Model Based Testing of Aspect-Oriented Software using UML Diagram

A THESIS PRESENTED

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE

DEGREE OF

MS IN SOFTWARE ENGINEERING

BY

SANA AKBAR
137-FBAS/MSSE/F06

SUPERVISED BY
ASSISTANT PROFESSOR
SALMA IMTIAZ

Department of Computer Science & Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University, H-10, Islamabad

(December 2011)

MS
005.3
SAM

1. Computer software
2. Application Software

Final Approval

Date: ________________

It is certified that we have read the research thesis report and have fully evaluated the research undertaken by **Sana Akbar** Registration No. **137-FBAS/MSSE/F06**. This research thesis fully meets the requirements of Department of Computer Science and Software Engineering and hence, International Islamic University Islamabad.

External Examiner

Dr. Muhammad Ramzan
Assistant Professor
Department of Software Engineering
FUIEMS, New Lalazar
Rawalpindi

[illegible]

Internal Examiner

Dr. Abdul Rauf
Assistant Professor
Department of Computer Science and Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University Islamabad

For [illegible]

Supervisor

Miss Salma Imtiaz
Assistant Professor
Department of Computer Science and Software Engineering
Faculty of Basic and Applied Sciences
International Islamic University Islamabad

[illegible]

DEDICATION

I would like to dedicate my research work to the

HOLIEST man Ever Born on Earth, ***PROPHET***

MUHAMMAD (Peace Be Upon Him) *and*

I also dedicate my work to my
PARENTS

Whose sincere love and prayers are a source of

strength for me and made me to do this research work

successfully.

SANA AKBAR
137-FBAS/MSSE/F06

A dissertation submitted to the

Department of Computer Science &

Software Engineering, Faculty of

Basic and Applied Sciences,

International Islamic University, Islamabad,

as a partial fulfillment of the requirements

for the award of the degree of

MS in Software Engineering (MSSE)

DECLARATION

I hereby declare that this Thesis **"Model Based Testing of Aspect-Oriented Software using UML Diagram"**, neither as a whole nor as a part thereof, has been copied out from any source. It is further declared that I have written this thesis entirely on the basis of my personal efforts, made under the proficient guidance of my thesis supervisor, **Miss Salma Imtiaz.**

If any part of this research thesis proved to be copied or found to be a research of some other individual, I shall standby the consequences.

No portion of the research work presented in this thesis report has been submitted in support of any other degree or qualification of this or any other University or institute of learning.

Sana Akbar
137-FBAS/MSSE/F06

ACKNOWLEDGEMENT

In the name of Allah, the most passionate and the most merciful whose blessings made it possible for me to complete this research work. It is a great pleasure for me to complete it successfully. It is all because of **Almighty Allah's** guidance that made me so able.

I owe my deepest gratitude to my thesis supervisor **Miss Salma Imtiaz** whose brain was behind the theoretical raw idea of this research work. Without her guidance and supports it was not possible to complete this research work. She has made available her support in a number of ways. Her commendable advices, sincere supervision and gracious attitude are worth mentioning and for which I am extremely grateful.

I would also like to show my gratitude to my parents, sisters (Aliya, Attiya, their husbands and their children), my husband and in-laws for their continuous support for the completion of my work. Especially my mother, whom is responsible for my intellectual curiosity and is always there for me in my hard time, this is the reason I am on this stage today.

Lastly, but by no means least, I would like to acknowledge **Miss Iram Rauf, Miss Irum Rubab and Sir Zafar Malik**, who were always there to support me in my research work and tolerated me.

For errors and inadequacies in this research work, I accept the responsibility.

Sana Akbar
137-FBAS/MSSE/F06

THESIS IN BRIEF

THESIS TITLE:	Model Based Testing of Aspect-Oriented Software Using UML Diagram
OBJECTIVE:	To discover the new way of model based testing technique in the evolving field of aspect-oriented programs
UNDERTAKEN BY:	***Sana Akbar*** 137-FBAS/MSSE/F06 Student of MS in Software Engineering Department of Computer Science & Software Engineering, Faculty of Basic and Applied Sciences International Islamic University, Islamabad
SUPERVISED BY:	**Miss Salma Imtiaz**
START DATE:	**April 10, 2011.**
COMPLETION DATE:	**December 20, 2011.**

Abstract

Aspect oriented programming (AOP) with increase in modularization and abstraction also increases risks of errors in both ways, statically and dynamically. Its constructs like pointcut, joinpoint, advice and introduction may affect the normal execution at compile and runtime. Therefore, well formed testing techniques are required to reduce maximum errors in an AOP. This thesis proposes a model based testing of aspect oriented programs to test the integration and interactions between different classes and aspects. UML 2.0 sequence diagram is used and extended for modeling the behavior of an AOP and its testing. Interaction of classes and aspects are shown using weaved sequence diagram, that is then converted to a Control Flow Graph (CFG) for the ease of testing from it. CFG behaves as a secondary model to test the classes, aspects and their weaving. The thesis proposes an algorithm to make CFG from weaved sequence diagram. It also proposes two coverage criteria as, all message sequence coverage criterion and all post-condition sequence coverage criterion. Both coverage criteria are applied on CFG, from which different test paths are generated. Also faults are inserted, explicitly, in a sequence diagram to assess whether the coverage criteria can cover the errors or not. It is stated that the proposed approach is capable to cover two faults such as, incorrect strength in pointcut patterns and failure to establish expected post condition. The technique is validated using two examples and is also partially automated.

Table of Contents **Page #**

List of Figures

Page #

Chapter 1

Introduction

1. INTRODUCTION

Testing helps in minimizing possible errors that can affect the operation and function of a system and the needs of the clients and end users [1]. It increases the quality of a system and confidence level of stakeholders on a system [2]. The whole testing process is comprised of analyzing the system, generating test cases using some testing technique , applying the generated test cases on the system under test (SUT) and finally analyzing the results obtained with the expected results i.e. test oracles [2,1]. The whole testing process should be automated [3] to get maximum benefits out of it. The benefits would be in terms of cost, effort and reduced amount of time.

Model based Testing (MBT), as evident by name, is the automatable testing process based on models [1], to facilitate the process of test selection and test results evaluation. It allows an early detection of faults in software, as design models are made before the implementation phase [4]. It is usually considered as black box testing technique [9], as mostly, there are no implementation details in models. It helps to catch subtle bugs that may create difficulty in the end. In MBT we first understand the system under test, choose the model to represent the behavior of a system, and build it. Test cases are generated meeting certain criteria and run on SUT. We perform MBT because models provide a common platform for developers and testers to communicate on and also help them to grasp the client's needs easily. Similarly, its automation [5] will provide more ease for the testers to test larger projects.

Aspect oriented programming (AOP) is an evolving paradigm and started a decade ago [2]. It helps to increase modularization and abstraction in a program thus avoiding appearance of repeated code segments, known as crosscutting concerns. It [3] extends object oriented programming and also proposes many of its own, new constructs. Crosscutting concern [2] is named as an aspect in AOP that is made of some constructs, such as pointcut, joinpoint, advice and introduction. Pointcut is a set of joinpoints and combine them with the help of few specified operators. Joinpoint is a point in base concern where implementation of an aspect has to weave in; an advice contains the implementation of crosscutting concern that is executed at a given joinpoint. There can be after, before, around, after throwing and after returning advices. Another construct i.e. introduction helps to add new fields, aspects, classes and interfaces to core concerns and affect the classes structurally. All these new constructs of AOP needs new modeling and testing techniques, other than OOP.

Aspect oriented modeling (AOM) allows to model the core concern, crosscutting concern and their weaving process. There is no standard profile or notation to model aspects [6] but many researchers have proposed different ideas to model it. The proposed, model based testing approach, in the thesis is based on a modeling technique [7] that extends sequence diagram of UML to represent aspect oriented program. UML is the standard modeling notation [8] to represent design of software and provides an extension mechanism such as stereotype, tagged values and constraints to model new paradigms. The approach in the thesis adapt sequence diagram as it shows the message sequence with respect to time. Therefore, we are able to clearly see at particular time and joinpoint, where and when an aspect is weaved in a class. Sequence diagram also shows the control flow between different objects and messages. Modeling of aspect oriented programs will help to clarify and simplify the testing process of the aspect implementation, their weaving mechanism and can facilitate in capturing faults that can originate statically or dynamically. This is how MBT help to reduce cost, time and maintenance.

The central point of this thesis is to perform model based testing of aspect oriented software using sequence diagram of UML. The model is converted to a control flow graph to make the UML model testable. Coverage criteria are applied and test paths are generated to see which faults can be caught by the approach. There is validation of the approach through examples and partial automation is also done.

1.1 PROBLEM DOMAIN

Aspect oriented programming besides keeping the distributed or scattered code in one place and helping in easy maintenance, also increases the possibility of faults in aspects, classes and weaving process [11]. Testing, consequently, is required to be strong enough to test the new programming paradigm. In the thesis we use model based testing to test AOP because it helps to test a program before its implementation; long before the deadline; when the developers are in the process of understanding it. Weaving is the process by which the encapsulated functionality in an aspect is inserted at a specified joinpoint of a class. Weaving of classes and aspects is a critical task which requires proper testing for finding static and dynamic errors. Modeling, thus, provide different behavioral views to capture the outlook of a system. Model based testing of aspect oriented programs requires models to be expressive and strong enough so that aspects can be tested separately, as well as their weaving with classes. Many faults can reside in an AOP that can also be located in its model; therefore MBT of AOP will help to decrease the appearance of faults in future.

The current techniques used for model based testing purpose do not cover most of the faults in aspect oriented software, resulting in weaving problems and ineffective testing [22]. We want to cover the faults that are not addressed, yet, by such coverage criteria which can provide effective testing technique.

1.2 RESEARCH QUESTIONS

The aim of the research is twofold. To motivate people towards aspect oriented programming paradigm because it helps to modularize implementation of crosscutting concerns (repeated code), easy maintenance of systems, greater code reusability, and ability to test application code automatically without disturbing the code. We emphasize the need for a model based testing technique for AOP because it [5, 12], explain requirements, enhances communication between developers and testers; design models, if available, eases the maintenance and can be used again for testing; the practice can be automated; and MBT can improvise error detection ability and minimizes the testing cost by automatically generating and executing loads of test cases.

Following are the questions that will be addressed by this research:

1. What faults are covered by existing techniques using different models and coverage criteria?
2. How model based testing can be performed using an aspect oriented model which can cover maximum number of faults?

1.3 RESEARCH METHOD

The research questions are answered by the following research process steps:

1. Literature review
2. Analysis of findings
3. Presenting an approach
4. Validation of the proposed approach

Following are the tasks that are carried out to achieve the specified goal:

- A study of literature to understand the concepts of Model Based Testing and the various techniques that are widely employed.
- A study of Aspect-oriented Programming issues with a focus on its constructs such as aspect, joinpoint, pointcut and advice and weaving between class and aspect and what can be the faults that can originate in weaving.
- An analysis and evaluation of different techniques proposed for model based testing of aspect oriented programs performed using a fault model and other generic evaluation parameters. The evaluation is used to discover which faults are being covered by the techniques, using which model and coverage criteria.
- A possible approach to increase the use of model based testing of aspect oriented programs.
- Validation of the proposed solution through examples.

- Partial automation of the proposed solution.

1.4 THESIS OUTLINE

The thesis is explained and organized using following chapters as:

Chapter 2: The second chapter explains the background of the problem domain, which is Model Based Testing, Aspect Oriented Programming and its modeling. Moreover, it gives a broad overview of its constructs and discusses different techniques to model these constructs in UML.

Chapter 3: The third chapter discusses the different techniques already proposed for model based testing of aspect oriented software. It performs evaluation of each individual technique proposed. The techniques in literature are compared and analyzed on a fault model and on different other parameters, to judge the contributions and limitations of each technique.

Chapter 4: The fourth chapter explains the technique that is proposed after the survey of literature and results are attained by applying model based testing on modeling of AOPs. The approach, proposed an algorithm to model an intermediate testable model, some coverage criteria and generation of test cases to cover maximum faults in an AOP, presented by a fault model.

Chapter 5: The fifth chapter validates the proposed technique in the prior chapter with the help of an example. The chapter provides an introduction to the selected example, which is used to explain the step by step process of the proposed technique. It is also evaluated by a fault model that how many different faults can be caught by the application of the proposed solution.

Chapter 6: the sixth chapter contains the result and evaluation of the proposed approach

Chapter 7: The seventh and final chapter of the thesis provides the conclusion of the work done. It discusses the contribution and limitation of the work and talk about future work that can support in further improving the model based testing of aspect oriented programs.

Chapter 2

Background

2 BACKGROUND

2.1 TESTING

Testing is an important task in software lifecycle to ensure its correct behavior. [13]. It is the only function that introduces faults in software to see if software is able to cover the problem or not. It helps to increase the efficiency and effectiveness of software. The testing process facilitates the confidence of developers, testers, vendors and clients towards software. The process is also a significant task as hundred percent testing cannot be done in any case, there remains possibility for hidden errors even an extensive testing have been performed.

2.2 MODEL BASED TESTING

Testing is now performed in many ways and MBT is one of the most popular and acceptable testing strategy [12]. Automating the process of test generation using model is widely known as Model Based Testing (MBT). MBT fulfilled the demand, of efficient and effective software performance, of different stakeholders such as customers, developers and testers. It also aids less cost and easy maintenance [14]. It is considered as a noteworthy process, as OMG also has proposed UML based testing profile [15]. The general procedure for performing model based testing is building a model, building a testable model, defining coverage criteria, generating test cases by applying coverage criteria and obtaining the test results in the end [14].

Model has got a fundamental position in MBT. Various models depict different behaviors of software, therefore, adapting and building of models must be performed with careful and expert advice [12]. In the literature [12] several models like finite state machine (FSM), state charts, Markov Chain [4], and UML [8] has been used for representing different software and their various behaviors. Sometimes, model of software does not represent an adequate amount of information that is required for testing, which can be a cause of failure of a system [16]. The thesis, with some modifications, adapts the modeling of [7] that extended UML which is a standard modeling notation and has got a diverse range of acceptance and usage. It allows representing both the dynamic and static nature of software.

A testable, intermediate model is build as sometimes the models are complex and test case generation from the design model becomes a laborious task and difficult. Occasionally it is difficult to test directly from a model if there is a lot of information present in the model, from which test cases cannot be derived. Therefore, at some places either data flow graph, control flow graph, or a simple flow graph is made, depending upon the requirements of a tester [4].

2.3 ASPECT-ORIENTED PROGRAMMING

Gregor Kiczales [36] and his team present the theory of AOP, at Xerox PARC. The need of aspect oriented programming (AOP) originated when there is demand of same type of security, exception handling, error handling etc at different points and levels of programming [17]. In contrast to procedural languages, AOP is not about executing step by step code of functionality [17]. Similarly, in contrast with OOP, AOP is not restricted to the boundaries of an object(s), its data and methods [17]. AOP enhances OOP but not replace it. AOP is meant to increase modularity and abstraction with the help of which maintenance, reuse and nonfunctional requirements of software are easy to implement.

The abstraction and modularization in AOP is increased when tangled code, dispersed throughout the software implementation, is encapsulated in an *aspect.* Aspect is the main modular unit of AOP that contains the crosscutting concerns, known as advice, and the points, known as pointcut, where a crosscutting concern has to weave in.

Pointcut is composed of different joinpoints that shows where the crosscutting functionality of an aspect will be executed at a particular point in a class. Pointcuts can be static or dynamic. Static pointcuts contain the joinpoints that are known before the execution time of a program. Dynamic pointcuts contain the joinpoints that appear in the flow of a program execution or at runtime.

An advice contains the body of a crosscutting concern and executes at the joinpoint with which it is attached. An advice can be before, after, around, after returning, before returning etc. Before advice executes before the specified jointpoint and after advice executes after the specified joinpoint. The around advice allows to cut down the normal execution of a class and replace the class' functionality with its own. It is primarily meant for handling exceptions in software.

Weaving is the process by which the concerns and crosscutting concerns work mutually with each other. The process of execution of an advice at some point of execution of a method of a class is known as weaving process.

Many programming languages support AOP and some new languages are also proposed for the implementation of aspect oriented programs. The languages are AspectJ, Perl, Aspect#, AspectC++ , AspectXML.

2.4 ASPECT ORIENTED MODELING

Aspect oriented modeling (AOM) facilitate AOP and help to represent it using models. Different techniques have been proposed for AOM [19], [30], [33], but there is no standard notation or profile for modeling AOP.

Banniasad and Clarke [18] gave an idea of "Theme" to represent advice and method. It combines UML sequence diagram and class diagram in a theme i.e. symbolize the dynamic and static nature of an advice and method. The technique also proposed the new weaving process by extending UML composition. It uses the operators such as merge and override for the weaving process.

Zakaria et al [19] presents a profile and extends UML to model AOP by using stereotypes, tagged values and constraints. The technique categorizes the aspects as abstract aspect, active and passive aspect. It also categorizes the composition or relationship of aspect and class using tagged values as control, report, track, validate, handle error, handle exception etc. Iconic representation is proposed for an aspect and pointcut. The aspect and pointcut, both, in the proposed modeling extend class' Meta model.

Basch and Sanchez [41], proposed to add two new constructs in UML to represent aspects and joinpoints using UML. Each aspect is represented with a UML package to be separated and encapsulated. The joinpoint is shown with a circle having cross in it. It is neither in the aspect nor in the core component but joins the two, to clarify that which aspect is weaving with which class. Each package of class and aspect contains the interaction diagrams such as sequence and collaboration diagram, which clearly shows that at which joinpoint which advice is invoked to be executed.

Chapter 3

Literature Survey

The chapter compares and analyzes different approaches proposed for model based testing of aspect oriented software, in the literature. It performs evaluation of each technique on a fault model proposed by Alexander [11] and on different other general parameters, to find out how approaches act upon model based testing issues of AOPs.

3 LITERATURE SURVEY

Model based testing of aspect-oriented programs using models is a new area of research. Few techniques have been proposed since 2005, which are assessed through different parameters, discussed below in the chapter.

In 2005 Xu and Xu [7] test the interactions between aspects and class using aspect diagram, class diagram and sequence diagram. The approach models AOP, both statically and dynamically, but tests using sequence diagram only. This is the only paper that discusses and gives algorithm for test data generation. It also tests polymorphic behavior of an object through polymorphic coverage criteria.

In the same year Xu et al. [20] propose first ever state based testing approach for AOP. They use FREE state model to depict the dynamic behavior of integration of class and aspects. Weaving mechanism shows the impact of aspects on class' state and transitions. An N+ coverage criterion is used to reveal all the sneak paths, corrupt states, wrong states and wrong paths.

Massicotte et al. [21] proposed an iterative and incremental, integration testing approach. It uses UML collaboration diagram to depict interactions between aspects and classes, in context of AspectJ. Multi-aspects are integrated incrementally in classes to localize the source of faults. The approach presents two phase aspect-class integration testing strategy: (1) **Static analysis:** generating test sequences from the dynamic interactions of aspects and classes. (2) **Dynamic analysis:** verification of execution of generated test sequences. An automated aspect is generated by the tool to track the sequences. Five testing criteria are adopted by the approach. These are:

- **Transition Coverage Criterion:** Test all transitions of objects at least once.
- **Sequence Coverage Criterion:** Test all the sequences i.e. group of transitions of objects at least once.
- **Modified Sequence Coverage Criterion:** all set of transitions i.e. all possible sequences that are affected by aspect must also be re-tested.
- **Simple Integration Coverage Criterion:** If only one aspect is weaved in a class then all the sequences should be tested at least once again, in which the affected method resides.
- **Multi Aspect Integration Coverage Criterion:** If a method is affected by many aspects then all the possible sequences containing the transition should be re-tested well.

Many researchers; till now have proposed many approaches for testing AOPs. Naqvi et al [22] in 2005 presents a survey on testing techniques for AOPs. The paper compares and analyzes three [34], [20], and [25] different techniques based on the fault model proposed by Alexander et al [11]. Deriving from their own learning Naqvi et al show that incorrect aspect precedence, failure to establish expected postconditions, incorrect focus of control flow are still not addressed by any approach and there is a lot of work to be done in the field.

Xu and Xu [23] in 2006 give an approach, based on FREE state model to test the impact of aspects on classes after incremental weaving. This is the only approach in literature that adopts regression testing. It defines rules to reuse the class tests on the aspect-oriented model. The aspect-oriented state model is not self explanatory; it does not show where an aspect and joinpoint resides. We have to see code to locate the classes and aspects in the model. The approach uses only branch coverage criterion to generate test cases.

Xu and Xu [24] in 2006, give the first ever state-based approach that introduces the concept of integration aspect in aspect-oriented model based testing. An integration aspect is composed of interactions between multiple classes with each other. Weaving mechanism is given, explicitly, that explains the integration of two classes in an aspect. Base classes and integrated classes can be unit tested, so this approach gives the paradigm to test the integration aspect. The aspect is tested through the interface of base classes. The test cases of classes are reused for integration aspect, but the approach does not give any specific principles to reuse them on integration. The approach claims to address two faults i.e. incorrect strength in pointcut patterns and advice implementation.

Xu et al. [25] in 2006, give the hybrid testing model based on state models and flow graphs. FREE state model is used to sketch the semantics of classes and aspects and flow graphs are used to reveal the essential details to test intra-class paths i.e. only methods and advices are converted to flow graphs. Control and data flow testing is performed on the resultant model.

Xu and He [26] in 2007, proposed to test an AOP by using its aspect oriented use cases. The paper focuses on formalizing the use cases to Petri nets. It gives an algorithm to model weaving between aspects and classes using Petri nets. No algorithm is given to generate test cases from the Petri nets of aspect oriented programs.

Jackson et al [27] target unit testing strategy, named as Kertheme. It tests core and crosscutting concerns, named as themes, separately, but not the interactions between them. It is a design validation technique, i.e. creating tests using design of software for avoiding late error identification. The approach tests the themes by merging sequence diagrams of test cases with the concerns. So, using semantic based weaving of scenarios and executable class diagram it is able to test theme approach. The benefits by the approach are: design can be easily validated and designer gets confident, the process of making and changing test cases becomes easy as there are isolated scenarios for tests, identification of errors becomes easier as expected and executed both will be in hand, change will cost little effort as themes are highly modular in nature, and its modularity also increases reusability. The approach does not point towards specific fault.

Its main drawback is that it does not check the interactions between themes, which can result in incorrect weaving and joinpoint selection, could be full of errors. The paper just goes for testing of individual theme but not for the whole integrated system. The integration of tested themes can also create many faults in a system.

Parizi and Ghani [28] give another survey on AOP testing strategies. This survey again is based on the fault model proposed by Alexander et al [11]. It includes five [34], [20], [25], [35], and [7] techniques in its studies and describes them; include their conclusions and future work. The comparison shown is same as that of the previous one i.e. of Naqvi et al [22].

Liu and Chang [29] in 2008, propose a new approach in the field that cater the dynamic nature of AOP. It is a state based testing approach. Object State Diagram (OSD) of state variables of classes is made using [6], Crosscutting Weaving Model (CWM) is proposed to model the weaving of aspects in classes and another algorithm is proposed to model the Aspect oriented Object State Diagram (AsOSD), that shows the state variables and their transitions between classes and weaved aspects. Based on AsOSD a procedure for test tree is given

Xu et al [42], in 2008, extended their own work [7]. They used the same models such as class diagram, aspect diagram and sequence diagram to build the aspect object flow tree. The aspect object flow tree is made by applying different coverage criteria such as condition coverage, polymorphic coverage and loop coverage (the new one) on the sequence diagram. They now claim to cover the three faults such as incorrect advice type, incorrect (weaker or stronger) pointcut strengths, and incorrect aspect precedence. The paper also gives ever first empirical study in the field.

Madadpour et al, [43] in 2011, propose an incremental testing strategy using UML activity diagram. The activity diagram of core concerns is made first and then the same diagram is made for aspects. Both the models are then weaved to represent an aspect oriented model using the extension mechanism of UML, such as stereotypes. The paper proposes three coverage criteria such as, action path coverage criterion, modified action path coverage criterion and multi-aspect integration coverage criterion. These coverage criteria are supposed to cover the three faults from the Alexander's fault model [11], such as, incorrect aspect precedence, incorrect advice type, and incorrect strength in pointcut patterns. The paper elaborates its technique using a case study and also automates the process.

3.1 PARAMETERS SELECTED:

The approaches in the survey are compared on various parameters. First six are from the fault model proposed by Alexander et al. [11]. The fault model describes six faults that can appear in aspect-oriented programs or models. The other ones are common to aspect-oriented programs and modeling. The approaches which have claimed to cover a fault(s) in the fault model and also those parameters which are found in the approaches are assigned tick. The parameters which are not found are assigned a cross. The parameters are described below:

3.1.1 Incorrect strength in pointcut patterns:

Pointcut selects different joinpoints where aspect has to weave. If the pointcut is too strong, some necessary joinpoints will not be selected and if the pointcut is too weak, additional joinpoints will be selected that should be ignored.

3.1.2 Incorrect aspect precedence

There can be joinpoints where more than one aspect can be weaved. Check whether the order of the aspects is according to the specification or not.

3.1.3 Failure to establish expected postconditions

Clients expect the post conditions according to the contracts, whether the aspects are weaved or not. Test if advices inserted in core concerns are affecting the post conditions or not.

3.1.4 Failure to preserve state invariants

Weaving of advices and introductions may cause change in the state invariants of core concerns. This fault may affect the post conditions also. Ensure that weaving does not cause violations of state invariants.

3.1.5 Incorrect focus of control flow

It should be tested whether execution reach the point where aspect is weaved or not.

3.1.6 Incorrect changes in control dependencies

Dynamic nature of aspects can change the control dependency at any time e.g. around advice. Check if there is alteration of behavior and if after alteration, program gives the expected results or not.

3.1.7 Incorrect Advice Implementation

Advice is the function that aspect weaves at a joinpoint. Check if implementation of an advice is correct or not.

3.1.8 Weaving Mechanism

It is a basic strategy of aspect oriented programming. Check if any weaving mechanism is given by the proposed approach or not.

3.1.9 Modeling Notation

As survey is on model based testing, it becomes important to check which modeling language is used. Whether standard UML, UML profile or any other modeling technique is used for the illustration of approach or not.

3.1.10 Artifact used

Which artifact (s) of a modeling technique is used by a solution?

3.1.11 Test Coverage Criteria

Coverage criteria facilitate testing by identifying paths that should be tested and also assess the testing technique. The parameter helps us to know which approach used which coverage criteria.

3.1.12 Static/Dynamic Testing

This parameter allows testing if an approach uses static diagram, dynamic diagram or both.

3.1.13 Case Study

It shows the implementation of proposed approach on a system, and how many approaches till now are validated by a case study.

3.1.14 Tool Support

It is automation of a proposed approach. The parameter checks how many of the approaches have been automated.

3.2 REVIEW TABLE:

Surveyed Papers	Fault Behavior						Generic Parameters							
	Incorrect strength in Pointcut patterns	Incorrect aspect precedence	Failure to establish Expected postconditions	Failure to preserve state invariants	Incorrect focus of Control flow	Incorrect changes in Control dependencies	Incorrect Advice Implementation	Weaving Mechanism	Modeling Notation	Artifact used	Test Coverage Criteria	Static/ Dynamic Testing	Case Study	Tool Support
Xu and Xu, 2005	×	×	×	×	×	√	×	√	AspectJ metamo del and UML	• Class • Aspect class • Sequenc e	• Branch • Polymorphi c	Dynamic	√	×
Xu et al. 2005	√	×	×	√	×	×	×	√	UML	FREE State model	N+	Dynamic	√	×
Badri et al. 2005	√	×	×	×	×	×	×	×	UML	Statecharts	• Transition • Sequence • Advice • Execution • Multi-Aspect Integration	Dynamic	√	×

Surveyed Papers	Fault Behavior						Generic Parameters							
	Incorrect strength in Pointcut patterns	Incorrect aspect precedence	Failure to establish Expected postconditions	Failure to preserve state invariants	Incorrect focus of Control flow	Incorrect changes in Control dependencies	Incorrect Advice Implementation	Weaving Mechanism	Modeling Notation	Artifact used	Test Coverage Criteria	Static/ Dynamic Testing	Case Study	Tool Support
Massicotte et al. 2005/ Massicotte et al. 2006	×	×	×	×	×	×	×	×	UML	Collaboration	• Transition, • Sequence, • Modified Sequences, • Simple Integration, • Multi-Aspect Integration	Dynamic	×	Partial
Xu and Xu 2006	√	×	×	×	×	×	√	√	UML	State model	• Branch	Dynamic	√	×

Surveyed Papers	Fault Behavior						Generic Parameters							
	Incorrect strength in Pointcut patterns	**Incorrect aspect precedence**	**Failure to establish Expected postconditions**	**Failure to preserve state invariants**	**Incorrect focus of Control flow**	**Incorrect changes in Control dependencies**	**Incorrect Advice Implementation**	**Weaving Mechanism**	**Modeling Notation**	**Artifact used**	**Test Coverage Criteria**	**Static/ Dynamic Testing**	**Case Study**	**Tool Support**
Xu and Xu, 2006	√	×	×	×	×	×	√	√	UML	State model	Modal Test Case Pattern	Dynam ic	√	×
Xu et al. 2006	×	×	×	×	×	√	×	√	UML	FREE state model	• Def-Use	Dynam ic	√	×
Xu and He, 2007	×	×	×	×	×	×	×	√	Petrinets	Petrinets	• Transition • State	Dynam ic	×	×

Surveyed Paper	**Fault Behavior**						**Generic Parameters**							
	Incorrect strength in Pointcut patterns	**Incorrect aspect precedence**	**Failure to establish Expected postconditions**	**Failure to preserve state invariants**	**Incorrect focus of Control flow**	**Incorrect changes in Control dependencies**	**Incorrect Advice Implementation**	**Weaving Mechanism**	**Modeling Notation**	**Artifact used**	**Test Coverage Criteria**	**Static/ Dynamic Testing**	**Case Study**	**Tool Support**
Jackson et al.2007	×	×	×	×	×	×	×	√	Theme/ UML	• Class • Sequence	×	Static and Dynamic	×	×
Liu and Chang 2008	×	×	×	×	×	×	×	√	UML	• Object state • Aspect object state	×	Dynamic	×	×
Xu et al 2008	√	√	×	×	×	×	√	√	UML	• Class • Sequence	• Condition • Polymorphic • Loop	Static and Dynamic	√	×

Surveyed Papers	Fault Behavior						Generic Parameters							
	Incorrect strength in Pointcut patterns	Incorrect aspect precedence	Failure to establish Expected postconditions	Failure to preserve state invariants	Incorrect focus of Control flow	Incorrect changes in Control dependencies	Incorrect Advice Implementation	Weaving Mechanism	Modeling Notation	Artifact used	Test Coverage Criteria	Static/ Dynamic Testing	Case Study	Tool Support
Madad pour et al 2011	√	√	×	×	×	×	√	√	UML	• Activity	• Action Path • Modifie d Action Path • Multi-Action Integrati on	Dynamic	√	√

Table 1 Review Table

√: Can be found by the approach.

×: Can not be found by the approach.

3.3 ANALYSIS:

We have analyzed the survey on the fault model proposed by Alexander et al [11], because it is the first model proposed in the field of AOP, it discusses general faults of AOP but not specific to pointcuts or other constructs and it is also referenced by two surveys [22, 28], performed already in the field. The results of two surveys [22, 28] are also considered. Some research papers in the field, also, referenced the fault model and claim to cover a fault(s) in it. The analysis is discussed as under:

1. Literature provides evidence that two faults of Alexander's fault model are not addressed by any of the approaches in the literature, the faults are: 'Failure to establish expected post conditions' and 'incorrect focus of control flow'. Every client is concerned about its decided and contracted post conditions whether or not any advice is weaved. Thus we have to design such advices that do not change the expected post conditions or produce the expected post condition in anyway, which is a difficult task. The advice inclusion can behave as a source of error in the core concern's post conditions that need to be tested thoroughly. 'Incorrect focus of control flow' is another fault that is not tackled by the literature. The static changes or additions done by an advice are being covered previously but it is hard to test the advice with its class in a dynamic context, which decides to be weaved at runtime. For example, in AspectJ the keywords cflow and cflowbelow are used to weave advice in the recursive calls. The keywords decide at runtime that which recursive call has to be weaved with the advice. Consequently, this type of weaving is likely to produce the said fault, which creates problems for the stakeholders.
2. Studies provide evidence that all of the work uses dynamic diagrams to model and test AOP, thus addressing the testing of dynamic nature of AOP. There is an exception of one approach [27], which tests using both static and dynamic part of program. The approaches used dynamic view of an AOP for testing because static diagrams only show the objects, attributes, operations and relationships of a system, they do not show the details of a system such as behavior of a system, an

action and its reaction, and how weaving affect the core concerns, which thing is affected and which is not. The dynamic diagrams show the collaborations between objects and changes to the internal states of objects, which clearly show the affect of an aspect to the core concern. Therefore, dynamic diagrams clearly represent the process and details of weaving and the affects of aspects on the core concerns.

3. Weaving mechanism is an important and a must part of AOP. All the work done in model based testing of AOP address weaving because there is still no standard notation or modeling language to model AOP and its weaving. Most of the researchers of model based testing have used newly proposed profiles of UML in the field of aspect oriented modeling. One of the testing techniques [26] has used petrinets to model AOP, but it results in weak testing strategy.
4. Empirical study and evaluation helps to observe the results of an approach by providing experimental results leading to the authenticity the approach. The existing literature gives only one evidence [42] of empirical study in this area, because AOP is a new programming paradigm and is not in use commonly in the field of programming. Consequently, there is very less work done in model based testing of aspect oriented programs which is hazardous to the empirical study.
5. Few of the techniques [21, 42] are supported by a tool, implemented their approach and showed the results. One of them has automated only the weaving process of AOP but the testing is manual and the other has automated only the testing process but not the weaving.
6. Most of the approaches [20, 23, 24, 25 and 29] have used state diagrams to represent the aspect oriented behavior of a system, because apparently, using the diagram, it becomes easy to represent the weaved behavior of an AOP, the weaving of before and after advice can be shown clearly and the also the detail of changing and making of new connections can be seen clearly. Furthermore, different state based techniques have covered different faults in a fault model.
7. Only one approach [7] has used sequence diagram to represent AOP. There is no standard modeling notation proposed yet, for AOP, therefore, it become difficult to represent AOP using sequence diagram. We cannot clearly model the different

types of advices in it but can clearly see that where and when an advice is invoked on a lifeline..We can also show the static insertion by an aspect to the classes. The thesis has also used sequence diagram to model AOP to fulfill the curiosity that why only one approach has used it and why the approach has not addressed the faults that are not covered yet at all.

8. Only one technique [43] has used activity diagram to perform testing on AOP. Model based testing of AOP is a new area of research and also needs different experimentations to know which model would be most helpful to cover which fault and as well which can cater maximum faults.

Chapter 4

Proposed Approach

4 PROPOSED APPROACH

Model based testing of aspect oriented program is a field of research which caters the problem and research of aspect oriented programming, modeling of aspect oriented programs, testing, model based testing and model based testing of aspect oriented software.

The proposed technique is adapted from the framework of model based testing. Block diagram of the proposed approach is shown in figure1 below. The proposed model based testing of aspect oriented software is based on sequence diagram of UML 2.0. The weaving of aspects and classes through pointcuts and advices, is shown by weaved sequence diagram proposed in [7], with some modifications.

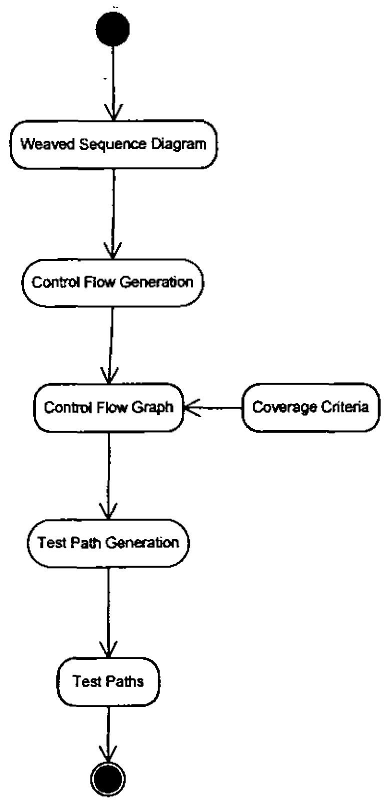


Figure 1 Activity Diagram of Proposed Approach

TH-8961

Weaved model of sequence diagram, adapted from [7], would behave as an input to the testing process. Control flow generation introduce an algorithm to build a control flow graph (CFG) from weaved sequence diagram which is an intermediary model of the technique. Control flow graph shows the sequence of control between various messages and advices. The thesis proposes an algorithm to generate CFG from weaved sequence diagram, that help further in testing. Faults specific to aspect oriented programming are considered and introduced, which affect CFG. Coverage criteria ensure an almost bugs free software and help to generate a finite number of test cases [4]. They are applied on CFG to see if a coverage criterion can cover the fault introduced or not. Test paths are generated, which are the end result of the technique. These help to track errors/faults present in weaved sequence diagram. The approach claims to cover two faults from the fault model [11], i.e. incorrect strength in pointcut pattern and failure to establish expected post condition

The steps of the proposed approach are explained using an example Rental Movie taken from [7]. It is about renting a movie. Whenever a transaction starts, an aspect authorizes the user, using an advice at pointcut. If the user is authenticated the transaction is continued, otherwise the transaction is truncated.

The proposed technique is also partially automated. The weaved sequence diagram is first converted to an XML format, then with some modifications the XML file is given as an input to the software to convert to a CFG. Coverage criteria are applied to the CFG and test paths are generated.

4.1 WEAVED SEQUENCE DIAGRAM

UML can be extended using stereotypes, tagged values and constraints [4]. There is no standard profile, notation or language to model aspect oriented software [30]. Different approaches are proposed to represent aspect oriented software by extension mechanism of modeling notations. In this thesis weaving of aspects and classes are represented by extending sequence diagram of UML 2.0. The idea of modeling is grasped from the approach proposed by [7], with some modifications. The aspect oriented modeling in [7] clearly models the process of weaving, create lifelines in sequence diagram for classes and advices and represents a joinpoint with writing comments on a link between messages and advices. The approach proposed in the thesis modifies the modeling in [7], by

- Treating aspects as classes in sequence diagram, similarly as it is treated in [30].
- Building separate lifelines for every aspect in a program.
- Representing pointcuts using stereotype on message arrow or link.

The example taken from [7], is shown using weaved sequence diagram in figure below with modifications.

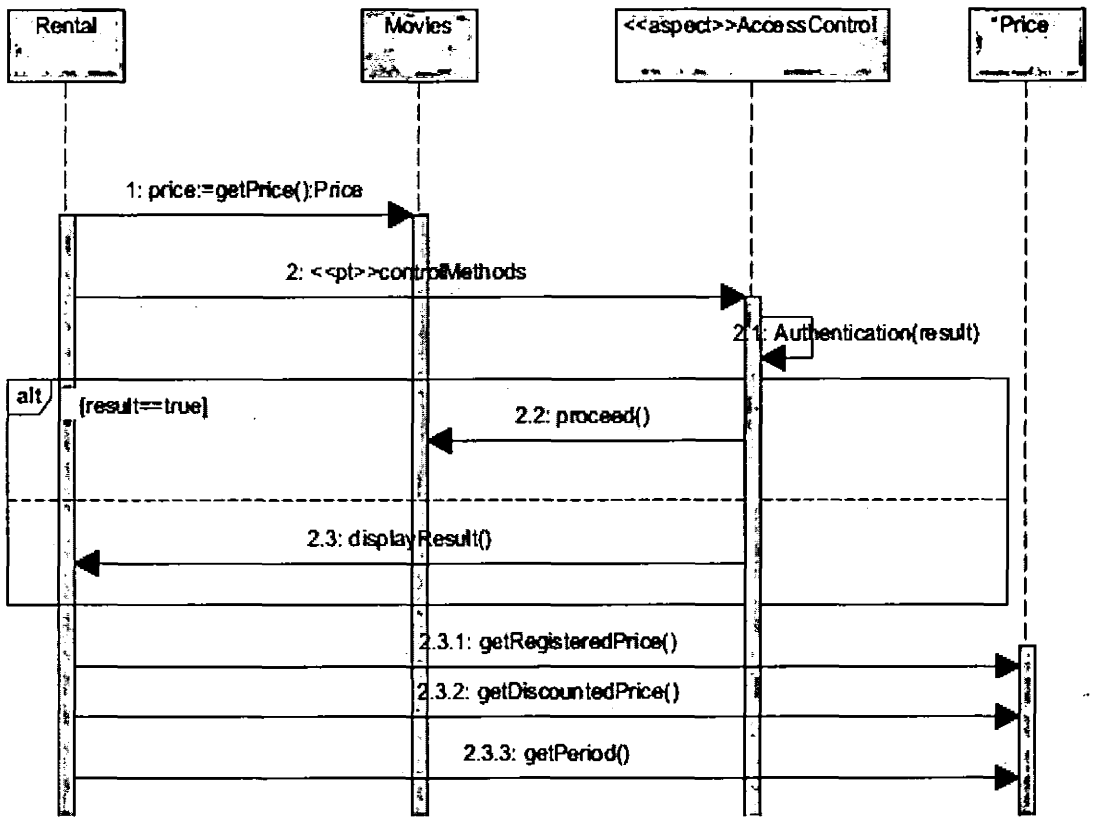


Figure 2 Weaved Sequence Diagram of RentalMovie

We can clearly see that the aspect AccessControl lays equivalent to the classes in the example, with a separate lifeline as well. Aspect and pointcut are represented using stereotypes <<aspect>> and <<pt>>. The placement of pointcut arrow shows that the advice in an aspect is weaved at the function call, as the pointcut is present right after the function call and its tail lays on the same lifeline as of the function call. Finally, weaved sequence diagram of RentalMovie symbolize that whenever a transaction starts, an aspect <<aspect>> AccessControl authorize the user, using an advice at pointcut <<pt>> controlMethods. The post conditions of the messages are written and saved in the description of each message that is why it is not visible in the sequence diagram.

Weaved sequence diagram is made using visual paradigm and XML of it, is generated automatically. The software of the system takes the XML of weaved sequence diagram as input and slightly modify the XML to be useful for the further operation.

4.2 CONTROL FLOW GENERATION

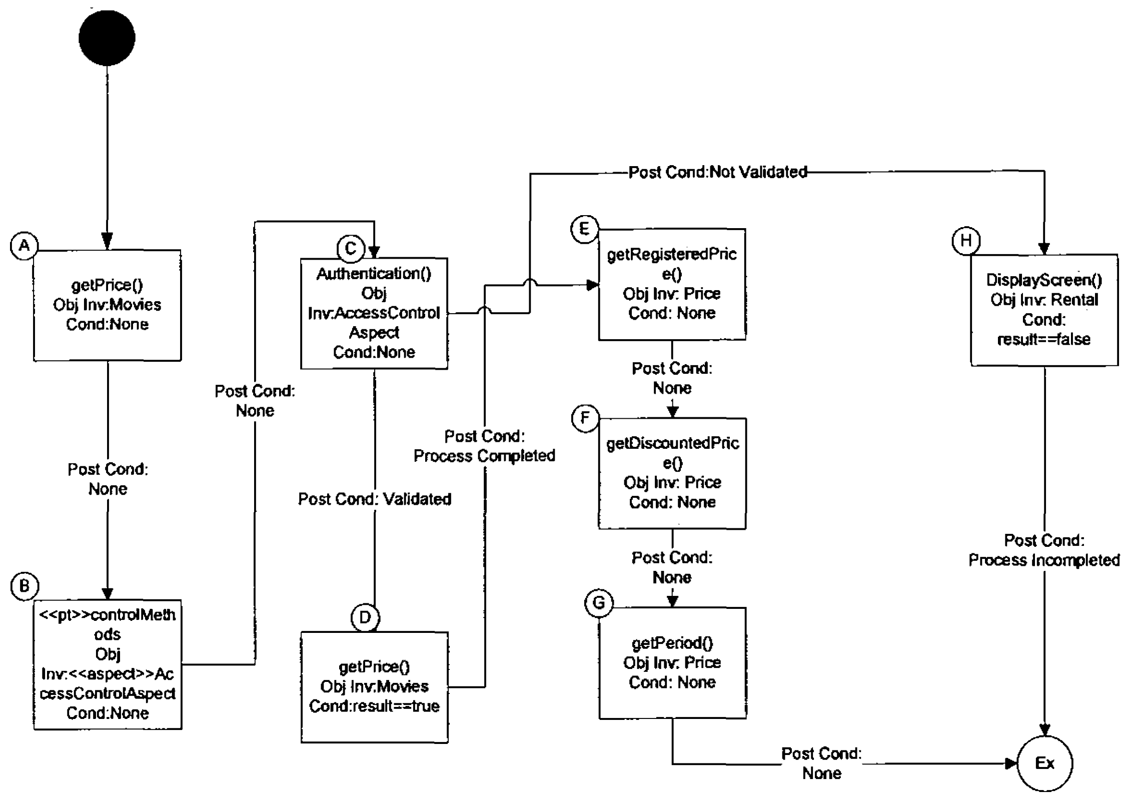
Control flow graph generation is used to test the aspect oriented sequence diagram and to test the control flow of a program that is affected by an aspect(s). Control flow graph is an intermediary output of the approach. It provides a simple way to track changes in a normal flow of program. It is helpful to generate the execution sequence of the design model. The control flow graph is composed of vertices and edges. In our control flow graph, the vertices represent messages of weaved sequence diagram and edges represent post condition of a message, if any. A general algorithm to construct a top to bottom flow graph from weaved sequence diagram is as under:

```
Algorihtm:  Create CFG (WSEQDIA (D))
Input:      WSEQDIA (D): Weaved Sequence Diagram of a problem/Software
Output:     CFG: Control Flow Graph for a sequence diagram
Begin:      M= set of messages in WSEQDIA (D)
                 Node [M];    Edge [M];

       For each message m Є M    do {
            Add {message no., message name, condition (if any), object invoked}
            to Node[m]
            Generate nodes in CFG for each Node[m]
            Add {post condition of m} to Edge[m]
            Connect Node[m] and Node [m+1] with Edge [m]
            }
End Create CFG
```

The sequence diagram of example RentalMovie shown in figure 2 is now converted to CFG using the algorithm illustrated above. Figure 3 below shows the control flow graph made by weaved sequence diagram shown in figure 2 above. The algorithm applied on sequence diagram puts first message of the sequence diagram to the first node of control flow graph with some information e.g. getPrice () is the first message in the sequence diagram and it is the first node in CFG. Message with stereotype is also made a node e.g. <<pt>> controlMethods. Each node contains the message number, message name, object invoked and condition (if any). The information is gathered to be used for performing further tasks proposed in the approach. Connect first node to the next node with an arrow. It is exited when there are no more messages in the sequence diagram. Each node is annotated with alphabets for ease of generating test paths from the CFG.

The software apply the proposed algorithm to the XML of weaved sequence diagram and build the required CFG out of it. The software shows it as under in figure 3:



A
getPrice() Obj Inv:Movies Cond:None
Post Cond: None
B
<<pt>>controlMethods Obj Inv:<<aspect>>AccessControlAspect Cond:None
Post Cond: None
C
Authentication() Obj Inv:AccessControl Aspect Cond:None
Post Cond: Validated
D
getPrice() Obj Inv:Movies Cond:result==true
Post Cond: Process Completed
Post Cond:Not Validated
E
getRegisteredPrice() Obj Inv: Price Cond: None
Post Cond: None
F
getDiscountedPrice() Obj Inv: Price Cond: None
Post Cond: None
G
getPeriod() Obj Inv: Price Cond: None
Post Cond: None
H
DisplayScreen() Obj Inv: Rental Cond: result==false
Post Cond: Process Incompleted
Ex

Figure 3 CFG build by Software using XML of Weaved Sequence Diagram

4.3 PROPOSED COVERAGE CRITERIA

Coverage criteria help to limit the test paths that can be infinite in number such as in case of loop structure [4]. The thesis proposes two coverage criteria that help to cover two faults from the sequence diagram of aspect oriented software. The proposed coverage criteria are as under:

4.3.1 All Message Sequence Coverage Criterion

A criterion *C* that traverses all nodes (messages) *Ni* in a CFG at least once, where *i=n* and *n* is the total number of nodes (messages) in a CFG. The idea of the criterion is adapted from [4]. The criterion in [4] provides larger coverage and tries to cover every possible node of a CFG. The thesis applies the proposed criterion on CFG and covers the fault incorrect strength in poincut patterns, proposed in the fault model [11]. It was necessary to propose the criterion because pointcut is the basic element of AOP, which directs an advice(s) to weave in. If the process of weaving is not correct then the whole execution is affected by it, which can create problems.

4.3.2 All Post Conditions Sequence Coverage Criterion

A criterion *C* that traverses all transitions (post conditions) *Ti* in a CFG at least once, where *i*=*n* and *n* is the total number of transitions (post conditions) in a CFG. It covers the fault failure to establish expected post conditions, present in the fault model [11]. It was necessary to propose the criterion because no other approach in the literature has addressed the fault before.

4.4 TEST CASE GENERATION

Here now test cases are generated after applying the proposed coverage criteria in the thesis.

4.4.1 Applying All Message Sequence Coverage Criterion

CFG is now used to generate test cases by using the coverage criteria. All message sequence coverage criterion help to generate the test cases constitute of all sequences of message. It is also automated and the test cases are shown as under:

- A-B-C-D-F-G-H-Ex
- A-B-C-E-Ex

4.4.2 Test Path Generation after Inserting Faults

The approach covers some faults specific to aspect oriented software. We now insert faults which affect the sequence diagram and as well as CFG and see by applying the coverage criterion, that the proposed testing approach is able to test the inserted fault or not. We claim that all message sequence coverage criterion after traversing all messages in a CFG can cover the fault i.e. incorrect strength in pointcut pattern, inserted in software. -

If pointcut "public pointcut controlMethods():getPrice(price Movie.price)", is modified to "public pointcut controlMethods():get*(*)" then the advice will execute at every function with the name started with get, that will create errors in a software. The CFG will now be extended by the following nodes, as under in figure 6:

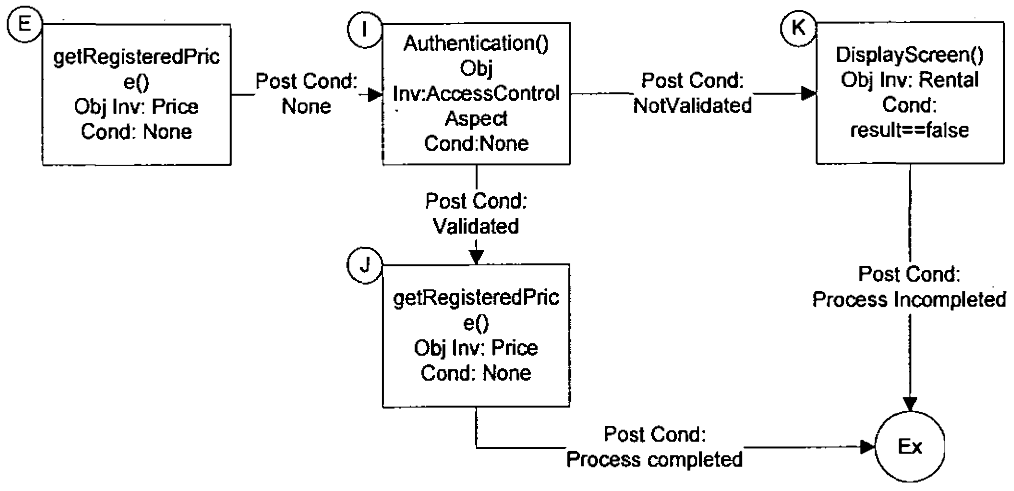


Figure 4 Extended part of CFG by the insertion of faults

Now the test cases generated by the all message sequence coverage criterion are:

- **A-B-C-H-Ex**
- **A-B-C-D-E-I-J-Ex**
- **A-B-C-D-E-I-K-Ex**

4.4.3 Applying All Post Condition Sequence Coverage Criterion

All post condition coverage criterion traverse all post conditions in CFG and compares expected sequence of post conditions to the obtained sequence of post conditions. Expected sequence of post conditions, before inserting faults, obtained from the generated CFG is shown as under:

- **None-None-Not Validated-Process Incomplete-None-None-Ex**
- **None-None-Validated-Ex**

4.4.4 Test Path Generation after Inserting Faults

If same fault is inserted and shown in CFG in figure, then the obtained sequence of post condition is as under:

- **None-None-Validated-Process Completed-None-Validated-Process Completed-None-None-Ex**
- **None-None-Validated-Process Completed-None-Not Validated-Process Incomplete-Ex**

It is claimed that after applying the coverage criterion to the CFG the fault, failure to establish expected post condition, can be easily caught.

4.5 Proposed faults

Beside already proposed fault models specific to aspect oriented software, the thesis also proposes new faults that can appear in aspect oriented software. The faults are:

4.5.1 Missing Advice

Different faults can arise if there is a pointcut without an advice i.e. there is no advice signature as well as its implementation.

4.5.2 Missing Advice Implementation

Different faults can arise if a pointcut has an advice signature but no advice implementation.

Chapter 5

Validation of Proposed Solution

5 VALIDATION OF PROPOSED SOLUTION

The proposed technique in this thesis is first elaborated in the previous chapter with the help of short example Rental Movie taken from [7]. It is also now validated with a large example, Transaction Management taken from [32]. The following sections give an introduction to the example, how it is modeled and how the partially automated, proposed testing technique is able to capture the faults, inserted in the example.

5.1 TRANSACTION MANAGEMENT

Transaction management is larger example taken from [32], as compared in the previous chapter, to demonstrate and validate the technique proposed in the thesis. Transaction management encapsulates the crosscutting concerns residing and touching the normal execution of banking system. Banking system constitutes common functionality, such as, debit, credit and transfer of currency between different accounts, where as transaction management monitors and facilitates the processes to move in a smooth way. The following text explains the core concerns and crosscutting concerns, their weaving and testing using coverage criteria.

5.1.1 Weaved Sequence Diagram

Weaved sequence diagram is made by weaving the different functionality present in the classes and aspects. It is shown in the figure below.

Figure 5 Weaved Sequence Diagram of Transaction Management System

5.1.2 Control Flow Generation

CFG is generated using the algorithm proposed in the thesis. CFG of above weaved sequence diagram generated by the software is shown below:

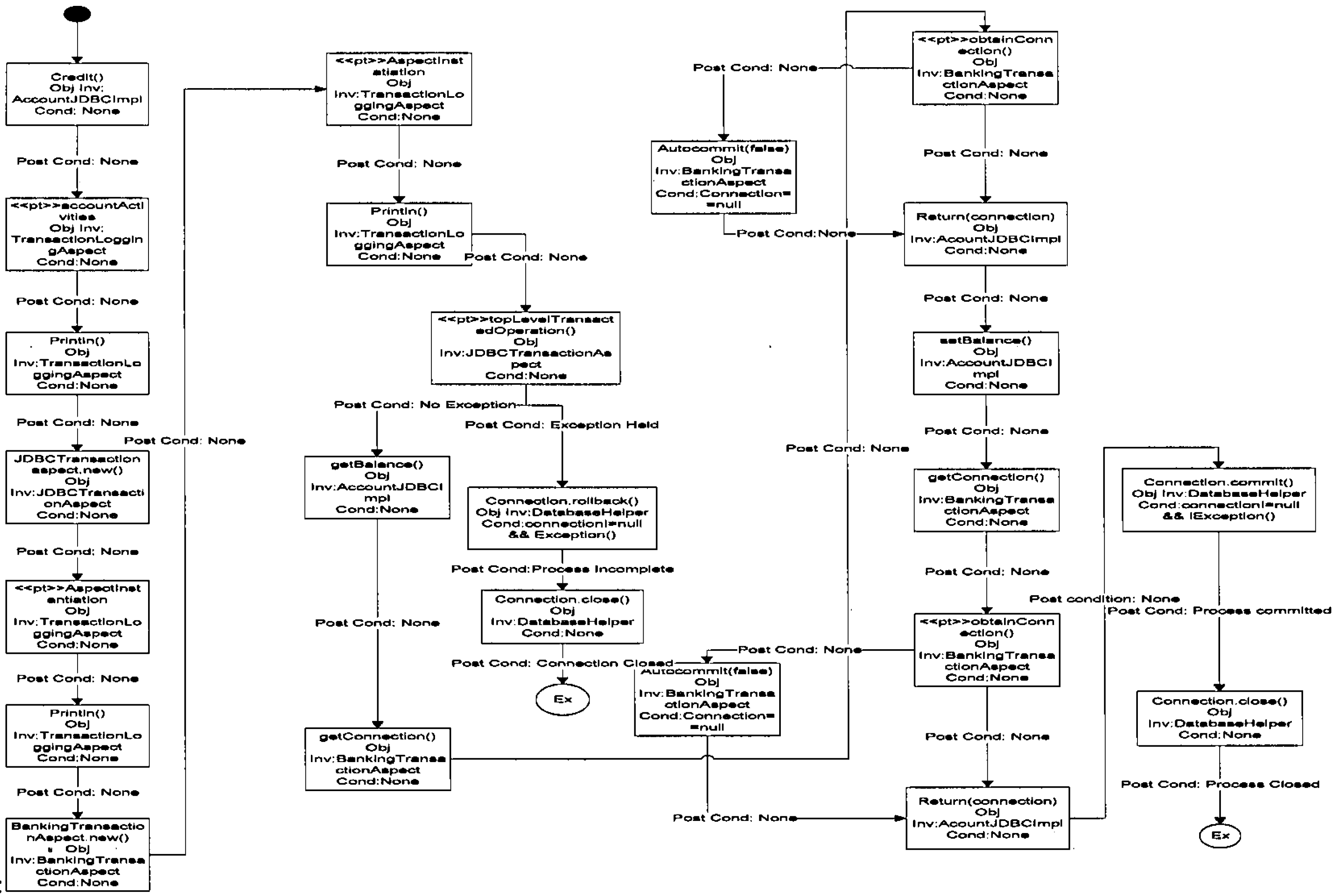

Credit() Obj Inv: AccountJDBCImpl Cond: None
Post Cond: None
<<pt>>accountActivities Obj Inv: TransactionLoggingAspect Cond:None
Printin() Obj Inv:TransactionLoggingAspect Cond:None
JDBCTransaction aspect.new() Obj Inv:JDBCTransactionAspect Cond:None
<<pt>>AspectInstantiation Obj Inv:TransactionLoggingAspect Cond:None
Printin() Obj Inv:TransactionLoggingAspect Cond:None
BankingTransactionAspect.new() Obj Inv:BankingTransactionAspect Cond:None
<<pt>>AspectInstatiation Obj Inv:TransactionLoggingAspect Cond:None
Printin() Obj Inv:TransactionLoggingAspect Cond:None
<<pt>>topLevelTransactedOperation() Obj Inv:JDBCTransactionAspect Cond:None
Post Cond: No Exception
Post Cond: Exception Held
getBalance() Obj Inv:AccountJDBCImpl Cond:None
getConnection() Obj Inv:BankingTransactionAspect Cond:None
Connection.rollback() Obj Inv:DatabaseHelper Cond:connection!=null && Exception()
Post Cond:Process Incomplete
Connection.close() Obj Inv:DatabaseHelper Cond:None
Post Cond: Connection Closed
Ex
Autocommit(false) Obj Inv:BankingTransactionAspect Cond:Connection==null
<<pt>>obtainConnection() Obj Inv:BankingTransactionAspect Cond:None
Return(connection) Obj Inv:AcountJDBCImpl Cond:None
setBalance() Obj Inv:AccountJDBCImpl Cond:None
getConnection() Obj Inv:BankingTransactionAspect Cond:None
Post condition: None
Post Cond: Process committed
Connection.commit() Obj Inv:DatabaseHelper Cond:connection!=null && !Exception()
Connection.close() Obj Inv:DatabaseHelper Cond:None
Post Cond: Process Closed
Ex

Figure 6 CFG created from Weaved Sequence Diagram of Transaction Management System

5.1.3 Test Case Generation

5.1.3.1 Test Path Generation by Applying All Message Sequence Coverage Criterion

Below are the test paths generated by applying all message sequence coverage criterion.

- A-B-C-D-E-F-G-H-I-J-N-O-P-Q-V-R-S-T-V-W-X-Ex
- A-B-C-D-E-F-G-H-I-J-N-O-P-V-R-S-T-V-W-X-Ex
- A-B-C-D-E-F-G-H-I-J-N-O-P-V-R-S-T-U-V-W-X-Ex
- A-B-C-D-E-F-G-H-I-J-N-O-P-Q-V-R-S-T-U-V-W-X-Ex
- A-B-C-D-E-F-G-H-I-J-K-L-Ex

5.1.3.2 Test Paths Generated by Applying All Post conditions Sequence Coverage Criterion

The test paths generated by applying the all post conditions sequence coverage criterion are as under:

- None- None- None- None- None- None- None- None- None- None-No Exception- None- None- None- None- None- None- None- None- None- Process Completed-Ex
- None- None- None- None- None- None- None- None- None- None-No Exception- None- None- None- None- None- None- None- None- Process Completed-Ex
- None- None- None- None- None- None- None- None- None- None-No Exception- None- None- None- None- None- None- None- None- None- Process Completed-Ex

- None- None- None- None- None- None- None- None- None- None-No Exception- None- None- None- None- None- None- None- None- None- None- Process Completed-Ex
- None- None- None- None- None- None- None- None- None- None- Exception Held- Process Incomplete-Ex

Chapter 6

Results

6 RESULTS OBTAINED

The results obtained after applying the two coverage criteria are evaluated in this chapter. Same set of paramete
the approaches of the literature were compared. The evaluation is shown using a table.

Proposed echnique	Fault Behavior						Generic Parameters					
	Incorrect strength in Pointcut patterns	Incorrect aspect precedence	Failure to establish Expected postconditions	Failure to preserve state invariants	Incorrect focus of Control flow	Incorrect changes in Control dependencies	Incorrect Advice Implementation	Weaving Mechanism	Modeling Notation	Artifact used	Test Coverage Criteria	Static/ Dynamic Testin
na Akbar	√	×	√	×	×	×	×	√	UML	• Sequence diagram.	• All Messages Sequence Coverage Criterion • All Post Conditions Sequence Coverage Criterion	Di Te

Table 2 Results Obtained

The table above clearly shows that the proposed technique uses UML sequence diagram to cover the two faults i.e. incorrect strength in pointcut patterns and failure to establish expected post condition. It uses two coverage criteria to capture the two faults in an AOP. The technique is validated with two examples and is also partially automated.

Literature survey of the field illustrate that only few of the researches have used UML sequence diagram and two faults of fault model [11] are not addressed by any of them. The research in the thesis found the gap and wanted to know why this gap is left and how to fill up the space, so that we can have arguments against or in favor of it.

The research only able to cover one uncovered fault using sequence diagram, but also we have come to the result that state diagram would be more specific to cover the fault ‘unable to establish post conditions.”

Chapter 7

Conclusion and Future Work

7 CONCLUSION AND FUTURE WORK

This work proposes a model based testing approach for aspect oriented programs. Aspects, classes and their weaving is shown using UML Sequence diagram. CFG is made by proposed algorithm, to facilitate the process of testing from the model. Coverage criteria are applied to CFG to generate test paths. The technique is validated using two examples, which prove that it helps to cover two faults of fault model [11], such as, incorrect strength in pointcuts and failure to preserve post conditions. The process is also partially automated.

Our future work will be to cater other faults in fault model [11], by some amendments in the same technique or by a new approach. It is also aimed that the process may be automated at its maximum, so as to make the process easily used by testers.

8 REFERENCES

[1] M. Utting, A. Pretschner and B.Legeard :"A Taxonomy Of Model-Based Testing," Working paper series ISSN 1170-487X, April 2006, Department of Computer Science, The University of Waikato, Private Bag 3105 Hamilton, New Zealand.

[2] M.J. Harrold, J.D. McGregor,, and K.J. Fitzpatrick, "Incremental Testing of Object-Oriented Class Structure," in Proceedings of the 11th International Conference on Sojtwam Engineering, pages 68-80, May 1992.

[3] IBM Research Report Software Debugging, Testing, and Verification Brent Hailpern, Padmanabhan Santhanam

[4] R. V. Binder, "Testing Object-Oriented Systems: Models, Patterns, and Tools," Addison-Wesley Professional, Boston, 1999.

[5] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "On Evaluation of Model-Based Testing and its Automation," in Proceedings of the 27th International Conference on Software Engineering (ICSE'05), 2005.

[6] D. C. Kung, P. Suchak, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen, "On Object State Testing," in Proceedings of the 18th Annual International Computer Software and Applications Conference, 1994, pp. 222-227.

[7] W. Xu and D.Xu, "A model-based approach to test generation for aspect-oriented programs," in Proceedings of the AOSD 2005 Workshop on Testing Aspect-Oriented Programs, Chicago, March 2005.

[8] http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

[9] B. Beizer, "Black-Box Testing Techniques for Functional Testing of Software and Systems," Wiley, 1995.

[10] J. Boberg, "Early Fault Detection with Model-Based Testing," in Proceedings of the 2008 SIGPLAN workshop on ERLANG

[11] R. T. Alexander, J. M. Bieman, and A. A. Andrews, "Towards the Systematic Testing of Aspect-Oriented Programs," Technical Report No. CS-4-105, Department of Computer Science, Colorado State University, Fort Collins, CO, 2004.

[12] I. K. El-Far, and J. A. Whittaker, "Model-Based Software Testing," in Encyclopedia on Software Engineering (edited by Marciniak), Wiley, 2001.

[13] Mrs. R. Jeevarathinam, Dr. A. S. Thanamani, "Test Case Generation using Mutation Operators and Fault Classification," (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 1, 2010

[14] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz, "Model-Based Testing in Practice," in Proceedings of the International Conference on Software Engineering, 1999.

[15] utp.omg.org/

[16] T. Dinh Trong, "A Systematic Procedure for Testing UML Designs," in Proceedings of ISSRE, 2003.

[17] Patent no. US 6467086 B1

[18] E. Baniassad and S. Clarke. "Theme: An Approach for Aspect-Oriented Analysis and Design," in Proceedings of the International Conference on Software Engineering, pages 158–167,2004.

[19] A. A. Zakaria, H. Hosny and A. Zeid, "A UML Extension for Modeling Aspect-Oriented Systems," in Proceedings of Aspect Modeling with UML workshop at the 5th International Conference on the Unified Modeling Language, the Language and its Applications, September 2002.

[20] D. Xu, W. Xu, and K. Nygard, "A State-Based Approach to Testing Aspect-Oriented Programs," in Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, 2005, pp. 366-371.

[21] P. Massicotte, M. Badri, and L. Badri, "Generating Aspects-Classes Integration Testing Sequences: A Collaboration Diagram Based Strategy," in Proceedings of the 3rd ACIS International Conference on Management and Applications, 2005, pp. 30-37.

[22] S. A. A. Naqvi, S. Ali, and M. U. Khan, "An Evaluation of Aspect Oriented Testing Techniques," in Proceedings of the IEEE Symposium on Emerging Technologies, 2005, pp. 461-466.

[23] D. Xu and W. Xu, "State-Based Incremental Testing of Aspect-Oriented Programs," in Proceedings of the 5th International Conference on Aspect-Oriented Software Development, 2006, pp. 180-189.

[24] W. Xu and D. Xu, "State-Based Testing of Integration Aspects," in Proceedings of the2nd Workshop on Testing Aspect-Oriented Programs, 2006, pp. 7-14.

[25] W. Xu, D. Xu, V. Goel and K. Nygard, "Aspect Flow Graph for Testing Aspect-Oriented Programs," http://www.cs.ndsu.nodak.edu/~wxu/research/436-1111jd.pdf

[26] D. Xu and X. He, "Generation of Test Requirements from Aspectual Use Cases," in Proceedings of the Workshop on Testing Aspect-Oriented Programs, 2007, pp. 7-14.

[27] A. Jackson, J. Klein, B. Baudry, and S. Clarke, "KerTheme: Testing Aspect Oriented Models," in Proceedings of the workshop on Integration of Model Driven Development and Model Driven Testing (ECMDA'07), Bilbao, Spain, July, 2007.

[28] R. M. Parizi and A. A. Ghani, "A Survey on Aspect-Oriented Testing Approaches," in Proceedings of 5th International Conference on Computational Science and Applications, IEEE, 2007.

[29] C. H. Liu And C. W. Chang, "A State-Based Testing Approach for Aspect-Oriented Programming," Journal Of Information Science And Engineering 24, 11-31, 2008.

[30] T. Aldawud, and A. Bader, "UML Profile for Aspect-Oriented Software Development," in Proceedings of the 3rd International Workshop on Aspect Oriented Modeling, 2003.

[31] D. Stein, S. Hanenberg, and R. Unland, "An UML-based Aspect-Oriented Design Notation for AspectJ," in Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD), 2002.

[32] R. Laddad, "AspectJ in Action". Manning, 2003.

[33] C. Chavez, and C. Lucena, "A Metamodel for Aspect-Oriented Modeling," in Proceedings of the Workshop on Aspect-Oriented Modeling with UML, 2002.

[34] J. Zhao, "Data-Flow-Based Unit Testing of Aspect Oriented Programs," in Proceedings of 27th Annual IEEE International Computer Software and Applications Conference (COMPSAC'2003), Dallas, Texas, December 2003, pp 188-197.

[35] C.C. Lopes and T. C. Ngo, "Unit-Testing Aspectual Behavior," in Proceedings of the Workshop on Testing Aspect-Oriented Programs (WTAOP), held in conjunction with the 4th International Conference on Aspect-Oriented Software Development (AOSD), 2005.

[41] M. Basch and A. Sanchez, "Incorporating Aspects into the UML," in Proceedings of the AOM workshop at AOSD, 2003.

[42] W. Xu, D. Xu,, and W. E. Wong, "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.

[43] S. Madadpour, S. H. M. Hosseinabadi and V. Abdelzad. Article: "Testing Aspect-Oriented Programs with UML Activity Diagrams," International Journal of Computer Applications 33(8):4-11, November 2011.