AN AUTOMATED APPROACH TO UML TESTING

Anbunathan R*

AnirbanBasu**

ABSTRACT

Model-based testing has been gaining attention among the software testing community for Functional Testing. With increasing use of UML methodologies, generating test cases from UML diagrams is being looked at as an attractive proposition. In this paper a systematic procedure is presented for functional test case generation from UML diagrams. Functional test cases are generated from UML Sequence diagrams and MC/DC test cases are generated from Activity diagrams, after Sequence and Activity diagrams are translated to corresponding XMI files. A case study is presented to illustrate the application of the method. The advantages of the proposed method with others are also discussed in details.

KEY WORDS: UML diagram, UML testing, Test case generation, Sequence diagram, Activity diagram, XMI files, Model Based Testing, Test automation.

* **Test Manager, and Research Scholar, Bharathiar University, Coimbatore, India**

** **Professor, Department of CSE, APS College of Engineering, Bangalore, India**

1. Introduction

Model-Based Testing (MBT) involves developing a model from functional requirements and helps in generation of test cases for all scenarios. Successful utilization of MBT requires software requirements to be precisely defined in order to accurately characterize the system behavior [2]. With the proliferation of OO systems and with the increasing use of UML diagrams, UML testing dealing with generation of test cases from Sequence diagram and Activity diagram is gaining acceptance [15].

Many methods have been proposed for UML Testing. The method proposed in this paper covers both manual and automated way of generating test cases from Sequence and Activity diagrams. Test Cases from Sequence diagram help to check if system behavior is as per the expected timing sequence. Activity diagram based test cases help to test dynamic behavior of the system, using different combination of test inputs. The method proposed here generates Multiple Conditions/Decisions coverage (MC/DC) [13] test cases from Activity diagram. These test cases help to achieve 100% test coverage. Also this method generates reduced number of test cases, using Pairwise testing method. The proposed method is discussed in Section 3. The method is illustrated in Section 4with an example and compared with other methods in Section 5.

2. Related Work

This section discusses other methods that have been proposed for UML Testing. Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems by Emanuela G. Cartaxo[4], introduced UML Sequence diagram for test case generation. Another approach is UML Sequence diagram along with OCL (Object Constraint Language)[6], which is used for generating test cases. Test framework for Model Based Testing (MBT) is proposed in [10] which incorporate UML test profile. One more approach is converting Sequence diagram into XML file, parsing XML file, and then generating test cases [12].Monalisa Sarma et al. [7] transformed a UML Use case diagram into a graph called Use Case Diagram Graph (UDG) and Sequence Diagram into a graph called the Sequence Diagram Graph (SDG) and then integrated UDG and SDG to form the System Testing Graph (STG). The STG is then traversed to generate test cases for system testing. They [7] have used state-based transition path coverage criteria for test case generation. There are some other methods [17][18][19] that derive test cases from UML

diagrams using a similar approach. But none of the methods discuss generation of MC/DC test cases.

In [4], Labeled Transition Systems (LTS) based test case generation has been proposed. Test case generation algorithms from LTS and tools to support this generation are also available. UMLAUT (Unified Modeling Language All pUrposes Transformer) is a tool for the manipulation of UML models that includes the UML Sequence diagram. It is a general framework for UML model transformation [20]. It is developed within the Triskell Project. This framework is used with the TGV tool to generated test cases. TGV (Test Generation with Verification technology) is a conformance test generator [21] [22]. Then, the transformation from UML Sequence diagram to LTS is done by UMLAUT tool and the test case generation by TGV tool [23].

In [24], test case generation by means of UML Sequence diagrams and Markov Chains has been presented. UML Sequence diagrams and Message Sequence Charts are transformed into Markov Chains. From the Markov Chains test cases are generated. Their work focuses on statistical usage testing.

3. Generation of Test Cases from UML Diagrams

In this section, a method is discussed on how test cases can be generated from Sequence diagram, Activity diagram and generating corresponding XMI files.

3.1 Sequence diagram

The input and output system events related to System Under Test (SUT), can be illustrated in UML Sequence diagrams [3].Before proceeding to a logical design of how a software application works, it is useful to define its behavior by a system Sequence diagram besides Use cases, and system contracts [3].

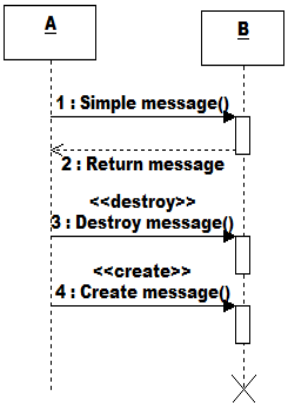

Figure 1: Sequence diagram example

A Sequence diagram consists of:

• **Object**: A Sequence diagram consists of sequences of interaction among different objects. It is a primary element involved in this diagram, being represented by rectangles labeled A and B (Figure 1).

• **Message**: The interactions between different objects in a Sequence diagram are represented by messages. A message is denoted by a directed arrow and the notation differs depending on the message type (Different message types are represented in the following manner. simple messages (with thick arrow), special messages to create (with create label) or destroy objects (with destroy label), and message response (dotted arrow)).

3.2 Generation of Test cases from Sequence diagram

The combination of simple message and return message is considered as one sequence. For example, the simple messages sent from object A to Object B constitute the 'test procedure'. The return message from Object B to Object A constitute the 'Expected result'. In this way each sequence is converted into one test case. The parser continuously scans till it finds 'return message' and extracts one sequence.

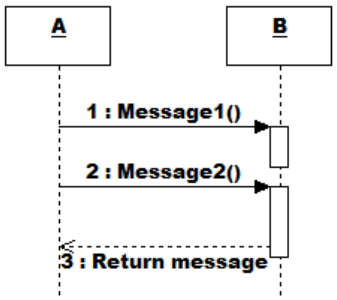

Figure 2: Reference Sequence diagram

For example, the Sequence diagram shown in Figure 2 can be converted into test case shown in Table 1.

Table1: Sample test case

Test Case ID	*Pre-Condition*	*Description*	*Expected Result*	*Actual result*	*Verdict*
TC1	Start Application	Message1 from ObjectA to ObjectB ->Message2 from ObjectA to ObjectB ->ReturnMessage from ObjectB to ObjectA	The following message is expected ReturnMessage from ObjectB to ObjectA		

Test case description captures all messages for a particular sequence. The expected result is the return message from the SUT.

StarUML [9] supports export of UML models into XMI file [14], which can be parsed in Java/Eclipse environment. After parsing XMI file, sequences, messages and return messages are extracted. Then the corresponding test cases are generated.

3.3Activity diagram

Activity Diagram is a special form of Statechart Diagram that offers rich notation to show a sequence of activities. It may be applied to any purpose (such as visualizing the steps of a computer algorithm), but is considered especially useful for visualizing business workflows and processes, or Use cases [3].

Some of the components of Activity diagram are initial state, final state, decision, transition, and action state (Figure 3).

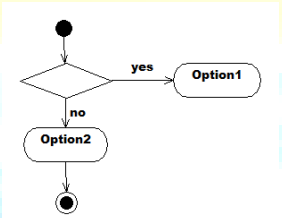

Figure 3: Activity diagram example

3.4Generation of Test Cases from Activity diagram

In this approach, decision and transitions play major role. Decision is expected to have variable used in SUT. Transitions are expected to have the corresponding values to these variables. For example, if 'customer type' is the variable in the system, then the corresponding values can be Employee and Non-Employee. This is illustrated in the Figure 4, which is a segment of the Activity diagram.

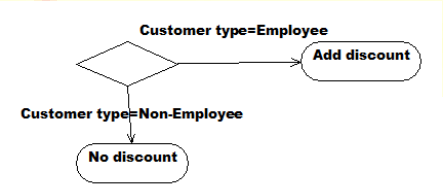

Figure 4: Segment of POS Activity diagram

This variable along with corresponding values constitutes a column in decision table as shown in Table2.

Table 2: Sample decision table

customer type	*Variable2*	*Variable3*
employee	Value1	Value1
non-employee	Value2	Value2

Activity diagram is exported into corresponding XMI file in StarUML environment. Parsing technique is used to parse XMI file and then extract variables and their corresponding values. A decision table with these variables and values can be created automatically. After decision table is created, test cases can be generated using Pairwise test tool called 'Allpairs' tool [25].

Expected result is extracted from Active state by traversing through transitions. For example, if 'customer type' is 'Employee', the corresponding Expected result is 'Add discount' as shown in Figure 4.

4. Illustration of the method

In this section, the proposed method is discussed with an example on testing Purchase Online System (POS) which is the SUT in this case [3]. Both Manual and Automated methods of generation of test cases from Sequence and Activity diagrams are discussed. Manual method is discussed first to illustrate complexity involved in creating test cases and helps to understand logic behind the creation of test cases. Automated method shows how same test cases can be easily generated with the click of a single button.

4.1 Manual Generation of Test Cases

Steps involved in creating test cases from Activity diagram manually are explained from section 4.1.1 to 4.1.4.

Similarly creating test cases manually from Sequence diagram is illustrated in section 4.1.5.

The objective of proposed method is to remove this manual effort in creating test cases.

4.1.1 Identification of Use case scenarios

The Use case diagram as shown in Figure 5 belongs to POS. This has actors such as cashier who takes money from customer and initiates new sale, and payment authorization service which authorizes the online payment.

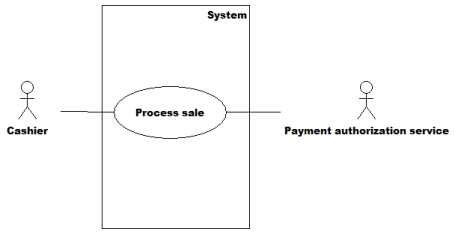

Figure 5: Usecase diagram for POS

The following Use case scenarios are identified:

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

4.1.2 Identification of Flows

From the above scenarios, basic and alternative flows can be identified manually.

Basic flows

B1. Customer identifies goods for purchase

B2. Cashier starts new sale

B3. Cashier enters goods id(s) in system

System returns description, price and running total with tax

B4. Customer pays (By cash)

System accepts payment

B5. System sends sale information to external accounting system and inventory system

B6. System presents receipt (Normal receipt)

B7. Customer leaves with goods and receipt

Alternative flows

A1. Customer can be Employee (eligible for discount) and Non-Employee

A2. System error, restart and recover (options: 1. Restart application 2. Restart PC)

A3. Invalid/Multiple ids (Options: 1. Invalid id 2. Add multiple ids 3. Remove id 4. Remove multiple ids 5. Cancel sale)

A4. Different payment modes (Options: 1. Credit card 2. Debit card 3. Cheque)

A6. Gift receipt (No price is displayed)

The Activity diagram as shown in Figure 6 represents basic flow along with alternative flow for POS:

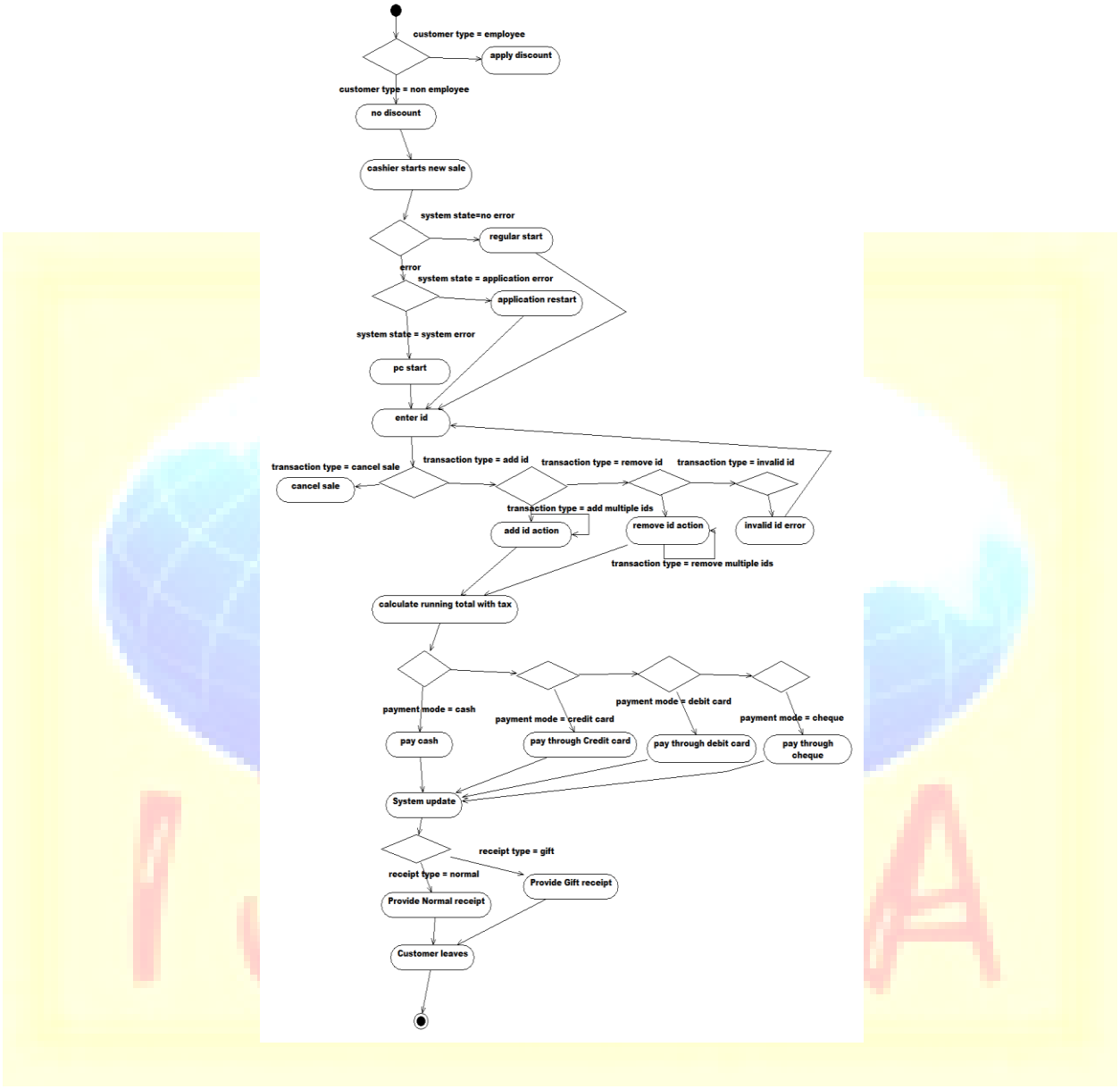

Figure 6: Activity diagram for POS

4.1.3 Identification of variables and values

Table 3lists all basic flows along with corresponding variables and options for each variable.

Table 3: Basic flow with variables and values

Basic steps	*Variable*	*Options to be tested*					
B1	Customer type	Employee	Non-Employee				
B2	New sale	No error	Application error	System error			
B3	Sale type	Add id	Invalid id	Add multiple ids	Remove id	Remove multiple id	Cancel sale
B4	Payment mode	Cash	Credit card	Debit card	Cheque		
B5	System update						
B6	Receipt type	Normal receipt	Gift receipt				
B7	Customer left						

4.1.4 Creation of test cases

The manual step to create the first test case, involves picking and connecting any options. To create the second test case, one of the other options that were not used in the first one is picked. This process of adding test cases is continued until all nodes of the graph are covered, which is illustrated in Figure 7.

Allocation of test cases can also be represented in the form of a test case allocation matrix, as shown in the Table 4.

Table 4: Test case allocation matrix

Basic steps	*Variable*	*TC1*	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*
B1	Customer type	Non-Emp	Emp	Non-Emp	Emp	Non-Employee	Non-Emp
B2	New sale	No error	App error	System error	No error	System error	App error
B3	Sale type	Invalid id	Add id	Add mult ids	Remove id	Remove mult ids	Cancel sale
B4	Payment mode	Cash	Credit card	Debit card	Cheque	Debit card	NA
B5	System update	System update	System update	System update	System update	System update	NA
B6	Receipt type	Normal receipt	Gift receipt	Normal receipt	Gift receipt	Gift receipt	NA

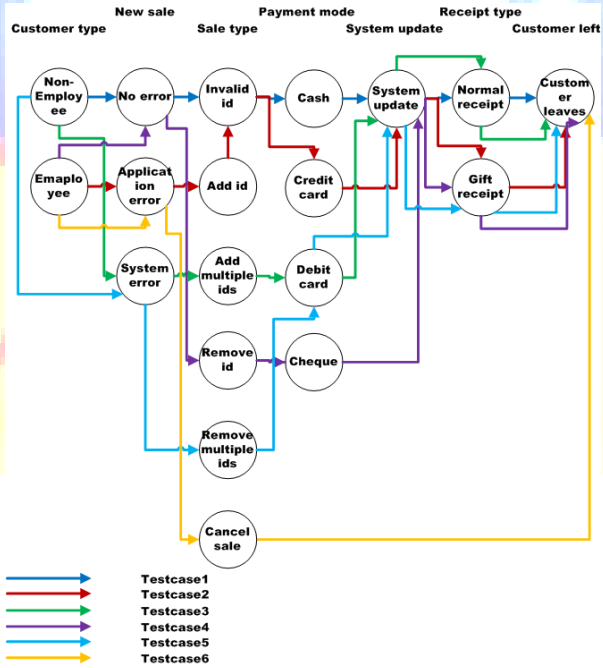

Figure 7: Process of adding test cases one by one

4.1.5 Creation of Test cases from Sequence diagram

The Sequence Diagram of POS as shown in Figure 8 illustrates the external actors that interact directly with the system, the system (as a black box), and the system events that the actors generate. Time proceeds downward and the ordering of events should follow their order in the Use case.

The following sequences can be identified from Sequence diagram

1. makeNewsale()->enterItem()->description, total
2. endsale()->total with taxes
3. makePayment->change due,receipt

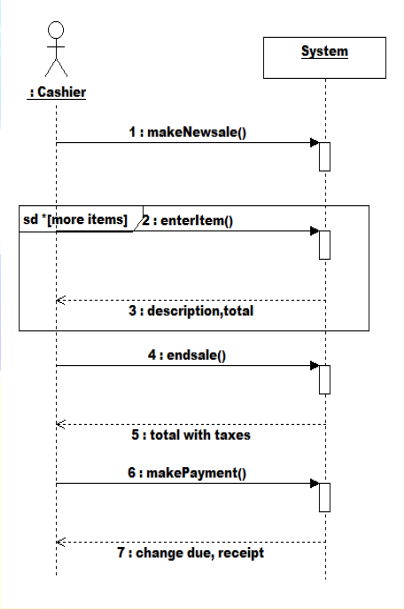

Figure 8: Sequence diagram for POS

A sequence starts with a forward message and sequence ends when a return message is found.To test these sequences the following test cases are identified as shown in Table 5:

Table 5: Created test cases from Sequence diagram

The second test case can be executed only if first test case is successfully executed. Similarly the thirdtest case can be executed after successful execution of first and second.

Test case id	*Pre-condition*	*Description (Sequence)*	*Expected result*
TC1	Open application	makeNewsale()->enterItem()->description, total	System needs to return item description along with cost
TC2	Items are entered	endsale()->total with taxes	Items are finalized and system should return total cost
TC3	Sale end	makePayment->change due,receipt	Payment should be made and receipt is received from system

4.2 Automatic Generation of Test Cases

The detailed steps for generating test cases from Activity diagram are given in section 4.2.1 and 4.2.2.

Similarly steps to generate test cases from Sequence diagram are given in section 4.2.3 and 4.2.4.

4.2.1 Process Flow Diagram from Activity diagram

4.2.2

Figure 9 illustrates the steps involved in generating test cases automatically from Activity diagram.

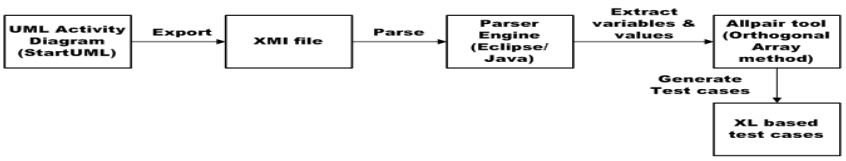

Figure 9: Process flow diagram from Activity diagram

4.2.3 Generation of Test cases from Activity diagram

The following two process steps are involved to generate test cases automatically from Activity diagram. First process step is to generate decision table by parsing XMI file, which is exported from corresponding Activity diagram in StarUML tool environment. Second, use this decision table to generate optimized test cases using Allpairs tool. These process steps are elaborated as given below.

Step1: Generation of decision table from Activity diagram

The following steps are required to generate the decision table from Activity diagram:

1. Draw Activity diagram using StarUML tool. Export Activity diagram from StarUML to XMI file.
2. Parse XMI file using Java code in Eclipse environment.
3. Identify variables from branches, corresponding values of the variables from transitions automatically.
4. Generate the following decision table, which will be used to generate test cases using All pairs tool.

Table 6 shows decision table which is automatically generated from Activity diagram, by parsing corresponding XMI file.

Table 6: Decision table generated from Activity diagram

System state	*receipt type*	*payment mode*	*customer type*	*transaction type*
no error	normal	cash	employee	add id
app error	gift	credit card	non-employee	remove id
system error		debit card		invalid id

		cheque		add multiple ids
				remove mult ids
				cancel sale

Step2: Usage of Pairwise test tool for generating test cases

Test cases can be generated using Pairwise test tool, which takes decision table as input. Allpairs [25] tool is one of the Pairwise test tools, which is used in this proposed method.

Table7 shows generated test cases by 'Allpairs' tool. Some of the test cases are invalid. For example, in case of 'Cancel sale' option, different payment modes are invalid, as payment is not required. Such cases can be excluded.

Table7: Generated test cases from Activity diagram

TC ID	*customer type*	*system state*	*transaction type*	*payment mode*	*receipt type*
1	employee	no error	add id	cash	normal
2	non employee	application error	add id	credit card	gift
3	non employee	application error	remove id	cash	normal
4	employee	no error	remove id	credit card	gift
5	employee	system error	invalid id	debit card	normal
6	non employee	no error	invalid id	cheque	gift
7	employee	application error	add multiple ids	debit card	gift
8	non employee	system error	add multiple ids	cheque	normal
9	employee	system error	remove multiple ids	cash	gift

10	non employee	no error	remove multiple ids	credit card	normal
11	non employee	no error	cancel sale	debit card	normal
12	employee	application error	cancel sale	cheque	gift
13	~employee	system error	add id	credit card	~normal
14	~non employee	system error	remove id	debit card	~gift
15	~non employee	application error	invalid id	cash	~normal
16	~employee	no error	add multiple ids	cash	~gift
17	~employee	application error	remove multiple ids	cheque	~normal
18	~non employee	system error	cancel sale	cash	~gift
19	~non employee	~no error	add id	debit card	~gift
20	~employee	~no error	remove id	cheque	~normal
21	~employee	~application error	invalid id	credit card	~gift
22	~non employee	~system error	add multiple ids	credit card	~normal
23	~non employee	~application error	remove multiple ids	debit card	~gift
24	~employee	~system error	cancel sale	credit card	~normal
25	~non employee	~system error	add id	cheque	~gift

Step3: Extraction of Expected result from Active state

The following steps are required to extract Expected result from Activity diagram:

1. Mapping table between transitions and corresponding Action state index is created. For example, for the transition ‘customer type = employee’, the corresponding Action state index is extracted as ‘UMLActionState.9’, during first parse.

2. Mapping table is updated with Action state name, for every Action state index as shown Table 8. For example, by using Action state index 'UMLActionState.9', the corresponding Action state name is extracted as 'apply discount', during second parse.

Table 8: Mapping table to extract Expected result

Transition	*Action state index*	*Action state name*
customer type = employee	UMLActionState.9	apply discount

3. Test case Table 7 is updated with Expected result by referring to mapping Table 8. Updated Test case table is shown in Table 9 for one test case.

Table 9: Updated Test case table with Expected result

TC ID	*customer type*	*system state*	*transaction type*	*payment mode*	*receipt type*	*Expected result*
1	employee	no error	add id	cash	normal	1.apply discount 2.regular start 3.add id action 4.pay cash 5.Provide Normal receipt

4.2.3 Process Flow Diagram from Sequence diagram

Figure 10 illustrates the steps involved in generating test cases from Sequence diagram.

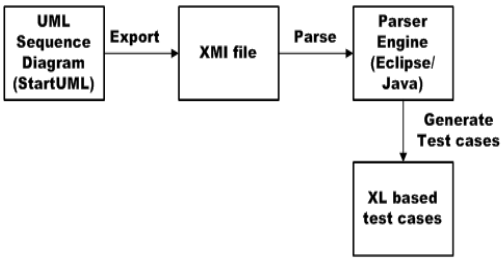

Figure 10: Process flow diagram from Sequence diagram

4.2.4 Generation of Test cases from Sequence diagram

The following steps are involved to generate test cases from Sequence diagram automatically.

1. Draw Sequence diagram using StarUML tool.
2. Export Sequence diagram from StarUML to XMI file.
3. Parse XMI file using XMIparser.java file in Eclipse environment.
4. Identify sequences automatically.
5. Generate XL based test cases using these sequences.

Table 10 shows generated test cases from Sequence diagram.

Table 10. Generated test cases from Sequence diagram

5. ‘’’Experimental results

Test case id	*Pre- condition*	*Description*	*Expected result*	*Actual result*	*Verdict*
TC1	Start Application	makeNewsale -->endItem -->description,total	The following message is expected description,total		
TC2	Successful completion of TC1	endSale -->total with taxes	The following message is expected total with taxes		
TC3	Successful completion of TC2	makePayment -->change due, receipt	The following message is expected change due, receipt		

The proposed approach is deployed in few applications and results are obtained. The following applications are considered for experimentation:

a. Account system
b. Borrow book
c. Currency controller
d. Ice vending machine
e. Safe home system
f. Simple ATM (SATM)
g. Triangle program
h. Wiper controller

5.1 Description

The brief descriptions of all applications are given in the following section:

5.1.1 Account system

An Account system helps user to open 'new' account. Once account is created user can do various transactions such as balance checking, debit money, credit etc. If the balance is maintained less than 0, then the status is changed to 'overdrawn'. If the account is not accessed for more than 5 years, then the status is changed to 'locked'. Also Account system allows user to close the account.

5.1.2 Borrow book

The Borrow book application allows user to search book in the database. If book is found in the database, user can reserve the book in his name. The Borrow book application has login feature and checks authentication of the user.

5.1.3 Currency converter

The Currency converter application allows user to convert currency from USD or Indian rupees to equivalent other country currencies. It allows user to enter input value and select target country. It throws error, if either input value not entered or target country is not selected.

5.1.4 Ice cream vending machine

The Ice cream vending machine allows user to purchase ice creams automatically. It allows user to select different flavor of ice creams such as Vanilla, Chocolate, Strawberry and Butterscotch etc. It calculates money based on selected flavor and number of ice creams ordered. When user inserts money into the slot, it calculates balance amount and returns back.

5.1.5 Safe home system

The Safe home system is a security system that helps user to monitor home. It alerts home owner in case of any intruder entering home, through various mechanisms such as sending SMS, making emergency call, activating alarm, video recording and blinking control panel.

5.1.6 Simple ATM system

The Simple ATM system provides banking transactions such as withdraw money, deposit money, balance checking, print mini statement etc. User requires a valid debit card and need to enter valid PIN number to avail banking services.

5.1.7 Triangle program

The Triangle program displays triangle type such as Isosceles, Scalene, Equilateral based on the values of the sides a, b, c. It displays an error message, in case of invalid entry.

5.1.8 Wiper controller

The Wiper controller allows user to set different wiper speed by changing position of lever and dial. The lever position can be changed to off, inter, low and high. When lever position is set to inter, dial positions can be changed to 1, 2 and 3.

5.2 Summary

Table 11 shows summary of Sequence and Activity diagram test cases. Column B shows MC/DC test cases possible to derive from Activity diagram. Column C shows optimized number of test cases using Allpairs tool.

Table 11.Summary of Sequence and Activity diagrams test cases

		Activity diagram Test cases		
Project name	*Sequence diagram Test cases (A)*	*Before optimization (B)*	*After optimization (C)*	*Total Test cases (A+C)*
Account system	9	96	13	22
Borrow book	6	4	4	10
Currency converter	14	567	23	37
Ice vending machine	5	72	18	23
Safe home	8	480	16	24
Simple ATM	7	240	16	23
Triangle program	5	128	8	13
Wiper controller	6	12	12	18

5.3 Mutation analysis

The effectiveness of generated test cases for Triangle program is checked using fault injection technique called mutation analysis [26][27]. Mutants are created from Triangle program after injecting errors in program to make program faulty. If test case set is capable of capturing these errors, then mutants are killed by tests. The mutation analysis report after creating mutants for Triangle program is shown in Table 12.

Totally 12 test cases failed because of 5 mutants. Mutant 1 has maximum failures of 4 out of 13 test cases executed. This shows even though test cases are generated from design, any defect injected by developer is easily captured by these cases.

Table 12.Mutation analysis for Triangle program test cases

Mutation number	*Change in code*		*Number of test cases failed in Mutants*		*Total Fails per Mutan*
	Correct	*Buggy*	*Sequence diagram*	*Activity diagram*	

					t
1	!(a>0 && b>0 && c>0)	!(a>0 || b>0 || c>0)	1	3	4
2	(a<b+c)&&(b<c+a)&& (c<a+b)	(a<b+c)||(b<c+a)|| (c<a+b)	1	1	2
3	a==b && b==c && c==a	a==b || b==c || c==a	1	1	2
4	(a==b && b<>c)||(a==c && a<>b)|| (b==c && a<>c)	(a==b && b<>c) &&(a==c && a<>b) &&(b==c && a<>c)	1	1	2
5	a<>b && b<>c && c<>a	a==b && b<>c && c<>a	1	1	2
Total fails category wise			5	7	12

6. Comparison with Other Methods

There are others who have considered a Sequence diagram as input for test case generation. An example can be seen in [17], where the Seditec tool generates automatically test stubs for the classes and methods whose behavior is specified in the Sequence diagrams. This approach is different from the proposed method because the method in [17] generates test stubs, while the proposed method generates test cases for sequences in the case of Sequence diagram and builds a decision table in the case of Activity diagram. From this decision table, optimized number of test cases is generated using 'Allpairs' test tool.

In [17], the test pattern "Round-trip Scenario Test Pattern" is presented. This pattern uses UML Sequence Diagrams as input, but it does not use combination of messages and return-messages for test case generation. It uses flow graphs. In the proposed method, combination of messages and return-messages is considered as one sequence. This method ensures 100% sequence coverage, while generating test cases.

There are several research projects [18][19] proposing concepts for UML based test tools. However, most of them rely on state diagrams and/or demand extensive additional effort from the designers by introducing new formalisms into the UML, which in our experience significantly lowers the chance of those concepts being accepted in industry projects. The proposed method addresses generating Multiple Conditions/Decisions Coverage (MC/DC) test cases from Activity diagram, which ensures 100% transitions and active states coverage. The usage of Allpairs tool ensures reduces number of test cases generated. For example, the test cases generated as shown in Table 7 are reduced to 25 cases, even though there 288 combinations are possible for the decision table shown in Table 6.

7. Conclusions

In this paper, a method has been proposed to generate functional test cases from UML Sequence diagram and MC/DC test cases from Activity diagram. As these diagrams are developed early in the development cycle, early generation of test cases is possible by the proposed method. Besides, the proposed method is more prone to automation and reduces effort for writing exhaustive test cases.

The case study discussed here has shown that the proposed method can be used for applications in both PC based and in embedded environments. As Sequence diagram represents sequences of combined atomic services, generating test cases for testing these atomic services is very useful. Similarly, as Activity diagram represent dynamic behavior of the system, and generating MC/DC test cases helps to test system behavior under various input conditions.

REFERENCES

[1] P. Jorgensen, Software Testing: A Craftsman's Approach. CRC Press,2002.

[2] B. Beizer, Software Testing Techniques, 2nd ed. Van NostrandReinhold, 1990.

[3] Craig Larman, "Applying UML and patterns ", Addison Wesley, 2000.

[4] Emanuela G. Cartaxo, Francisco G. O. Neto and Patr´ıcia D. L. Machado, "Test Case Generation by means ofUML Sequence Diagrams and Labeled Transition Systems", IEEE 2007.

[5] A.V.K. Shanthi, Dr.G.Mohan Kumar,"Automated Test Cases Generation from UML Sequence Diagram", International Conference on Software and Computer Applications 2007.

[6] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong ," Test Case automate Generation from UML Sequence diagramand OCL Expression", International Conference on Computational Intelligence and Security 2007, pp 1048-52.

[7] Monalisa Sarma, Debasish Kundu, Rajib Mall,"Automatic Test Case Generation from UML Sequence Diagrams", 15th International Conference on Advanced Computing and Communications 2007.

[8] Qaisar A. Malik, Dragos¸ Trus¸can, Johan Lilius,"Using UML Models and Formal Verification in Model-Based Testing", 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems 2010.

[9] StarUML Tool. http://staruml.sourceforge.net/en/, Jul. 2011.

[10] Padma Iyenghar1, Elke Pulvermueller1, Clemens Westerkamp,"Towards Model-Based Test Automation for Embedded SystemsUsing UML and UTP", IEEE ETFA 2011.

[11] Object Constraint Language 2.0 is available from Object Mangement Group's web site http://www.omg.org/

[12] Vinaya Sawant, Dragos¸ Ketan Shah,"Automatic Generation of Test Cases from UML Models", International Conference on Technology Systems and Management 2011.

[13] G.J. Myers, C.sandler, T.Badgett, and T.M.Thomas. "The art of softwareTesting" , 2nd Edition.Wiley,2004

[14] OMG, "XML Metadata Interchange (XMI),v2.1",2004.

[15] Santosh Kumar Swain, Durga Prasad Mohapatra, Rajib Mall,"Test Case Generation Based on Use caseand Sequence Diagram", Internatonal Journal of Software Engineering, Vol.3 No.2 2010.

[16] I. K. El-Far and J. A. Whittaker, "Model-based software testing,"Encyclopedia on Software Engineering, 2001.

[17] F. Fraikin and T. Leonhardt, "Seditec—testing basedon sequence diagrams," 2002. [Online]. Available: citeseer.ist.psu.edu/fraikin02seditec.html

[18] Jean Hartmann, Claudio Imoberdorf, Michael Meisinger,"UML-Based Integration Testing", Proceedings of theInternational Symposium on Software Testing and Analysis,Portland, Oregon, 2000, pp. 60-70

[19] J. Offutt and A. Abdurazik, "Generating Tests from UMLSpecifications", Second International Conference on theUnified Modeling Language, Springer, New York 1999, pp.416-429

[20] W. M. Ho, J.-M. Jquel, A. L. Guennec, and F. Pennaneac'h,"UMLAUT: An extendible UML transformation framework," inAutomated Software Engineering, 1999, pp. 275–278. [Online].Available: citeseer.ist.psu.edu/ho99umlaut.html

[21] T. J´eron and P. Morel, "Test generation derived from model-checking,"in CAV '99: Proceedings of the 11th International Conference onComputer Aided Verification. London, UK: Springer-Verlag, 1999,pp. 108–121.

[22] C. Jard and T. Jéron, "Tgv: theory, principles and algorithms:A tool for the automatic synthesis of conformance test cases for nondeterministic creactive systems," Int. J. Softw. Tools Technol. Transf.,vol. 7, no. 4, pp. 297–315, 2005.

[23] L. Bousquet, H. Martin, and J. Jzquel, "Conformance testing from umlspecifications." [Online]. Available: citeseer.ist.psu.edu/683853.html

[24] F. Z. M. Beyer, W. Dulz, "Automated ttcn-3 test case generation bymeans of uml sequence diagrams and markov chains," in Asian TestSymposium, 2003, pp. 102–105.

[25] ALL PAIRS Tool. http://www.satisfice.com/tools.shtml.

[26] S. Kansomkeat and W. Rivepiboon, "Automated-generating test case using UML statechart diagrams", Proc. SAICSIT 2003, ACM 2003 pp. 296 – 300, 2003.

[27] Demillo, Lipton and Sayward, "Hints on Test Data Selection:Help for the Practicing Programmer", IEEE, 1978.