Analyzing Impact of Change in UML Sequence Diagrams on State Machine based Regression Testing

Conference Paper · February 2010

CITATIONS
0

READS
2,083

3 authors, including:

Nosheen Sabahat
Forman Christian College
33 PUBLICATIONS **146** CITATIONS

SEE PROFILE

Proceedings of the IASTED International Conference
Software Engineering (SE 2010)
February 16 - 18, 2010 Innsbruck, Austria

ANALYZING IMPACT OF CHANGE IN UML SEQUENCE DIAGRAMS ON STATE MACHINE BASED REGRESSION TESTING

Nosheen Sabahat
College of Electrical and Mechanical Engineering, NUST
Rawalpindi, Pakistan
nosheen@ce.ceme.edu.pk

Qurat-ul-Ann Farooq
Technische Universtät Ilmenau
Ilmenau, Germany
qurat-ul-ann.farooq@tu-ilmenau.de

Zafar I Malik
Academy of Educational Planning and Management
Islamabad, Pakistan
zafarimalik@acm.org

ABSTRACT
Regression testing is an important activity to test the validity of the evolving software systems. UML design models can be used for early change identification and regression test selection. Whenever a change occurs in one model element in a diagram, it may cause changes in other UML diagrams due to dependencies among model elements. In such a case, change identification and testing of the corresponding parts of the system also becomes necessary to prevent the system from the affect of these changes. In this paper, we extended our existing state-based regression testing approach by analyzing impact of change in sequence diagram on state-based regression testing. We determined the model elements of class diagram and state machine, impacted by sequence driven changes and analyzed the effect of these changes on the test suite. The approach supports UML 2.1 meta-model. To prove the applicability and usefulness of this work, we applied it on a comprehensive case study. Our results prove that a significant number of test cases should be retested due to changes in sequence diagrams to make the regression test suite more accurate and reliable.

KEY WORDS
UML, Regression Testing, Model Based Testing

1. Introduction

Regression testing is the activity of rerunning the test cases from existing test suites to build confidence that software changes have no unintended side-effects [1, 12]. Exercising the complete existing test suite to test the modified system is very expensive. A more feasible option is to select a subset of the existing test suite that corresponds to the modification. This strategy is known as selective regression testing. For selective regression testing change identification and change impact analysis is required. Impact analysis is defined as the process of identifying the potential consequences of a change, and estimating what needs to be modified to accomplish a change [2, 12, 14].

UML based regression testing [1] provides a way for early change identification, impact analysis and regression test selection. This results in saving the project cost because the test preparation can be started in early phases of software development. For UML based regression testing UML analysis and design models are used to detect the changes in the software system. A change in one UML diagram can have impact on other UML diagrams as well due to dependencies between models [1, 2, 12, 14]. These dependencies should be considered during change identification and impact analysis for effective regression testing.

In this paper, we reported an extension to our previous UML based approach for selective state machine based regression testing [1]. Our previous regression testing methodology considers the changes in the class diagrams and the impact of these changes on the corresponding state machines. We are extending our existing state based regression testing approach by analyzing impact of change in the sequence diagrams on state machine based regression testing. Our work is based on the assumption that operations defined in the class diagrams are realised using the sequence diagrams. Thus, a change in a sequence diagram will change its corresponding operation in the class diagram. We also assume that there exist a behavioural state machine for defining the behaviour of a class and the attributes and operations defined in the class diagrams are used by its corresponding state machine.
The class and can in turn affect state machines.

For change identification, we compare two versions of a sequence diagram and check the effect of changes in sequence diagram on class diagrams and the corresponding state machines. After that we classify the test cases considering the change to generate the regression test suite. Our approach supports UML 2.1 meta model.

We summarize our contributions in extending the approach in the following points.

- We are extending our previous state based regression testing approach and provide a set of change definitions for sequence diagrams (sequence-driven changes) for UML 2.1 [15] by analyzing elements of UML sequence diagrams.

- These changes are then identified by comparing baseline and delta versions of the sequence diagram.

- We have enhanced the definition of class-driven changes provided in our previous approach by extending the definition of modified operation. In our case, an operation is considered changed if its corresponding sequence diagram is changed; which was not defined by the previous approach.
- We developed a comprehensive case study of library management system to prove the scalability of our approach. Our case study includes 11 classes, 3 state machines and 5 sequence diagrams.

The rest of the paper is structured as follows: Section 2 discusses the related work. Section 3 describes our approach for change analysis and regression test selection. Section 4 presents the change definitions. Section 5 elaborates the test suite classification and section 6 presents the case study Section 7 describes results and discussions. Section 8 and section 9 present conclusions and acknowledgements respectively.

2. Related Work

In this section, we will discuss the related work on UML based and state based regression testing. Recently, we reported our proposed model based methodology for state based regression testing [1]. Our proposed regression testing technique was based on state machine and class diagrams for change identification [1]. We identified the changes in UML 2.1 class diagrams and analyzed its effect on state machines and the corresponding test suites. We classified our baseline test suite into obsolete, reusable and re-testable test cases for regression testing. The changes in sequence diagrams were not catered in the previous work. We extended our work to incorporate sequence driven changes as reported in this paper.

A limited research is also available in literature in which state machine like models are used for regression testing. These techniques are not UML based but, since our work employs UML state machines which are extensions of state charts, therefore, we are also including them.

The technique presented by Beydeda and Gruhhn [16] is for class level regression testing based on specification as well as implementation information. A class specification implementation graph (CSIG) is used as a model. Regression test cases are selected by comparing the base line and delta version of CSIG.

Korel et.al [13] used the EFSM (Extended Finite State Machines) as the model and their test case reduction technique is based on dependence analysis of this model. The focus of their work is on regression test suite reduction by performing the dependence analysis of the EFSM. The authors have constructed a dependence graph considering the data and control dependencies of the transitions. The interaction pattern of each test case is calculated based on these modifications and test suite is reduced based on these interaction patterns. The above discussed state based regression testing approaches lacks to cater the relationship of state machines with other system models as we did in our work.

Briand et.al proposed a UML model-based approach to impact analysis and changes in UML designs and has discussed sequence driven changes in their work [2]. They have presented the impact analysis rules and classification of change definitions for UML class, state and sequence diagrams for UML 1.4; but the changes are only discussed for decision-making and change planning process. We have provided the changed definitions for our work according to UML 2.1 in context of regression testing of UML.

Gorthi et al. [18] presented an approach similar to Chen et al. [9] for regression testing. They presented the requirements using activity diagrams corresponding to use cases annotated with information regarding priority of each activity. They also made test case prioritization by considering higher priority modified activity nodes.

Mansour et al. [20] presented a regression testing approach based on class diagrams and interaction overview diagrams (IOD). The IOD depicts the systems requirements. Test case selection is performed considering the changes in both artefacts. Naslavsky et al. [19] presented an idea for regression testing using class diagrams and sequence diagrams for model driven architecture (MDA). They make use of model transformations for regression testing.
There are also some other UML based regression testing techniques in the literature. We already discussed these techniques in our previous work [1]; hence, we are not discussing these techniques here to avoid redundancy; as this approach provides an extension to our previous work.

3. Proposed Approach

In the UML based system development, various design models are developed to model different views of the system. These UML models are inter-dependent with each other. For example, the class diagrams are related to state machines as state machines define the behaviour of a class. Whenever a change occurs in one model, it can cause changes in other models due to these interdependencies. It is important to deal with these interdependencies at the time of change identification and impact analysis during regression testing.
Our regression testing approach is based on the assumption that changes in sequence diagrams can change the corresponding class diagrams, thus affecting the behavioural state machines defined for the classes in a class diagram. We already discussed, in our previous work that [1] how changes in class diagrams can affect

the corresponding state machines and how important it is to test those changes. In this paper, we present the idea that changes in sequence diagrams can also effect the class diagram and their corresponding test suites.

The operations defined in the class diagrams are elaborated using the sequence diagrams. These operations can be used in the state machines as call actions and call events. In this paper, we provide the change definitions for sequence diagrams according to UML 2.1 specifications [15]. Our work defines the sequence-driven changes that can affect class diagram and state machines and enhances the previous definition of changes. Figure 1 describes the extended model of the state based regression testing approach [1] by incorporating sequence-driven changes in it. The grey elements in the diagram show the extensions we have provided in the previous state based regression testing approach.

According to Figure 1, the *sequenceDiagramComparator* takes two versions of sequence diagrams i.e. baseline sequence diagram and delta version of the sequence diagram for comparison.
The *sequenceDiagramComparator* produces the set of sequence-driven changes. To classify the sequence diagram changes, the properties of each model elements in the UML meta-model are considered so that the possible changes in the sequence diagram can be determined. The *classDiagramComparator* takes sequence-driven changes, baseline and delta version class diagram and generates *the set of classDrivenChanges*.

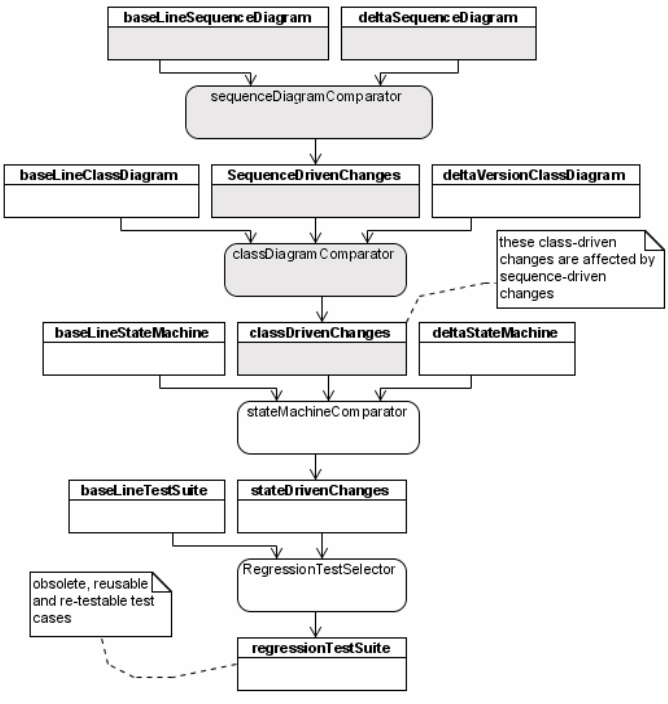

Figure 1: Extended Abstract Model of State Based Regression Testing Approach

The set of class driven changes also contains those elements of the state machines which are impacted by the changes in sequence diagram. The state machine comparator takes the base line and delta version of state machines for comparison and generates the set of state driven changes. It also uses the set of class driven changes to find those elements of the state machines which are impacted by the changes in the class diagram.

The state driven changes along with the baseline test suite are fed into Regression Test Selector to obtain test cases. In the next section we will discuss change definitions which are necessary to identify changes between two versions of UML models. We are providing the definitions for the sequence driven changes only. For the other change sets, i.e. class driven and state driven changes, the interested readers can refer to our previous work [1]. In the following section we will discuss the sequence-driven changes and some extended change definitions for class driven changes.

4. Change Definitions

In this section, we will discuss the sequence-driven changes and some extended class-driven changes necessary for change identification.

4.1 Sequence Driven Changes

The sequence-driven changes are those changes which are obtained by comparing the base line and delta version of the sequence diagrams. The sequence-driven changes will be used by the sequence diagram comparator for comparison.
A sequence diagram describes an interaction that is a sequence of messages passed between objects to accomplish a task [11]. Whenever any message of sequence diagram is modified, it affects operations of the class diagram and some parts of state machine may also be modified due to the impact of this change.
We refer to the baseline and delta version of the sequence diagram as SD and SD′ respectively. A sequence diagram will be changed if any of its objects, their lifeline or the messages among those objects gets changed [15].

Following we define these changes for our state based regression testing approach.

ModifiedMessage: We define the set of messages in a sequence diagram $S \in SD$ as MSG^C and the messages in the sequence diagram $D \in SD'$ as MSG^D. The set $MSG^D{}_M$ denotes the set of modified messages in D. A message msg $\square$ MSG^D will be considered modified in the following cases:
A message will be modified if the instance of class MessageKind or MessageSort is changed. All the modifications in the messages are shown in the form of a change model in figure 2.

ModifiedMessageKind: According to UML 2.1, an instance of Message class will be modified if kind of message is changed. The MessageKind can be complete, lost, found, or unknown. Default value of message kind is unknown. The instance of Message class will also be changed if the default value is changed or any other property associated with the Message instance is changed.

ModifiedMessageSort: The instance of Message class will also be changed if the sort of communication used to generate message is changed i.e. MessageSort. In this case the message can be of the form synchCall, asynchCall, asynchSignal, deleteMessage, reply; where the default value is synchCall. Also when the default value of MessageSort gets modified, it will modify the Message.

AddedMessage: A message is said to be added into modified version of a sequence diagram, if it does not exist in the baseline version. Messages are added when interaction between two objects takes place.

DeletedMessage: A message not present in the modified version of the sequence diagram and present in the baseline version is a deleted message of the sequence diagram. Messages are deleted when the interaction among two objects is deleted.

ModifiedInteraction: The set of interactions in a sequence diagram S□SD are defined as IN^C and the interactions in the sequence S□ SD′ as IN^D. The set IN^D M denotes the set of modified interactions in D.
If messages are added or deleted, then the corresponding interaction among connectable elements gets modified. Also the interaction among two objects changes if lifeline among the objects changes; i.e. if the lifeline is deleted.
An interaction in□ IN^D will be considered modified if message type is changed. Message type refers to the MessageKind or MessageSort where the MessageKind can be complete, lost, found, or unknown. The default value of MessageKind is unknown. If the MessageKind changes from unknown to lost, complete or found then the corresponding interaction gets changed.
Interaction can also be modified, if the MessageSort is changed i.e. if the default value of MessageSort synchCall is changed to asynchCall, asynchSignal, deleteMsg or reply.

DeletedInteraction: Considering the baseline version of sequence diagram, an Interaction will be deleted if the corresponding messages of that interaction are deleted or the lifeline of the objects being interacted is deleted.

AddedInteraction: An interaction is said to be added into a sequence diagram if it is not present in the baseline version of the sequence diagram. A new interaction will be created among two objects if any message is added from one object to the other.

ModifiedLifeline: The set of lifelines in a sequence diagram S□SD are defined as LF^C and the lifelines in the sequence S□ SD′ as LF^D. The set LF^D M denotes the set of modified interactions in D.
A lifeline in a sequence diagram will be modified if any message that is not present in the baseline version of sequence diagram is added into it. Similarly if any message is deleted, it modifies the lifeline.
A lifeline is also said to be modified if its name in the delta version of sequence diagram gets changed and is different from that of the baseline version.

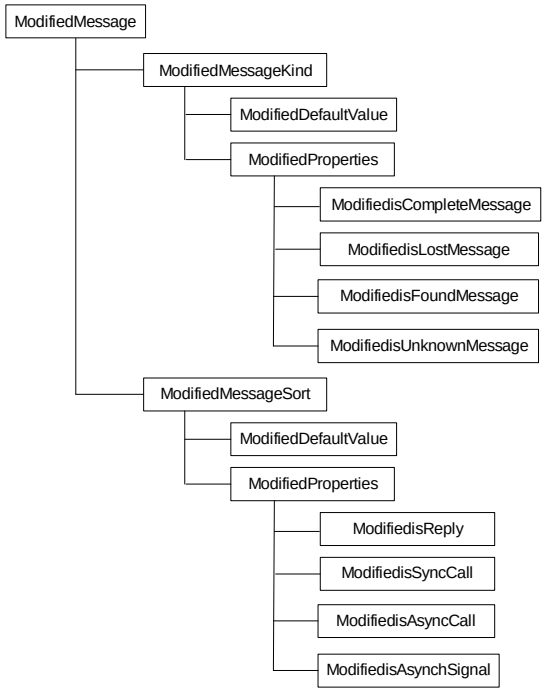

Figure 2: Change Model for ModifiedMessage

Added Lifeline: Lifeline will be added if messages are instantiated among elements, or there takes place some interaction among objects i.e. when instance of Interaction class is created, then a new lifeline is added.
A lifeline is said to be added into the delta version of the sequence diagram if it is not present in the baseline version. When a new lifeline is added into a sequence diagram messages are also added.

DeletedLifeline: Lifeline will be deleted if messages among two corresponding elements are deleted. A lifeline provides a means to identify the order of events that affect the condition and behavior of an object during the execution of an interaction. So if the lifeline is deleted, then the order of events and behaviour of objects get affected. Lifeline also gets deleted when the instance of interaction class is deleted.

ModifiedCombinedFragment: The instance of CombinedFragment will be changed if the default value of its attribute InteractionOperatorKind is changed i.e.

from seq to alt, opt, par, loop, critical, neg, assert, strict, ignore, consider.

4.2 Extension in the class-driven changes

An important definition which needs to be modified from the definitions of class-driven changes discussed in our previous approach [1] is the definition of "ModifiedOperation". Following is a part of the definition of ModifiedOpeartion previously defined [1]

"We define the set of operations in a class $C \in CD$ as OP^C and the operations in the class $D \in CD'$ as OP^D. The set $OP^D{}_M$ denotes the set of modified operations in D"

We are adding the following in the previous definition of ModifiedOperation to cater the effect of change of a sequence diagram on class diagram. Other parts of the definition are same.

"An operation $op \in OP^D$ if its corresponding sequence diagram is modified. "

5. State Based Regression Test Suite Classification

In this section, we will discuss our modified test suite classification using the above definitions for the sequence-driven changes and modified class-driven changes. According to our approach, whenever a message is changed in a sequence diagram, the corresponding operation in the class diagram is also considered changed. This changed method can affect the state actions and effects in turn [1].

Table 1: Changes in sequence diagrams of Library Management System

Changed → Element Sequence Diagram↓	Message	Lifeline	Combined Fragment
addBooks()	Modify 1	Delete 1	-
orderBooks()	Add 1	-	Delete 1
issueBooks()	Delete 1	Delete 1	Add 1
returnBooks()	Modify 1	-	Modify 1
payFine()	Modify 1	Delete 1	-

After considering both types of changes, the baseline test suite is classified into obsolete, reusable and re-testable test cases as discussed in our earlier work [1].

For the construction of our experimental baselines test suite, we are using the transition tree methodology [7]. A few test case from the baseline test suite of class LibraryMember are depicted in Table 2.

By considering sequence-driven changes, some additional testcases will also become re-testable. This aspect was not addressed by our previous approach. This is because we have enhanced the previous definition of ModifiedOperation; as discussed in section 4.2.

Table 2: An example of some of the test cases from baseline test suit for Library Member class

ID	Test Path
1	To,T1,T4,T5,T21,T6,T9,T10,T16
2	To,T1,T4,T5,T21,T6,T9,T11,T22
3	To,T1,T4,T5,T21,T7,T12,T17,T18
4	To,T1,T4,T5,T21,T7,T13,T22
5	To,T1,T4,T5,T21,T8,T14,T19,T20
6	To,T1,T4,T5,T21,T8,T15,T22
7	To,T1,T4,T6,T9,T10,T16,T21,T5
8	To,T1,T4,T6,T9,T10,T16,T21,T7,T12,T17,T18
9	To,T1,T4,T6,T9,T10,T16,T21,T7,T13,T22
10	To,T1,T4,T6,T9,T10,T16,T21,T8,T14,T19,T20
11	To,T1,T4,T6,T9,T10,T16,T21,T8,T15,T22
12	To,T1,T4,T6,T9,T11,T22,T21,T5
13	To,T1,T4,T6,T9,T11,T22,T21,T7,T12,T17,T18
14	To,T1,T4,T6,T9,T11,T22,T21,T7,T13,T22
15	To,T1,T4,T6,T9,T11,T22,T21,T8,T14,T19,T20

6. Case Study

To prove the feasibility of our approach, we performed a case study of a library management system. The baseline version of class diagram is shown in figure 3. We constructed three state machines for the Library, LibraryMember and Book class. The state machine of class LibraryMember is shown in figure 5. We are not showing the other state machines due to the space constraints However, the total number of states and transitions of these state machines are shown in table 3.

There are 5 sequence diagrams of our library system; from which two sequence diagrams are of class Librarian and the rest of the diagrams are of our class LibraryMember. Following types of changes were identified in the delta version of the sequence diagrams as described in table 1:

- 1 message is modified and one lifeline is deleted from the sequence diagram corresponding to the operation addBooks()
- 1 message is added and one combined fragment is deleted from the sequence diagram corresponding to the operation orderBooks()
- 1 message and 1 lifeline is deleted from the operation issueBooks() and one combined fragment is added.
- 1 message and 1 combined fragment are modified from the sequence diagram corresponding to the operation returnBooks ().
- 1 lifeline is deleted and 1 message is modified from the sequence diagram corresponding to the operation payFine ().

The baseline sequence diagram of operation payFine () is depicted in figure 4. The forth lifeline showing the FN object of class Fine is deleted in the delta version and

message number 7 is modified by passing the parameter fineAmount in the delta version. We compared the baseline and delta version of sequence diagrams and analyzed the effects in the class diagram and state machine for the changes discussed. As a result of modified operation, the transition T10, T17 and T19 are modified in the state machine of the LibraryMember class. Table 2 shows an excerpt of the test cases for baseline version of class Library Member. These test cases are obtained by using transition tree methodology [7]. The modified transitions T10, T17 and T19 of the state machine of the LibraryMember class, in the result of the modified payFine() sequence diagram, made the test cases 1.3,5, 7, 8, 9, 10, 11, 15 from table 2 re-testable.

7. Results and Discussions

In this section, we report the results of application of our regression testing methodology on the library management system case study.

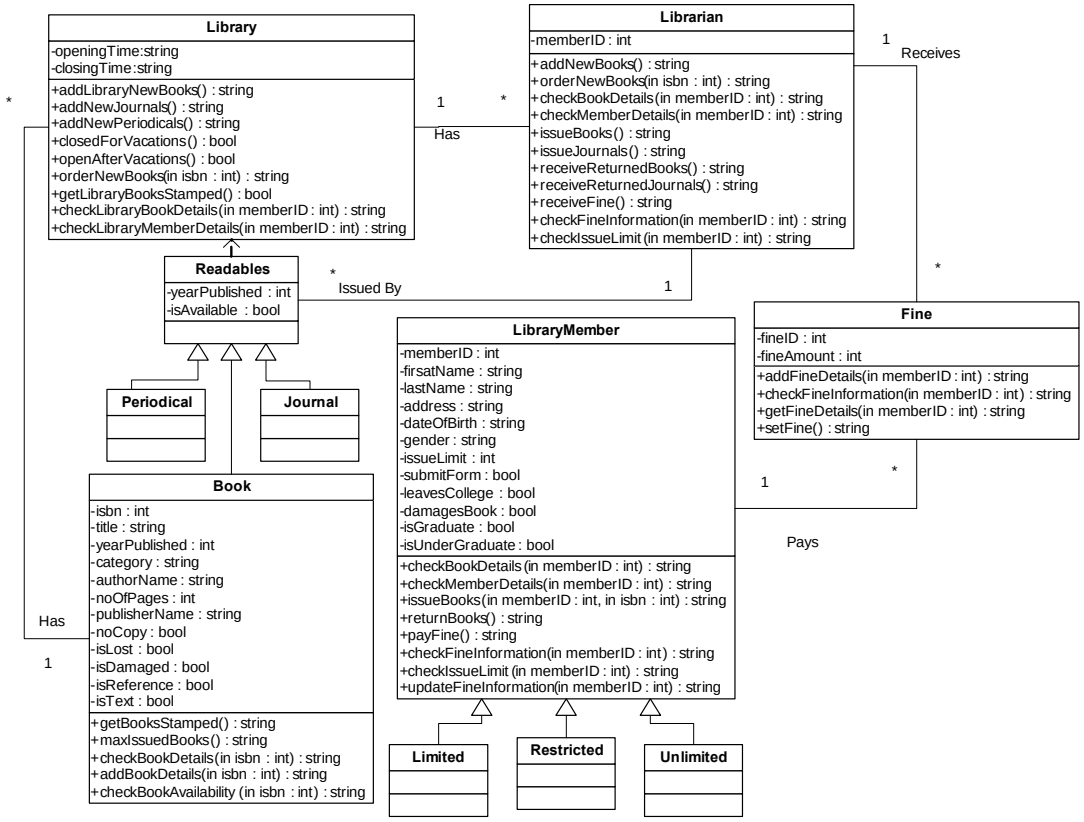

Figure 3: Class Diagram for Library Information System

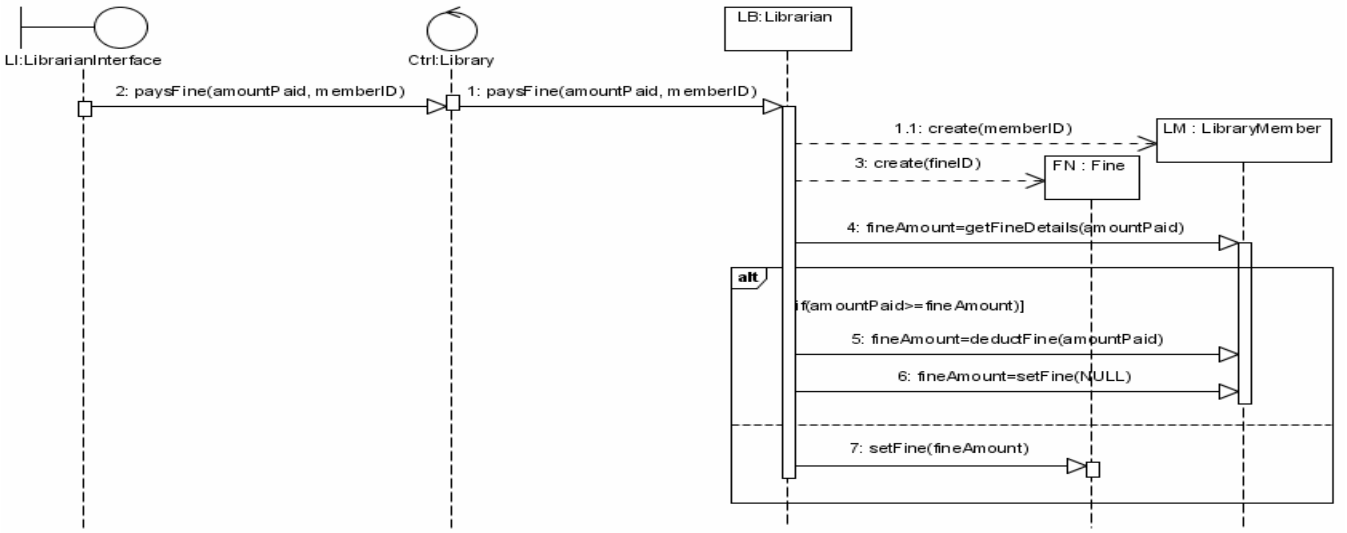

Figure 4: Baseline Sequence Diagram for Pay Fine Operation

Table 3 shows the results of our complete case study after applying our proposed methodology. There were total 72 test cases in the baseline test suite of LibraryMember class; of which 48 test cases are re-testable and 24 test cases are reusable. Similarly, there were 333 test cases in the baseline test suite of Library class of which 188 test cases are re-testable and 145 test cases are reusable. There were total 8 test cases in the Book class of which 7 are re-testable and 1 is reusable.

Table 3: Classification of test cases

State Machine	No of states	No of transitions	Total Baseline Test Cases	Re-testable Test Cases	Reusable Test Cases
Library	10	20	333	188	145
Library Member	15	21	72	48	24
Book	9	13	8	7	1

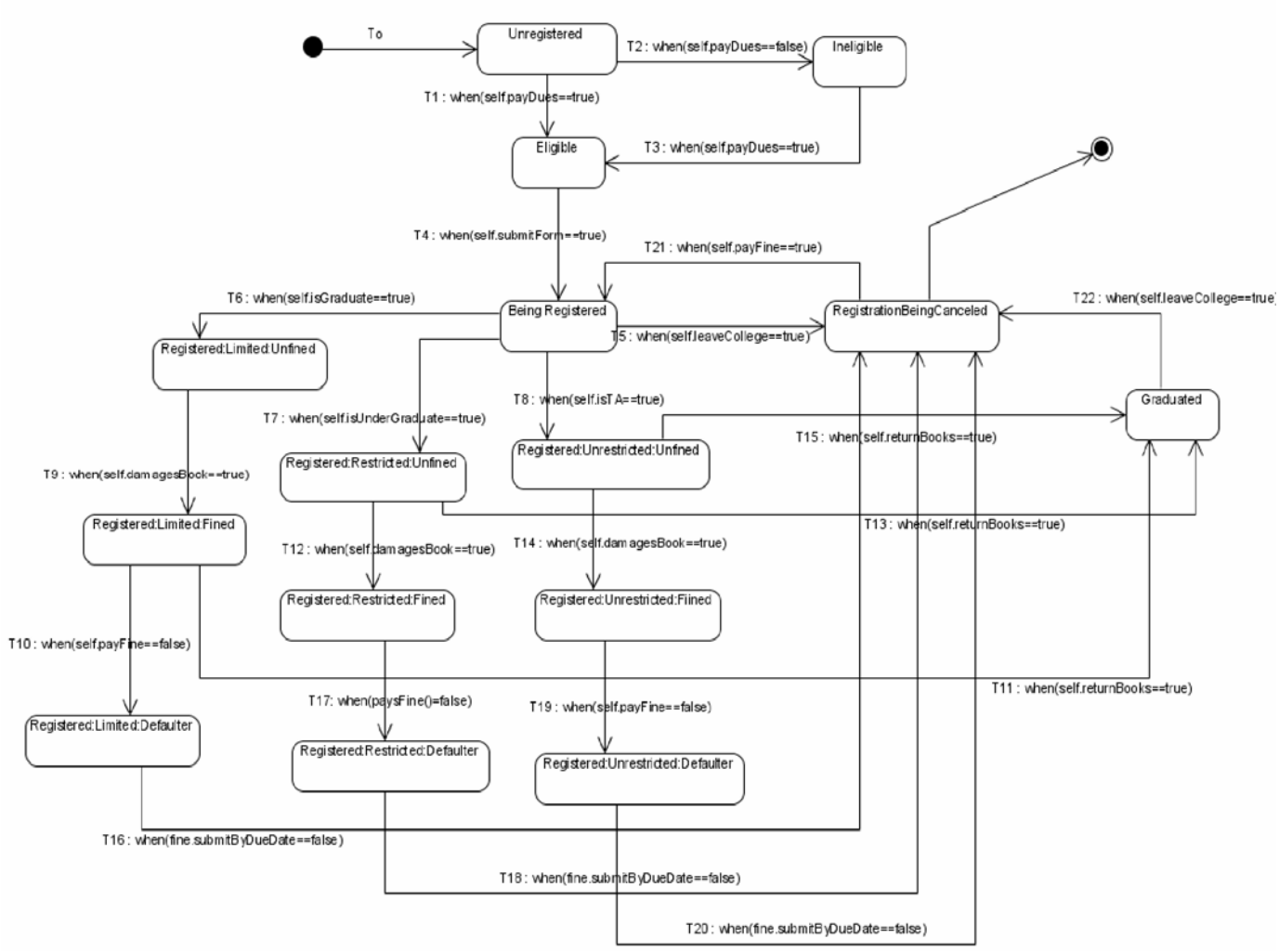

Figure 5: State Machine of LibraryMember Class

In our case we do not have any obsolete test cases because there is no case of the deletion of operations in our case study which makes the test cases obsolete.

As compared to our previous approach for state based regression testing [1], this approach is more safe as it also deals with those fault revealing test cases that were not handled in the previous approach. If we run the same case study on our previous approach, it will fail to detect all the re-testable test cases shown in table 3. We have also developed a prototype tool to prove the applicability of our approach. The tool takes XMI of the UML 2 sequence diagrams, class diagrams and state machines for comparison. The test cases are given as input in form of XML for regression test selection. We used the Eclipse platform and [18] EMF and UML 2.1 plug-in of Eclipse which is an implementation of UML 2.1 meta-model.

8. Conclusion

We presented a state based approach for safe regression testing that extends our previous work on state based regression testing by incorporating change information form sequence diagrams with it. We defined changes in the baseline and delta version of the sequence diagrams based on UML 2.1 meta-model and used them to extend our state based regression testing technique. Our work advocates that changes in sequence diagram can modify the corresponding operations defined in the class diagram; thus affecting the state based regression testing. We applied our approach on a case study of library management system. Our results show that our proposed enhancements make the existing technique safe as they also identify those fault revealing test cases which were not handled by the previous approach.

Acknowledgements

Our thanks to Dr. S.A. Bhatti, Mr. Usman Habib, Mr. Anees-ul-Haq and Mr. Sohail Razzaq for their support and cooperation throughout this work.

References

[1] Qurat-ul-ann Farooq, Muhammad Zohaib, Z. Iqbal, Zafar I Malik and Aamer Nadeem " *An Approach for Selective State Machine based Regression Testing*", Proceedings of the 3rd international workshop on Advances in model-based testing, Pages: 44 - 52, ISBN:978-1-59593-850-3, London, United Kingdom, 2007

[2] L. C. Briand, Y. Labiche, L. O'Sullivan, " *Impact Analysis and Change Management of UML Models*" Carleton Technical Report SCE-05-02, Version 3, October 2006

[3] Rothermel, G., and Harrold, M.J. (1997). *A safe, efficient regression test selection technique,* ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 6, Issue 2, pp: 173 - 210 , Publisher: ACM

[4] Sajeev, A.S.M, and Wibowo, B, *Regression test selection based on version changes of components*. Tenth Asia-Pacific Software Engineering Conference, pp 78- 85, ISBN: 0-7695-2011-1, Dec 2003.

[5] D. Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C., Kim, Y.S., and Song, Y.K. (1995). *Developing an object-oriented software testing and maintenance environment*, Communications of the ACM, Volume 38, Issue 10, p: 214 , ISBN:0-7695-1819-2

[6] Traon, YL. Jeron, T. Jezequel, JM. Morel, P. *Efficient object-oriented integration and regression testing.* , IEEE Transactions on reliability, Volume: 49, Issue: 1, ISSN: 0018-9529,

[7] Binder, R. (2000). *Testing Object-oriented Systems: Models, Patterns, and Tools.*, Published by Addison-Wesley, ISBN 0201809389

[8] Samar Mouchawrab, Lionel C. Briand, Yvan Labiche, " *Assessing, Comparing, and Combining Statechart- based testing and Structural testing: An Experiment*", Carleton University, TR SCE-06-15, September 2006.

[9] Ali, S., Briand, L.C., Rehman, M.J., Asghar, H., Iqbal, M.Z.Z., and Nadeem, A. (2006). *A State-based Approach to Integration Testing based on UML Models*, Journal of Information and Software Technology, Volume 49, Issue 11-12, pp: 1087-1106

[10] G. Antoniol, L. C. Briand, M. D. Penta, and Y. Labiche, " *A Case Study Using the Round-Trip Strategy for State-Based Class Testing*," Proc. 13th IEEE International Symposium on Software Reliability Engineering (ISSRE'02), pp. 269-279, November 12-15, 2002.

[11] Tom Pender, *UML Bible*, Published by: Wiley Publishing, Inc, ISBN:0764526049

[12] Briand, L.C., Labiche, Y, and Soccar, G. " *Automating Impact Analysis and Regression Test Selection Based on UML Designs*". Carleton University, TR SCE-02-04, Version 2

[13] Korel, B., Tahat, L.H., and Vaysburg, B. (2002). *Model Based Regression Test Reduction Using Dependence Analysis,* Proceedings of the International Conference on Software Maintenance (ICSM.02)

[14] Briand, L.C, Y. Labiche, L. OSullivan, M.M. So´wka. , "Automated Impact Analysis of UML Models", *The Journal of Systems and Software 2005*

[15] OMG, UML 2.1: Super structure Specifications, OMG, April 2006: Available: http://www.omg.org/cgi-bin/doc?formal/07-02-03

[16] Beydeda, S., and Gruhn, V. *Integrating White- and Black-Box Techniques for Class-Level Regression Testing.* Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, Pages: 357 –362, ISBN: 0-7695-1372-7, 2001.

[17] Eclipse, Available at: *http://www.eclipse.org/downloads/*, The Eclipse Foundation, Last Visited: November 2007

[18] *Ravi Prakash Gorthi, Anjaneyulu Pasala, Kailash KP* Chanduka and Benny Leong, Specification-based Approach to Select Regression Test Suite to Validate Changed Software, Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference, pp 153-160 , Year of Publication: 2008

[19] Leila Naslavsky, Debra J. Richardson, Using Traceability to Support Model-Based Regression Testing, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 567-570, 2007

[20] N. Mansour and H. Takkoush, "UML based regression testing technique for OO software," in Proceedings of the IASTED International Conference on Software Engineering and Applications, pp. 96–101, Boston,Mass, USA, November 2007.