

A Rational Approach on Model Based Slicing

Tamanna¹, Dr. Sukhvir Singh²

Computer Science and Engineering, N.C. College of Engineering, Israna
Panipat, India

¹grover.tanu13@gmail.com

²boora_s@yahoo.com

Abstract: Aim of Software testing is to evaluate an attribute or capability of system and determine that whether it meets required prospects. The most rationally challenging part of testing is to design of test cases. Now a days, UML has been extensively used for object oriented modeling and design. This is due to the fact that UML metamodel extends support to describe structural and behavioural aspects of an architecture. However, it is still difficult to understand this behaviour, because the size of automatically generated model diagrams tends to be huge. To overcome this problem of software visualization model based slicing technique has been developed. Model based slicing is a breakdown technique to extract and identify relevant model parts or related elements across diverse model views. We have proposed a novel procedure to extract the sub- model from a big model diagrams on the basis of slicing criteria. The proposed methodology use the concept of model based slicing to slice the sequence diagram to extract the desired chunk. In the presented approach UML, conversion of UML into XML, Java DOM API for parsing and slicing has been used. Then by using the Editor Extracted Sequence Diagram has been generated.

Keywords- Model Based Slicing, Sequence Diagram, Parsing, Slicing, UML.

I INTRODUCTION

Due to the rise in size and complexity the importance of architectural design has been increased of software products. The architecture of software system defines its design structure and allows an architect to find about various properties of the system which are at the higher level of Abstraction. For doing this, Unified Modeling Language (UML) is the best choice and it is extensively used to signify and construct the architecture with the help of its various model diagrams of software system. UML diagrams tell us about the structural and behavioural aspects of architecture [1, 2]. Various relations among objects, such as Aggregation, association, composition and generalization/specialization etc. can be defined by Structural models (e.g., class diagrams, object diagrams, component diagrams). On the other hand, sequence of actions, states and their interaction, through which a use case is realized, can be defined through the behavioural models (e.g. communication and sequence diagrams, activity diagram, state diagrams). Analysing UML Models is bit challenging task since the information about the system can be spread across several model views. The concept of model based slicing came into existence to overcome this problem. To extract and identify relevant model parts or related elements a decomposition technique is used ant that is Model Based slicing. It takes the user defined slicing criteria as input and slices the architecture, as a view of interest [3]. Slicing is helpful in reengineering, software maintenance, testing, recompilation, program comprehension, decomposition and integration, recompilation, program comprehension, and debugging. The goal of software testing is to confirm quality. Highly reliable systems are produce by Software Testing, since static verification techniques agonise from

several difficulties in detecting all software mistakes [15]. The most challenging part of testing is the design of test cases. With the help of program source code Test Cases are generated. Another approach is to generate test cases developed using formalisms such as UML models. Without affecting their core structure and functionality, structure can be decomposed into sub models and slicing is best technique for this. It helps the developer to gain the perfect view of software according to their requirement.

II RELATED WORK

Model based slicing has been applied to state machines in early stage of development [4] where similar benefits as those listed above have been claimed. State machine slicing is that when applying slicing to a model of a system rather than to the system implementation. However, system models which are represented in terms of the UML-family of languages are more complex than state machines (and contain state machine sub-languages). Some approaches and efforts have been made for slicing UML diagrams. The approach in [5] define context-free slicing of UML class models where the matter of context has been defined to be object location, which is a dynamic property of the situation therefore for the structural model context free slicing is a static slice. As described in [5] the procedure used for slicing a program or state machine is not much complex than that of slicing a model since there are more types of elements and relationships in program slicing or state machine. OCL (object constraint language) should be used to express the slicing criteria. A similar approach has been used to modularize the UML meta-model into groups of constituents that are related to the different UML diagram types in [6] although the predicate that are used to find out the slicing criteria has been

secure in terms of traversing the meta-model elements starting with a collection of supplied classes. In conjunction with OCL invariants Class Models has been sliced, in [7] thereby decreasing the state-space explosion that would otherwise occur after using a model-checker to verify a class-model. UML state-charts can be sliced as described in [8] [9] [10] although these methods do not simplify the results to include other parts of the UML language family. Both static and dynamic aspects of UML can be combined and sliced as described in [11] [12] where class and sequence diagrams are merged into a single representation (a model dependency graph MDG) that can be consequently sliced to show partial dynamic and structural information resulting from criteria containing both structural and dynamic constraints. In order to generate test cases Slicing UML sequence diagrams has been described in [13] [14]. UML sequence diagrams (or scenarios) are basically an integral part of implementations of a program. It shows the objects and classes involved in the situation and the sequence of messages exchanged between the objects. Sequence diagrams are usually associated with use case realizations in the Logical View of the system under progress. It has been methodically analysed that for the procedure of slicing sequence diagram no consolidate technique have been developed to extract the point of interest from architecture of software to ease the software visualization that uses conditional predicate for finding out a relative slice. Generic view of functional behaviour of software models in the form of sequence diagram can be shown by Figure 1. Rectangular box states the objects within model diagram that are communicating with each other. Dotted lines denote the life line of the objects on which instances of the objects has been created. Arrow tells about the particular action (in the form of messages) of objects and their direction.

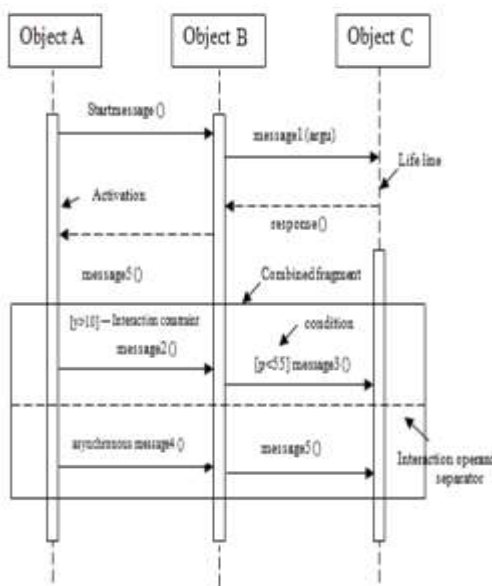


Figure 1: Generic view of Sequence diagram [1]

III PROPOSED METHODOLOGY AND IMPLEMENTATION

The proposed work addresses the slicing of sequence diagram to ease the software visualization by using conditional predicate for finding relevant slices.

In the proposed methodology, following steps has been followed:

1. From a particular requirement specification UML (Sequence) diagram has been generated.

1.1. Visual paradigm for UML, Rational rose and Magic-draw, etc can be used to generate the UML diagrams.

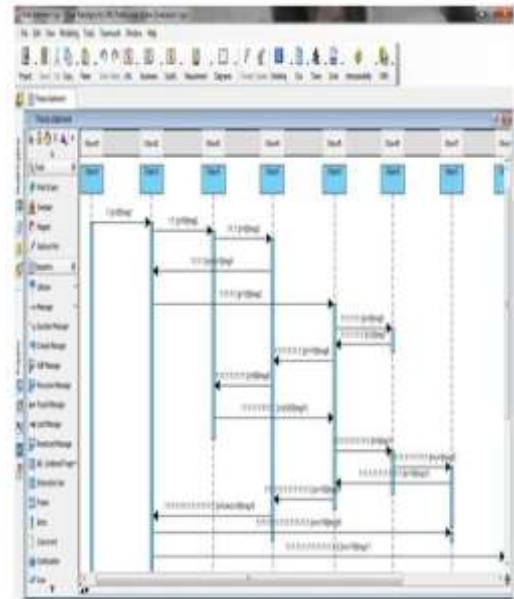


Figure 2: Designing sequence diagram using visual paradigm

2. From the specified UML diagram (Sequence diagram) create XML.

2.1. Visual paradigm for UML 10.0 version provides the in-built functionality to export the diagrams into XML format.



Figure 3: XML file of Sequence diagram

3. Document Object Model (DOM) parser for parsing XML code and generate an output file (with .txt extension) having Object name, identifier, message name, message to & fro information.

3.1. Java API DOM is used to parse the XML code file generated in step 2.

3.2. DOM parser uses the function Document-Builder-Factory () to create the instance of the class to parse the file.

3.3. DOM parser will generate a txt file having information regarding object name and its identifier. This file also contains the information related to all the messages and the objects among which the message is floating.

3.4. All the information generated by parser will be stored in separate .txt file.

```
Object-Name=Object1 Object-Id=0HL4XyFYH5DwNl
Object-Name=Object9 Object-Id=0FL4XyFYH5DwNf
Object-Name=Object8 Object-Id=A5L4XyFYH5DwNX
Object-Name=Object7 Object-Id=25L4XyFYH5DwNQ
Object-Name=Object6 Object-Id=5pL4XyFYH5DwNJ
Object-Name=Object5 Object-Id=4pL4XyFYH5DwNC
Object-Name=Object4 Object-Id=0DL4XyFYH5DwM7
Object-Name=Object3 Object-Id=0HL4XyFYH5DwM0
Object-Name=Object2 Object-Id=0HL4XyFYH5DwMf
Object-Name=Object1 Object-Id=0HL4XyFYH5DwMn
Message=[c<20]msg1 (TO-Object-Id= y0L4XyFYH5DwMf & From-Object-Id= 0HL4XyFYH5DwMn)
Message=[c<50]msg2 (TO-Object-Id= #HL4XyFYH5DwM0 & From-Object-Id= y0L4XyFYH5DwMf)
Message=[c<50]msg3 (TO-Object-Id= 0DL4XyFYH5DwM7 & From-Object-Id= #HL4XyFYH5DwM0)
Message=[c<7<=0]msg4 (TO-Object-Id= y0L4XyFYH5DwMf & From-Object-Id= 0DL4XyFYH5DwM7)
Message=[c<120]msg5 (TO-Object-Id= 4pL4XyFYH5DwNC & From-Object-Id= y0L4XyFYH5DwMf)
Message=[c<40]msg6 (TO-Object-Id= 5pL4XyFYH5DwNJ & From-Object-Id= 4pL4XyFYH5DwNC)
Message=[c<20]msg7 (TO-Object-Id= 4pL4XyFYH5DwNC & From-Object-Id= 5pL4XyFYH5DwNJ)
Message=[c<100]msg8 (TO-Object-Id= 0DL4XyFYH5DwM7 & From-Object-Id= 4pL4XyFYH5DwNC)
Message=[c<90]msg9 (TO-Object-Id= #HL4XyFYH5DwM0 & From-Object-Id= 0DL4XyFYH5DwM7)
```

Figure 4: Output-file generated by parser

4. Passing file obtained from step 3 and slicing criteria to a .java program (which act as slicer) for getting the relative/required chunk of information in a separate .txt file.

4.1. Generated .txt file in step 3 as input taken by the Slicer.

4.2. Slicer will ask user to tell about the slicing criteria at run time to generate the slice as per requirements.

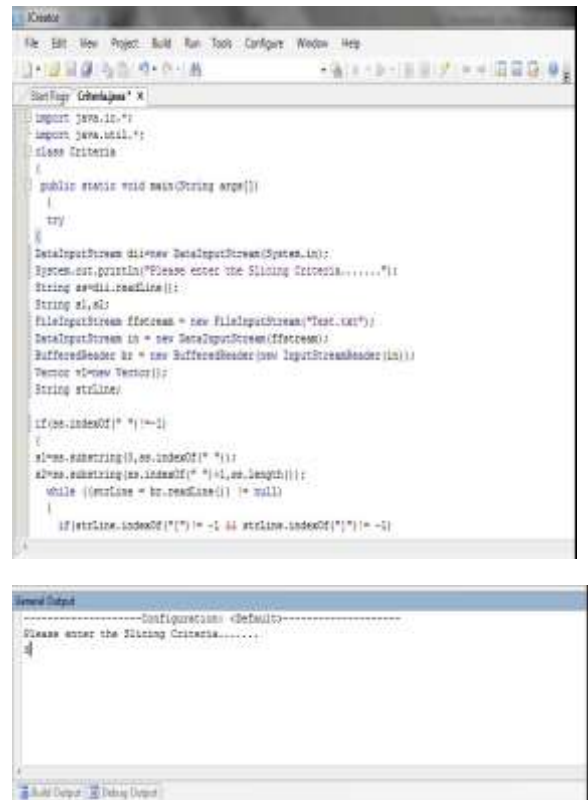


Figure 5: Java program for finding out the specified chunk

4.3. Computed slices will be store in separate .txt file which holds the information of messages, their guard condition and objects id's among which messages are being passed.

5. Changing object id with relative object name among which message is passing so that information can be retrieved easily (this step will only deal with sliced part).

5.1. To ease the retrieved of information objects id's will replaced by their corresponding object name (in the file retrieved from step 4.3).

5.2. All the information will store in separate .txt file which holds the information of messages and the objects name (among which they are communicating relative to user defined slicing criteria).

```

Message=[z<50]msg3 (TO-Object= Object4 & From-Object= Object3)
Message=[x+y+z=0]msg4 (TO-Object= Object2 & From-Object= Object4)
Message=[z=90]msg9 (TO-Object= Object3 & From-Object= Object4)
Message=[z+p>200]msg10 (TO-Object= Object5 & From-Object= Object3)
Message=[x+z>100]msg16 (TO-Object= Object7 & From-Object= Object2)
Message=[x+z<100]msg17 (TO-Object= Object8 & From-Object= Object2)
Message=[x=40 and z=40]msg20 (TO-Object= Object9 & From-Object= Object8)
Message=[y=50 and z=40]msg21 (TO-Object= Object10 & From-Object= Object9)
Message=[z=0]msg22 (TO-Object= Object9 & From-Object= Object10)
Message=[y*z>0]msg23 (TO-Object= Object7 & From-Object= Object9)
Message=[z-p>0]msg24 (TO-Object= Object5 & From-Object= Object7)
Message=[z>100]msg25 (TO-Object= Object6 & From-Object= Object5)
Message=[z-q>0]msg26 (TO-Object= Object4 & From-Object= Object6)
    
```

Figure 6: computed slice after the conversion of object-id to object-name

6. Passing txt file as obtained from step 5, to a .java program so that it can be converted into input file format for Quick Sequence Diagram Editor.

```

< Object4>:<int>[v]
< Object3>:<int>[v]
< Object2>:<int>[v]
< Object5>:<int>[v]
< Object7>:<int>[v]
< Object8>:<int>[v]
< Object9>:<int>[v]
< Object10>:<int>[v]
< Object6>:<int>[v]

< Object3>< Object4>.<[z<50]msg3 >
< Object4>< Object2>.<[x+y+z=0]msg4 >
< Object4>< Object3>.<[z=90]msg9 >
< Object3>< Object5>.<[z+p>200]msg10 >
< Object2>< Object7>.<[x+z>100]msg16 >
< Object2>< Object8>.<[x+z<100]msg17 >
< Object8>< Object9>.<[x=40 and z=40]msg20 >
< Object9>< Object10>.<[y=50 and z=40]msg21 >
< Object10>< Object9>.<[z=0]msg22 >
< Object9>< Object7>.<[y*z>0]msg23 >
< Object7>< Object5>.<[z-p>0]msg24 >
< Object5>< Object6>.<[z>100]msg25 >
< Object6>< Object4>.<[z-q>0]msg26 >
    
```

Figure 7: Input file for quick sequence diagram editor

7. Tool will generate the final and relatively small sequence diagram.

7.1. Tool will take the input format defined at step 6 as input to convert into its equivalent diagram.

7.2. Refined slice (small sequence diagram) will be generated as final output according to slicing criteria as per requirement to ease the software visualization.

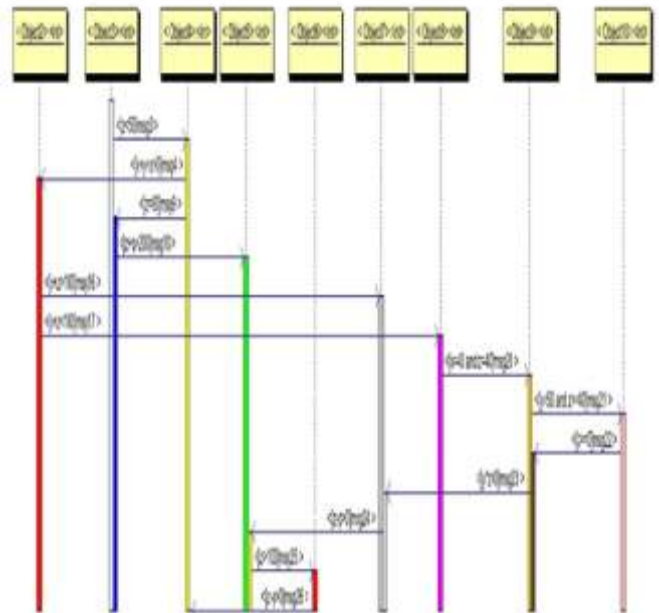


Figure 8: Computed Sliced Sequence diagram

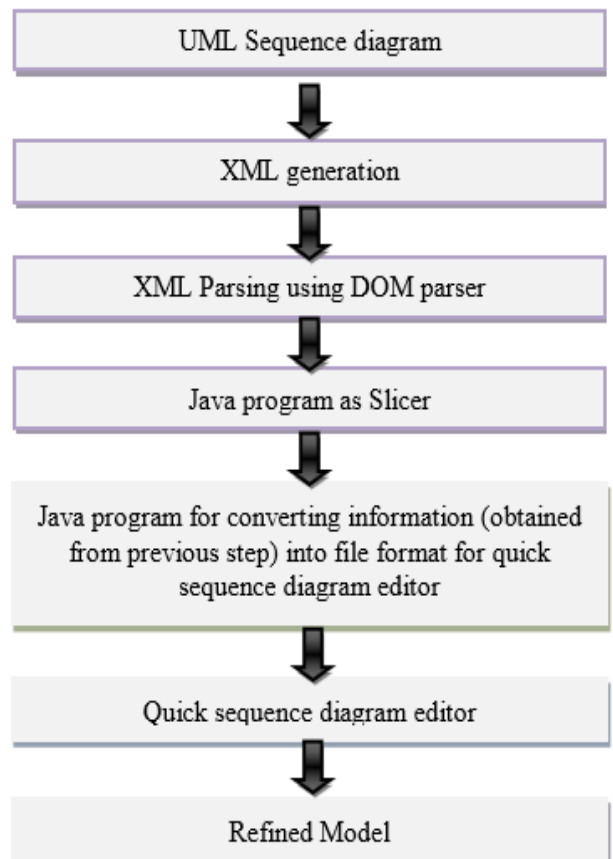


Figure 9: Overview of Methodology

IV CONCLUSION

The key contribution of the technique is to generate the refined model slices related to slicing criteria using

conditional predicate in sequence diagram. Practical implementation of technique that will extract the sub-model from architecture of software to ease the software visualization has been discussed. The foundation of the proposed technique is ‘UML’ and ‘Slicing’. With this, the problem of visualization of large and complex software can be handled efficiently. The proposed technique has focused on the generation of chunk using model based slicing but still there are the few points that can be explored further like model reduction in concurrent and distributed programming.

- [14] P. Samuel, R. Mall, and S. Sahoo, “UML Sequence Diagram Based Testing Using Slicing,” IEEE Indicon 2005 Conference, pages 176–178, IEEE, 2005.
- [15] R. V. Binder, “Testing object-oriented software: a survey,” Software Testing Verification and Reliability, vol. 6(3/4), pp: 125 – 252, 1996.

REFERENCES

- [1] Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide,” 2nd Edition, May 2005, Publisher. Addison Wesley.
- [2] Jianjun Zhao, “Slicing Software Architecture,” Technical Report 97-SE-117, pp.85-92, Information Processing Society of Japan, Nov 1997.
- [3] Rupinder Singh and Vinay Arora, “Literature Analysis on Model based Slicing,” International Journal of Computer Applications, vol. 70(16), pp: 45-51, May 2013. Published by Foundation of Computer Science, New York, USA.
- [4] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, and L. Tratt. Control dependence for extended finite state machines. Fundamental Approaches to Software Engineering, pp. 216–230, 2009.
- [5] H. Kagdi, J.I. Maletic, and A. Sutton, “Context-Free Slicing of UML Class Models,” Proc. 21st IEEE Int’l Conf. Software Maintenance, pp. 635-638, 2005.
- [6] J.H. Bae, K.M. Lee, and H.S. Chae. Modularization of the UML metamodel using model slicing. In Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on, pages 1253–1254. IEEE, 2008.
- [7] A. Shaikh, R. Clarisó, U.K. Wil, and N. Memon, “Verification-driven slicing of UML/OCL models,” In Proceedings of the IEEE/ACM international conference on Automated software engineering, pages 185–194. ACM, 2010.
- [8] Kevin Lano Crest, “Slicing of UML State Machines,” Proceedings of the 9th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '09), 2009.
- [9] V. Ojala, “A slicer for UML state machines,” Helsinki University of Technology, 2007.
- [10] S. Van Langenhove, “Towards the Correctness of Software Behavior in UML: A Model Checking Approach Based on Slicing,” Dissertation, Department of Mathematics, Ghent University, 2006.
- [11] J.T. Lallchandani and R. Mall, “Slicing UML architectural models,” ACM SIGSOFT Software Engineering Notes, vol.33, no.3, pp. 1–9, 2008.
- [12] J.T. Lallchandani and R. Mall, “Integrated state-based dynamic slicing technique for UML models,” Software, IET, vol. 4, no. 1, pp. 55–78, 2010.
- [13] P. Samuel and R. Mall. A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams. e-Informatica Software Engineering Journal Selected full texts, vol. 2, no. 1, pp. 61–77, 2008.