International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 7, July - 2013

Panorama For Automated Testing Of Aspect-Oriented Programs Based On Unified Modeling Diagrams

Sunidhi Sharma
Research Student

Geetanjali Babbar
Associate Professor

Abstract

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 problem of crosscutting concerns. AOP is an approach for modularizing the 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 in order to improve separation problem. 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 our research we will propose 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.

1. Introduction

In software development life cycle [11], testing is an important part. The IEEE definition of testing is "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results". The quality of any software product is checked through the testing. There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following rote procedure. Another definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are things the tester tries to do with the product, and the product answers with its behavior in reaction to the probing of the tester. More than half of budget of a software project spend on testing even though, it is not guaranteeing the correctness of software. There has been a high level of interest to automate the testing process in software development. To assuring the quality of aspect oriented projects, testing is the only process. The process of automated software testing requires an approach to select the test case. The main aim of testing is to cover the programming features.

Aspect-oriented programming provides new features that capture crosscutting concerns into one separate units known as Aspects [6], The language introduces new constructs (such as join points, advice, intertype declarations, point cuts and aspects). The behavior of an aspect in AspectJ programs can be categorized into two types [12], first is aspectual behavior which is defined as behavior implemented in pieces of advice and second is aspectual composition which is defined as behavior implemented in pointcuts for composition between base and aspectual behavior. In automated software testing, there are two major activities i.e test-input generation and test oracles [5]. Test-input generation generates test inputs for the program under test. After the test inputs are generated and executed, we need to have test oracles to determine whether these test executions are correct. In this paper, testing is performed over UML diagrams

A Use Case provides a unit of functionality given by the system. Use Case Diagrams [22] can be used to describe the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of your system, UCDs can be used to show all of its available functionality. The main purpose of the use-case diagram is to develop the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use-case diagrams generally show groups of use cases -- either all use

cases for the complete system, or a breakout of a particular group of use cases with related functionality (e.g., all security administration related use cases) [1] [2]. To show a use case on a use-case diagram, you draw an oval in the middle of the diagram and put the name of the use case in the center of, or below the oval. To draw an actor on a use-case diagram, you draw a stick person to the left or right of your diagram. Use simple lines to depict relationships between actors and use cases, as shown in Fig 1.

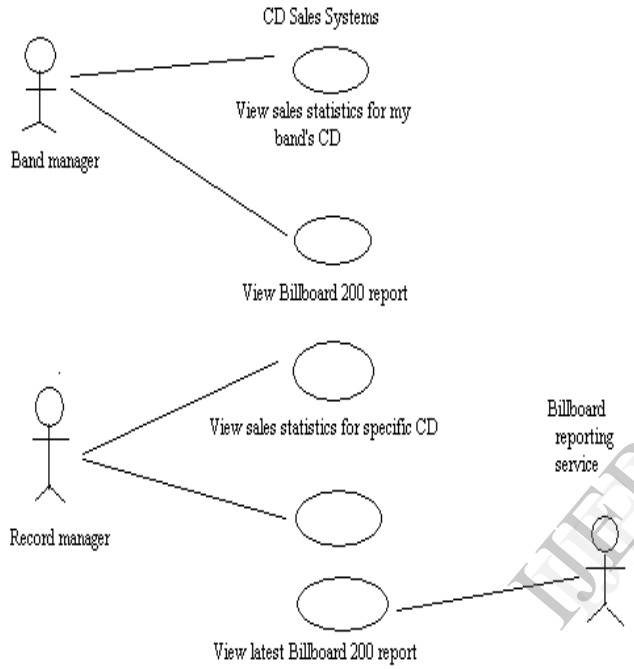

Fig.I: Relationship in UML diagram[1]

A use-case diagram is typically used to communicate the high-level functions of the system and the system's scope. As shown in Figure 1, we can easily tell the functions that our example system provides. This system lets the band manager view a sales statistics report and the Billboard 200 report for the band's CDs. It also lets the record manager view a sales statistics report and the Billboard 200 report for a particular CD. The diagram also tells us that our system delivers Billboard reports from an external system called Billboard Reporting Service [1][2].

Aspect-Oriented Programming [4] is a software engineering approach that uses several new constructs, such as join points, point cuts, advices, and aspects in order to improve separation of crosscutting concerns [16]. The new constructs bring new types of programming faults with respect to crosscutting concerns, such as incorrect point cuts, advice, or aspect precedence. In fact, existing object-oriented testing techniques are not adequate for testing aspect-oriented programs. Aspect Oriented Programming (AOP) is an emerging discipline in Software Engineering. AOP [4] is a programming paradigm which isolates secondary functions from the main program's logic. The definition given by Gregor Kiczales "Modular units that cross-cut the structure of other modular units." The central idea of Aspect Oriented Programming as an emerging discipline of post-object technology is to provide strong support to the separation of the repeated, scattered or entangled concerns at every stage of software development, introducing a new modular unit to encapsulate them to facilitate extensibility, changeability and reuse.

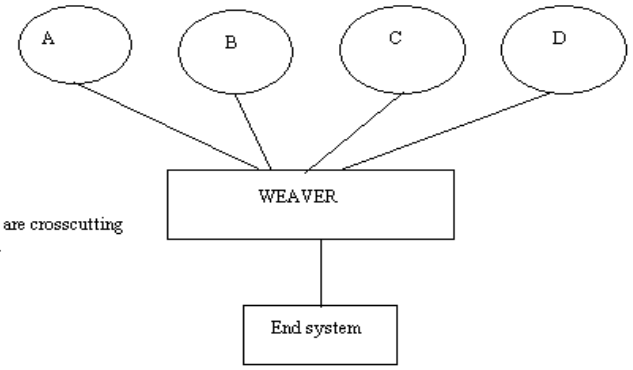

Fig. II: Structure of Aspect Oriented programming

In software engineering if we want to define a concern, then it is defined as a property or interest point of a system. From the system point of view, concerns are defined as those interests belonging to the system and its operation, or other aspects which are critical or important for the stakeholders. Simply, a concern is a kind of requirement needed by the system. Some concerns can be easily encapsulated within classes or modules, but some concerns affects the functionality of several modules, those are called crosscutting concerns and they are not easy to separate. They cannot be easily encapsulated into new functional units as "implicit functionality" because they crosscut the whole system and are implemented in many classes or modules producing an entangled or scattered code, difficult to understand and maintain. The goal of AOP is to encapsulate them into a modular unit, called aspect, to handle these requirements at implementation level. AOP is an extension of Object oriented programming and it is a future programming technique.

2. Related work

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

Philippe Massicotte, Linda Badri, Mourad Badri [23] 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.

Swati Tahiliani, Pallavi Pandit [24] 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.

Somayeh Madadpour, Seyed-Hassan Mirian [16] 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.

Rothermal and harrold [13] proposes an approach which is based on graph traversal algorithm. In their approach aspect oriented features (such as point cuts, join points, aspects, advice [14]) are added into object oreinted programs, and the program is regression tested to make sure that the newly introduced features do not affect the code. He assumes two versions of program P and P' such that P is the original program which is basically a java program and P' is the modified program (aspect is added to java program). He presents a graph traversal algorithm which runs the test suite for the original program and obtains the coverage information and constructs the java interface graph (JIG)[20] for both original and modified program and then compare the CFG's .Comparison helps in detecting the dangerous arcs. An arc is dangerous if target of the CFG of both original and modified program differ. Dangerous arcs are then rerun and the test cases are selected safely.

Guoquing Xu [14] 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.

3. Overview of Our Approach

Aspect-Oriented Programming is emerging as a very popular language because of the separation of crosscutting concerns. It is a software engineering approach which includes following new constructs, such as join points, pointcuts, advices, and aspects. With the addition of these constructs there is a great possibility of arriving various new faults corresponding 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. As a result, there is an adequate need for developing new testing techniques. The Related approach focuses on automatic integration of one or several crosscutting concerns to a primary concern and testing and finding the various errors automatically.

In our research, we will focus on the testing of Aspect oriented programs with UML state based diagram. Particularly for testing we will use AGRO UML tool. In our research we will focus on faults finding for Aspect Oriented Programs with help of flow diagram based on state base of UML. Then test sequences will be generated based on the attributes available in input Aspect oriented code. Test sequences will also be based on interaction between aspects and primary models, and verifies the execution of the selected sequences. In our research we will propose an automatic fault detection mechanism with help of uml diagrams which can detect incorrect advice type errors, weak pointcuts, incorrect Precedence errors. Sequences will be generated by proposed scheme automatically while detection of faults in Aspect Oriented codes which will validate the coming results.

4. Automated Testing on Aspect Oriented Programs

In this paper, we present a framework for automated generation of test data for aspect oriented programs.

This framework is based on existing framework for object oriented programs. The main objective of this framework is to generate test data to find out the various faults and calculate total effort. Also the main motive of this study is to automatically correct it. We use AspectJ programs to implement this framework.

In this framework, we first have AspectJ code which will be converted into plain java code. As all the contents are not relevant to test, so the irrelevant content are identified and removed from the base class. With the help of this step, we reduce visible faults. Now the actual testing is performed on the relevant content of produced code and generates the test data. After testing, next step will be of generation of test sequence which is an automatic step. After then execution is done and discovered errors will be automatically corrected.

The major steps of our method are described in the following:
1. Building an aspect oriented code with the separation of crosscutting concerns.
2. Building state based model of the primary concern
3. Testing the primary concern separately.
4. Integrating an aspect. As long as there are aspects which are not integrated
a. Building aspect model and weave it into primary model.
b. Generating the test sequences affected or created by the aspect.
c. Testing the primary concern with the integrated aspect and executing it.
d. If there is no problem encountered, return to step 4.
4. Testing entirely the primary concern including aspects.
5. End.

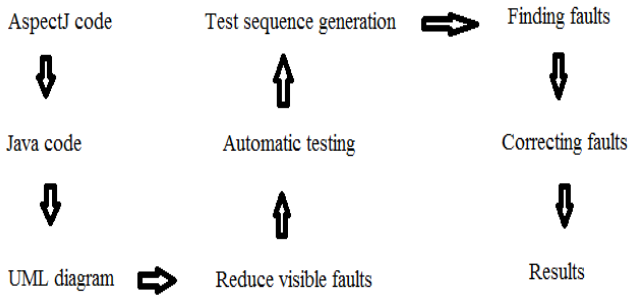

Fig.. III: Framework of automated testing of AOP

To convert code from AspectJ to Java code, AspectJ Compiler 1.0.6 has been used. The code is converted into UML diagram with AGRO UML tool. This framework has been implemented with a software tool which can test Java programs using various testing techniques. With the code we make UML diagram with the help of AGRO UML tool. Then the test sequences is generated followed by the execution of the test sequences. Finally, with the automated tool, faults will be detected and automatically corrected.

5. Conclusion and Future Work

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 from the aspect-oriented programs. The faults are of various kinds like incorrect advice type, weak or strong pointcut expressions and incorrect precedence.

Our research will focus on the testing of aspect programs by generating test sequences based on state based diagrams which will validate the correct working and starting malfunctioning(errors). Further aspect model will be generated and integrate with basic aspect model with flow diagrams in various UML diagrams. With the help of UML diagrams, we will be able to test the code automatically. Future scope of this research will be handling the testing with some other UML diagram.

6. Acknowledgement

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.

7. References

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

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

[3] 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

[4] 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

[5] 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.

[6] 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.

[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] Blackburn, M., Busser, R., Nauman, A., Knickerbocker, R., and Kasuda, R., Mars Polar Lander fault identification using model-based testing, *In Proc. of the Eighth International Conference on Engineering of Complex Computer Systems*, 2002. .

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

[10] 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.

[11] Software Engineering websites namely *www.simple.wikipedia.org/wiki/Software_engineering* andhttp://en.wikipedia.org/wiki/Aspect-oriented_programming.

[12] Zhao, J. and Rinard, M., System dependence graph construction for aspect-oriented programs, MIT-LCSTR-891, *Laboratory for Computer Science*, MIT, 2003.

[13] M. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, Regression Test Selection for Java Software, *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp.312-326*, October 2001.

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

[15] G. Xu and A. Rountev, Regression test selection for AspectJ software, *In ICSE '07: Proceedings of the 29th international conference on Software Engineering, pages 65–74*, 2007.

[16] 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

[17] 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

[18] 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.

[19] 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.

[20] 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

[21] 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

[22] Website of Use case diagram in UML *http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html*

[23] 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

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