Volume 3, Issue 6, June 2013 ISSN: 2277 128X
**International Journal of Advanced Research in Computer Science and Software Engineering**
Research Paper
Available online at: www.ijarcsse.com

# Technique for Extracting Subpart from UML Sequence Diagram

**Rupinder Singh*, Vinay Arora**
*CSED Thapar University Patiala*
*Punjab, India*

***Abstract— Software visualization is one of the promising technique aimed at helping developers and testers to understand the behaviour of object-oriented systems effectively. However, it is still difficult to understand this behaviour, because the size of automatically generated model diagrams tends to be beyond the analyze capacity. In this paper, a novel methodology to ease the problem of software visualization has been proposed, that uses the concept of slicing UML diagrams and model based slicing technique.***

***Keywords— Slicing, UML, Software visualization, Sequence diagram, Model based slicing***

## I. INTRODUCTION

In recent years, due to the increase in size and complexity of software products the importance of architectural design has been increased. The architecture of an object-oriented software system defines its high level design structure and allows an architect to reason about various properties of the system at higher level of abstraction. For this, Unified Modeling Language (UML) is the best option and widely used to represent and construct the architecture of software system with the help of its various model diagrams. UML diagrams describe structural and behavioral aspects of architecture [1, 2]. Structural models (e.g., class diagrams, object diagrams, component diagrams) are used to describe various relations among objects, such as aggregation, association, composition and generalization/specialization etc. On the other hand, the behavioral models (e.g. communication and sequence diagrams, activity diagram, state diagrams) are used to describe a sequence of actions, states and their interaction, through which a use case is realized. The task of analyzing UML Models is bit challenging since the information regarding system can be distributed across several model views. To overcome this problem the concept of model based slicing came to existence. Model Based slicing is a decomposition technique to extract and identify relevant model parts (or fragments) or related elements across diverse model views. It takes the user defined slicing criteria as input and slices the architecture, as a view of interest [3]. Slicing is useful in software maintenance, reengineering, testing, decomposition and integration, decompilation, program comprehension, and debugging. The goal of software testing is to ensure quality. Software testing is necessary to produce highly reliable systems, since static verification techniques suffer from several hurdles in detecting all software faults [16]. The most intellectually challenging part of testing is the design of test cases. Test cases are usually generated based on program source code. An alternative approach is to generate test cases from specifications developed using formalisms such as UML models. Slicing is best technique which can decompose the structure into submodels without affecting their core structure and functionality. It helps the developer to gain the perfect view of software according to their requirement.

## II. LITERATURE ANALYSIS ON MODEL BASED SLICING

In early stages of development, Model based slicing has been applied to state machines [4] where similar benefits as those listed above has been claimed. State machine slicing is an example of applying slicing to a model of a system rather than to the system implementation. However, system models such as those represented in terms of the UML-family of languages are much more complex than state machines (and contain state machine sub-languages). Several approaches and attempts have been made for slicing UML diagrams. The approach in [5] describe context-free slicing of UML class models where the issue of context has been defined to be object location, which is a dynamic property of the scenario therefore context free slicing is a static slice of a structural model. As noted in [5] the criteria used for slicing a model is more complex than that used in program or state machine slicing since there are more types of elements and relationships. This has been noted that 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 collections of components that are relevant to the different UML diagram types in [6] although the predicate used to determine the slicing criteria has been fixed in terms of traversing the meta-model elements starting with a collection of supplied classes. Class models has sliced in conjunction with OCL invariants in [7] thereby reducing the state-space explosion that would otherwise occur when using a model-checker (in this case Alloy) to verify a class-model. UML state-charts can be sliced as described in [8] [9] [10] although these approaches do not generalize 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 subsequently sliced to show partial dynamic and structural information resulting from criteria containing both structural and dynamic constraints. Slicing UML sequence diagrams in order to generate test cases has been described in [13] [15]. UML sequence diagrams (or scenarios) are essentially an integral part of executions of a program. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. It has been thoroughly analyzed that for the process 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. Figure 1 shows the generic view of functional behaviour of software models in the form of sequence diagram. Rectangular box specifies the objects within model diagram that are interacting with each other. Doted lines denote the life line of the objects on which instances of the objects has been created. Arrow represents the particular action (in the form of messages) of objects and their direction. Where round brackets specify the guard condition that specifies the truth and false value to interact. Sign of cross specifies the end of life line where all the instances of the objects will destroy after the completion of their respective action.

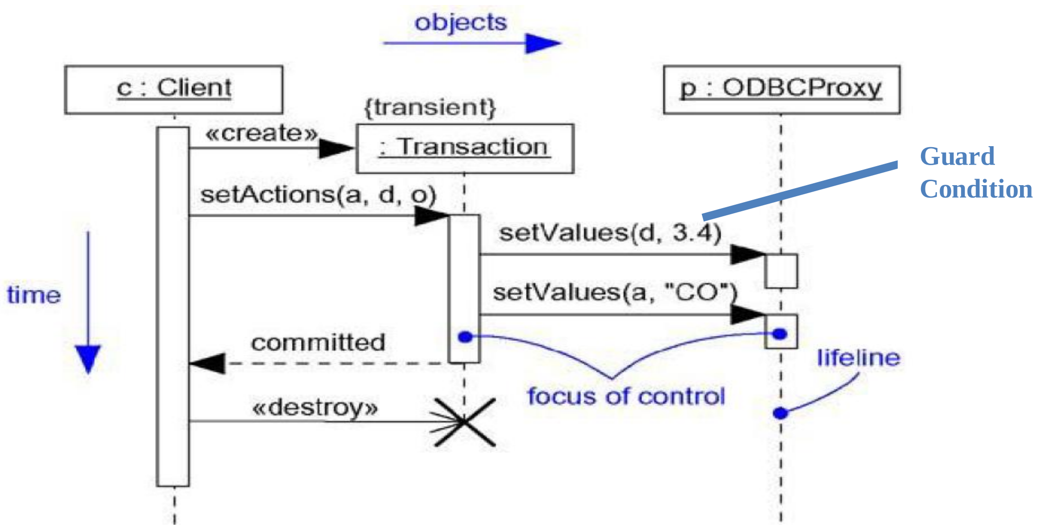

Figure 1: Generic view of Sequence diagram [1]

## III. METHODOLOGY

To extract subpart from UML Sequence diagram following technique has been proposed:

1. Generation of UML (Sequence) diagram of/from a particular requirement specification.

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

2. Create XML from the specified UML diagram (Sequence diagram).

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

3. Document Object Model (DOM) parser for parsing XML code and generating 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 DocumentBuilderFactory ( ) 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.

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. Slicer will take .txt file generated in step 3 as input.

4.2. Slicer will ask user to define the slicing criteria at run time to generate the chunk/slice as per specified requirements.

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).

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*.

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.

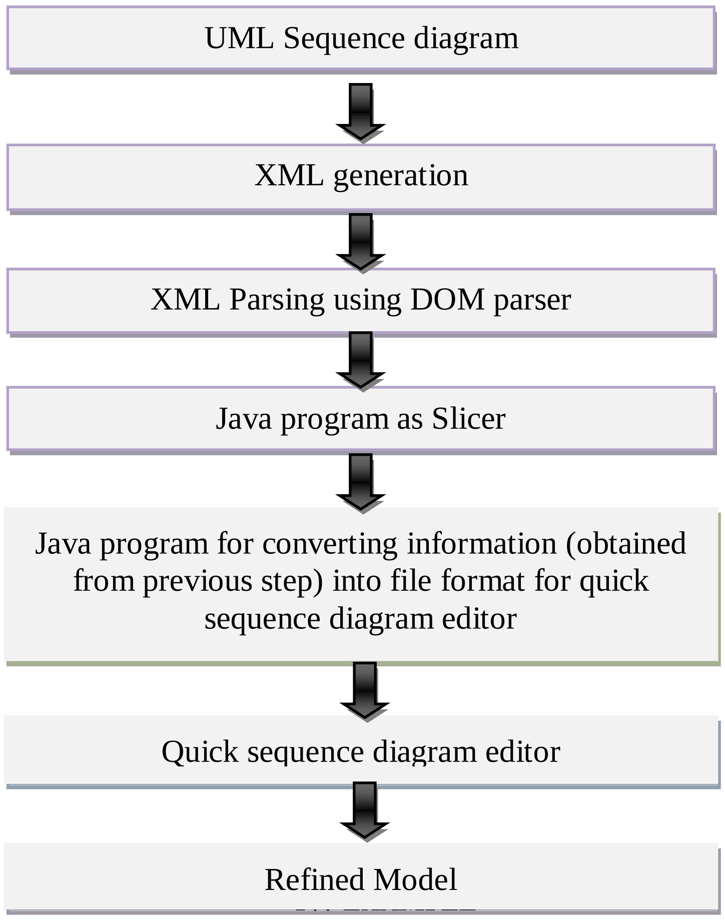
UML Sequence diagram
XML generation
XML Parsing using DOM parser
Java program as Slicer
Java program for converting information (obtained from previous step) into file format for quick sequence diagram editor
Quick sequence diagram editor
Refined Model

Figure 2: Overview of Methodology

Consider an example UML sequence diagram as shown in figure: 3. the purpose of selecting this example is to demonstrate the concept of proposed methodology. In the example there are four objects which are interacting with each other thorough message passing (using guard condition as true to interact with each other). We illustrate our methodology by explaining the generation of chunk or refined model diagram as shown in figure 4 with respect to slicing criteria. Here in this example let the slicing criterion is variable ‘c’. Given criteria is a variable used in conditional predicate of message guard condition. True and false value of these guard conditions are used by objects to interact with each other. According to user defined slicing criteria, proposed methodology will slice the model diagram shown in figure 3 and generate the resultant small chunk or refined sequence diagram as shown in figure 4.

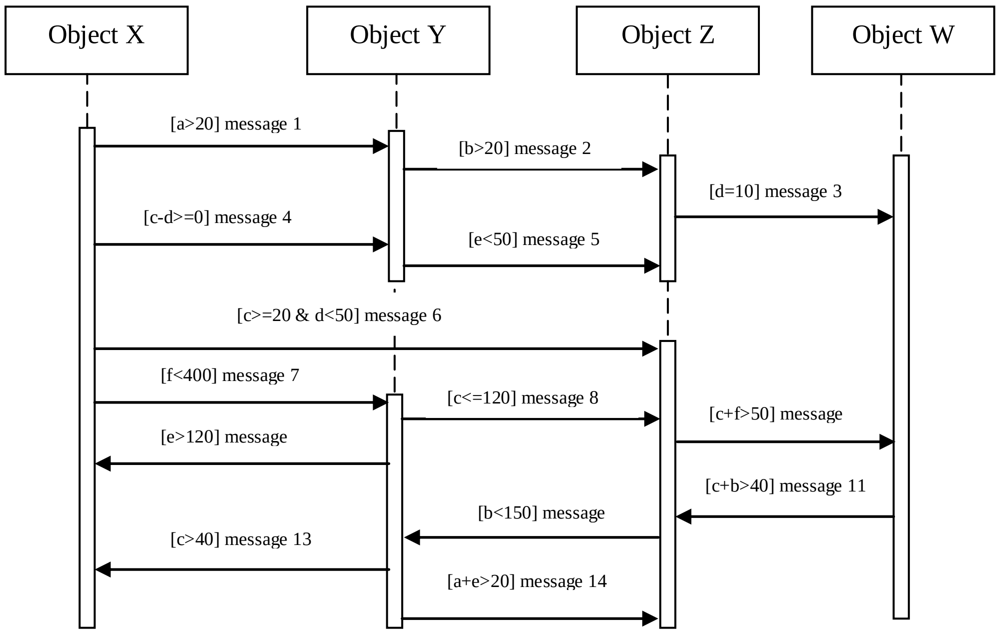

Figure 3: Example Sequence diagram

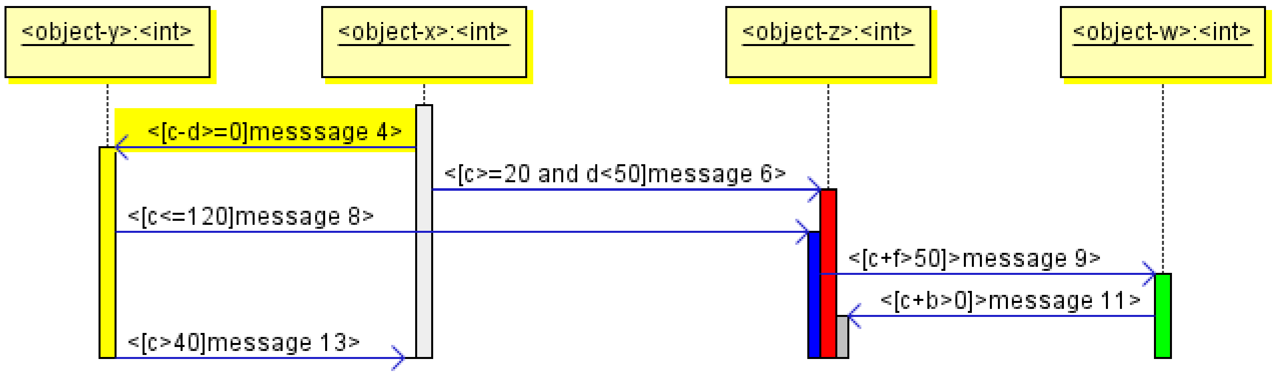

Figure 4: Refined Sequence diagram

## V. Conclusion

A new technique for model based slicing has been proposed that will extract the sub-model from architecture of the software to ease the software visualization. The key contribution of the technique is to generate the refined model slices related to user defined slicing criteria using conditional predicate in sequence diagram. Sequence diagrams capture time dependent (temporal) sequences of interactions between objects. The foundation of the proposed technique is 'UML' and 'Slicing'. With this, the problem of visualization of large and complex software can be sorted out. This technique can further be enhanced and applied in the domain of concurrent programming.

## Refrences

**[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. Wiil, 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.

[14] J. Qian and B. Xu, "Program slicing under UML scenario models," ACM SIGPLAN NOTICES, vol. 43, no. 2, 2008.

[15] P. Samuel, R. Mall, and S. Sahoo, "UML Sequence Diagram Based Testing Using Slicing," IEEE Indicon 2005 Conference, pages 176–178, IEEE, 2005.

[16] R. V. Binder, "Testing object-oriented software: a survey," Software Testing Verification and Reliability, vol. 6(3/4), pp: 125 – 252, 1996.