International Journal of Computer Science and Informatics

Volume 4 | Issue 2 Article 3

October 2014

ASPECT MINING USING UML COMMUNICATION DIAGRAM

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

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

DISHANT MUNJAL
Department of Computer Science and Engineering, NIT Rourkela, 111cs0609@nitrkl.ac.in

ASPECT MINING USING UML COMMUNICATION DIAGRAM

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

[1,2,3]Department of Computer Science and Engineering, NIT Rourkela
E-mail: 511cs109@nitrkl.ac.in, alinamishra88@gmail.com, 111cs0609@nitrkl.ac.in

Abstract- Aspect-Oriented Programming (AOP) focuses on precise constructs for modularization of the crosscutting concerns of a program. Crosscutting concerns can be defined as the functionalities that navigate the principal decomposition of software and therefore cannot be assigned to a single modular unit. Aspect mining attempts to find and isolate crosscutting concerns dwelling in legacy systems which in turn help in the adoption of an aspect-oriented design. Functionalities originally scattered across different modules and tangled with each other can be factored out into a distinct, separate unit, called an aspect. Identifying and refactoring the existing system into AOP considerably ease the debugging, testing and maintenance of the large legacy system. The goal of this paper is to find the aspects at the design level using aspect mining techniques for already existing non-aspect applications. The main advantage of this approach is that without understanding the underlying code, we can separate the crosscutting concerns at the architectural level. The novelty in our approach is that we are finding the aspects of an existing system at the architectural level using UML Communication Diagram. Also, the number of nodes in Control Flow Graph (CFG) drawn from the existing Communication diagram is reduced after finding the aspects resulting in the new reduced CFG.

Keywords- *Concerns, Scattered, Tangled, Object,Fan in.*

I. INTRODUCTION

Aspect-oriented programming (AOP) is a recent programming paradigm that targets crosscutting concerns: characteristics of a software application that are tough to segregate, and whose implementation is stretched across several modules [1]. This notion was first introduced by Kiczales et al. [2]. Different examples of aspects comprise user interface, logging, security, data storage, threading, exception handling [3] etc. The important objective of AOP is to facilitate dealing with crosscutting concerns of a software application as independently as possible. Due to inherent complexity of software, it is ineffective for dealing with many concerns which have a crosscutting impact on the components of the system [4]. As a new opportunity to reduce complexity, AOP offers a paradigm for modularizing such crosscutting concerns [5]. The weaving method of AOP automatically combines primary concerns and crosscutting concerns into an executable intact. Aspect mining aims to identify the aspect opportunities in existing and non aspect-oriented system [6]. Different authors have presented various automated code mining techniques, normally referred to as aspect mining techniques, which are capable to discover crosscutting concerns in the source code [7]. The purpose of these techniques is to present an outline of the source code entities that participate in a particular crosscutting concern. A significant difficulty with such crosscutting concerns is that they influence the understandability of the software application, and therefore reduce its evolvability and maintainability. To begin with, crosscutting concerns are complex to understand, because their implementation can be spread over different packages, classes and methods. Next, in the occurrence of cross-cutting concerns, regular concerns become challenging to understand as well, since they get tangled with the crosscutting ones.

In this paper, we have focused on two issues while providing support for modeling crosscutting concerns. The first step is to identify the several concerns that cut across different modules and then understand where and how these concerns cut across. Lack of this information, makes it very hard to represent and explain simultaneously on the subject of the crosscutting structure and the behavior in the system. Secondly, we provide a way to differentiate crosscutting from non crosscutting concerns and encapsulate the former into aspects.We have used a communication diagram to demonstrate our approach. A communication diagram is comprised of objects and their associations that demonstrates the communication between the objects [8]. The advantage of using communication diagram is that it mainly focuses on the objects and its interaction and not on the sequencing of the messages. The messages are sequenced by use of numbering techniques and not on the basis of the timeline as it is in sequence diagram. For this purpose, we have considered the communication diagram of UML 2.0 for the IssueBook usecase of Library Management System (LMS). In the first step, we generate the Control Flow Graph from the Communication Diagram. Next, we used program slicing technique to identify the messages with similar functionality by analyzing the dependency in the execution trace and objects involved in the communication of the messages. Then, we have merged similar functions into one node in the CFG. Hence, the new representation obtained is smaller than the original one. To maintain the information about the merged nodes we have

stored it in a table at the time of merging. We named the new representation as Concern Graph for Communication Diagram (CGCoD). In the next step, we have used Fan in analysis to compute the concerns in the existing system. It is done by calculating the Fan in metric [9], [10] of each node in the control flow graph obtained from

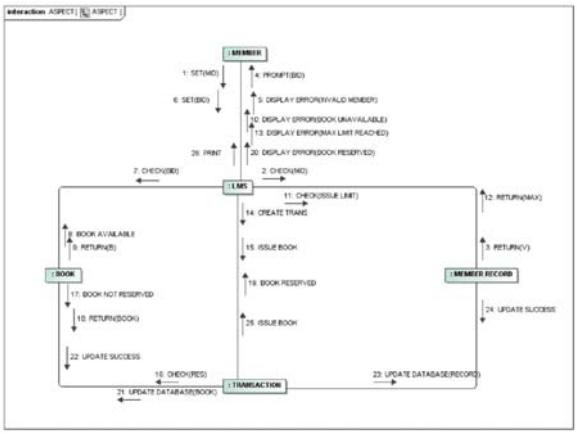

Fig. 1. Communication diagram for IssueBook scenario of LMS

the communication diagram in previous step. Once the concerns are identified, we check the crosscutting property of the concerns in order to define it as an aspect. Thus, we have successfully separated the crosscutting concerns from the architectural model at a higher level of abstraction. The rest of the paper is organized as follows: Section 2 gives a brief overview of the basic concepts used in our technique. Section 3 gives the detail steps to generate the concern graph CGCoD from the Communication Diagram. Section 4 describes the proposed algorithm and the working of the algorithm with an example of IssueBook usecase in Library Management System. In this section, we also compute the aspects considering the crosscutting features of the concerns. In Section 5, we compare our work with other existing and related works. Section 6 concludes the paper and specifies some work to be carried out in future.

II. BASIC CONCEPTS

In this section, we explain some basic definitions, notations and terminologies associated with the working of our proposed approach.

Program Slicing: The idea of program slicing is based on the principle of obtaining an executable subset of the program under consideration with respect to the slicing criterion. A slicing criterion is given as hs; vi, where s is the statement number and v is the variable that is either defined or used at s. Thus, the slice of the program with respect to hs; vi consists of all those statements that either gets affected by the value of variable v at s or affect the value of v. On the basis of direction of influence, slicing can be computed either in the forward direction or in the backward direction of the slicing criterion. A program slice that is obtained by the static analysis of the program is known as static slicing. A static slice contains all components of a program that may affect the variable specified in the slicing criterion and does not depend upon any specific input to the program. But, by analyzing the runtime information of an input to the program, we can obtain a much smaller slice in comparison to the slice obtained by static analysis. The slice thus obtained is known as dynamic slicing and is valid for a single input to the program. The slicing criterion to compute the dynamic slice of a program is given as hs; v; Ii, where s is the statement number, v is the variable at s and I is the input given to the program. Program slicing can thus be applied in a number of fields such as program debugging, testing, program comprehension, maintenance, etc. As the object-oriented software has increased in size and complexity, high level architectural designs have become very useful in comprehending large systems.

In this regard, Unified Modeling Language (UML) is most widely used to construct the architectural models of large and complex software. It has a wide range of options to model the various aspects of a system. A system can be easily comprehended if the interactions among its objects (behavioral aspect) can be under- stood properly. Therefore, communication diagrams are useful to handle, analyze and comprehend these interactions among the objects. Many researchers have applied program slicing techniques on UML diagrams [11], [12] for understanding, testing, reengineering and reuse of large software architectures. Our slicing algorithm traverses the edges of the proposed inter- mediate graph to identify and separate the messages into their equivalence classes.

Aspect-oriented Programming (AOP): Aspect-Oriented Programming provides specific language mechanisms to explicitly capture the crosscutting structure. To better support the expression of crosscutting design decision, AOP uses a component language to describe the basic functionality of the system and an aspect language to describe the different crosscutting properties [13]. The components and the aspects are then combined into a system using an aspect weaver. The aspect weaver makes it possible for an advice to be activated at an appropriate join point during run time. Thus a source code is modified by inserting aspect specific statement at join point.

Aspects: An aspect is an ordinary feature that's usually scattered across methods, classes, object hierarchies, or even whole object models. An aspect is a crosscut- ting type, defined by aspect declarations. Aspects may have methods and fields just like any other class. They may also contain pointcut, advice, and introduction (inter-type) declarations.

Aspect mining: Aspect mining is an approaching area of research, where we need to find the crosscutting concerns from the legacy system. To find we can use various aspect mining techniques like Fan in analysis, identifier analysis and dynamic analysis. This leads to more modularized software products and ease the work of maintenance of already existing non-aspect legacy systems.

Fan in analysis: Fan in analysis is intended to identify crosscutting concerns whose implementation is scattered over various elements, and composed of method invocations. Fan in of a message m (represented as a node) is defined as the number ofdistinct message bodies that can invoke m. Due to polymorphism, one method call can affect the fan in of several other methods. Our proposed technique intends at identifying such messages by calculating the Fan in metric for each message using the Concern Graph for Communication Diagram (CGCoD). It depends on the inspection stating scattered, crosscutting functionality that most influences the modularity. It is likely to produce high Fan in values for key messages employing this functionality.

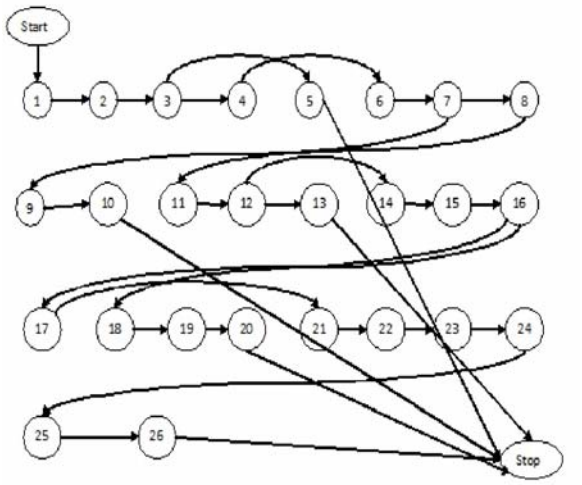

Fig. 2. Control Flow Graph for Figure 1.

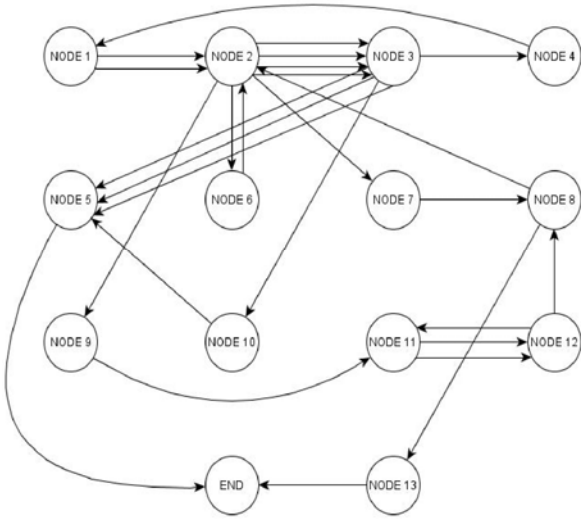

Fig. 3. CGCoD for the Communication Diagram in Figure 1.

TABLE I
TABLE TO SHOW DATA ABOUT THE NODES OF CGCOD

ORIGINAL NODE	*MESSAGE*	*NEW NODE*
1, 6	*set*	*NODE*1
2, 7, 11, 16	*check*	*NODE*2
3, 9, 12, 18	*return*	*NODE*3
4	*prompt*	*NODE*4
5, 10, 13, 20	*displayerror*	*NODE*5
8	*bookavailable*	*NODE*6
14	*createtrans*	*NODE*7
15, 25	*issuebook*	*NODE*8
17	*booknotreserved*	*NODE*9
19	*bookreserved*	*NODE*10
21, 23	*updatedatabase*	*NODE*11
22, 24	*updatesuccess*	*NODE*12
26	*print*	*NODE*13

III. INTERMEDIATE REPRESENTATION

In this section, we describe our Concern Graph for Communication Diagram (CGCoD) representation of the UML Communication Diagram. CGCoD represents the reduced graph that is obtained from the Control Flow Graph (CFG) after finding the concerns. The first step is to draw the communication diagram for the given problem scenario. There are different tools available for drawing the communication diagram like Rational Rose, Magic Draw, AgroUML [16], Rational Software Architecture etc. We have used MagicDraw [17] to draw the communication diagram needed for our paper. Next step is to generate the Control Flow Graph from the Communication diagram. For this purpose, we have considered each message passing between the objects as one node. Then the dependencies among the nodes are found and are represented as edges. In order to find the concerns, we have stored the execution trace and separated the functions having similar functionality. The messages having similar functionality are then combined to form one node in the CGCoD. For maintaining the information about the nodes that are merged in the previous step, we have used TABLE I to store the information.

A. An Example of CGCoD To explain the construction of CGCoD, we have considered the IssueBook() scenario of Library Management System (LMS) system. In the first step, we find the objects concerned with the scenario and the communication among them. We have used MagicDraw tool to draw the communication diagram as shown in Figure 1. Using the communication diagram, we have drawn the CFG as shown in Figure 2. All the messages are represented by a node. The node number is equal to the label of the messages in the communication diagram. The edges of the graph represent the dependency among the nodes. Once the CFG is constructed, the dynamic slicing algorithm is used to isolate the similar functions. The dynamic slicing is based on the criteria that the message passed from one object to the other have same name and perform similar functions in the given problem scenario. The algorithm initially does the parsing and then by using token analysis finds the message with the same name. The node numbers corresponding to the message with

the same name are grouped together into equivalence classes to form a single node in CGCoD. Simultaneously, a table is constructed at the run time to store the original node number, message name and modified node number as shown in TABLE I. From TABLE I, by using the new node number an intermediate representation is generated. This representation is named as CGCoD. In CGCoD, the new node numbers represent the nodes and the edges are drawn based on the dependency among the messages communicated among the various objects concerned in the given problem scenario. The advantage of CGCoD over CFG is that the number of nodes in CGCoD is less than CFG. As a result, the space required to store the CGCoD decreases. The CGCoD for the above problem scenario is shown in Figure 3.

IV. ASPECT MINING PROCEDURE FOR COMMUNICATION DIAGRAM

This section details the procedure to find the aspects from non aspect-oriented systems at the architectural level. This is achieved by a three step process: CGCoD Construction, Identification of Concerns and Finding Aspects. In the first step, we draw the communication diagram for a given scenario by identifying the objects and communication messages involved between them. We then derive the control flow graph by considering each of the messages as a node and the dependency among them as edges. We derive the CGCoD by applying dynamic program slicing technique. In the second step, we analyze the CGCoD to find the various concerns that are present in the given scenario. Here, we have used the Fan in analysis on the intermediate representation to identify the concerns. In the third step, we classified those concerns that can be designed as aspects. To find that whether a concern is an aspect or not we have analyzed the scattered and tangled property of each concern identified in the previous stage. Here, we draw a Concern Scattering Graph (CSG) to check the crosscutting features of the identified concerns. In the CSG, concerns form the top layer of nodes and objects form the bottom layer of nodes. An edge exist from a node in upper layer to the nodes in lower layer only if the concern is realized by the objects in the graph. This graph is then used in phase 3 of our proposed algorithm. Our algorithm named Node marking Aspect Mining for Communication Diagram (NMAMCoD), identifies the scattering and tangling property among the various concerns in the scenario.

Algorithm NMAMCoD

Input: Communication Diagram
Output: Aspects
Notations used:
M - Set of messages in the communication diagram.
N - Set of nodes in CGCoD.

Step 1. Construction of CGCoD

a. $CFG = constructCFG(C_G)$ //Call a procedure for CFG construction
b. $\forall\ m \in M$, Do the following
   i. Identify the tokens from the messages in Communication Diagram
   ii. Group the messages into equivalence class using any standard slicing algorithm.
c. Assign an unique node number to each of the equivalence class.
d. $CGCoD = constructCGCoD(CFG)$ //Call a procedure for CGCoD construction

Step 2. Identification of Concerns

a. $\forall\ n \in N$, Do the following
   i. find the fan_in Metric
   ii. $Threshold\_fan\_in = (\sum fan\_in\,(n))\,/n$
   iii. $Concern = \{n \in N | fan\_in(n) > Threshold\_fan\_in\}$

Step 3. Finding of Aspects

a. for each concern found in Step 2, do the following
b. find all the objects with which the concerns are linked.
c. $CSG = constructCSG()$ // call a procedure to construct CSG
d. Finding crosscutting concerns:

   Let c1, c2 ϵ Concern, c1 $\neq$ c2. we say c1 crosscuts c2 if
   i. $Card(f(c1)) > 1$
   ii. $\exists\ o\ \in f\,(c1) : c\ \in g\,(o)$
e. Express each crosscutting concern identified in Step 3(d), as an aspect.

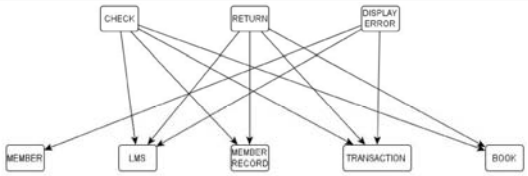

Fig. 4. Concern Scattering Graph for Figure 1.

A. Working of NMAMCoD Algorithm

We illustrate the working of the algorithm with the help of an example. Consider the communication diagram IssueBook usecase of a LMS given in Figure 1. The CFG for the communication diagram is shown in Figure 2. The slicing algorithm is implemented after separating the tokens. Let us consider the second row of Table I, where nodes 2, 7, 11, 16 are combined to form NODE 2. The concern for which this merging is done is *check*. Then the CGCoD is drawn as shown in Figure 3. Next, the Fan_in of each node is calculated. For NODE 2 the calculated Fan_in is 4. Likewise, the Fan_in of each node is calculated and the Threshold_Fan_in is calculated by using the formula given in the proposed algorithm, which is coming to be 2. The nodes having Fan_in value of more than 2, are separated as concerns. In the next step, the CSG is drawn considering all the identified concerns and the objects involved with it, as shown in Figure 4.

Next, we identify the scattering and tangling between the concerns to find the crosscutting concerns. Crosscutting occurs when a concern is spread over various objects and at least one of the object is tangled. In Figure 4, we have two domains related to each other through mapping. One domain has a set of three concerns *check*, *return* and *displayerror*. The other domain has four objects *member*, *lms*, *memberrecord*, *transaction* and *book*. The concern *check* is mapped to four objects *lms*, *memberrecord*, *transaction* and *book*. So the cardinality of the concern *check*, $Card(check) = 4$. Therefore, we can say that the concern *check* is scattered over four objects. We now need to prove that any one of the four objects is tangled. Let us pick up any one object, for example *lms*, over which *check* is scattered. On analyzing the CSG, we find that the object *lms* is also mapped with another concern *return*. Thus, the object *lms* is tangled with two concerns *check* and *return*. Therefore, the intersection of the set of mapping for both the concerns *check* and *return* is not equal to null.

$\Rightarrow f(c1) \cap f(c2) \neq \Phi.$

$\Rightarrow f(check) \cap f(return) \neq \Phi.$

Hence, we can say that the concern check crosscuts concern return. Therefore, check is a crosscutting concern and a probable candidate to be extracted as an aspect. Similarly, the result obtained for other two concerns reveal that return and displayerror are also identified as Aspects.

V. IMPLEMENTATION AND RESULTS

In this section, we briefly enlighten the implementation of our algorithm. The motivation for our implementation is to validate the correctness and the preciseness of our algorithms. We have tested our algorithm on different communication diagram given as input. We have coded our algorithm in Java. First, we are generating the Control Flow Graph (CFG) of the communication diagram as shown in Figure 2. From CFG, we are generating the Concern Graph for Communication Diagram (CCGoD) as shown in Figure 3. We are accumulating the whole information about the CGCoD in a file. While storing the different information about the CGCoD, we store the data in a structured data type. Some snapshots of implementation are shown below. Figure 5 shows the Fan in value calculated for each node in the graph of Figure 3. In Figure 6, we show the calculated Threshold Fan in value of all the nodes. The identified concerns that are possible candidates to be separated as aspects are shown in Figure 7. Finally, the aspects that have been mined from the UML communication diagram are shown in Table II.

TABLE II
RESULT OF ASPECT MINING

Sl. No.	Aspect	Objects on which aspect is scattered	Concerns with which aspect is tangled
1	CHECK	LMS	RETURN
			DISPLAYERROR
		MEMBERRECORD	RETURN
		TRANSACTION	RETURN
			DISPLAYERROR
		BOOK	RETURN
2	RETURN	LMS	CHECK
			DISPLAYERROR
		MEMBERRECORD	CHECK
		TRANSACTION	CHECK
			DISPLAYERROR
		BOOK	CHECK
3	DISPLAYERROR	MEMBER	
		LMS	CHECK
			RETURN
		TRANSACTION	CHECK
			RETURN

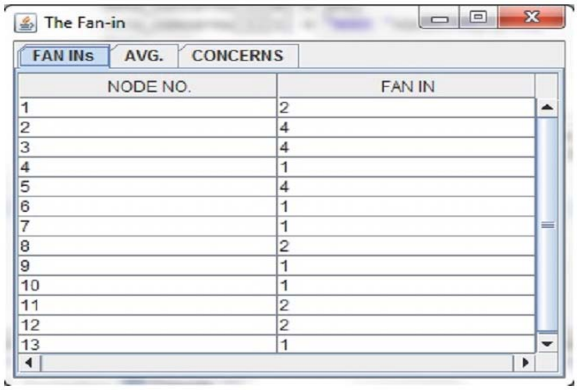

Fig. 5. Snapshot showing Fan_in of each node

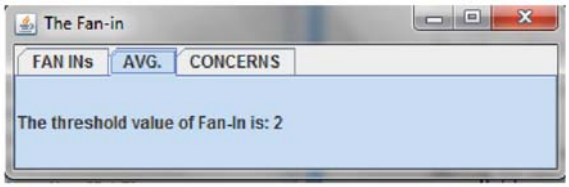

Fig. 6. Snapshot showing the threshold-fan-in

VI. COMPARISON WITH RELATED WORKS

To the best of our knowledge, no work has been done on aspect mining from UML diagrams. In the absence of any directly related work, we compare our work with the work based on aspect mining from source code. Tonella et al. [18] investigated interfaces those are likely to be crosscutting each other. This was done by string matching for interface name, call relationships between the methods of the classes that implemented the corresponding interface. But, we have used Fan-in analysis to find the crosscutting concerns shown by the UML communication diagram for the given problem scenario. Shepherd et al. [19] proposed aspect mining in Java source code by using clone

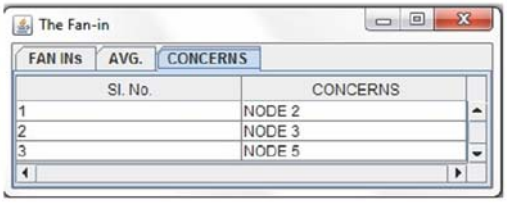

Fig. 7. Snapshot showing the identified Concerns

detection based technique on the Program Dependence Graph and comparing each statements abstract syntax tree representation. The tool is automated but list of the aspects mined are not reported in the paper. We have implemented our algorithm to find the aspects at the design level by using the UML 2.0 communication diagram. Tourwe et al. [20] proposed an aspect mining technique based on concept analysis. Some other approaches used clone detection technique to identify the crosscutting behavior in the source code of non-aspect oriented programs. But in our work, we have used the communication diagram which led to identification of aspects at design stage much earlier to coding. Ophir [21] used control-based comparison to find the initial re-factoring candidates. It then finds similar data dependencies in sub-graph representing the code clones. It also used clone detection technique to identify the aspects from code. Again, our approach finds the aspect from the communication diagram.

VII. CONCLUSION AND FUTURE WORK

Aspect mining is being usually done at the code level. In our approach, we have found the aspects at the design level which helps in better understanding of the crosscutting concerns. It also help programmers to easily refactor the existing non-aspect applications.We have taken UML 2.0 communication diagram to compute the slices. We have first developed the Control Flow Graph (CFG) from the communication diagram. From CFG, we have developed a Concern Graph for Communication Diagram (CGCoD) as an intermediate representation. Then, we have developed an algorithm to identify the aspects from the communication diagram by using the Fan in metric. Also, we have generated Concern Scattering Graph (CSG) to find the scattered and tangled concerns. Finally, we have found the aspects that are crosscutting in the given problem. In future, we will extend our work to find aspects from other UML diagrams like activity diagram, state-chart diagram etc. We will also implement other metrics to identify the aspects more accurately and efficiently.

REFERENCES

[1] S Hanenberg, C Oberschulte, and R Unland. Refactoring of Aspect-Oriented Software. In In Proceedings of the 4th Annual International Conference on Object-Oriented and Internet-based Technologies, Concepts and Applications for a Networked World (Net.ObjectDays), pages 19–35, September 2003.

[2] G Kiczales, J Irwin, J Lamping J M Loingtier, V C Lopes, C Maeda, and A Mendhekar. Aspect-Oriented Programming. In In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), pages 220–242, Finland, June 1997.

[3] M Lippert and V C Lopes. A Study on Exception Detection and Handling using Aspect-Oriented Programming. In In Proceedings of the International Conference on Software Engineering (ICSE), page 418427. ACM Press, 2000.

[4] P Soni. Analysis of Component Composition Approaches. International Journal of Computer Science and Communication Engineering, 2(1):8–13, 2013.

[5] G W Griswold, M Shonle, K Sullivan, Y Song, N Tewari, Y Cai, and H Rajan. Modular software design with crosscutting interfaces. Software, IEEE, 23(1):51–60, 2006.

[6] L Moonen. Dealing with crosscutting concerns in existing software. In Frontiers of Software Maintenance, 2008. FoSM 2008., pages 68– 77. IEEE, 2008.

[7] A Kellens and K Mens. A Survey of Aspect Mining Tools and Techniques. Technical Report INGI 2005-07, Universite catholique de Louvain, Belgium, 2005.

[8] A Abdurazik and J Offutt. Using UML Collaboration Diagrams for Static Checking and Test Generation. In The 3rd International Conference on the UML, pages 383–395, New York, October 2000.

[9] M Marin, A D Van, and L Moonen. Identifying aspects using fan-in analysis. In Reverse Engineering, 2004. Proceedings. 11th Working Conference on, pages 132–141. IEEE, 2004.

[10] M Marin, A D Van, and L Moonen. Identifying crosscutting concerns using fan-in analysis. ACM Transactions on Software Engineering and Methodology (TOSEM), 17(1):1–34, 2007.

[11] P Samuel and R Mall. Slicing-based test case generation from UML activity diagrams. ACM SIGSOFT Software Engineering Notes, 34(6):1–14, 2009.

[12] J T Lallchandani and R Mall. Slicing UML Architectural Models. ACM SIGSOFT Software Engineering Notes, 33(3):1–9, 2008.

[13] J Zhao and M Rinard. System Dependence Graph Construction for Aspect-Oriented Programs. Technical Report 7HFKQLFDO 5HSRUW 0, Laboratory Of Computer Science, Massachusetts institute of Technology, USA, 2003.

[14] J M Conejero, J Hernandez, E Jurado, and K Berg. Croscutting, what is and what is not? A Formal definition based on a Crosscutting Pattern. Technical Report TR28/07, University of Extremadura, Spain, 2007.

[15] J M Conejero, E Figueiredob, A Garciac, J Hernndeza, and E Juradoa. On the relationship of concern metrics and requirements maintainability. Information and Software Technology, 54(2):212– 238, 2012.

[16] http://www.agrouml.tigris.org. In ArgoUML Quick Guide, 2004.

[17] http://www.magicdraw.com. In Magicdraw UML v11.6, 2010.

[18] P Tonella and M Ceccato. Migrating interface implementations to aspects. In Proceedings of 2oth International Conference on Software Maintenance (ICSM), pages 220–229. IEEE Computer Society, 2004.

[19] D Shepherd, E Gibson, and L Pollock. Design and evaluation of an automated aspect mining tool. In In International Conference on Software Engineering Research and Practice, pages 601–607. Citeseer, April 2004.

[20] T Tourwe and K Mens. Mining aspectual views using formal concept analysis. In In Proceedings IEEE International Workshop on Source Code Analysis and Manipulation, pages 97–106. IEEE Computer Society, 2004.

[21] D Shepherd and L Pollock. Ophir : A Framework for Automatic Mining and refactoring of Aspects. Technical Report 2004-03, University of Delaware, 2003.