A Model-Based Regression Testing Approach for Evolving Software Systems with Flexible Tool Support

Qurat-ul-ann Farooq
Technische Universität Ilmenau, Germany
Qurat-ul-ann.farooq@tu-ilmenau.de

Muhammad Zohaib Z. Iqbal
Simula Research Laboratory, Norway
zohaib@simula.no

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

Matthias Riebisch
Technische Universität Ilmenau, Germany
Matthias.riebisch@tu-ilmenau.de

Abstract

Model-based selective regression testing promises reduction in cost and labour by selecting a subset of the test suite corresponding to the modifications after system evolution. However, identification of modifications in the systems and selection of corresponding test cases is challenging due to interdependencies among models. State-based testing is an important approach to test the system behaviour. Unfortunately the existing state-based regression testing approaches do not care for dependencies of the state machine with other system models. This paper presents the tool support and evaluation of our state-based selective regression testing methodology for evolving state-based systems. START is an Eclipse-based tool for state-based regression testing compliant with UML 2.1 semantics. START deals with dependencies of state machines with class diagrams to cater for the change propagation. We applied the START on a case study and our results show significant reduction in the test cases resulting in reduction in testing time and cost.

1. Introduction

Evolution is inherent in the software systems. Due to the growing size and complexity of modern systems, the evolving nature of a system can cause adverse effects and even system failures. Besides many other measures to prevent these unintended effects of evolution, one very important measure is to test the systems after introducing the modifications which is often referred to as *Regression Testing*. Repeating the entire testing activity whenever the system is modified is a very costly task. A large system may have a huge number of test-cases and test-execution requirements. Executing all these test-cases is generally not an option. Hence, it is necessary for regression testing to select a subset of the system corresponding to modifications. This is known as the selective strategy for regression testing and is a more feasible solution in terms of cost and time [3].

Model-based regression testing has several advantages compared to the conventional code-based regression testing approaches. The traditional code-based testing approaches fail to scale when the size of the system increases. Model-based testing approaches provide better *complexity management* and *better comprehension* of the system, the relevant test suites and test cases. In model-based testing, the testing activity can be started in the early phase of software development which results in *effort reduction* in terms of time and labour. Further, *traceability maintenance* between test cases and models is easy and finally, *portability* and *platform independence* is a major benefit for evolving systems to adopt the rapid changes in technology and operational environment [4].

In model-based development, several artefacts are inter-related. A change in one artefact can affect other related artefacts; hence, it is necessary to cater for these relationships and dependencies for effective regression testing. In UML-based development, the class diagram shows the structural aspects of classes and state machines reflect the behavioural aspects of the classes [15]. State machine-based testing is widely used in practice to test class behaviour. When a class is modified, due to any change in the specification, both structural and behavioural aspects can be modified and it is necessary to retest the corresponding test cases. The relationship between class diagram, state machine and the corresponding test suite is shown in Figure 1.

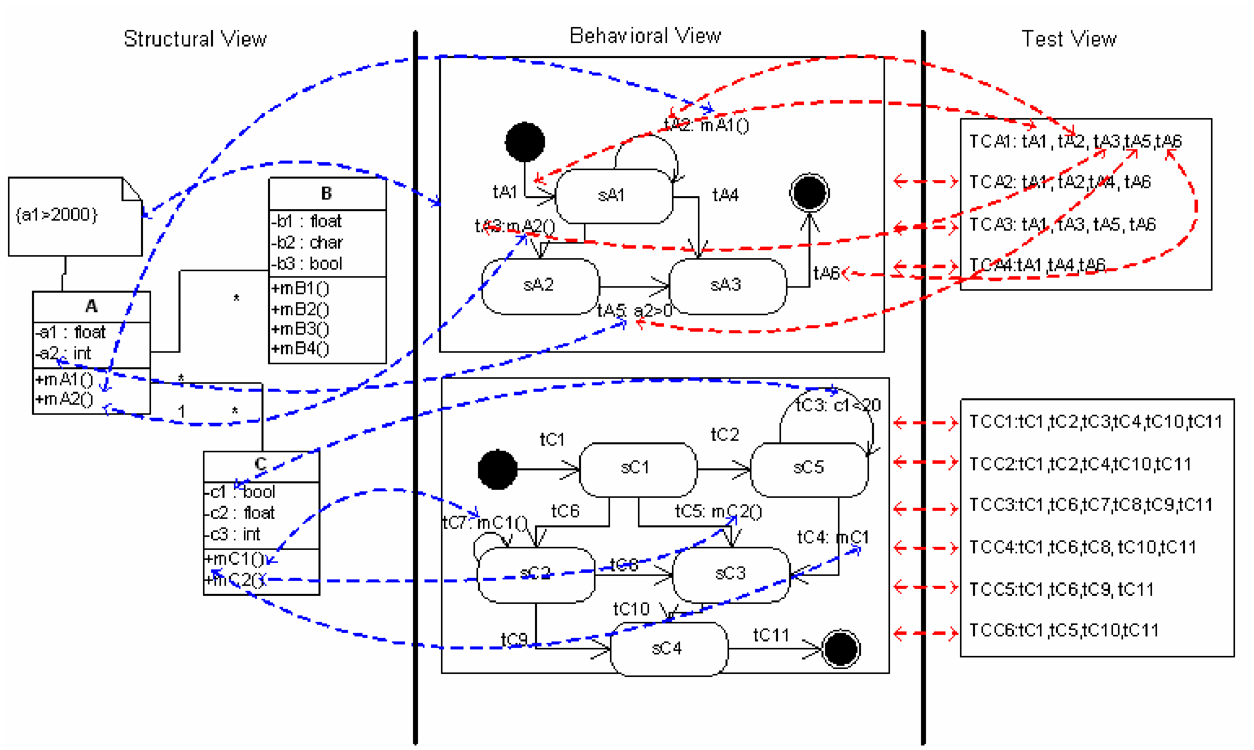

Figure 1: Relations between structural, behavioural and test view

According to Figure 1, relationship exists between the class diagram (structural view) and state machines (behavioural view). For example, attributes defined in the classes can be referred in the state machine's events, guards and actions. Similarly methods defined in class diagrams can be used in events and actions on state machine as well. The state machines should also conform to the corresponding class invariants and operation contracts defined in the class diagram.

Likewise, relationships exist between state machines and the test suite derived from the state machine. For example, the test case with ID "TCA1" depicted in Figure 1, corresponds to transitions tA1, tA2, tA3, tA5 and tA6 of the first state machines. It is important to understand and use these relationships for safe and effective regression testing.

In our previous work [1] on state-based regression testing we employed these relationships between class diagram and state machines for dealing with change propagation during regression testing. This paper presents our regression testing tool START (The *ST*Ate-based *R*egression *t*esting *T*ool) which is implemented using concepts of our state-based regression testing approach [1]. We also present the application of our tool on a case study. The case study is executed on tool and we obtained reasonable reduction in the test suite for regression testing. The tool supports the UML 2.1 standard. START provides a flexible means of manipulating models by using an XMI representation of models. It can be easily integrated with any modelling environment and can be integrated as a plug-in with the IDE and a testing tool like JUnit [24].

The rest of the paper is organized as follows; Section 2 describes our previous state-based regression testing methodology. Section 3 describes the implementation of our methodology using our tool START. Section 4 reports our case study of the Student Enrolment System. Section 5 discusses and evaluates the related work in the field and section 6 concludes our work.

2. The State-based Regression Testing Approach

In this section, we discuss our previous state-based regression testing methodology [1] to clarify the fundamental concepts behind the implementation of START. As shown in Figure 1, we use the relationships between class diagram and state machine and to the corresponding test suite to deal with change propagation. Figure 2 provides an overview of our approach. Following are major task involved in the process as expressed in Figure 2.

2.1 Change Identification:

First of all, changes in the class diagram are obtained by comparing the baseline version of class diagrams and the delta version of class diagrams along with class invariants and operation contracts. We refer this change set as class-driven changes. According to Figure 2 *ClassDiagramComparator* performs this task of comparing two versions of a class diagram.

2.2 Change Impact Analysis:

After computing the class-driven changes the *StateMachineComparator* reflected in Figure 2 compares the baseline and delta version of state machines along with state invariants. Changes in both versions are detected and class-driven changes are also used to obtain the affected elements of the state machine. For example, a state transition will be marked as affected if it is using any changed attribute or operation of the corresponding class in its guards, events or actions. We refer to these changes as state-driven changes.

2.3 Regression Test Selection:

Finally, the set of affected test cases from the baseline test suite are selected using *RegressionTestSelector* by tracing the state-driven changes to the corresponding test-cases. Our test suite is classified into three types of test cases; obsolete, reusable and re-testable [13].This classification is adopted by several regression testing techniques in the literature [4, 9]. Obsolete test cases are no more valid for the delta version. They usually correspond to elements in the system that are deleted and are not accessible in the delta version.

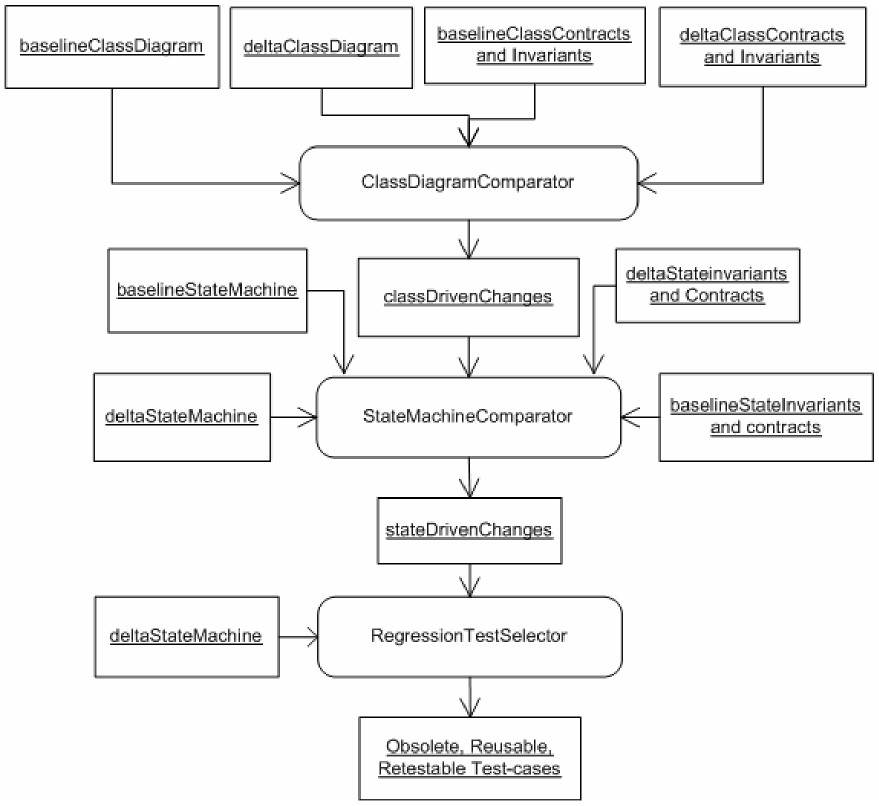

Figure 2: Overview of our state-based regression testing methodology

Re-testable test cases need to be executed for regression testing as they correspond to modified parts of the system. Reusable test cases correspond to unchanged parts of the system. They are valid but they are not required to be re-executed for regression testing.

For performing the change identification and change impact analysis, it is necessary to define the changes in the artefacts precisely. In the next section, we discuss the change definitions required for our approach.

3. Change Definitions

As discussed in the previous section, our approach requires the definition of two types of changes; class-driven changes and state-driven changes. The change definitions and change models used in our approach are inspired by the work of Briand et al. [4, 21, 22].

3.1 Class-driven changes

Class-driven changes are those changes which can be obtained by comparing a baseline and delta version of the class diagram. These changes are sometimes directly visible on the state machines, such as deletion of an attribute from a class may cause change in an action using this attribute on the corresponding state machine. Some times these changes are not reflected on the state-machine and this information can be obtained only by analysing the class diagrams. For example, the change in type of an attribute will not be reflected on the state machine but we need to consider all those transitions using this attribute in some events, actions or guards for this change.

3.2 State-driven Changes

State-driven changes are obtained by comparing the base line and delta version of the state machine and by using the set of class-driven changes. Figure 3 presents an example of a simple change model defining a modified transition. According to Figure 3, the *ModifiedTransition* class is a sub class of *CompositeModifiedElement*.

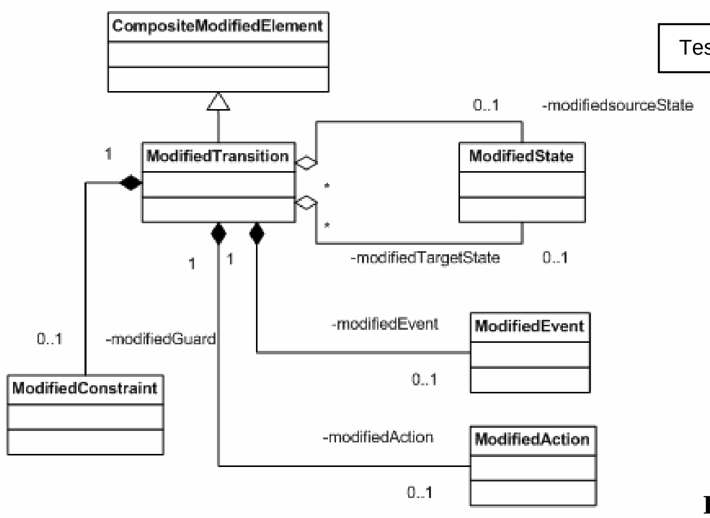

Figure 3: The change model for modified Transition

The *CompositeModifiedElement* class is an abstract class and it refers to the modified elements which are further composed of other composite and atomic elements. A transition can be modified if it refers to some source or target modified state, as depicted by the association ends *modifiedSourcestate* and *modifiedTargetState*.

A transition can also be considered modified if it refers to some modified event, guard or action as depicted by the association ends *modifiedEvent*, *modifiedAction* and *modifiedGuard*.

The *ModifiedState*, *ModifiedEvent*, *ModifiedConstraint* and *ModifiedAction* are themselves sub classes of *CompositeModifiedElement* and are defined by separate change models. Due to space constraints we can not present all change definitions in this paper. Interested readers should refer to our previous work on the topic [1].

The *ClassDiagramComparator* and *StateMachineComparator* use these definitions for the comparison of two versions of class diagram and state machines and construct the corresponding change models reflecting the changes between two versions. This change information is used by the *RegressionTestSelctor* for the selection of regression test suite. The following section discusses our regression test selection process.

3 Test Case Selection

As discussed earlier, we classify our test suite into obsolete, reusable and re-testable test cases. Figure 4 shows the classification of test paths of using our selection methodology.

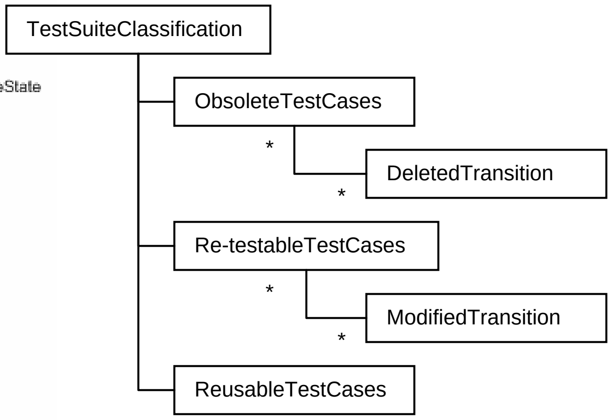

Figure 4: State-based Test Suite Classification

According to Figure 4, all obsolete test cases will correspond to some deleted transitions and all re-testable test cases will correspond to some modified transitions. A simple definition of a modified transition is already discussed in the previous section. For more rigorous and detailed definitions of modified and deleted transitions, interested readers can refer to our previous work [1]. We define the baseline test suite as

TS, baseline state machine as SM and delta version state machine as SM′. The set of deleted transitions in SM′ refers to T'_D and the set of modified transitions in SM′ refer to T'_M.

Our base line test suite consists of the test paths extracted from the SM by applying any of the state-based testing strategies [3]. Each test path $ts \in TS$ in the test suite will represent a sequence of transitions $t_i \in T$ in the SM.

The set of obsolete test cases is defined as $TS_O \subset TS$. The set of reusable test cases is defined as $TS_{RE} \subset TS$ and $TS_{RT} \subset TS$ is the set of re-testable test cases. A test case $ts \in TS$ will be added to TS_O if some transition $ti \in TS$ exists in T'_D. A test case $ts \in TS$ will be added to TS_{RE} if some transition $ti \in ts$ exists in T'_M and $ti \notin T'_D$. All the other test case in TS belong to TS_{RT} and are not included in $TS_O \cup TS_{RE}$.

4 START: STAte-based Regression-testing Tool

In this section, we will discuss how we implemented our regression testing tool START, based on the concepts discussed in the previous sections. As stated earlier, START is built on the Eclipse platform. We used an EMF plug-in which conforms to the UML 2.1 meta-model [12]. The benefits of adopting an Eclipse-based platform include the ease of possible integration of the technique with other modelling environments and testing tools. We will explore these possibilities later in this section in detail. Another benefit is the plug-in centric development supported by Eclipse.

START uses input models in XMI v2.1 format to ease the diagram interchange. The baseline test cases are also taken in an XML format to make it possible to integrate the tool with other state-based test generation tools and to make the interchange of test cases easy. START consists of three major components; A *Parser*, a *Comparator* and a *TestSuiteAnalyzer*. *Parser* consists of a number of sub components (*XMIParser, OCLParser, ClassDiagramParser, StateMachineParser*). The Comparator component is further divided in *ClassDiagramComparator* and *StateMachineComparator*.

XMI parser is used to read the XMI of class diagrams and state machines, and the *OCL parser* will be used to parse the contracts written in OCL. The *ClassDiagram parser* and *StateMachineParser* parse the XMI files and populate the instances of UML 2 meta-model of classes, associations and state machines.

StateMachineComparator uses these instances and the baseline and delta version of state machines to generate instances of *ModifiedStateMachine*. TestSuiteClassifier uses the information of *ModifiedStateMachine* and baseline test suite and classifies the baseline test suite into obsolete, reusable and re-testable test cases [13]. Figure 5 shows the package diagram of *START*. *START* consists of 5 major packages; Parser, Comparator, Regression Test Selector, Change and Test Container.

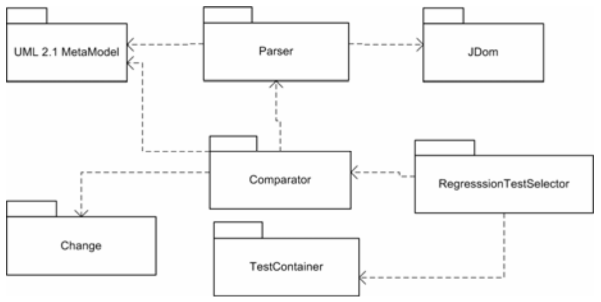

Figure 5: Package Diagram for START

The parser uses an EMF-based [12] implementation of the UML meta-model to populate model instances and JDom to manipulate the XML test cases and the XMI of models. The comparator uses the parser package to obtain the instances of class diagram and state machine models for change identification. It uses the change package to populate change models described in section 2.2. Finally, the regression test selector uses the test container package which provides the baseline test cases and using the change information obtained from the comparator to perform the regression test selection.

Figure 6 depicts the models and their respective meta-models employed by START. According to figure 6, the packages above the horizontal strong line represent the meta-models used by START. START uses three distinct meta-models. The UML 2.1 meta-model is implemented as EMF plug-in by Eclipse [12]. The base line and delta version class diagram and state machines conform to the UML meta-model. The Change meta-model which we discussed in section 3 is required to represent change information. The change definitions provided in section 3 from this Change meta-model. The change information between baseline and delta version of models obtained by comparator is stored in the change models conforming to the Change meta-model. Finally Test case meta-model is required for representation of test suite. The baseline test suite required by the test selector will conform to the Test case meta-model. Similarly, the regression test suite constructed by the test selector will also conform to the Test case meta-model.

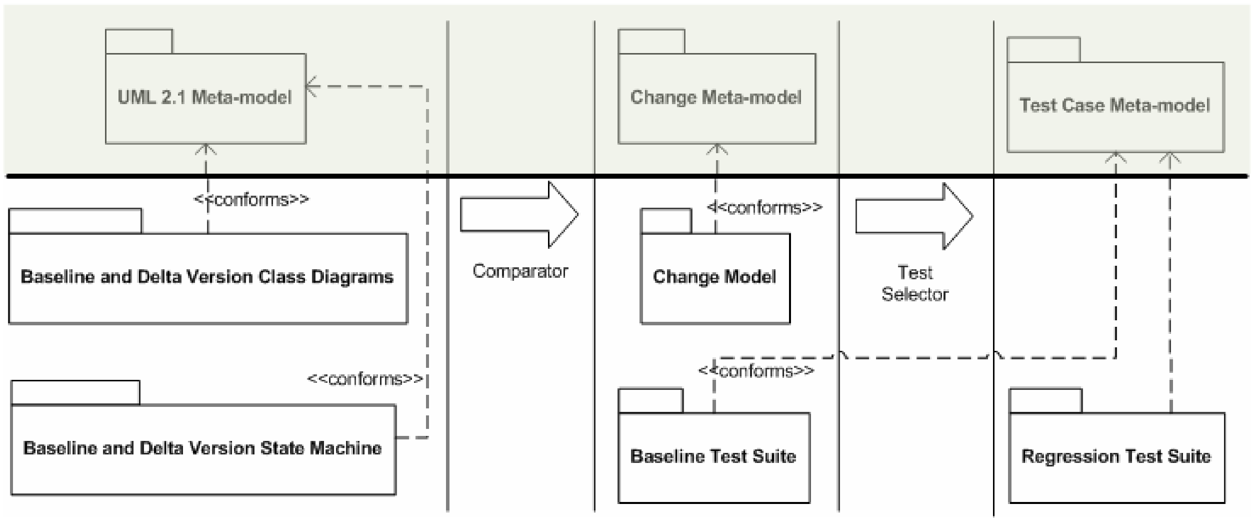

Figure 6: The relationships between models used by *START* and their meta-models

Following we will discuss how START provides a flexible way of integration with other testing tools and modelling environments.

.**Integrating START with Testing Tools**

START provides a flexible architecture and can work with other testing tools like JUnit [24]. In Figure 5, the package diagram depicts the package "*Test Container*". Test Container is the package that contains test classes of the baseline version.

The test container package contains an interface TestSuite which represents the test suites corresponding to the state machines. For integrating JUnit with our approach, the JUnit test class can implement the test suite interface and we have to define a method getAllTestCases() that will return the JUnit test cases. The ID of JUnit test cases should depict the corresponding state-machine transitions so that the implicit traceability between state machine transitions and test cases could be retrieved at the time of test selection.

Integrating *START* with Modelling Environments
Due to inconsistencies in the XMI representation of models in several available modelling tools; it is hard to cater for all possible XMI representations. We are working on a module that can transform multiple XMI formats in one standard simple format to be processed by START. At presents, START supports XMI format used by the Visual Paradigm modelling environment.

5 Case Study

In order to validate the applicability of our approach and our regression testing tool START, we present a case study of a Student Enrolment System. The baseline version of the system consists of seven classes. The behaviour of the Student and Course classes of the system is described by using the state machines. Figure 7 shows an excerpt of the course state machine.
The student state machine consists of 9 states and 18 transitions. The course state machine is rather complex and consists of 14 states and 26 transitions. We constructed the baseline test suite for the classes in the system using transition tree methodology (Binder, 2000). The baseline test suites of the Student and Course classes consist of 58 and 723 test cases respectively. The delta version of the system contains following changes

- The type of an attribute is changed in the Student class from String to Boolean (the "defaulter" attribute)
- A state and two transitions are removed from the Student state machine (State 2 and transition T1 and T3)
- A new parameter is passed to the function closeRegistration() in the Course class (course code)

Table 1 depicts the results computed by executing our case study on *START*. From the 723 test cases of the

Course class 447 are required to re-execute for the regression testing.

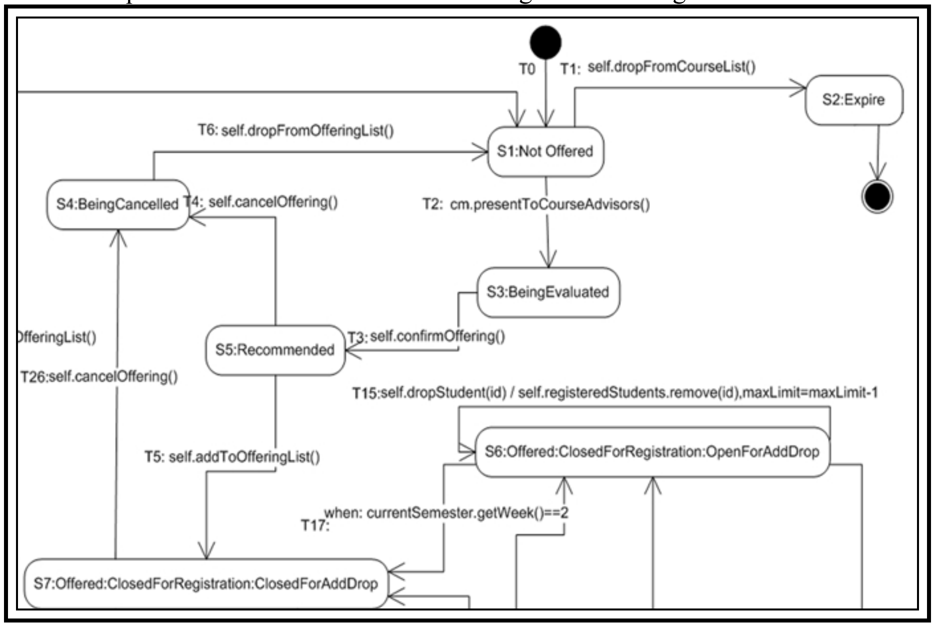
T0
T1: self.dropFromCourseList()
S2:Expire
T6: self.dropFromOfferingList()
S1:Not Offered
S4:BeingCancelled
T4: self.cancelOffering()
T2: cm.presentToCourseAdvisors()
S3:BeingEvaluated
OfferingList()
S5:Recommended
T3: self.confirmOffering()
T26:self.cancelOffering()
T15:self.dropStudent(id) / self.registeredStudents.remove(id),maxLimit=maxLimit-1
S6:Offered:ClosedForRegistration:OpenForAddDrop
T5: self.addToOfferingList()
T17: when: currentSemester.getWeek()==2
S7:Offered:ClosedForRegistration:ClosedForAddDrop

Figure 7: An Excerpt of course state machine

None of the test cases are obsolete and 276 test cases are preserved for future use. From 58 test cases of the Student class 15 test cases need to be re-executed for regression testing, 29 test cases become obsolete and are required to be excluded from the baseline test suite and 14 test cases are preserved for the future use.

Table 1: Results of the Student Enrolment System case study

Classes	**Total base-line test cases**	**Reusable**	**Re-testable**	**Obsolete**
Course	723	276	447	0
Student	58	14	15	29

6 Related Work

In this section, we discuss the related work on UML-based regression testing and state-based regression testing. Various UML-based regression testing approaches are reported in the literature. However, all these approaches do not deal with the state-based testing aspects.

Following we discuss these approaches in brief to see how the UML-based regression testing techniques present in literature deal with change identification, impact analysis and test selection activities.

Naslavsky et al. [21] 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. Briand et al. [4] presented a technique for regression testing based on use case diagrams, sequence diagrams and class diagrams. They provide impact analysis among these different diagrams to deal with change propagation and provide a mechanism for test case selection. They implemented a tool RTS for the proof of concept. For change identification and change definitions the change models used in our approach are inspired by Braind et al.'s work on change impact analysis and consistency management [21, 22]. However, the change definitions provided by Briand et al. are based on UML 1.4 and are focusing on model refinements and consistency checking and are not complete for our purpose

Ali et al. [18] presented an approach for regression testing using class and sequence diagrams. They construct an extended control flow graph using both diagrams. The change identification is based on comparison of both diagrams and then the selection of corresponding test cases is performed. Jeron et al. [6] present an approach for integration and regression testing from the UML class diagrams. They constructed a Test Dependency Graph (TDG) as their test model to show the dependencies among the classes. The authors proposed several coverage criteria for regression testing if a class is modified. However, how changes will be identified is not addressed by the authors. Wu and Offutt [7] discuss UML-based regression testing of component-based software using class diagrams, collaboration diagrams and state charts for specification of components. The authors provide some guidelines of using these artefacts for regression testing but further rework is required for implementations of these guidelines.

Chen et al. [9] presented a regression testing strategy using activity diagrams. They used activity diagrams for describing the system requirements. They treated activity diagrams as a control flow graph and perform change identification by comparing nodes in the graph. The test cases corresponding to the nodes in the graph are selected by using a traceability matrix. The test cases are also prioritized based on important functional requirements and risk prone requirements.

Gorthi et al. [19] 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. Deng et al. [10] presented a rule-based approach for regression testing using use case diagrams, class diagrams, sequence diagrams and activity diagram. The test cases are generated based on activity diagrams. However, rework is required to apply the technique to a real scenario due to fewer details.

Mansour et al. [21] 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.

Palkins et al. [16] presented an approach for regression testing of UML models, their work is not related to implementation testing, and hence, it is not of interest for us. Sajeev et al. [17] presented an approach for modelling version data with UML. The actual input of the approach are not UML models, hence, their work is also not very related to our work.

A limited research is also available in literature in which state machine like models are used for regression testing. Korel et al. [14] presented a regression testing technique using EFSM (Extended Finite State Machine). They perform change identification based on two elementary modifications, addition and deletion of a transition. A dependence graph considering data and control dependence is constructed and test suite reduction is performed by calculating interaction patterns based on the dependence graph.

Beydeda and Gruhhn [8] present a regression testing technique using specification and implementation information. They used a class state machine (CSM) similar to a state machine. A control flow graph is generated using CSM and the methods presented in the implementation. Both graphs are comparing for change identification and test selection. One major drawback of their approach is that they require implementation and code level details along with the specification.

These existing state-based regression testing techniques are not compliant with any standard UML meta-model. Besides, these techniques do not deal with the interdependencies of state machine with other system artefacts as we did. Hence, they fail to deal with the change propagation phenomena. The tool support provided in the area is very limited. None of the state-based testing techniques provide any tool support to ease the regression testing process.

7 Conclusions and Future work

In this paper, we reported our tool support for our state-based regression testing methodology. START is our regression testing tool that uses UML 2.1 state machines and class diagrams and their dependencies for regression test selection. START is based on the Eclipse platform and provides an EMF-based solution. START can be easily integrated with other modelling environments and testing tools.

START classifies state-based test suite into obsolete, reusable, and re-testable test cases for regression testing. We applied a case study of Student Enrolment System on START to prove the applicability of our approach. Our results show significant reduction in the size of the baseline test suite for regression testing.

In future, we intend to investigate the possibility of the application of our approach using the model driven development methodology. We want to investigate the required transformations and to see how the relationships between models at different levels of abstractions can affect our original methodology.

8 References

[1]. Qurat-ul-ann Farooq, Muhammad Zohaib, Z. Iqbal, Zafar I Malik, Aamer Nadeem, "An approach for selective state machine based regression testing", ACM 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]. Heiko Stallbaum, Andreas Metzger, Klaus Pohl, "An Automated Technique for Risk-based Test Case

Generation and Prioritization", Proceedings of the 3rd international workshop on Automation of software test, pp. 67-70, 2008

[3]. Binder, R.. Testing Object-oriented Systems: Models, Patterns, and Tools, Published by Addison-Wesley, The Addison-Wesley object technology series, ISBN 0201809389, 2000

[4]. L.C. Briand, , Y. Labiche, S. He, Automating regression test selection based on UML designs, Information and Software Technology, Volume 51 , Issue 1, January 2009

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

[6]. Thierry Jéron, Jean-Marc Jézéquel, Yves Le Traon, Pierre Morel, , Efficient Strategies for Integration and Regression Testing of OO Systems, 10th International Symposium on Software Reliability Engineering, p. 260, 1999

[7]. Wu, Y., Offutt, J. Maintaining Evolving Component based software with UML, In Proceeding of 7th European conference on software maintenance and reengineering, IEEE, pp: 133- 142, ISBN:0-7695-1902-4, Publisher: IEEE Computer Society, 2003

[8]. 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, pp. 357 – 362, 2001

[9]. Chen, Y., Probert, R.L., and Sims, D.P. Specification-based Regression Test Selection with Risk Analysis, Proceedings of the conference of the Centre for Advanced Studies on Collaborative research, pp: 1, 2002

[10]. Deng, D., Sheu, P.C.Y., and Wang, T. Model-based testing and maintenance, Proceedings of IEEE Sixth International Symposium on Multimedia Software Engineering, Volume 00, pp. 278- 285, ISBN:0-7695-2217-3, 2004

[11]. Eclipse, Available at: http://www.eclipse.org/downloads/, The Eclipse Foundation, Last Visited: November 2009

[12]. UML2. UML Plugin for Eclipse, Available at: http://www.eclipse.org/modeling/mject=uml2, Last Visited: November 2009

[13]. Leung, H.K.N. White, L. , Insights into regression testing software testing, Proceedings of Conference on Software Maintenance., , On page(s): 60-69, ISBN: 0-8186-1965-1, Oct 1989.

[14]. 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)

[15]. OMG, UML 2.1: Super structure Specifications, OMG, Available at: http://www.omg.org/cgi-bin/doc?formal/07-02-03, Last visited: November 2009

[16]. Pilskalns, O., Andrews, A. Regression Testing UML Designs, In Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM'06), Publisher: IEEE Computer Society, Pages: 254 - 264 , ISBN :1063-6773, 2006

[17]. 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, 2003

[18]. Atifah Ali, Aamer Nadeem, Muhammad Zohaib Z. Iqbal, Mohammad Usman, Regression Testing based on UML Design Models, Proceedings of the 13th Pacific Rim, International Symposium on Dependable Computing, pp. 85-88, 2007

[19]. 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

[20]. 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

[21]. 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.

[22]. Briand,L.C, Labiche,Y, Yue, T, "Automated Traceability Analysis for UML Model Refinements", Technical Report: TR SCE-06-06, Version 2, Carleton University, Ottawa, , Canada, http://sce.carleton.ca/squall, August 2006.

[23]. Briand, L.C., Labiche, Y, Sullivan, L.O, and So´wka, M. M. "Automated impact analysis of UML models". Journal of Systems and Software, Volume 79, Issue 3, Pages: 339 - 352, ISSN: 0164-1212, March 2006.

[24]. Junit, available at: www.junit.org, Last visited: Novemebr 2009