ISSN No. 0976-5697

Volume 4, No. 11, Nov-Dec 2013

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Testing Aspect Oriented Programs by Automated Process Based on Unified Modelling Diagrams

Sunidhi Sharma
Department of Computer Science
Chandigarh Engineering College, Landran
Mohali, India
sunidhisharma24@yahoo.com

Geetanjali Babbar
Department of Computer Science
Chandigarh Engineering College, Landran
Mohali, India
cecm.cse.gbi@gmail.com

Abstract: Aspect Oriented Programming is an emerging software engineering paradigm that offers new constructs in order to improve separation of crosscutting concerns into single units called aspects. AspectJ, the most used aspect-oriented programming language, represents an extension o java. In all the previous years, object oriented language has been widely used but today Aspect Oriented Programming is gaining a lot of popularity because this language has solved most of the problems of crosscutting concerns, which makes the code understandable and simplifies software maintenance and evolution. It provides new constructs namely join points, point cuts, advices and aspects. Because of new constructs, new types of faults may rise. So, existing testing techniques are not adequate for testing aspect-oriented programs. As a result, we need to add new techniques in order to get an appropriate solution. In this paper, an approach based upon UML diagrams for testing aspect-oriented programs automatically is presented. The approach focuses on generating test sequences based on the attributes available in input aspect oriented code. In the research, I have proposed an automatic fault detection mechanism with the help of UML diagrams which can detect various errors. Sequences will be generated automatically and fault detection in Aspect-Oriented codes will validate the correct working and errors.

Keywords: Aspect Oriented Programming, Joinpoints, Object Oriented Programming, Pointcuts, UML Class Diagrams, Test Sequences

I. INTRODUCTION

Aspect Oriented Programming is a software engineering paradigm[1] that offers new constructs, such as join points, pointcuts, advices, and aspects in order to improve separation of crosscutting concerns. The new constructs[2] bring new types of programming faults with respect to crosscutting concerns, such as incorrect pointcuts, advice, or aspect precedence. In fact, existing object-oriented testing techniques are not adequate for testing aspect-oriented programs[3]. As a result, new testing techniques must be developed. The Related approach focuses on integration of one or several crosscutting concerns to a primary concern and tests whether or not an aspect-oriented program[4] conforms to its expected crosscutting behavior.

In our scheme, test sequences[5][6] will be generated based on the attributes available in input Aspect oriented code. Previously, test sequences will be generated manually. But in the present research test sequences will be based on interaction between aspects and primary models[7][8], and verifies the execution of the selected sequences automatically. In our research we will propose an automatic fault detection mechanism with help of uml diagrams[9] which can detect incorrect advice type errors, weak pointcuts, incorrect Precedence errors. We have proposed an automatic scheme [10]rather than the manual selection which was in use previously. Sequences will be generated by proposed scheme automatically while detection of faults in Aspect Oriented codes which will validate the correct working and starting malfunctioning (errors)[1].

Purpose of the research work is:

a) To find the faults that specific to aspectual structures.
b) To provide solution for incorrect advice type, strong or weak pointcut expressions, and incorrect aspect precedence.
c) To find the faults that are hidden in the Aspect Oriented Program. Hidden faults are those which does not effect the working of the program but the actual results does not match with the expected results.
d) To make whole procedure automatic one, so that with the simplicity we can get our results and find the faults easily.

II. RELATED WORK

Testing of aspect oriented programs is a new programming paradigm. Many researchers had contributed their research in the field of testing AOP.

Somayeh Madadpour, Seyed-Hassan Mirian [1] explained that Aspect-Oriented Programming is a software engineering paradigm that offers new constructs, such as join points, point cuts, advices, and aspects in order to improve separation of crosscutting concerns. This paper provides an activity-based testing approach for aspect-oriented programs. Proposed approach can help testers reveal several types of faults that specific to aspectual structures, such as incorrect advice type, strong or weak point cut expressions, and incorrect aspect precedence.

Philippe Massicotte, Linda Badri, Mourad Badri [2] described that Aspect-Oriented Programming is an emerging software engineering paradigm. It offers new constructs and tools improving separation of crosscutting concerns into single units called aspects. Authors present, in this paper, a new aspects-classes integration testing strategy and the associated tool. The adopted approach consists of two main phases: (1) static analysis: generating testing sequences based on dynamic interactions between aspects and classes, (2) dynamic analysis: verifying the execution of the selected sequences. Authors focus, in particular, on the integration of one or more aspects in the control of collaborating classes.

Mayank Singh, Shailendra Mishra [4] says that testing of aspect oriented programs is a new programming paradigm. Many researchers had contributed their research in the field of testing AOP. Mutation testing is an emerging area of research in testing of aspect oriented programming. The effectiveness of mutation testing depends on finding fault types and designing of mutation operators on the basis of faults identified. Therefore the effectiveness of testing depends upon the quality of these mutation operators. We already have the mutation operators for procedural and object oriented languages, but for aspect oriented language only a few researchers have contributed. In this paper we will study in detail about the fault types and related mutation operators for AspectJ language. This paper also proposes the implementation framework to implement these mutation operators automatically.

Swati Tahiliani, Pallavi Pandit [9] explained that apart from application modeling, the Unified Modeling Language (UML) is also used for designing the tests on various levels (unit, integration, system tests). Authors have listed various approaches based on UML diagrams, and the Use Case based approaches have been described too. As future work, these approaches could be further compared and analyzed for determining the best approach.

Guoquing Xu [13] approach given by Rothermal and Harrold is based only on static analysis but most of the time dynamic analysis is required because of calling of external methods. Guoquing Xu gave an approach which is based on RETSA framework. He gave another approach on aspect oriented program. This approach is an extension of JIG used by Rothermal and Harrold i.e. AJIG (Aspect-J Inter Module Graph). It is a new control flow representation for aspect-J softwares which captures the semantic intricacies of aspect-related interactions.

III. PROPOSED APPROACH: AUTOMATED TESTING ON ASPECT ORIENTED PROGRAMMING

Our research is focused on providing a testing environment for AOP based program[11][12]. Aim of this research is to test an AOP code to find Strong and weak Point-Cuts and weak join-Points.

In this paper, we present a framework for automated generation of test data[14] for aspect oriented programs. Research starts with designing the UML class diagram for the classes and aspect. UML diagram[15] contains the useful information about the classes and aspect. We have used UML design tool i.e. ArgoUml. This tool help in creating the UML diagram for a class by reading its source code. By using the source code of the program which is used for testing, we generate a UML class diagram. As ArgoUml do not generate UML diagram for aspects, we have created it our-self by modifing ArgoUml class diagram. After creating class uml diagram and uml diagram for aspect, we have created a XMI file based on the uml diagram. This xmi file is used during testing to get information about our main classes and aspects.

After creating uml diagram and generating xmi file from that, next we have developed a program our main testing program. This program reads the XMI files node by node, extract useful information, send information to a helping program to get some information from aspect file and perform testing based on information. XMI file contain information in the form of various nodes, and starting from first node our program can read whole XMI node by node. In the beginning our program read XMI file and display node names and attributes on the screen as output. While reading the XMI file our program looks for main class and aspect data in the XMI file. Whenever program gets either main class or aspect data in XMI file it extracts main class method names and return types and aspect's point-cut name from the file. This information is stored by the program as it is needed for testing. After getting required information from XMI file, aspect name and point-cut name is forwarded to another program. This program reads the aspect file and fetch useful information from the file. Program will use point-cut's name to find the required point-cut in the file. This point-cut information will be used for testing.

To extract only required information from the point-cuts our program uses a processing block, that process the point-cuts that are stored after separating the user defined point-cut and parse the unwanted data from the point-cuts. After parsing the point-cut we are left with the name of join point(i.e. method that will generate this join-point) and return type. After getting this information, control will be returned to main testing program where information gathered from aspect file and XMI file will be compared. In comparison we will check if a point-cut has a join point in the main program and all the required join-points are picked up in aspect or not .

In main testing program we have defined a method findFault() this method compare the information collected from xmi and aspect file and display results in the form of Strong, weak point-cuts and weak join-points. Following is the flow chart for flow of control:

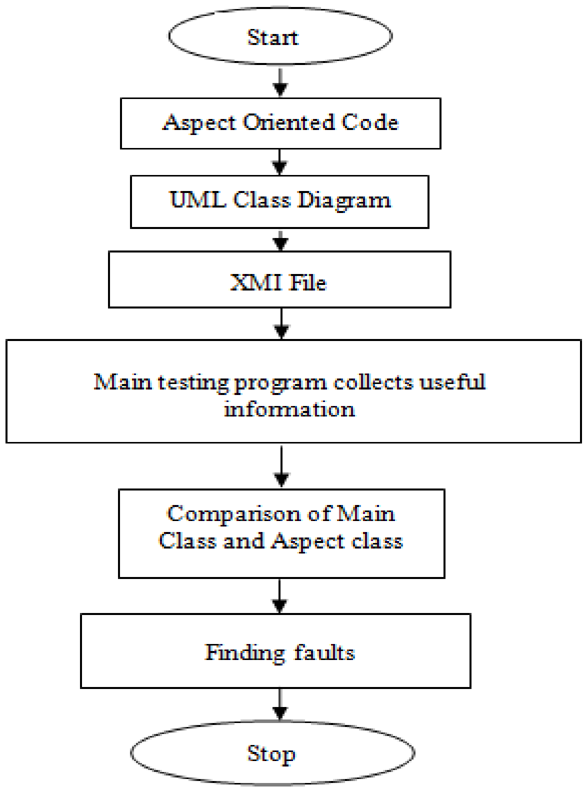

Start
Aspect Oriented Code
UML Class Diagram
XMI File
Main testing program collects useful information
Comparison of Main Class and Aspect class
Finding faults
Stop

Figure 1. Flow Chart of Flow of Control

IV. PROPOSED ALGORITHM

Following are the steps of the algorithm:

Step1: Building class model of primary concern.

Step2: Building an Aspect Oriented Code with the main class and the aspect class.

Step3: Main class containing the actual code of the problem and Aspect class containing the authentication part.

Step4: Uml diagram will be produced for the Aspect Oriented Code (main class) with the AgroUML tool.

Step5: Uml diagram will be created for the aspect class.

Step6: XMI file will be created based on the UML diagrams.

Step7: XMI file will corresponds to main class.

Step8: Main testing tool (program) will be created.

Step9: Extracts useful information from XMI and aspect file separately.

Step10: Comparison between aspect class and main class automatically.

Step11: Finding faults.

Step12: End.

V. RESULTS AND DISCUSSIONS

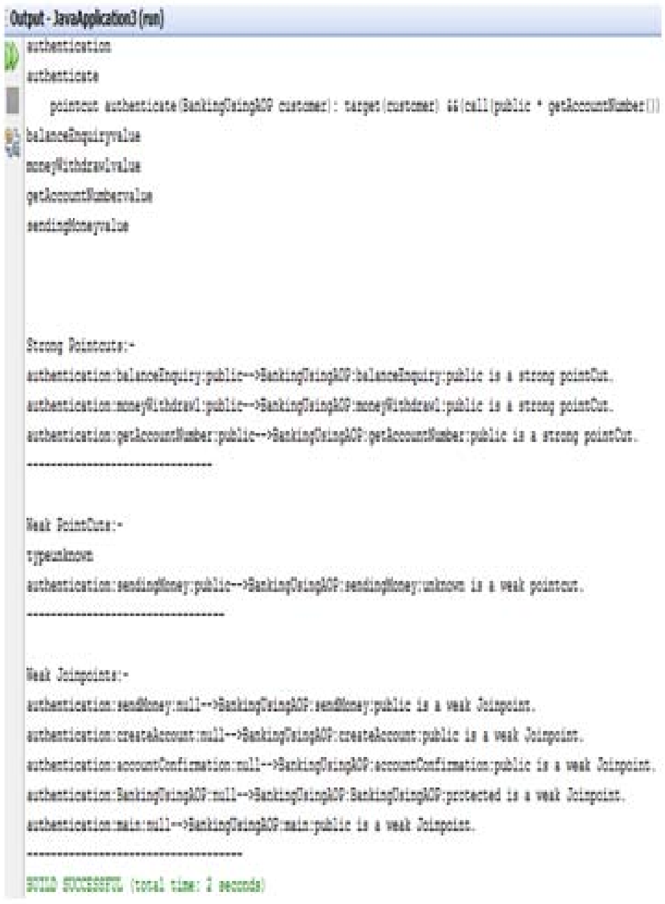
Our research is focused on providing a testing environment for AOP based program. Aim of this research is to test an AOP code to find Strong and weak Point-Cuts and weak join-Points automatically. We will collect the useful information from the main class and aspect class through XMI File. We will extract the results by comparing the XMI from aspect file. In main testing program we have defined a method findFault(), this method compares the information collected from xmi and aspect file and display results in the form of Strong, weak point-cuts and weak join-points.

Strong point-cuts are those point-cuts which corresponds to exactly same method in the main program, weak point-cuts are those point-cuts which are defined in our aspect but don’t have a corresponding method in the main program. Weak point-cuts can be a result of type e.g. we have given a wrong method name in the point-cut or we have specified a wrong access-specifier. Weak join-points refer to methods from main class which do not have any point-cut in the aspect. That means when execution control will be transferred to these methods aspect code will not be notified.

If a join-point from the main class is picked up by a point-cut in the aspect then it is a strong point-cut. Because when join-point will occur in main program, in aspect corresponding point-cut will be notified and associated advice will run and perform the required action. That means a strong point-cut will result in the expected action.

If a point-cut defined in aspect has no corresponding join-point in main class then it is a weak point-cut. Because it is defined unnecessarily and it will never pick any join-point from the main class. This type of point-cuts are weak as they have no purpose in the aspect. A weak point-cut may be a result of a method removal, suppose a method has been removed from main class, or programmatically error like programmer has defined a wrong name. Weak point-cut are hard to find as compiler will not flash any error for a weak point-cut.

If there is no point-cut defined in aspect for a join-point from main class then that is a weak join point. Aspect will never run any advice for a weak join-point as it will never picked up by any point-cut. As aspect will never get any notification for a weak join-point therefore aspect will not perform any action regarding this weak join-point. Now there are two scenarios, a join-point can be left on a purpose like we don't need our aspect to perform any action on the occurrence of the join point. Or it may be a programmatically error, that join point is left by mistake. In latter case, where a join-point should be picked out by point-cut but it is left by mistake, we may expect some unexpected behaviour and result by the main program. Our testing program will show all the weak join-points to the programmer, therefore if any join point left by mistake program modify aspect accordingly.



```
authentication
authenticate
   pointcut authenticate(BankingUsingAOP customer): target(customer) &&(call(public * getAccountNumber())
balanceInquiryvalue
moneyWithdrawlvalue
getAccountNumbervalue
sendingMoneyvalue




Strong Pointcuts:-
authentication:balanceInquiry:public-->BankingUsingAOP:balanceInquiry:public is a strong pointCut.
authentication:moneyWithdrawl:public-->BankingUsingAOP:moneyWithdrawl:public is a strong pointCut.
authentication:getAccountNumber:public-->BankingUsingAOP:getAccountNumber:public is a strong pointCut.
--------------------------------


Weak PointCuts:-
typeunknown
authentication:sendingMoney:public-->BankingUsingAOP:sendingMoney:unknown is a weak pointcut.
----------------------------------


Weak Joinpoints:-
authentication:sendMoney:null-->BankingUsingAOP:sendMoney:public is a weak Joinpoint.
authentication:createAccount:null-->BankingUsingAOP:createAccount:public is a weak Joinpoint.
authentication:accountConfirmation:null-->BankingUsingAOP:accountConfirmation:public is a weak Joinpoint.
authentication:BankingUsingAOP:null-->BankingUsingAOP:BankingUsingAOP:protected is a weak Joinpoint.
authentication:main:null-->BankingUsingAOP:main:public is a weak Joinpoint.
-------------------------------------
BUILD SUCCESSFUL (total time: 2 seconds)
```

Fig ure 2. Result showing the strong pointcuts, weak pointcuts and weak joinpoints for Aspect Oriented Code.

VI. CONCLUSION

We present in this paper, the UML based testing approach on Aspect-oriented programs. Our approach is helpful in finding out the aspectual faults automatically[16] from the aspect-oriented programs. The faults are of various kinds like incorrect advice type, weak or strong pointcut expressions and incorrect precedence[17].

We have worked for this testing in four phases: (1) Firstly, we will have Aspect Oriented Code which will have to be tested automatically. (2) Secondly, UML Diagram will be created by me based on Aspect Oriented Code. Further, XMI file will be created based on the UML diagram. (3)

Thirdly, the main testing application has been developed through which testing of Aspect Oriented Program will be done. (4) Finally, in the main step, comparison of Aspect Class with the main class is done. We have defined a method named findFault which will find out the hidden faults and give us the directions to correct it.

Currently, our approach is based on the automatic testing of aspect code where with the help of single java application, we are able to find the hidden faults such as weak pointcuts, weak joinpoints and strong pointcuts. In this approach, we have used UML Class Diagram. Furthermore, for the future scope, we will try this automatic procedure on some other UML Diagrams.

VII. ACKNOWLEDGMENT

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior and acts during the course of study.

First and foremost, I would like to express my sincere gratitude to my guide Mrs. Geetanjali Babbar, Assistant Professor. I was privileged to experience a sustained enthusiastic and involved interest from her side. I am thankful for her support, cooperation and motivation provided to me during the training for constant inspiration, presence and blessings. Lastly, I would like to thank the almighty and my parents for their moral support and my friends with whom I shared my day-to-day experience and received lots of suggestions that improved my quality of work.

VIII.REFERENCES

[1]. Somayeh Madadpour, Seyed-Hassan Mirian-Hosseinabadi, Vahdat Abdelzad, Testing Aspect-Oriented Programs with UML Activity Diagrams, International Journal of Computer Applications, Volume 33, No-8, pp 23-29, November 2011

[2]. Philippe Massicotte, Linda Badri, Mourad Badri, Towards a Tool Supporting Integration Testing of Aspect-Oriented Programs, Journal of Object Technology, Volume 6, no. 1 (January 2007), pp. 67-89

[3]. Liu, C. H., and Chang, C. W., A State-Based Testing Approach for Aspect-oriented Programming, In Journal of Information Science and Engineering , pp. 11-31, 2008

[4]. Mayank Singh, Shailendra Mishra, Mutant Generation for Aspect Oriented Programs, Indian Journal of Computer Science and Engineering, Vol 1, No 4, pp 409-415, 2011.

[5]. Reza Meimandi Parizi, Abdul Azim Abdul Ghani, Rusli Abdullah, and Rodziah Atan, On the Applicability of Random Testing for Aspect-Oriented Programs, International Journal of Software Engineering and its Applications, Vol. 3, No. 4, October, 2009

[6]. Hilsdale and J. Hugunin, Advice weaving in AspectJ, In proc 3rd International Conference on Aspect-OrientedSoftware Development pages 26–35, 2004

[7]. Dalal, S. R., Jain, A., Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., and Horowitz, B. M., Model-based testing in practice, In Proc. of the 21st International Conf. on Software Engineering (ICSE'99), 1999.

[8]. Xu, W., Xu, D., and Wong, W. E., Testing Aspect-Oriented Programs with UML Design Models, International Journal of Software Engineering and Knowledge Engineering, Vol. 18, No. 3, pp. 413-437, May 2008.

[9]. Swati Tahiliani, Pallavi Pandit, A survey of UML- based approaches to testing, International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 5

[10]. Grady Booch, Dr. James Rumbaugh, Dr. Jacobson, The Unified Modeling Language(Addison-Wesley Professional, sept 1998).

[11]. Marlon Dumas and Arthur H.M ter Hofstede, UML Activity diagrams as a Workflos Specification Language, In proceedings of the UML’2001 conference

[12]. Cui, Z., Wang, L., and Li, X., Modeling and integrating aspects with uml activity diagrams, Proceedings of the 2009 ACM symposium on Applied Computing, 2009

[13]. G. Xu, A regression tests selection technique for aspect oriented programs, In Workshop on Testing Aspect-Oriented Programs, pages 15–20, 2006.

[14]. Badri, B., Badri, L., Fortin, M. B., Automated State-Based Unit Testing for Aspect-Oriented Programs: A Supporting Framework, International Journal of Object Technology, vol. 8, no. 3, pp. 121-126, 2009

[15]. T. Xie and J. Zhao, A framework and tool supports for generating test inputs of AspectJ programs, InProc. 5th International Conference on Aspect-Oriented Software Development pages 190–201, March 2006.

[16]. Xie, T., Zhao, J., Marinov, D., and Notkin, D., Automated test generation for AspectJ programs, AOSD 2005 Workshop on Testing Aspect-Oriented Programs, Chicago, 2005

[17]. Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R., and Stauner, T., One evaluation of model-based testing and its automation, In Proc. of the 27th International Conf.on Software Engineering (ICSE'05), 2005.