Automated Test case generation using UML diagrams based on behavior

Ashish Verma
Research Scholar, NITTTR, Chandigarh, India

Dr. Maitrayee Dutta
Associate Professor, Department of Computer Science, NITTTR, Chandigarh, India.

Abstract - Although several approaches for automated test case generation have been proposed over the last few years, automated test case generation for object oriented application based on their behavior is still in an infant stage. This thesis proposed a new framework for automated test case generation from UML diagrams for object oriented applications based on the behavior. This thesis implements this framework with the development of a tool integrates all types of automated test case generation from four UML diagrams (Class Diagram, Sequence diagram and State-chart Diagram and Use case Diagram) in an object oriented application. Parsing the Petal files to generate the collaborative Test cases from these UML Diagrams using Pattern Discovery and Information Retrieval is discussed.

Keywords: Software Testing, Automatic Test cases, UML (Unified Modeling Language), Class Diagram, Sequence diagram and State-chart Diagram, Use Case Diagram, Pattern Discovery.

I. INTRODUCTION

Now-a-days, computer applications have diffused into every sphere of life, for manipulation of several sophisticated applications. Many of these applications are of very large, complex and safety critical. Thus, highly reliable software is essential. In other words, the good quality software with high reliability is most essential. Apart from existence of many techniques for increased reliability, software testing is an important and common methodology followed. So, testing remains the most important part of quality assurance in the practice of software development. Although so many quality assurance techniques like formal specifications, design reviews, model checking, and inspection, exists till today, further furnishing method of testing is required for effective testing. Because software now has such an important role in our lives both economically and socially, there is pressure for software professionals to focus on quality issues. Poor quality software that can cause loss of life or property is no longer acceptable to society.

The need for better quality means more pressure for software testing and for test engineers taking care of it. Software Testing is the process of executing a program with the intention of finding errors. Every software code has been reviewed and verified through Software Quality and Assurance activities but these activities are not sufficient. Every time the software delivered to the client has been thoroughly tested by client before sending it to the production. Thus QA Team has to test the software before it gets to the client. Testing information flow is said to be as a testing technique which specifies the strategy to select input test cases and analyze test results. Most software engineers would agree that testing is a vital component of a quality software process, and is one of the most challenging and costly activities carried out during software development and maintenance. The software Testing process has been described as a series of phases, procedures, and steps that result in Delivering Quality Software Product to the production. Testing thus can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.

Some Researcher has derived the automatic test cases using class diagram, sequence diagrams and state chart Diagram, but not considered Use case diagram functionality. In this paper, we use UML class diagram, sequence diagrams and state chart Diagram, Use case diagram as design specifications, and implemented the idea of parsing the Petal files to generate the collaborative Test cases from these UML Diagrams using Pattern Discovery and Information Retrieval process. Rational Rose tool has been used to produce a model of system design. We proposed a novel approach for automated test case generation from these four UML diagrams for object oriented applications based on the behavior. This thesis implements this framework with the development of a tool integrates all types of automated test case generation from four UML diagrams in an object oriented application.

The paper is structured in the following way: In next section, we presented related work of generating test cases by using different UML Diagrams. In section 3, we proposed methodology for Test case generation, while section 4, we discussed the implementation of our approach with a case study and describe the experimental evaluation of the proposed model. Then after a clear Conclusion, last section concluded the paper and gives an overview of our work.

II. Related Work

Philip Samuel, Rajib Mall and Sandeep Sahoo have presented a novel testing methodology to test object-oriented software based on UML sequence diagrams [7]. This paper has presented an approach to generate test cases automatically from UML sequence diagrams using dynamic slicing technique [7]. Monalisa Sarma, Debasish Kundu, Rajib Mall have presented a novel approach of generating test cases from UML design diagrams [9].

Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall have presented a strategy for integration testing which combines information from use case and sequence diagram. This methodology predominantly uses the sequence diagram of UML 2.0 for generation of test cases [3].

Supaporn Kansomkeat, Jeff Offutt, AynurAbdurazik, Andrea Baldini presents a single project experiment on the fault revealing capabilities of model- based test sets [4].

Puneet E. Patel, Nitin N. Patil generates test cases from activity diagram. Also a test coverage criterion, called activity path coverage criterion has been implemented [1].A.V.K. Shanthi1 and G. Mohan Kumar focuses on test cases generation by means of UML Sequence diagram using Genetic Algorithm which best test cases are optimized and the test cases validated by prioritization. For generating the test case from the sequence diagram, they first extract the necessary information from the diagram. For that write parser in java to extract all possible information from file. Based on the extracted information, a Sequence Dependency Table (SDT) is generated. With the help of SDT test path are generated, by applying the GA most prioritized test case are generated [13].

Supaporn and Wanchai have proposed the automatic testing technique to solve partially the testing process [11]. This research paper has presented an attempt to generate test cases from TFG (Testing flow graph) and then select test cases to detect any fault. Firstly, state chart diagram has been altered to graph called TFG (Testing flow graph) [11]. Secondly, test cases have been generated by traversing the flow graph and using the coverage criteria of the state and transition of diagrams. Finally, using mutation analysis test cases have been evaluated to detect any fault [11].

Li Bao-Lin, Li Zhi-shu, Li Qing and Chen Yan Hong have presented a new test cases generation approach that is based on UML sequence diagrams and Object Constraint Language/OCL [8]. The sequence diagram has been transformed to a tree representation. Firstly by selecting conditional predicates from sequence diagram whole constructed tree has been traversed. Then, pre and post conditions have been applied with the help of OCL. OCL alter the conditional predicates on sequence diagram and thus test cases have been generated by applying function minimization technique [8].

A.V.K Shanthi, Dr. G. Mohan kumar proposed setting up several tests adequacy criteria with respect to class diagrams, an automatic approach is presented to generate test cases for Object oriented programs using Data mining searching techniques[14]. Yi-Tin Hu and Nai-Wei Lin use UML class diagrams and the Object Constraint Language to describe the specification of Java methods. The automatic generation of test cases is based on the unification mechanism and the powerful constraint solving mechanism of constraint logic programming [37].P. Samuel R. Mall A.K. Bothra has developed a novel method to automatically generate test cases based on UML state models. The test cases generated satisfy transition path coverage criteria and can be used to test cluster-level state-dependent behaviors [12]. Chayanika Sharma discussed the applications of GA in different types of software testing. The GA is also used with fuzzy as well as in the neural networks in different types of testing. It is found that by using GA, the results and the performance of testing can be improved [15].

Pakinam N. Boghdady, Nagwa L. Badr, Mohamed A. Hashim, Mohamed F. Tolba[2] proposes an enhanced XML-based automated approach for generating test cases from activity diagrams. The proposed architecture creates a special table called Activity Dependency Table (ADT) for each XML file.

M. Prasanna and K. R. Chandran [6] suggested “Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm. Class collaboration testing will be accomplished in similar way as testing of individual class occurs. They have used language such as Object Constraint Language/OCL [17] or a natural language, and/or as a state transition diagram to generate test cases for a class.

Information from all these diagrams have been accumulated and whichever form is most consistent is used to develop test cases. They used operations of classes to generate test cases and execution-based testing to test the class. In execution based testing assertion checks has been added to the class code to find the bugs.

Some more research has been reported to generate test cases based on class [18]. For testing these two types of diagrams a control flow graph has been developed that contains multiple entities. Now the entire traditional graph based test coverage techniques can be applied to this control flow graph as outlined in [18]. This includes branch coverage, path coverage and round trip scenario coverage criteria [19]. Since UML diagrams are always more abstract and provides ease to generate test cases than control flow graphs so researchers have started using UML diagrams to generate automated test cases from UML diagrams.

Mary V. Ashley, Isabel C. Caballero [16] presented a new optimization framework for sibling reconstruction from single generation microsatellite genetic data. Binder has also presented some work to generate test cases from state chart diagram. He also proposes a new method called state reporter method that effectively access and report internal state information whenever invoked [19]. Ranjita swain, P. K. Behara, D. P. Mohapatra also presented an approach for generated the Automatic Test Case from UML State Chart Diagram from function minimization [20].

III. Methodology for Test Cases Generation

Generally test cases have been generated from individual UML diagram, but the test cases generated by single diagram are not effective and efficient because test cases that have been generated depend on single view only. If combination of diagrams has been used then test cases generated will be more efficient and effective as they will cover both static and dynamic aspects of the system.

Thus the system focuses on the different UML diagrams for test cases generation: Class Diagram, Sequence Diagram, State chart Diagram and use case Diagram. Various types of information can be extracted from these diagrams to generate test cases.

For static information class diagrams have been used and for dynamic type of information sequence and state chart diagrams have been used. Class diagram has been used to get the information related to Classes, Relationship between classes/Association, Dependency, Generalization/ Parent-child relationship, Operations, Cardinality, and Attribute. Sequence diagram has been used to get the information about Object interactions i.e. messages sent to other objects, Precondition for a particular message, Post-condition for a particular message.

State chart diagram has been used to get the information about Initial state of a system, final state of a system, Guard conditions, Transitions. Combination of class, sequence and state chart diagrams i.e. all the required information from each diagram has been extracted to generate test cases for a particular system.

Use Case Diagrams 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. Use case diagrams represent the functions of a system from the user's point of view. This shows business use cases, actors, and the relationships between them. The relationships between actors and business use cases state that an actor can use a certain functionality of the business system.

Following are the major steps in prepared technique:

1) Input UML Diagram Petal Files: The Petal file of class, sequence and state chart diagrams and Use Case Diagram has been given as input to the developed tool GUI.
2) Parsing petal File: Petal file has been parsed by the tool by parsing tokens and string matching mechanism has been used to find a pattern like class name, class attributes, class cardinality, class operations, inheritance, dependency etc.
3) Is pattern found: If pattern has been found then it is temporarily stored in the designated buffer else next pattern has been found.
4) Store it in a Buffer: All patterns found have been stored in a designated buffer. There are different temporary buffers for every pattern like class name buffer, class attributes buffer, class cardinality buffer, class operations buffer, inheritance buffer, dependency buffer etc.

5) Search for another pattern: Then searches for various patterns in petal file. For example if class name has been found then class name has been entered into the class name buffer else if class attributes has been found, then appends it to the class attributes buffer similarly so on.
6) Is EOF (End of File): System keeps on searching the patterns until EOF?
7) Create bulk insert query from buffer and store it in database: When end of file has been reached, then generates the bulk insert query to minimize the Database access which contains all data about class diagram in form of tuples which can be executed and insert the information into database.
8) Parsing all Petal files: After class diagram, Petal files of sequence diagram and state chart diagram and Use case Diagram have been parsed sequentially through same process.
9) Extract strings to generate test cases: The database which contains the information from class diagram, sequence diagram, state chart diagram and Use case Diagrams has been used to generate test cases by extracting the correlating information. System has been implemented to generate automated test cases from these diagrams.

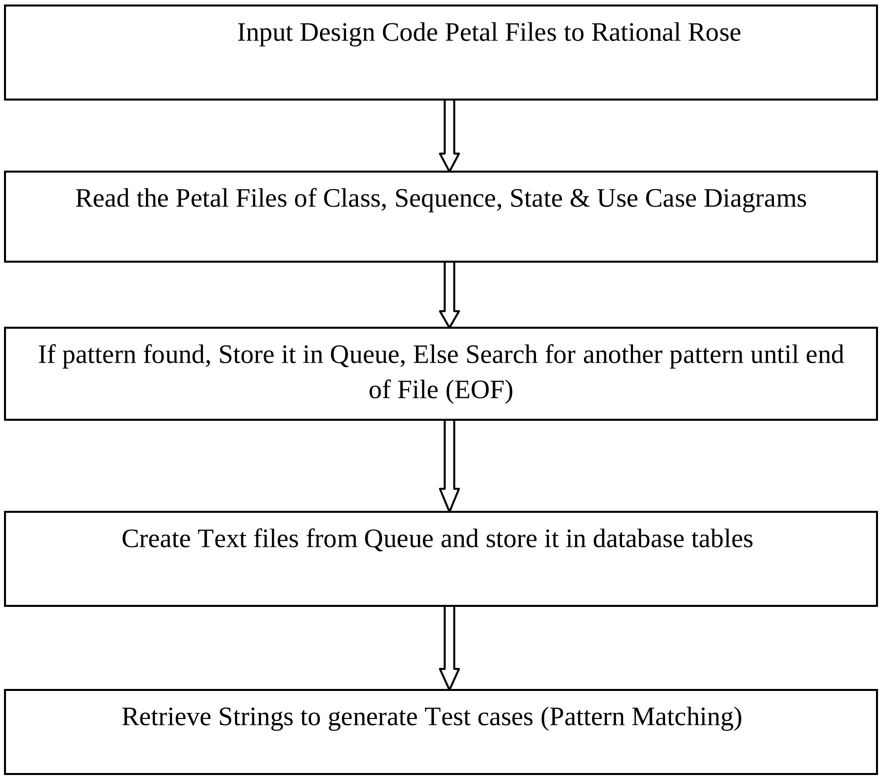

Input Design Code Petal Files to Rational Rose
Read the Petal Files of Class, Sequence, State & Use Case Diagrams
If pattern found, Store it in Queue, Else Search for another pattern until end of File (EOF)
Create Text files from Queue and store it in database tables
Retrieve Strings to generate Test cases (Pattern Matching)

Fig 3.1 Methodology for Designing the Framework

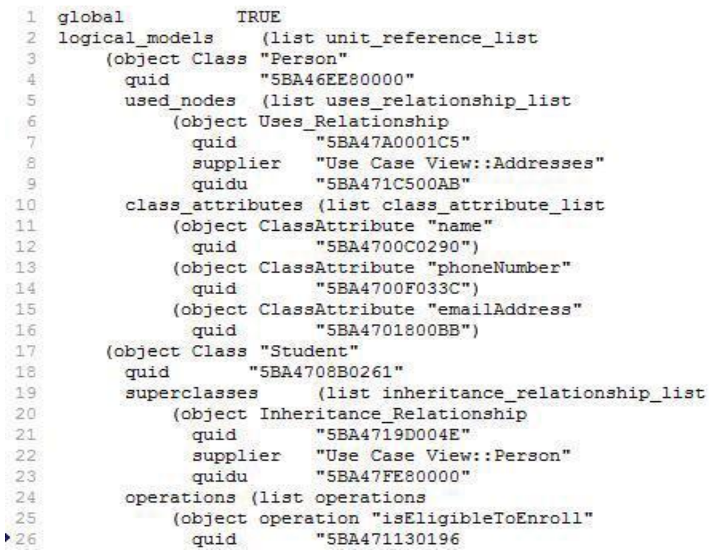
Class diagram and State Chart diagram for a particular application has been drawn using Rational Rose or any CASE tool. Class diagram has classes, which in turn have operations and attributes.

For a particular operation of a class in a class diagram, draw a sequence diagram. These diagram’s petal files have been entered as input to the tool, which are the model structure of software application. Petal file are model structured language file contains the entities and their relationship corresponding to each other. Figure shows the Petal File of class diagram. In Line 3 “Person” is class name, in line 6 shows the Dependency relation, in line 11, “name” is class attribute, in line 20 shows the inheritance relationship and in line 25 “isEligibleToEnroll” shows the class operations.

Tool reads the petal file line by line. It tokenizes the line into words and then matches the string with the pattern to find the class name, class attributes, class cardinality, class operations, inheritance, and dependency. When class name has been found, it has been entered to database with the related class attributes along with i.e. its attributes, its operations, its inheritance classes, its dependency, its cardinality and every string has been stored. There are 6 different attributes has been fetched for class diagram petal file i.e. class name, attributes, operations, inheritance classes, dependency, and cardinality.

Tool keeps on reading the file and searching the pattern until EOF (End of File). As EOF has been reached, two text files have been generated by tool that contains information from class diagram. First one contains information about class name, class attributes, class operations, inheritance, dependency and second one contains class's cardinality information.

.



```
global          TRUE
logical_models    (list unit_reference_list
    (object Class "Person"
      quid        "5BA46EE80000"
      used_nodes  (list uses_relationship_list
          (object Uses_Relationship
            quid        "5BA47A0001C5"
            supplier    "Use Case View::Addresses"
            quidu       "5BA471C500AB"
      class_attributes (list class_attribute_list
          (object ClassAttribute "name"
            quid        "5BA4700C0290")
          (object ClassAttribute "phoneNumber"
            quid        "5BA4700F033C")
          (object ClassAttribute "emailAddress"
            quid        "5BA4701800BB")
    (object Class "Student"
      quid       "5BA4708B0261"
      superclasses     (list inheritance_relationship_list
          (object Inheritance_Relationship
            quid        "5BA4719D004E"
            supplier    "Use Case View::Person"
            quidu       "5BA47FE80000"
      operations (list operations
          (object operation "isEligibleToEnroll"
            quid        "5BA471130196
```

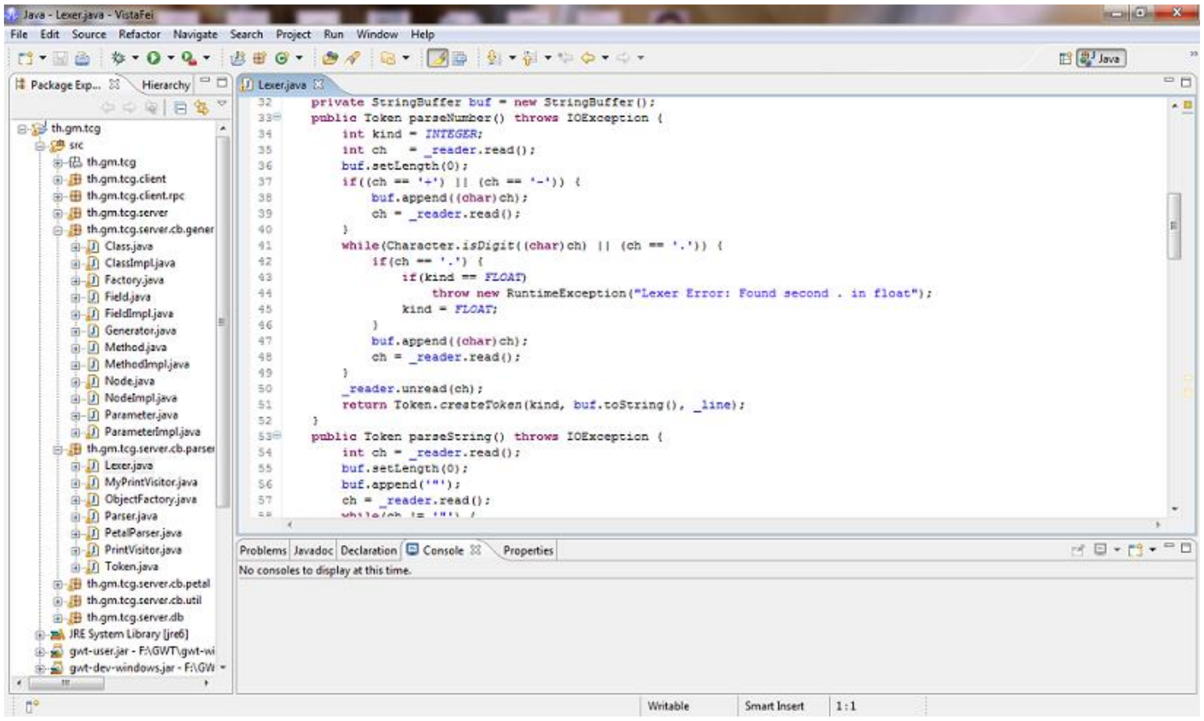
Figure 3.2 Sample Class Diagram Petal File

Similarly the Petal files of sequence diagram and State Chart diagram and Use case Diagram have been read by the tool. From Sequence diagram Pre-conditions, Post-conditions and messages send to other objects have been searched by the tool, and from State Chart diagram initial state, final state, guard conditions have been searched by the tool developed. In last, tool searched the actor, use cases, Associations, Subsystem boundaries, relationships (Include, Extend and Generalization) from the Use Case diagram.
Once searching has been over all the data in the form of text file has been generated by the tool. Text file of sequence diagram contains information in the form of tuples as Pre-condition, Post-conditions and messages. Similarly text file of State Chart diagram contains information in the form of tuples as guard conditions, initial state, and final state. Similarly text file of Use Case diagram contains actor, use cases, Associations, Subsystem boundaries, relationships (Include, Extend and Generalization) Tables have been created to log the application as well as the related data to store in with all test cases.

IV: CASE STUDY

The tool developed has been tested with an application of Student Enrollment System (SES). The class diagram represents the static view of the system, now each operation, attribute, association in class can be used to derive test cases. A single class in a class diagram can be used to derive test cases. Similarly State Chart Diagram for whole system can be drawn and test cases can be derived by covering all states, all events, all actions and guard conditions. A snapshot of Class Diagram Petal file of Student Enrollment system is depicted.
The novelty in proposed work is that single operation in a class has been used along with its sequence diagram, state chart diagram and Use Case Diagram of overall system to generate test cases. As in this case study, the class student has one operation Registered() has been used and corresponding to that operation, data of sequence diagram has been mapped. If pattern is found in other Diagrams tables then data of the same has been appended to the Test case. So sequence and state diagram and Use case's petal file for this operation has been used to derive test cases.
Test cases have been generated with the help of stored strings in database in the form of tables. In database the table of sequence diagram has three columns named Pre-condition, Test Sequence, and Post-condition.
Pre-conditions and post-conditions in sequence diagram have been given in documentation of each particular message of sequence diagram.

Column Precondition and Post condition in sequence diagram represents pre condition and post condition from sequence diagram. Test sequence column in sequence diagram is nothing but a message from one object to another in a sequence diagram. State Chart diagram data has three columns named Transition, initial state, final state.
Use case Diagram has 4 tables showing actor, use cases, Associations showing Subsystem boundaries. Use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. An association exists whenever an actor is involved with an interaction described by a use case.
Column transition contains the guard condition of each transition. Column initial state and final state contains information of initial state and final state of a particular transition. Test cases have been generated using inner join of two tables sequence diagram table and State Chart diagram table.



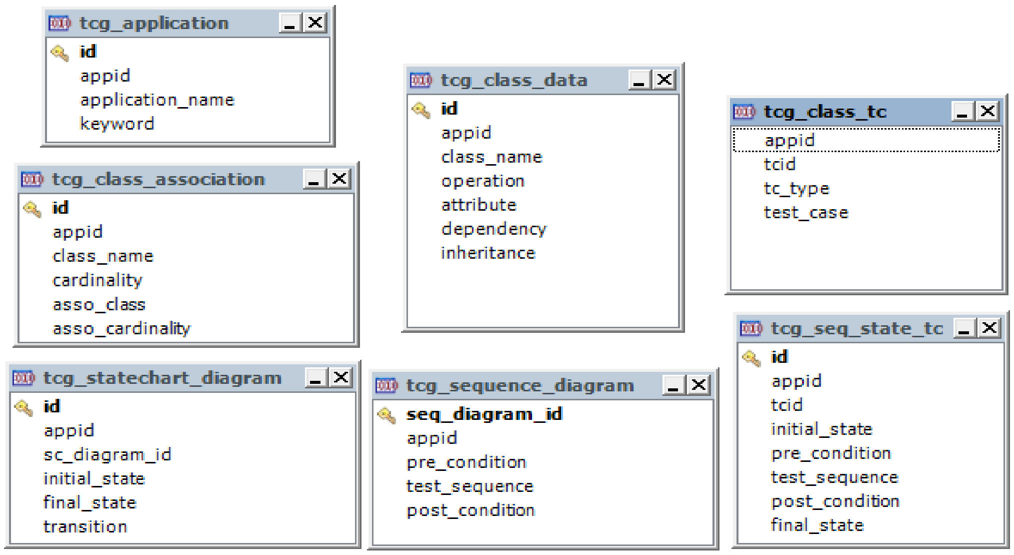
```
    private StringBuffer buf = new StringBuffer();
    public Token parseNumber() throws IOException {
        int kind = INTEGER;
        int ch   = _reader.read();
        buf.setLength(0);
        if((ch == '+') || (ch == '-')) {
            buf.append((char)ch);
            ch = _reader.read();
        }
        while(Character.isDigit((char)ch) || (ch == '.')) {
            if(ch == '.') {
                if(kind == FLOAT)
                    throw new RuntimeException("Lexer Error: Found second . in float");
                kind = FLOAT;
            }
            buf.append((char)ch);
            ch = _reader.read();
        }
        _reader.unread(ch);
        return Token.createToken(kind, buf.toString(), _line);
    }
    public Token parseString() throws IOException {
        int ch = _reader.read();
        buf.setLength(0);
        buf.append('"');
        ch = _reader.read();
        while(ch != '"') {
```

```
No consoles to display at this time.
```

Figure 3.3: Eclipse Text Editor View and Running the Main Project

The implementation of algorithm to parse the petal file for generating test cases is carried out in Java using Eclipse IDE. Figure 3.3 shows the environment of Eclipse and program running process The steps of implementation are given above.

Parsing process is a process to extract a small item from the original item. Different attributes involved in this process are class object attribute, inheritance attribute, relationship attribute such as Association, Association class, generalization, aggregation and composition. Extracted attributes has been stored as data in appropriate tables. These tables have been used to implement the tool generation process in Automated Test case generation that is proposed in our main research. The rules are syntactic, semantic and pragmatic. UML notations can be identified graphically via diagrams such as class diagram, state chart diagram, sequence diagram and Use Case Diagram. Tool reads the petal file line by line. It tokenizes the line into words and then matches the string with the pattern. When class name has been found, it has been entered to database with the related class attributes along with i.e. its attributes, its operations, its inheritance classes, its dependency, its cardinality and every string has been stored.



tcg_application
- id
- appid
- application_name
- keyword

tcg_class_association
- id
- appid
- class_name
- cardinality
- asso_class
- asso_cardinality

tcg_statechart_diagram
- id
- appid
- sc_diagram_id
- initial_state
- final_state
- transition

tcg_class_data
- id
- appid
- class_name
- operation
- attribute
- dependency
- inheritance

tcg_sequence_diagram
- seq_diagram_id
- appid
- pre_condition
- test_sequence
- post_condition

tcg_class_tc
- appid
- tcid
- tc_type
- test_case

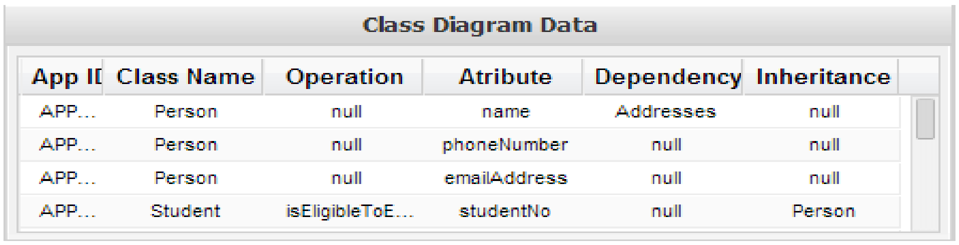
tcg_seq_state_tc
- id
- appid
- tcid
- initial_state
- pre_condition
- test_sequence
- post_condition
- final_state

Figure 3.4: Some Database Table Structure for the Tool

if precondition column of one Table from one diagram is similar to Transition column of other Table from other diagram , thus inner join query is used to generate test cases from sequence diagram and State Chart diagram and Use case Diagram. All the data from two or more tables are pulled out to generate test cases. Similarly test cases have been derived for other operations in class diagram.

Now class diagram has been combined to generate more effective test cases. Every class diagram has dependency, inheritance and association relationships.

The data fetched from class diagram has been shown in Table 4.1. There are two tables for class diagram, first Table which contains information about Class name, Operation, Attribute, Dependency, and Inheritance and other Table contains the information of cardinalities of class and associated class.

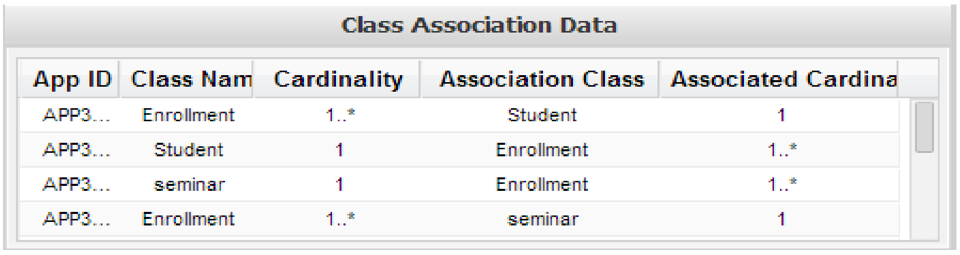


Class Diagram Data

App II	Class Name	Operation	Atribute	Dependency	Inheritance
APP...	Person	null	name	Addresses	null
APP...	Person	null	phoneNumber	null	null
APP...	Person	null	emailAddress	null	null
APP...	Student	isEligibleToE...	studentNo	null	Person

Table 4.1 Class Diagram Data

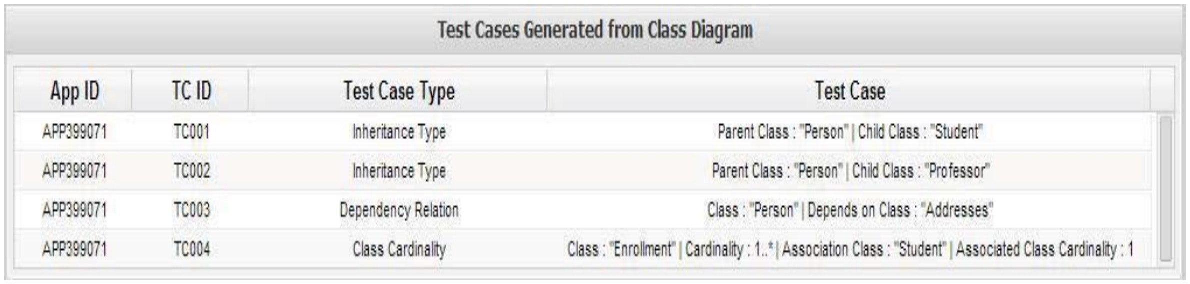
The table for association contains columns Class name, Cardinality, Association class and associated cardinality. Now from the table, the information about the inheritance relationship between classes can be extracted. It can be found which is parent and which is child as shown in table, person class is parent of student and professor. Thus any method that has been overridden in child can be found.



Class Association Data

App ID	Class Nam	Cardinality	Association Class	Associated Cardina
APP3...	Enrollment	1..*	Student	1
APP3...	Student	1	Enrollment	1..*
APP3...	seminar	1	Enrollment	1..*
APP3...	Enrollment	1..*	seminar	1

Table 4.2 Class Diagram Association Data



Test Cases Generated from Class Diagram

App ID	TC ID	Test Case Type	Test Case
APP399071	TC001	Inheritance Type	Parent Class : "Person" | Child Class : "Student"
APP399071	TC002	Inheritance Type	Parent Class : "Person" | Child Class : "Professor"
APP399071	TC003	Dependency Relation	Class : "Person" | Depends on Class : "Addresses"
APP399071	TC004	Class Cardinality	Class : "Enrollment" | Cardinality : 1..* | Association Class : "Student" | Associated Class Cardinality : 1

Table 4.3: Test Cases Generated from Class Diagram

These relationships have been used now to generate test cases. A class person depends on a class addresses so any test case derived for class addresses has been also used in class person for testing.
Similarly Test Cases from State chart, Sequence and Use case diagrams has been derived by parsing and pattern matching within its tables.
Extract strings to generate test cases as database which contains the information from class diagram, sequence diagram, state chart diagram and Use Case Diagram has been used to generate test cases by extracting the correlating information. System has been implemented to generate automated test cases from these diagrams.
Similarly class student and class professor has inheritance relationship with class person, so all test cases derived for operations of class person has been used as such in all child classes for testing child classes, also if there is any other method in child class or there exists any method in parent class that has been overridden in child class then it is needed to derive test cases for that method too.

V. CONCLUSION & FUTURE SCOPE

either test cases have been generated manually or generated from only one type of diagram, or these test cases do not shows input, output, pre-condition and post condition, or these test cases depend on only one aspect (dynamic or static) of system. This thesis derives motivation from this concept and uses integrated tool to generate test cases from all UML diagrams automatically from the behavior of the system. There is a good potential for reducing overall effort for test case generation from object oriented applications automatically from UML diagrams as the system features are broadly classified during design phase.

Test case generation from design specifications has the added advantage of allowing test cases to be available early in the software development cycle, thereby making test planning more effective.

This thesis proposes a new framework for automated test case generation from UML diagrams for object oriented applications based on the behavior. This thesis implements this framework with the development of a tool integrates all types of automated test case generation from different UML diagrams in an object oriented application.
This thesis performs studies for identifying which UML diagrams are most suited for test case generation for a system to perform functional testing, integrated testing, module testing, Work flow testing etc. based on behavior of the application. Furthermore, the development of integrated tool to generate test cases from all diagrams is need as future expect.

VI. REFERENCES

[1] Patel, Puneet E., and Nitin N. Patil. "Test cases Formation Using UML Activity Diagram." *Communication Systems and Network Technologies (CSNT), 2013 International Conference on*. IEEE, 2013.

[2] Boghdady, Pakinam N., et al. "An enhanced test case generation technique based on activity Diagrams. "Computer Engineering & Systems (ICCES), International Conference on IEEE, 2011.

[3] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram. Int.J. of Software Engineering, IJSE Vol.3 No.2 July 2010.

[4] SupapornKansomkeat, Jeff Offutt, AynurAbdurazik, Andrea Baldini, A Comparative Evaluation of Tests Generated from Different UML Diagrams", Technical Report, George Mason University, 2008.

[5] Y.G.Kim, H.S.Hong, D.H.Bae, and S.D. Cha. "Test cases generation from UML state Diagrams". IEEE Proceedings Software, 146(4), 1999, pp 187-192.

[6] M. Prasanna, K. R. Chandran, "Automatic Test Case Generation for UML Objectdiagrams using Genetic Algorithm" Int. J. Advance. Soft Computer. Appl., Vol. 1, No. 1, July 2009.

[7] Philip Samuel, Rajib Mall and SandeepSahoo"UML Sequence Diagram BasedTesting Using Slicing", IEEE Indicon 2005 Conference, Chennai, India, 11-13 Dec. 2005, pp 176-178.
[8] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong, "Test Case automate Generation from UML Sequence diagram and OCL Expression" School of Computer Sichuan University.Chengdu 610064, China. pp 1048-52.
[9] MonalisaSarmaDebasishKunduRajib Mall, "Automatic Test Case Generation fromUML Sequence Diagrams", 15th International Conference on Advanced Computing and communications pp 60-64.
[10] P. Samuel R. Mall A.K. Bothra, "Automatic test case generation using unifiedmodeling language (UML) state diagrams", IET soft. 2008 Vol. 2, No 2, pp 79-93/doi: 10.1049/iet-sen: 20060061.
[11] SupapornKansomkeat and WanchaiRivepiboon "Automated-Generating Test CaseUsing UML State chart Diagrams" Proceedings of SAICSIT 2003.
[12] PSamual, R. Mall, A.K. Bothra, "Automatic Test Case Generation using UML StateDiagrams" IET Software., 2008, Vol. 2, No 2, pp 79-93/79.
[13] A.V.K. Shanthi1 and G. Mohan Kumar "Automated Test Cases Generation fromUML Sequence Diagram", 2012 International Conference on Software and Computer Applications (ICSCA 2012,)IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore.
[14] A.V.K. Shanthi1 and G. Mohan Kumar "A Novel Approach ForGenerationof Test Cases Using TABU SEARCH ALGORITHM", ISSN 2250-0987 *Shanthi*et al, UNIASCIT, Vol. 2 (2), 2012, 222-224.
[15] Chayanika Sharma 'A Survey on Software Testing Techniques using Genetic Algorithm" IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 1, No 1, January 2013.
[16] Mary V. Ashley, Isabel C. Caballero "New Optimization Model and Algorithm for Sibling Reconstruction from Genetic Markers", INFORMS Journal on Computing, Articles in Advance, pp. 1–15, ©2009 INFORMS.
[17] Jos Warmer and AnnekeKleppe. "The Object Constraint Language: Precise Modeling with UML".Boston, MA: Addison-Wesley. 1999.
[18] B. Beizer. "Black-Box Testing, Techniques for Functional Testing of Software and Systems." Wiley, New York, 1995.
[19] R. Binder. "Testing Object-Oriented Systems: Models, Patterns and Tools". Addison-Wesley, 2000.
[20] Ranjita swain, P. K. Behara, D. P. Mohapatra, "Automatic Test Case Generation FromUML State Chart Diagram", International Journal of Computer Application (0975- 8887), Volume 42- No. 7, 2012.