International Journal of Computer Science and Informatics

Volume 4 | Issue 2 Article 4

October 2014

DYNAMIC SLICING OF ASPECT-ORIENTED UML COMMUNICATION DIAGRAM

ALINA MISHRA
Department of Computer Science and Engineering, NIT Rourkela, alinamishra88@gmail.com

SUBHRAKANTA PANDA
Department of Computer Science and Engineering, NIT Rourkela, 511cs109@nitrkl.ac.in

DISHANT MUNJAL
Department of Computer Science and Engineering, NIT Rourkela, dishantmun@nitrkl.ac.in

DYNAMIC SLICING OF ASPECT-ORIENTED UML COMMUNICATION DIAGRAM

ALINA MISHRA[1], SUBHRAKANTA PANDA[2], DISHANT MUNJAL[3]

[1,2,3]Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India
E-mail: [1]alinamishra88@gmail.com, [2]511cs109@nitrkl.ac.in, [3]dishantmun@nitrkl.ac.in

Abstract- Aspect-oriented Programming (AOP) is a recent programming paradigm that focuses on modular implementations of various crosscutting concerns. The typical features of AOP such as join-point, advice, introduction, aspects etc. make it difficult for applying slicing. Program slicing is a method to extort statements that are relevant to a particular variable. Program slicing, as initially defined by Weiser is based on static dataflow examination on the flow graph of the program. Program slicing has been applied in parallel processing, program debugging, program integration,program understanding , software testing and maintenance etc.In this paper, we proposed a technique for dynamic slicing of aspect-oriented software based on the UML 2.0 Communication diagram. Here, we can use slicing at the architectural level to find the slicing of communication diagram at an earlier stage of software development. To represent the various unique features of AOP in UML diagrams is very difficult and complex in nature. For this purpose, we first draw the communication diagram for aspect-oriented program considering the classes, pointcuts, advices and aspects. Next, we have generated an intermediate representation from the communication diagram. Then, we proposed an edge marking dynamic slicing algorithm that traverses the intermediate representation and finds the slices for the communication diagram of the given problem scenario. The novelty in our approach is that we present the communication diagram for the aspect-oriented software. Also, we proposed an algorithm to find the slice from it. The advantage of using communication diagram is that it mainly focuses on the interaction between the objects involved in the problem scenario.

Keywords- *Communication diagram; Aspect-oriented Programming (AOP); Communication Aspect Dependency Graph(CADG)etc.*

I. INTRODUCTION

Program slicing was introduced by Weiser [1] is supported by the fact that programmers have various abstractions regarding the program during debugging. Program slicing is a program disintegration technique focusing on selecting statements significant to a variable or computation, even if they are scattered all over the program. A program slice composed of statements extracted from the original program based on the slicing criterion. A program slice can be classified as static or dynamic. A static slice contains every statement of a program that might influence the value of a variable at a point of interest for all possible inputs. On the other hand, a dynamic slice [4] contains only those statements which truly have an effect on the value of a variable at a particular point of interest for a defined set of inputs. A foremost target of any slicing technique is to recognize as small a slice with regard to a slicing criterion as possible because smaller slices are more useful for various applications [15]. To a large extent the literature on slicing is concerned with improving the algorithms for slicing in terms of sinking the size of the slice and improving the effectiveness of the slicing algorithm. Generally the slices are calculated from the source code of a program i.e. in the coding phase of software development. A substitute of this approach is to derive slices from the design specifications using UML models. Therefore, the slices can be derived at the analysis or design stage. This has the benefit of computing the slices at an early stage of in the software development cycle, thereby building the design components more reusable. In addition to this, it also makes automatic code generation achievable for AOP systems with advanced levels of separation of concerns at the generated code. There are a number of applications of slicing [4, 8] in the different context of software development: Debugging can be automated using backwards program slicing starting with the point at which fault is detected as the slicing criteria. Program Comprehension is greatly needed whilw maintaining the legacy systems. For this purpose, we can use both backward and forward slicing to browse the code, looking for dependence between code units, flow of data between program statements etc. Also, in case of Testing forward or backward slicing can be used to find the regression test cases. Also, slicing can be used to locate the statements that are impacted by changes at any other point of program. Aspect-oriented programming (AOP) is an emerging programming paradigm that focuses on modular implementation of cross-cutting concerns [9] such as exception handling, synchronization, security, data access, logging. This technique was first introduced by Kiczales et al. [6]. Representation of such cross-cutting concerns by use of standard language constructs gives poorly structured code because these concerns are tangled with the crucial functionality of the code. This considerably increases the complexity and creates difficulty in maintenance. AOP [2, 3] intends to resolve this difficulty in maintenance by allowing the programmer to build up cross-cutting

concerns as complete stand-alone modules called aspects. The central idea behind AOP is to build a program by unfolding each concern separately. Aspect-oriented programming languages provide unique opportunities for program analysis schemes. For instance, to implement program slicing on aspect-oriented software, definite aspect-oriented features such as join-point, advice, aspect, introduction must be taken care of appropriately. Even though these features adds great strengths to model the cross-cutting concerns in an aspect-oriented program, they also introduce difficulties to analyze the program. In this paper, we use a communication diagram to capture the dynamic interactions between objects. A communication diagram consists of objects and associations between them that represents communication between objects [5]. One benefit of considering communication diagram is that it does not take into account the timing aspect of the interaction, rather it emphasizes on objects and the communication between them. Also, communication diagram is compact in size since timeline is not used in it. In case of sequence diagram due to timeline the objects are needed to be placed on the right side of the diagram which makes it unwieldy. Hence, a communication diagram is a useful extension of the object diagram. Object diagram shows the snapshot of a system at any point of time, while communication diagram appends the dimension of time and provides the snapshot of the system at various points of time. Therefore, we have used the communication diagram for analyzing the dynamic behavior of the system.

For a given problem scenario, we first draw the communication diagram. Once we are done with the diagram, the diagram is converted into an intermediate representation, which we named Communication Aspect Dependency Graph (CADG). In the next step, the CADG is traversed according to the algorithm proposed on the basis of marking and unmarking of the dependence edges in the graph. The algorithm is named Aspect-Oriented Edge Marking Algorithm (AOEM). The traversal of the graph is based on the slicing criterion. As a result, we obtained the required slice from the architectural model at a higher level of abstraction. The rest of the paper is organized as follows: Section II describes an overview of the basic concepts used in this technique. In Section III, we provide details to generate the dependency graph CADG from the Communication Diagram. Section IV describes the proposed algorithm and the working of the algorithm with an example of IssueTicket usecase in Online Railway Reservation System. In this section, we also compute the slice for all nodes in the dependency graph. Section V analyses the time complexity and the space complexity of AOEM algorithm and compares it with other's work. Section VI concludes our work and suggests some future work to be taken up.

II. BASIC CONCEPTS

This section gives a brief overview about some basic definitions and terminologies related to the intermediate representation and the working of the algorithm. Aspect: An aspect is defined as a crosscutting type, which can have a similar form of class declaration [7]. These are also known to be units of modular implementations of crosscutting concerns and consists of advices, pointcuts and general Java member declarations. Advice: This is a method like construct that is required to define crosscutting behavior. Advice is classified into three types in AspectJ: before, after, around. On a defined joinpoint After advice executes after the program proceeds with that joinpoint. Before the program proceeds with that joinpoint Before advice executes. Around advice executes when the joinpoint is reached and has a explicit control over the program, whether the program proceeds with that joinpoint or not.UML Communication Diagram: UML communication diagram is used to show the communication between the different objects involved in a given problem scenario. The ordering of the messages is shown with the help of numbering technique. In order to draw the communication diagram, we identify the objects that are participating in the scenario. Then all objects are connected using a link. Through the link the messages are sent between the objects to carry out the given scenario. In our work, we have considered aspect-oriented based communication diagram to compute the dynamic slice at the architectural level. As we know, aspects in AOP are similar to classes in OOP, we have represented the objects of the aspects similar to OOP. In the example of aspect-oriented (woven) communication diagram shown in Figure.1. we have considered three base classes (Passenger, ServiceController and TrainDatabase), one Aspect (AccessController), and one Around advice (login). The aspect in the scenario has a vital impact on the security of the system as it allows only the authenticated users for the enquiry process. The method p_information() of base class passenger act as joinpoint through "pass" pointcut designator. We have represented the pointcut i.e. the first message in dotted line from base class to aspect. At this point of time, the message is sent and the execution is temporarily stopped until acknowledgement is received. After the passenger information are validated the control moves back to the suspended base class and the execution resumes for enquiring the train details. Communication Aspect Dependency Graph (CADG) We define Communication Aspect Dependency Graph (CADG) as a directed graph with (N, E), where N is a set of nodes and E is a set of edges. CADG demonstrates the dependency of a node under consideration on the others. In this case, a node represents either a message or a note in the communication diagram and edges represent either control or data dependency among nodes. The CADG

is used to represent the dynamic dependencies among the messages in the communication diagram for IssueTicket usecase in Online Railway Reservation System. Considering the communication diagram of *IssueTicket* usecase as shown in Figure.1, its CADG is shown in Figre.2.

RecentDef (x): For all variable x, RecentDef (x) gives the pointer to the node or label number of the node equivalent to the latest definition of the variable under consideration. Each time a variable in the message is redefined, the RecentDef(x) for the variable changes.

d_Slice(n): For every node n of the CADG C_G i.e. message m of the communication diagram, d_Slice(n) stores the dynamic slice with respect to the most recent execution of the node n. This is calculated at the run-time of the algorithm. We define the dynamic slice d_Slice as follows:

$d\_Slice(n) = \{v_1, v_{2...}v_k\} \cup d\_Slice(v_1) \cup d\_Slice(v_2) \cup \ldots \cup d\_Slice(v_k)$.

III. COMMUNICATION ASPECT DEPENDENCY GRAPH

In this section, we present our CADG representation of UML 2.0 Communication diagrams for aspect-oriented software. CADG demonstrates the dependency between the messages that are passed linking various objects in a given problem scenario. Dependencies considered in the CADG are classified into two different types namely data dependency edges and control dependency edges. We have used different notations to represent the different types of edges in the Communication Aspect Dependency Graph.

CADG is confined to represent only the dynamic dependencies existing among various objects concerned in the scenario. Dynamic dependencies show a discrepancy with time (example data dependencies).

The flow of data among the messages that are passed between objects in the communication diagram are represented by Data dependence edges of CADG. Additionally, it shows the effect of the calling messages on the return value of that call. Control dependence edges of CADG represents flow of control in the communication diagram.

We follow the steps mentioned below to construct CADG from communication Diagram:

1. Draw the communication diagram for the given scenario in the system.

2. Represent each message of the communication diagram as node in CADG and label it with the message number.

3. Identify the different control dependencies and data dependencies existing between the messages passed among the objects in the communication diagram and represent them as edges in CADG.

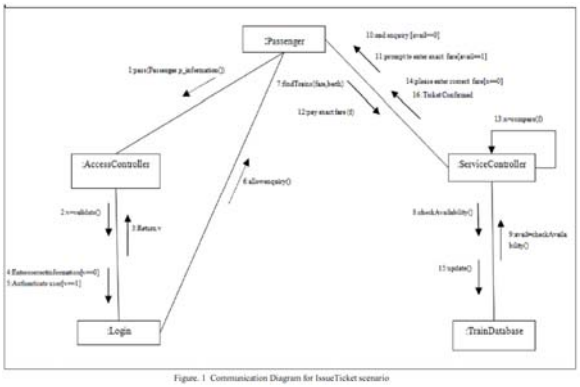

Figure. 1 Communication Diagram for IssueTicket scenario

A. An Example of CADG:

To explain the construction of CADG, we have considered the IssueTicket() scenario of Online railway Reservation System. In the first step, we identified the objects concerned with the scenario and the communication among them. Then we draw the communication diagram as shown in Figure.1.

From the communication diagram, the CADG is drawn as shown in Figure.2. Each message is represented by a node. The label of the messages in the communication diagram is identical to the node number in CADG. The edges of the graph represent the dependencies among the nodes.

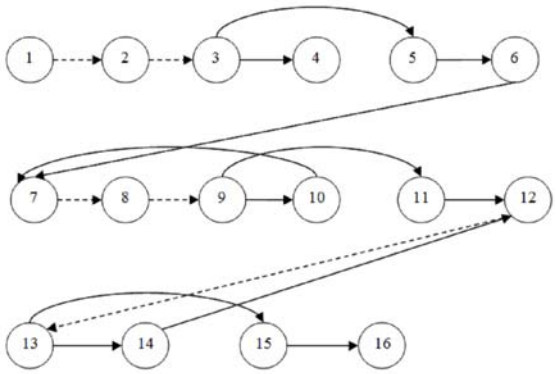

Figure.2. Communication Aspect Dependency Graph for Figure.1

The control dependence edges are represented by solid edges and data dependence edges are represented by dashed edges. After each execution of a message the graph is updated to find the recent dependencies.

IV. ASPECT-ORIENTED EDGE MARKING ALGORITHM (AOEM)

This section presents our proposed algorithm (AOEM) to compute the slice at the architectural level using the aspect-oriented based communication diagram. Our proposed algorithm works in three steps: CADG Construction, Initialisation and Traversal of CADG. Let the node n in CADG correspond to the message m in communication diagram CG . During execution of CG, let d_Slice(n)

denote the dynamic slice with respect to variable x, for the most recent execution. Let (n,v1), (n,v2)…(n,vk) be all the marked edges in the CADG after the execution of message corresponding to node n. Then the dynamic slice for the current execution of the statement corresponding to node u with respect to variable x is given by:

d_Slice(n) = {v1, v2…vk} d_Slice(v1) d_Slice(v2) …
d_Slice(vk).

Once a slicing criterion is specified, the AOEM algorithm directly produces the dynamic slice with respect to any given slicing criterion. The slice is computed by looking up at the corresponding dynamic slices computed during run-time. We now discuss our AOEM algorithm for communication diagram in the pseudo code form.

Algorithm: AOEM

Step1: CADG Construction:

1. For each message m create a node n in CADG
2. CADG = constructCADG(C_G) //Call a procedure for CADG construction

Step2: Initialisation:

1. Unmark all the data and control dependence edges of CADG.
2. Mark each control dependence edge (x,y) for which x is not a loop control node.
3. Set d_Slice = ϕ, for each node of CADG
4. Set RecentDef(x) = ϕ, for every variable x of CADG

Step3: Traversal of CADG: Traverse the Communication Diagram sequentially with the given set of values and do the following after each message *m* of a communication diagram is processed.

1. for each variable *x* used at *n* do the following:
   a. Unmark the marked dependence edge associated with the variable *x* corresponding to the previous execution of message *m*.
   b. Mark the data dependence edge (n,v) where v = RecentDef(x).
2. end for
3. Let $(n,v_1),(n,v_2),\ldots,(n,v_k)$ be the marked dependence edges in updated CADG, then Update d_Slice(n) = $\{v_1,v_2,\ldots,v_k\}$ ∪ d_Slice(v_1) ∪ d_Slice(v_2) ∪…∪ d_Slice(v_k).
4. end for
5. If n = Def (x) _message then Update RecentDef (x) = n;
6. If n is a loop control node then
   a. Mark each dependence edge (x, n) if the current execution of n is the entry of the loop.
   b. Unmark each incoming dependence edge if the current execution represents an exit of loop.
7. If m = terminate_message
8. exit;
9. else continue;

A. Working Of The AOEM Algorithm

We now demonstrate the working of AOEM algorithm with the help of our previously discussed example in Section III. Consider the communication diagram of IssueTicket scenarioin Figure.1 and the corressponding CADG shown in Figure.2. While execution of the initialization step, AOEM algorithm first unmarks all the dependence edges and set d_Slice(n) = , for every node of CADG. Now consider the following set of values: passenger_details = valid, avail=1, f= yes. For these set of given values, the model will have the execution trace as 1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16. Let us assume that a slicing command (16, ticket) is given. This command requires us to find the backward dynamic slice for the ticket variable at node 16. According to the AOEM algorithm, the dynamic slice at statement 16 is given by the expression d_Slice(16) = 15 d_Slice(15). By evaluating the expression in a recursive manner, we can get the final dynamic slice at statement 16. The shaded vertices in Figure.3 represents the dynamic slices with respect to ticket variable at node 16.B. Complexity Analysis of AOEM Algorithm Space Complexity: Each message in the Communication Diagram will be represented by a single vertex in CADG. A graph with n nodes require O(n2) space. So, the space requirement for the CADG is O(n2). In addition, we also need the following additional run-time space for manipulating the CADG:

1. At most O(n) space is require to store the d_Slice(m) for every message of the communication diagram CG. So, for n messages, the space requirement for d_Slice(m) becomes O(n2).

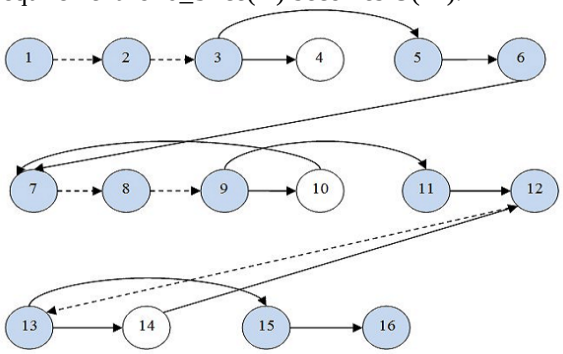

Figure.3 Updated CADG showing d_Slice(16) for Figure.1.

2. To store the RecentDef(var) for every variable var of C_G, at most O(n) space is required.

So, the space complexity of the AOEM algorithm is $O(n^2)$, where n is the number of messages in the communication diagram C_G.

Time Complexity: The total time complexity of the AOEM algorithm is found considering two factors: one is due to the execution time requirement for the run-time maintenance of CADG and the second one resulted from the time requirement to calculate d_Slice(n). The time requisite to store the required information at each node is O(n), where n is the number of messages in the communication diagram. The time considered necessary for traversing the CADG and reaching the specified nodes is $O(n^2)$, where n is the number of messages in the communication diagram. Thus, run-time complexity of the AOEM algorithm for computing the dynamic slice is $O(n^2S)$, where S is the length of the execution.

V. IMPLEMENTATIONS AND RESULTS

In this section, we briefly share the implementation of our AOEM algorithm. The main motivation for our implementation is to validate the correctness and the preciseness of our algorithm. We have tested our algorithm on different input programs with various executions and slicing criterion. We have coded our algorithm in JAVA. First we are generating Communication Dependence Graph (CADG) from the communication diagram. We have taken a node in the CADG for each message in communication diagram. These nodes are connected with the help of dependency edge. The dependency edge represents two types of dependencies: data dependency and control dependency. We are storing the whole information about the CADG in a file. While storing the different information about the CADG, we store the data in structure data type. This data structure contains various information about each node in the CADG like control dependency, data dependency, status: marked or unmarked. The snapshot for d_Slice(16) is shown in Figure.4.

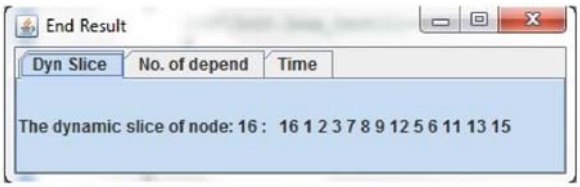

Figure. 4. Dynamic slice of Figure.1 with respect to node 16.

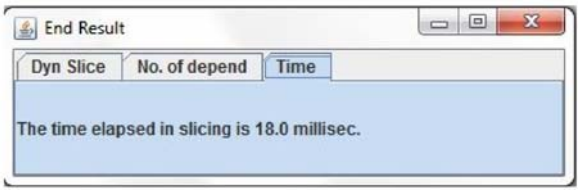

Figure. 5. Snapshot showing the elapsed time to calculate d_Slice(16)

VI. COMPARISONWITH RELATEDWORK

To the best of our knowledge no one has done slicing of aspect based UML Communication Diagram. In the absence of any directly related work, we compare our work with the work based on slicing of object-oriented UML models and code based slicing of aspect-oriented software.Korel et al. [4] proposed an algorithm for slicing of state-based models. The architectures used in their procedures do not make a distinction between the structural and behavioral aspects of a system. Also, it was designed for OOP. In our approach, we have considered the dynamic nature of the aspect-oriented system. Mohapatra et al. [17] proposed an algorithm called Trace Based Dynamic Slice TBDS algorithm. This algorithm computes the dynamic slice from the code of aspect-oriented programs using a dependence based graph called Dynamic Aspect-Oriented Dependence Graph DADG. Our algorithm intends to find the dynamic slice from the design stage using communication diagram at architectural level. Zhao [11,12] initially proposed a static slicing algorithm for AOP through aspect mining technique. Then, it was extended by constructing Sequence Dependence Graph for aspect-oriented programs. But this technique was unable to handle the pointcut. In our approach, CADG is able to find the dynamic slice as well as considers the pointcut feature of aspect-oriented programming. Braak [13] extended the technique proposed by Zhao [11,12] by including the Inter-type declarations in the graph. Then the two phase algorithm proposed by Horwitz et al. [16] was used to find the static slice of the aspect-oriented program. But our algorithm finds the dynamic slice for aspect-oriented programs at the architectural level. Lallchandani et al. [10] proposed an algorithm to find the dynamic slice combining class diagram and sequence diagram. The representation is named as Model Dependency Graph (MDG) for Object-oriented programs. Whereas we have considered the communication diagram for aspect-oriented programs.

VII. CONCLUSION

We have used UML 2.0 aspect-oriented based communication diagram to compute the dynamic slices. We have developed a Communication Aspect Dependence Graph (CADG) as an intermediate representation. Then, we propose an algorithm Aspect-Oriented Edge Marking Algorithm (AOEM). We have shown that the time complexity of our algorithm is O(n2S), where S is the length of the execution. The space complexity of our algorithm is O(n2), where n is the number of messages in the communication diagram. We have also proved that our algorithm computes the correct dynamic slice. We have implemented the algorithm to demonstrate their correctness. Our current work can be extended to compute dynamic slices of other UML 2.0 diagrams like activity diagrams, state-chart diagrams, interaction overview diagrams. Computing dynamic slices of a combination of two or more UML 2.0 diagrams like communication diagram and class diagram, communication diagram and activity diagram etc. is another direction for extension.

REFERENCES

[1] M. Weiser,“Programmers uses slices when debugging” Communications of ACM, 25(7), July 1982, pp.446-452.

[2] Aspect-Oriented Programming. www.wikipedia.org.

[3] AspectJ. www.eclipse.org/aspectj.

[4] B. Korel, I. Singh, , L. Tahat, and B. Vaysburg, ” Slicing of state-based models”, In Proceedings of the International Conference on Software Maintenance, pp. 34–43, 2003.

[5] A. Abdurazik and J. Offutt, “Using UML Collaboration Diagrams for Static Checking and Test Generation,”The

third international conference on the UML'00, New York, pp.383-395, October 2000.

[6] G. Kiczales, J. Irwin, J. Lamping, J. M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar. Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming(ECOOP), Finland, June 1997.

[7] Priya Gupta, Sushil Garg, and K.S.Khalon. Designing aspects using various UML Diagrams in resource-pool management. International Journal of Advanced Engineering Sciences and Technologies (IJAEST), 7(2):228-233, 2011.

[8] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar, "An edge marking dynamic slicing technique for object-oriented programs," In Proceedings of 28th IEEE Annual International Computer Software and Applications Conference, IEEE CS Press, pp. 60-65, September 2004.

[9] G. Kiczales and M . Mezini. Aspect-Oriented Programming and Modular Reasoning. In Proceedings of the 27th International Conference on Software Engineering, ICSE'05, May 2005.

[10] Jaiprakash T. Lallchandani and R. Mall. A dynamic slicing technique for UML architectural models. IEEE Transactions on Software Engineering, 37(6),December2011.

[11] J. Zhao. Slicing Aspect-Oriented Software. In Proceedings of 10th InternationalWorkshop On Program Slicing, pp. 251-260, June 2002.

[12] J. Zhao and M. Rinard. System Dependence Graph Construction for Aspect-Oriented Programs. Technical report, Laboratory Of Computer Science, Massachusetts institute of Technology, USA March 2003.

[13] Timon ter Braak. Extending Program Sliccing in Aspect-Oriented Programming with Inter-Type Declarations. InProceedings of 5thTwente Student Conference on IT, June 2006.

[14] F.Tip. A Survey of Program Slicing Techniques. Journal of Programming Languages, 3:121-189, 1995.

[15] Durga Prasad Mohapatra. Dynamic Slicing of Object-oriented programs. PhD Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, 2005.

[16] S.Horwitz, T.Reps, and D.Binkley. Interprocedural Slicing Using Dependence Graphs. ACM Trans. On Programming Languages and Systems, 12(1): 26-61, January 1990.

[17] D.P.Mohapatra, M. Sahu, R.Kumar, and R.Mall. Dynamic Slicing of Aspect-Oriented Programs. Informatica, 32(3): 261-274, 2008.