4]

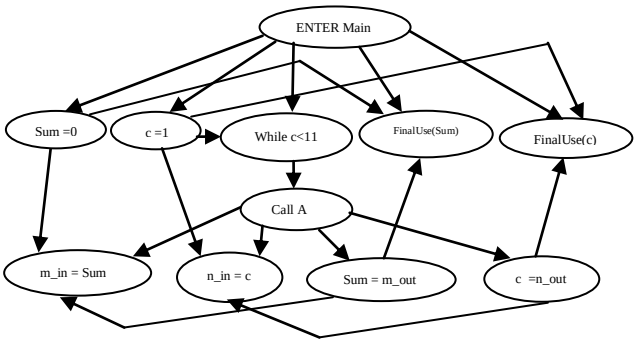


Fig 2: An example program and its SDG [5, 6]

B. System Dependence Graph

Inter-procedural dependences are modeled by a SDG (System dependence graph), a combination of PDG's. Each PDG represents the method or main() method in the class of the program. Additional arc shows direct or indirect dependence between a call site and the requested method and transitive inter-procedural data dependencies. Procedures have a well known entry node and a return statement without argument. The Fig 2 shows the system dependence graph of given example program.

```
Program Main ()                 procedure A(m, n)
Sum =0;                           call Add(m, n);
c =1;                              call Increment(n)
While c <11 do                    return
call A (Sum, c)
end (Sum, c)
```

C. Class Dependence Graph

The data dependence and control dependence of a distinct class is represented by ClDG(Class dependence graph). The PDG's are used to implement the method in ClDG, but the entry vertices of the PDG's are known as method entry vertices. A ClDG have a class entry vertex that is connected to entry method vertices by class member edges. Fig 3 shows the ClDG of classes SimpleCalc and AdvancedCalc. The class dependence edge which passes between them indicated inheritance. AdvancedCalc needs to be linked to its own specific data members and methods although it inherits all of the data members and methods belonging to SimpleCalc. The class dependence edge is traversed along the class membership data member edges of SimpleCalc to compute the inherited data members and methods.

D. Object Oriented Program Dependency Graph

The control flow, data dependencies and control dependencies in the object oriented program is represented by OPDG (Object Oriented program dependency graph). The OPDG representation of an object-oriented program is created in three stages, which are: Class Hierarchy Sub-graph (CHS), Control Dependence Sub-graph (CDS) , and Data Dependence Sub-graph (DDS). The CHS shows inheritance relationship between classes and the methods into a class. A CHS have a single class header node and a method header node for each method which is defined in the class. The static control dependence relationship which occurs within and among the different methods of a class is represented by CDS. The data dependence relationship among the statements and predicates of the program are represented by DDS. The OPDG of an object-oriented program is the union of three sub-graphs: CHS, CDS and DDS. The main advantage of OPDG is that the representation has to be generated only once during the entire life of the class. It does not require to be changed whereas the class definition remains unchanged. [8]

E. Call Graph

A call graph represents a static view of the relationship between object classes. In the call graph each node represent individual method and edges represent call sites. Though, a call graph does not provide important object-oriented concepts like inheritance, polymorphism and dynamic binding. Call graphs can be used for human understanding of programs, or used for further analyses that follow the flow of values between procedures. Call graphs used to finding procedures that are never called.

F. Object Oriented Dependency Graph

The ODG is a multi-diagraph which is extension of directed graph. The directed graph is extended by augmenting multiple edge types, vertex properties, and their relations. Due to object encapsulation, the ODG can avoid some dependencies.

G. Dynamic Dependence Graph

During program execution, the dynamic data and control dependences can be captured by dynamic dependence graph. The dynamic dependence graph (DDG) only have a fixed number of nodes similar to the basic blocks in the program. Dynamic data dependence graph of an example program shown in Fig 4. Dynamic dependences graph have the following dependence edges: Data dependence edge and Control dependence edge

```
y = a;                          1
while (y < 7) {                 2
y = y + z;                      3
if (y == 8)                     4
d(y);                           5
}                               6
```

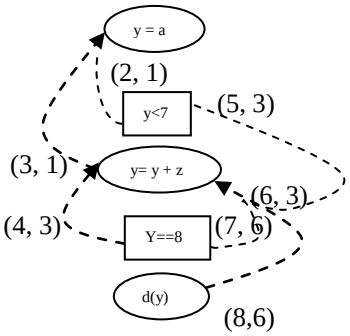


Fig 4: Dynamic data dependence graph for above program [9, 10]

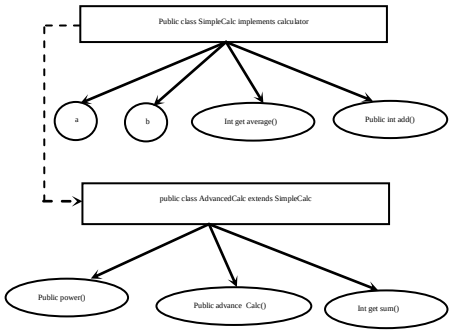
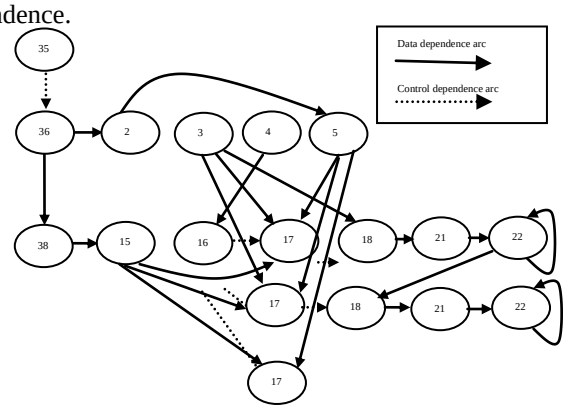


Fig 3: The ClDGs of the SimpleCalc and AdvancedCalc classes [7, 13]

H. Dynamic Object Oriented Dependence Graph

The DODG is an arc-classified diagraph (V, s), where V is the flow graph vertices, and s is the set of arcs representing dynamic dependencies and data dependencies between vertices. DODG is based on dynamic analysis of control flow and data flow of the program as shown in Fig 5. The execution of the statement or expression depends on control condition which is represented by control dependence. The data flow between statement and expression is represented by data dependence.



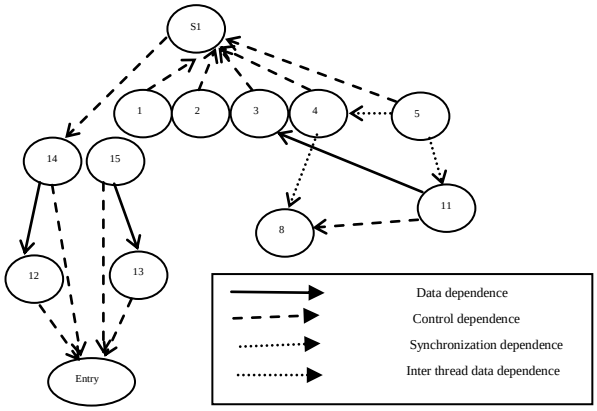


Fig 5: Dynamic Object Oriented dependence Graph [11] Fig 6: Concurrent Program Dependence Graph [12, 19, 20]

I. Object Program Dependence Graph

It is an approach produced by combining forward analysis and backward analysis. The forward process, it chose nodes on the OPDG and computes intermediate dynamic slices at the required points during the program execution. In the backward process, it traverses the OPDG to obtain the final dynamic slice. [1]

J. Extended System Dependence Graph

The extended system dependence graph is used to represent the object oriented programs with features like data hiding, inheritance, polymorphism etc. It is also known as a Class dependence graph (ClDG). A ClDG captures the control and data dependence relationships of the class without the information of calling environments. Every method in a ClDG is shown by a procedure dependence graph. The entry into the method is through method entry vertex. In a ClDG, each method entry is stretched by adding formal-in and formal-out vertices. Formal-in vertices are used for every formal parameter which is added and formal-out vertices for every formal reference parameter which is modified by the method.

K. Method Dependence Graph

The method dependence graph is equivalent to the program dependence graph. Based on static analysis of the produced bytecode, each node is represent method and edge represent their mutual calls. Method dependence graph have two

special types of dependence arc. Synchronization dependence arc and communication dependence arc: Synchronization dependence arc represent dependence relationships between different threads because of inter-thread synchronization and communication dependence arc represent dependence relationships between different threads because of inter-thread communication.

L. Multi-Threaded Dependence Graph

The Multi-Threaded dependence graph is combination of thread dependence graph (TDG) and some special kinds of dependence arcs to show thread interactions between different threads. The TDG is used to represent a single thread in a concurrent Java program and is equivalent to the SDG. The TDG is an arc-classified diagraph which contain number of method dependence graphs each representing a method, and some special type of dependence arcs to show direct dependencies between a call and the called method and transitive inter-procedural data dependencies in the thread.

M. Concurrent Program Dependence Graph

Program dependencies in a concurrent Java program is shown by concurrent program dependence graph (CPDG). The CPDG is a diagraph that have a collection of dependence graphs that represent Java methods, java classes and their extensions and interactions, interfaces and their extensions, packages, and complete programs respectively. It also have few additional vertices and arcs for parameter passing between different methods in a class and inter-thread synchronization and communication between different threads. Here Fig 6 shows the example of concurrent program dependence graph.

N. Control Flow Graph

A control flow graph (CFG) is a kind of flow chart which includes all path that is traversed during program execution. In this each node is the basic block, i.e. a straight-line piece of code without any jumps(End a block) or jump targets(Start a block) as in Fig 7. Jumps in the control flow are represented by directed edges. There are basically two blocks: the entry block (control enters) and the exit block (All control flow leaves).

```
M = c-d;
N= c-d;
If(n) {
M=M*M;
N=N*2;
}
Else {
M=M-1;
N=N*N;
}
O=M*N;
```

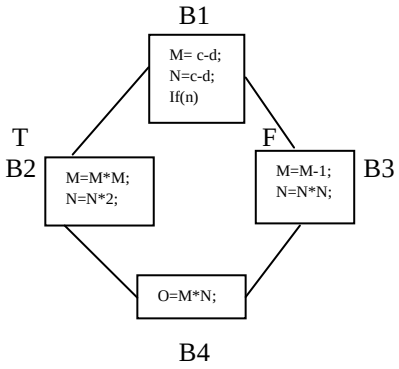


Fig 7: Control flow graph of program

O. Control Dependence Graph

A control dependence graph represents the control conditions necessary for a statement during execution of program. It also identifies those statements that are surely to execute when a given statement has executed. It consists of different types of nodes like statement node, predicate node, region node etc as in Fig 8. CDG is more flexible method of representing control dependencies than CFG. Here the example program given below.

```
S1.    read m
S2.    sum = 0

P3.     while m <= 4
S4.        read  n
P5.        if n >= 0 then
S6.           sum = sum + n
           endif
        endwhile
S7.   print sum
```

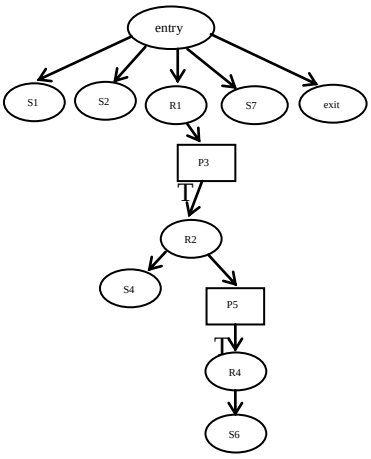


Fig 8: Control dependence graph of program

P. Task Synchronization Reachability Graph

TSRG insistently explain synchronization and concisely represents the execution for single task. TSRG gives the behaviour of an entire concurrent program and is constructed from the TGS's of the task that create the program. [1]

Q. TSRG based program dependence Graph (RPDG)

Dependencies in RPDG have property in transitivity, which is not present in traditional program dependence graph (PDG) in which dependencies are defined between statements. Dependence analysis is more precise in RPDG due to accurately detecting synchronization activities. So, RPDG used in many different software engineering activities like program understanding, slicing, debugging, optimisation, complexity measurement, maintenance and so on. [14]

R. Concurrent Control Flow Graph

A concurrent CFG (or CCFG) is a CFG that consist of fork and join nodes having expression. Fork and join nodes start and collect groups of threads as shown in Fig 9. Control flows out all edges leaving a fork and starting a collection of threads which will wait at a same join node before continuing. Fork and join can be nest however control cannot pass between threads. Specially, all paths from a individual fork must meet for a first time at a unique join. [15]

S. Concurrent System Dependence Graph

A concurrent system dependence graph (CSDG) of a concurrent object-oriented program P is a directed graph (N; E) where each node n∈ N represents a statement in P. For x, y ∈ N, (m, n) ∈ E iff one of the following holds:

1. n is control dependent on m. That edge is called a control dependence edge.
2. n is data dependent on m. That edge is called a data dependence edge.
3. n is synchronization dependent on m.

That edge is called a synchronization dependence edge.

4. n is communication dependent on m. That edge is called a communication dependence edge. [21]

T. Dynamic Multi-threaded Dependence Graph (DMDG)

The DMDG is an arc-classified diagraph (S, a), where S is the multi-set of flow graph vertices, and a is the set of arcs which representing dynamic control and data dependencies, synchronization and communication dependencies between the vertices. [19]

U. Distributed Program Dependence Graph (DPDG)

Let P be a distributed C++ program, and Pi be a sub program of P. P is represented using a set of DPDGs. The distributed program dependence graph GDi of the component-program Pi is a directed graph (Ni, Ei) where each node n represents a statement in Pi. For m, n 2 Ni, (n, m) 2 Ei iff any one of the following holds:

1. n is control dependent on m. That edge is called a control dependence edge.
2. n is data dependent on m. That edge is called a data dependence edge.
3. n is fork dependent on m. That edge is called a fork dependence edge.
4. n is communication dependent on m. That edge is called a communication dependence edge. [1]

V. Distributed Control flow Graph (DCFG)

A distributed control flow graph (DCFG) of a program is a flow graph, where each node represents a statement of program, and each edge represents potential control transfer among the nodes. In this the Start and Stop are two unique nodes that represent the entry and exit nodes of the program respectively. There is a directed edge representing a control flow from node to node.

W. Java System Dependence Graph (JSysDG)

A JSysDG is a multi-graph that represents the control and data dependencies between the statements of a Java program. Statements are classified as whether they devote to either the structure of a program means they are headers representing methods, classes, interfaces and packages or the program's behaviour means they refer to a method body. Graph based program representation can be produced by merging different dependence graphs [13].

X. Interface Dependence Graph

The interface dependence graph (InDG) includes an interface entry vertex. The interface entry vertex is associated to a set of method entry vertices, which further associated to parameter, that describe their input parameters. Every method entry vertex is associated to the method entry vertex of the method implementing it through an implement abstract method edge. The class is associated to the interface through an implements edge, when it implements an interface.

Y. Package dependence graph

A package describes a set of classes that are conceptually identical or are devoted to a identical purpose. This is described as a package dependence graph (PaDG). Packages are essential in case of slicing, because they are required to precisely compute data visibility. A package entry vertex shows the package, which is associated to each class and interface entry vertex refer to the package through a package member edge. [13]

Z. Model dependency graph

The structural as well as the behavioural aspects of a modelled system can be represented by Model dependency graph (MDG). The MDG is created by combining the behavioural model information along with the structural information

using an incremental process. An MDG have different types of nodes that can either correspond to model elements based on the classes and their relationships or objects and their interactions as shown in Fig 10. Each dependencies shown in an MDG has a pair of classes, attributes, operations and its parameters and return values associated with it. All such CA, AT, OA, PR, and RT nodes that represent classes and its features using the class information to the MDG. Model based graph that represents modular design and show different path or all path [16].

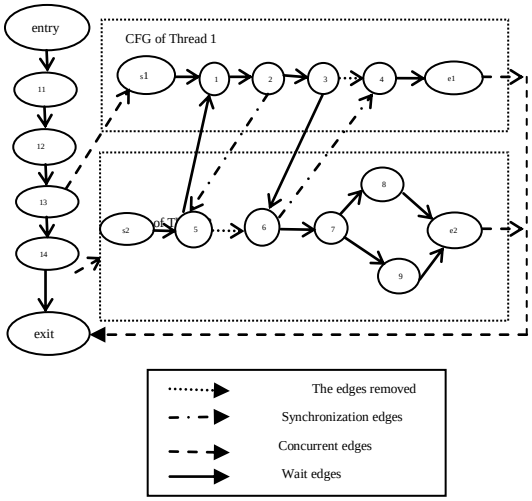


Fig 9: Concurrent Control Flow Graph [20]

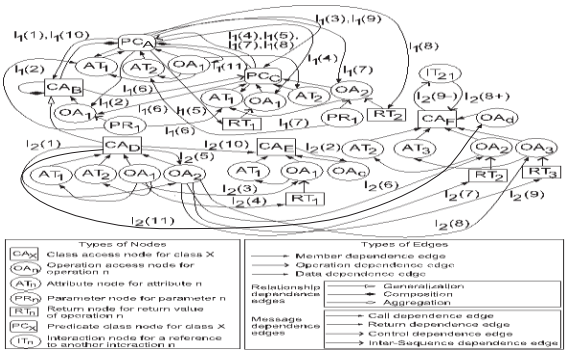


Fig 10: MDG of generic system [16]

TABLE I
COMPARISON OF DIFFERENT GRAPHS

Sr. no.	Name of graph	Platform Independent	Slicing	Complexity	Coverage Criteria	Practical Implementation	Kind of Testing	Usage
1.	Program Dependence Graph[3,4]	Yes	Yes	No	Data and control dependency	Possible	Static	Compiler Optimisation, program understanding, debugging, testing.
2.	System Dependence Graph[5,6]	Yes	Yes	Yes	Data, control and inter-procedural dependency	Possible	Static	Inter-procedural slicing
3.	Class Dependence Graph[7]	Yes	No	Yes	Data and control dependency	Possible	Static	Represent Object Oriented program
4.	Object Oriented Program Dependency Graph[8]	No	Yes	Yes	Data and control and control flow dependency	Possible	Static	Slice object oriented program, program debugging, testing tool etc.
5.	Call Graph	Yes	No	No	Caller-Callie relationship	Possible	Both static & dynamic	To understand and analyse program
6.	Object Oriented Dependence Graph	No	Yes	No	Data and control dependency	Possible	Static	Slice object oriented program

7.	Dynamic Dependence Graph[9,10]	Yes	Yes	No	Dynamic Data and control dependency	Possible	Dynamic	space and time efficient algorithm for dynamic slicing, improving software quality and security
8.	Dynamic Object Oriented Dependence graph[11]	Yes	Yes	No	Dynamic Data flow and control flow dependency	Possible	Dynamic	To compute dynamic Slice of object oriented program
9.	Object Program Dependence Graph	Yes	Yes	Yes	Data and control dependency	Possible	Both static & dynamic	To compute dynamic slice
10.	Extended System Dependence Graph	Yes	Yes	Yes	Data and control dependency	Possible	Dynamic	To compute Slice of object oriented program
11.	Multi-Threaded Dependence Graph	No	Yes	Yes	Transitive inter-procedural data dependencies	Possible	Static	Developing and understanding large scale software system in java.
12.	Method Dependence Graph	N	Yes	Yes	Synchronization and communication dependency	Possible	Static	To compute static slice of concurrent java program
13.	Concurrent Program Dependence Graph	No	Yes	Yes	Synchronization and communication dependency	Possible	Static	To compute static slice of concurrent java program
14.	Control Flow Graph	Yes	Yes	No	Static control flow relationship	Possible	Static	To generate test data
15.	Control Dependence Graph	Yes	No	No	Control and Control flow dependency	Possible	Static	To generate test data
16.	Task Synchronization Reachability Graph[14]	No	Yes	No	Synchronization dependency	Possible	Static	Determine synchronization activities in a program.
r17.	TSRG based Program Dependence Graph[14]	No	Yes	Yes	Transitive dependency	Possible	Static	Program understanding, slicing, optimization, maintenance
18.	Concurrent Control Flow Graph[15, 20]	No	Yes	No	synchronization and communication dependency	Possible	Static	Static slices of concurrent Java programs can be computed efficiently.
19.	Concurrent System	No	Yes	Yes	synchronization and	Possible	Dynamic	To computing dynamic

	Dependence Graph[21]				communicatio n dependency			slices of concurrent object-oriented programs.
20.	Dynamic Multi-threaded Dependence Graph	No	Yes	Yes	Data, control, synchronizati on and communicatio n dependency	Possible	Dynami c	To compute dynamic slice with a monitor like synchronizati on primitive.
21.	Distributed Program Dependence Graph	No	Yes	No	Data, control, fork and communicatio n dependency	Possible	Dynami c	To compute dynamic slice with fast response time
22.	Distributed Control Flow Graph	No	Yes	No	Control flow dependency	Possible	Dynami c	To develop slicing tool
23.	Java System Dependence Graph[13]	No	Yes	Yes	Data, control and call flow analysis	Possible	Both static & dynami c	Used to develop software engineering tool.
24.	Interface Dependence Graph[13]	No	No	No	Interface dependency	Theoretical	Both static & dynami c	To implement interface.
25.	Package Dependence Graph[13]	No	Yes	No	Data and control dependency	Theoretical	Static	To compute data visibility and slicing.
26.	Model Dependency Graph[16]	Yes	Yes	Yes	Structural and behaviour aspect	Possible	Both static & dynami c	To compute slice

III. CONCLUSIONS

Software testing is one of the most favourable and complex area in today's software development. In today's scenario all the test cases are developed at early stages of development where we have only architecture of software. To test the architecture in semantic and syntactic way, there is need of some intermediate graph. After analysis the various graphic techniques for software testing, we concluded that model dependency graph is one of favourable and most suited graph. It tells the both behavioural and structural dependences of the object and their attributes. We can further use this graph to ease the visualization and testing of software with the help of slicing. It can also be extended further in the region of concurrent programming.

REFERENCES

[1] Mohapatra,Durga Prasad, Rajib Mall, and Rajeev Kumar, "An overview of slicing techniques for object-oriented programs", Informatica (Slovenia) 30.2 (2006): 253-277

[2] Mark Weiser, "Program Slicing", 352 IEEE Transactions on Software Engineering, vol. SE-10, no. 4, July 1984

[3] Ferrante Jeanne, Karl J. Ottenstein, and Joe D. Warren. "The program dependence graph and its use in optimization", ACM Transactions on Programming Languages and Systems (TOPLAS) 9.3 (1987): 319-349

[4] Horwitz, Susan, Thomas Reps, and David Binkley, "Interprocedural slicing using dependence graphs", ACM Transactions on Programming Languages and Systems (TOPLAS) 12.1 (1990): 26-60

[5] Zhao Jianjun, and Martin Rinard, "System dependence graph construction for aspect-oriented programs", 7HFKQLFDO 5HSRUW 0 7 (2003)

[6] Susan Horwitz, Thomas Reps, David Binkley, "Interprocedural Slicing Using Dependence Graphs", Proceeding of the ACM SIGPLAN 88 conference on programming Language Design and Implementation, 1988

[7] Kovács, Gyula, Ferenc Magyar, and Tibor Gyimóthy, "Static slicing of java programs", University, 1996

[8] Anand Krishnaswamy, "Program Slicing : An Application Of Object-Oriented Program Dependency Graph", thesis

[9] Xiangyu Zhang, Rajiv Gupta, "Cost Effective Dynamic Program Slicing", Proceeding of the ACM SIGPLAN 2004 conference on programming Language Design and Implementation, page no. 94-106 , 2004

[10] Christian Hammer, Martin Grimme, Jens Krinke, "Dynamic Path Conditions in Dependence Graphs", Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, page no. 58-67, 2006

[11] Zhao Jianjun, "Dynamic slicing of object-oriented programs", Wuhan University Journal of Natural Sciences 6.1-2 (2001): 391-397

[12] Zhao Jianjun, and Bixin Li., "Dependence-Based Representation for Concurrent Java Programs and Its Application to Slicing", Proceedings of ISFST2004, Xian, China (2004)

[13] Walkinshaw Neil, Marc Roper, and Murray Wood, "The Java system dependence graph", Source Code Analysis and Manipulation, Proceedings of third IEEE International Workshop on. IEEE, 2003

[14] Qi Xiaofang, and Baowen Xu, "Dependence analysis of concurrent programs based on reachability graph and its applications", Computational Science-ICCS, Springer Berlin Heidelberg, pp.405-408, 2004

[15] Edwards Stephen A., "Method and apparatus for converting a concurrent control flow graph into a sequential control flow graph", U.S. Patent No. 7,100,164. 29 Aug. 2006

[16] Lallchandani Jaiprakash T., and Rajib Mall, "Slicing UML architectural models", ACM SIGSOFT Software Engineering Notes 33.3 (2008)

[17] Kagdi Huzefa, and Jonathan I. Maletic, "Onion Graphs for Focus Context Views of UML Class Diagrams", Visualizing Software for Understanding and Analysis, 4th IEEE International Workshop on IEEE, 2007

[18] Sindre Guttorm, Bjørn Gulla and Håkon G. Jokstad, "Onion graphs: aesthetics and layout", Visual Languages, Proceedings 1993 IEEE Symposium on. IEEE, 1993

[19] Zhao Jianjun, "Multithreaded dependence graphs for concurrent java programs", Software Engineering for Parallel and Distributed Systems, proceedings International Symposium on. IEEE, 1999

[20] Chen Zhenqiang, and Baowen Xu, "Slicing concurrent java programs", ACM Sigplan Notices 36.4 (2001): 41-47

[21] Sahu Madhusmita and Durga Prasad Mohapatra, "A Node-Marking Technique for Slicing Concurrent Object-Oriented Programs", (2009)

[22] Chen Jian-Liang, Feng-Jian Wang, and Yung-Lin Chen, "An object-oriented dependency graph for program slicing", Technology of Object-Oriented Languages, TOOLS 24 proceedings IEEE, 1997

