IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 12, Issue 4 (Jul. - Aug. 2013), PP 18-26
www.iosrjournals.org

A practical approach for model based slicing

Rupinder Singh[1], Vinay Arora[2]
[1]*(CSED, Thapar University, Patiala, India)*
[2]*(CSED, Thapar University, Patiala, India)*

Abstract : *Software testing is an activity that will aim at evaluating an attribute or capability of system and determine that whether it meets required expectations. Test cases can be designed at model level of software development. But to visualize the software model or architecture is difficult due to its large and complex structure. We have presented a novel methodology to extract the sub-model from model diagrams correspond to point of interest to ease the software visualization. The proposed methodology use the concept of model based slicing to slice the sequence diagram to extract the desired chunk.*
Keywords – *Software Testing, Sequence diagram, Model based slicing.*

Submitted date 17 June 2013 **Accepted Date: 22 June 2013**

I. INTRODUCTION

Software testing is an evaluation process to determine the presence of errors in code snippet. Software testing cannot completely test software because exhaustive testing is not possible due to time and resource constraints. According to *ANSI/IEEE* 1059 standard [1, 2], Testing can be defined as "A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item". The prime objective of testing is to discover faults that are preventing the software in meeting customer requirements. Moreover, testing requires planning and designing of test cases. The testing process progresses from component level to system level in an incremental way. The most intellectually challenging part of testing is the design of test cases. Test case generation from design specifications has the added advantage of allowing test cases to be available early in the software development cycle. Now days, UML has been widely used for object oriented modeling and design. This is due to the fact that a UML metamodel extends support to describe structural and behavioral aspects of an architecture. For instance, the structural models (e.g. class diagrams) are used to describe various relations among classes, such as aggregation, association, composition, and generalization/specialization. On the other hand, the behavioral models (e.g., communication and sequence diagrams) are used to depict a sequence of actions in an interaction that describe how the objects are interacting to perform their respective action [3]. But due to large and complex architecture of software products it's hard to visualize and test the software. To overcome this problem of visualization large models, 'Model based slicing' technique came into existence. Model Based slicing is a decomposition technique to extract and identify relevant model parts (or fragments) or related elements across diverse model views. Slicing can be referring as process or strategy to break body of information into smaller parts to examine it from different viewpoints that can yield more precise information so that one can understand it better [35]. The term is also used to mean the presentation of information in a variety of different and useful ways to ease the visualization. But in term of automation of software testing, the direct use of UML diagrams is not feasible. Conversion of UML into XML is one of the integral part of model based slicing for automation of whole technique. The core principle of model based slicing technique is to 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. RELATED WORK

In Model based slicing several types of model relations and dependency such as class-class, class-operation, operation-operation, class-object, object-object, guard condition in sequence diagram , conditional predicate, control flow , data flow etc., need to be taken into account as discussed in previous paper [34]. In this work, sequence diagram has been taken into account and various approaches present till date for slicing UML diagram have been listed. Zhao [4] introduced the concept of architectural slicing which operates on architectural description of software system. As an extension of his previous work Zhao [5] introduced Architectural Information Flow Graph with three types of information flow arcs: Component-connector, Connector-component, internal flow arcs to apply the slicing technique on software architecture precisely. Wang et.al [6] presented a method for slicing hierarchical automata. The importance of Wang's algorithm is its ability to remove the hierarchies and concurrent states, which are irrelevant to the properties of the hierarchical automata. Kagdi et al. [7] proposed the concept of model slicing to support maintenance of software through the understanding, querying and analyzing large UML models. Langehove [8] presents an algorithm for reducing the number of interference dependencies in state charts by using the concept of slicing with concurrent states. The proposed approach considers data dependencies from the definition and use of variables that are common to parallel executing statements. Lallchandani et al. [9] propose a technique for constructing dynamic slices of UML models using the integrated state-based information. In order to achieve this they proposed an algorithm for Architectural Model Slicing through MDG Traversal (AMSMT). By using the same algorithm (AMSMT) researchers had implemented a prototype architectural slicing tool called SSUAM [10] to generate static slices for UML Architecture models. Later on, in another approach [11] they proposed a DSUAM algorithm which uses the MDG representation to compute dynamic slices.

Samuel and Mall [12] presented a scheme to generate slice and test cases with the help of edge marking dynamic slicing algorithm for activity diagrams. They used the flow dependency graph (FDG) which shows the dependencies among activities that arise during run time. Noda et.al [13] proposed a sequence diagram slicing method to visualize the object oriented program's behavior. In order to achieve this, a tool has been proposed that named as ‘Reticella’ which is implemented as eclipse plug-in. The proposed tool take java program as input and after analyzing, fetch the static information and draw B-model tree.

Swain et.al [14] proposed an approach to generate test cases from UML interaction diagram by using the condition slicing. In their approach they identify the message guard condition from interaction diagram and use the condition slicing to generate test cases. J. Kim et.al [15] proposed an approach to address the hierarchy and orthogonality problems while tracing the data dependency in slicing of UML State machine diagram. Yatapanage et al. [16] focused their work on Model Checking as fully automated technique to reduce the size of model with the help of slicing. They used Behavior Tree dependency graph (BTDG) to capture all functional requirements and dependency between components and attributes.

Korel et.al [17] dedicated their work on slicing the state based models, such as EFSMs (Extended Finite State Machines). As a result two types of slicing came to existence—deterministic and nondeterministic slicing. Lano [18] defined that slicing can be carried out for UML state machines, using data and control flow analysis by factoring the model on the basis of features. Archer et al. [19] proposed a novel slicing technique on the feature model by taking cross-tree constraints into account with respect to set of features which are acting as slicing criteria. Julliand et.al [20] proposed an approach based on domain abstraction for generating test cases on the basis of syntactic abstraction and variable elimination with the help of model slicing.

Shaikh et.al [21] proposed a verification technique to check the correctness of model with the help of slicing. The proposed technique increases the scalability of verification by partitioning the original model into sub model. Kim [22][23] introduced the slicing technique called dynamic software architecture slicing (DSAS) using ADL (Architecture description language). Kim's work is very efficient there because it’s able to generate a smaller number of components and connectors in each slice according to slicing criteria. Lano et al. [24] defined the technique for slicing of UML model using Model Transformation, particular for restriction of model to those parts which specifies the properties of subset within. Zoltán et.al [25][26] proposed dynamic backward slicing of model transformations technique with respect to program slicing. The proposed technique take three inputs, the model transformation program, the model on which the MT program operates and the slicing criterion and generate the output as transformation slices and model slices. Blouin et.al [27] [28] proposed a DSML (Domain Specific Model Language) 'Kompren' to model the model slicers for particular domain. Kompren refers to the selection of the set of classes and relations from the input metamodel expressed using an object-oriented meta-language. Table 1 depicts the list of model based slicing tools extracted from literature analysis. Table 2 shows the comparison of different approaches of model based slicing.

Table 1: Tools for model based slicing

Year	Tool Name	Technique Used
2003	EFSM Slicing Tool	Control and Data flow analysis.
2007	UTG	Data Flow and Control Flow dependency, Communication Tree.
2008	SSUAM	Model Dependency Graph.
2008	UML Slicer	MetaModel Diagram, Key Elements.
2009	Reticella	B-Model dependency Graph.
2011	Archlice	Model Dependency Graph.
2011	Safe Slicer	System Model Language, Traceability Links and Rules.
2012	UOST	UML + OCL Constraints.

Table 2: Approaches/Techniques/Tools for model based slicing

Approach/Technique	**Slicing Process**		**Model**	**Usage**
	Syntactic	Semantic		
Feature model slicing [19, 29]	✓	✓	Feature model	Separation of concerns
UML slicer [30]	✓	×	UML metamodel	Modularization
UML statechart [18, 31]	✓	✓	State charts	Reactive system, Model checking
Safe slicer [32]	✓	×	System Models	Safety
Domain specific model language [27, 28]	✓	✓	UML metamodel	Dynamic model slicing
Context free UML slicing [18]	✓	×	UML class diagram	Sub model extraction
Dynamic software architecture slicing [22, 23]	✓	✓	Architecture description language	Architectural slicing
EFSM slicing [17]	✓	×	State based models	Size reduction
DSUAM [11]	✓	✓	UML	Separation of concerns
UML activity diagram [12]	✓	✓	Activity diagram	Test case generation
UML/OCL slicing [21, 33]	✓	✓	UML/OCL	Verification
Model transformation [24, 25, 26]	✓	✓	Graph and UML diagram	Program slicing
Behavior tree slicing [16]	✓	✓	Behavior tree	Model checking

III. Preliminary Work

We had proposed a method for extracting subpart from UML sequence diagram, based on conditional predicate in our previous work titled "*Technique for extracting subpart from UML Sequence diagram"*. This paper provides the detailed view on the practical implementation of the already proposed technique, for finding the chunk from a given sequence diagram.

IV. Proposed Methodology and Implementation

After reviewing the literature of software testing techniques, slicing techniques, software visualization and unified modeling language, it has been analyzed that slicing UML diagrams is one of the major area in which work can be extended for various constructs like sequence diagram, state transition diagram, activity diagram, class diagram, etc. 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.

Consider an example UML sequence diagram as shown in figure: 1. 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 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 1 and generate the resultant small chunk or refined sequence diagram.

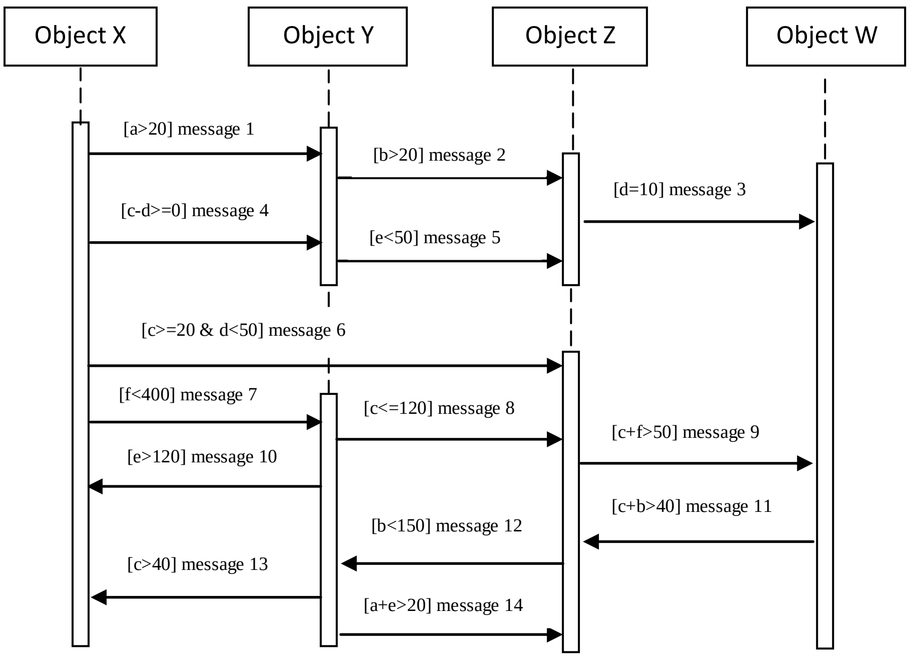


Fig 1: Example Sequence diagram

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, can be used to generate the UML diagram for specific requirement specifications. Figure 2 shows the designing of UML sequence diagram using visual paradigm tool.

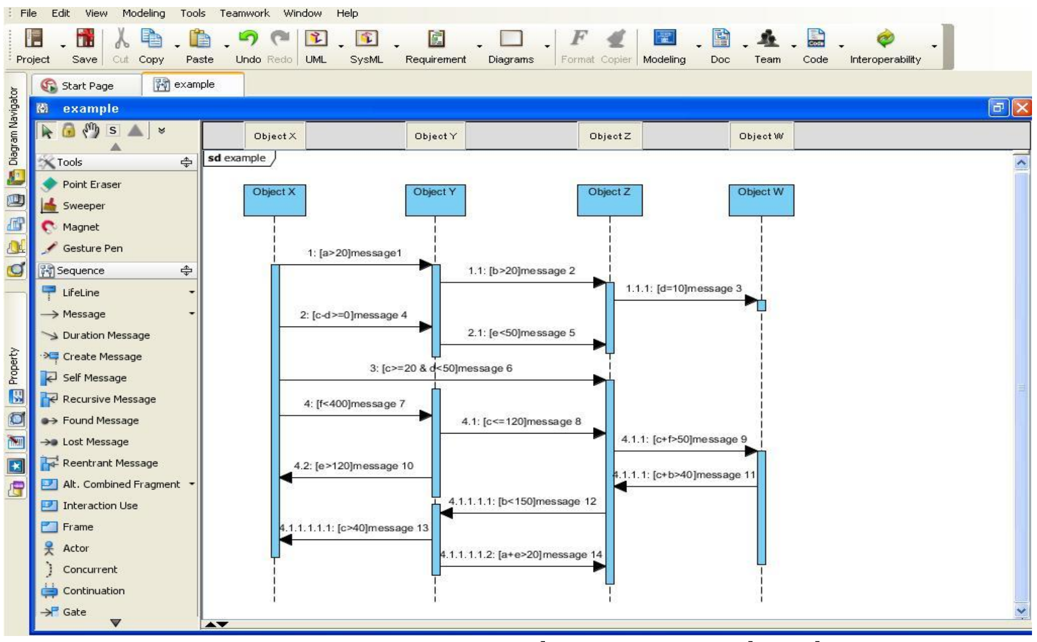
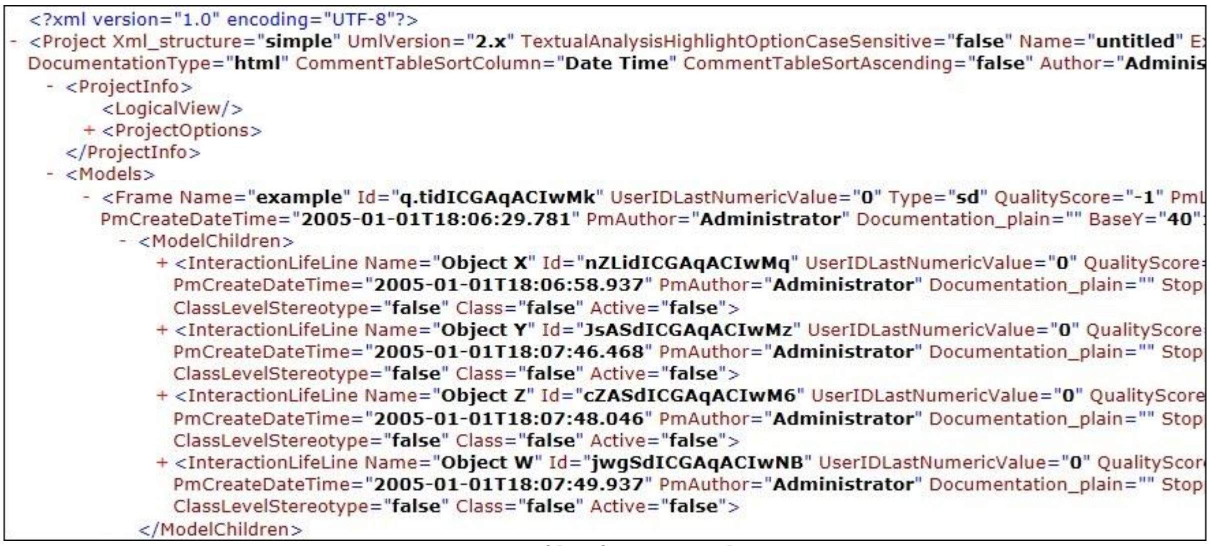


Fig 2: Designing sequence diagram using visual paradigm

II. Create XML from the specified UML Sequence diagram.

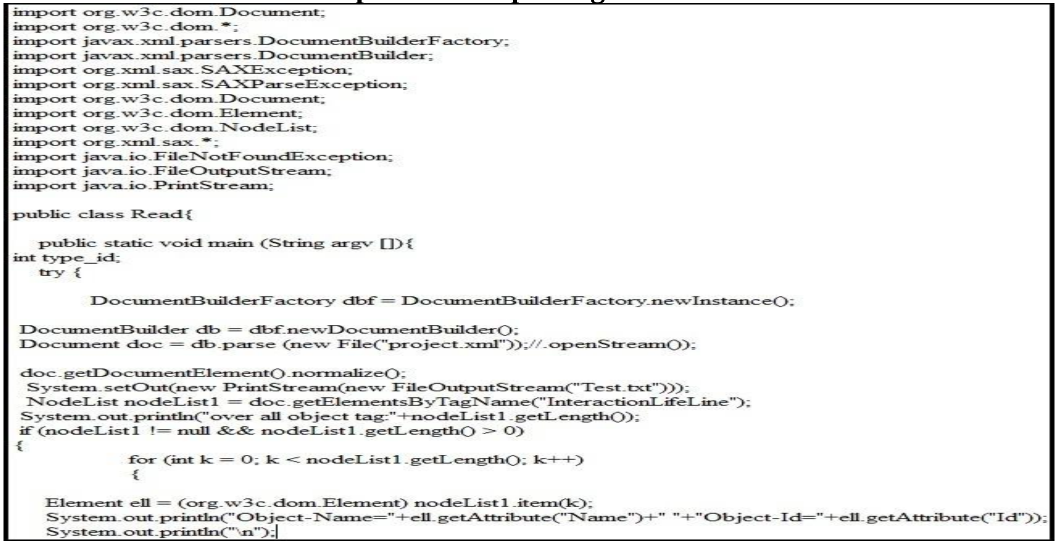
2.1. Visual paradigm for UML 10.0 version provides the in-built functionality to export the diagrams into XML format.



```
  <?xml version="1.0" encoding="UTF-8"?>
- <Project Xml_structure="simple" UmlVersion="2.x" TextualAnalysisHighlightOptionCaseSensitive="false" Name="untitled" E
  DocumentationType="html" CommentTableSortColumn="Date Time" CommentTableSortAscending="false" Author="Adminis
  - <ProjectInfo>
      <LogicalView/>
    + <ProjectOptions>
    </ProjectInfo>
  - <Models>
    - <Frame Name="example" Id="q.tidICGAqACIwMk" UserIDLastNumericValue="0" Type="sd" QualityScore="-1" PmL
      PmCreateDateTime="2005-01-01T18:06:29.781" PmAuthor="Administrator" Documentation_plain="" BaseY="40"
      - <ModelChildren>
        + <InteractionLifeLine Name="Object X" Id="nZLidICGAqACIwMq" UserIDLastNumericValue="0" QualityScore
          PmCreateDateTime="2005-01-01T18:06:58.937" PmAuthor="Administrator" Documentation_plain="" Stop
          ClassLevelStereotype="false" Class="false" Active="false">
        + <InteractionLifeLine Name="Object Y" Id="JsASdICGAqACIwMz" UserIDLastNumericValue="0" QualityScore
          PmCreateDateTime="2005-01-01T18:07:46.468" PmAuthor="Administrator" Documentation_plain="" Stop
          ClassLevelStereotype="false" Class="false" Active="false">
        + <InteractionLifeLine Name="Object Z" Id="cZASdICGAqACIwM6" UserIDLastNumericValue="0" QualityScore
          PmCreateDateTime="2005-01-01T18:07:48.046" PmAuthor="Administrator" Documentation_plain="" Stop
          ClassLevelStereotype="false" Class="false" Active="false">
        + <InteractionLifeLine Name="Object W" Id="jwgSdICGAqACIwNB" UserIDLastNumericValue="0" QualityScor
          PmCreateDateTime="2005-01-01T18:07:49.937" PmAuthor="Administrator" Documentation_plain="" Stop
          ClassLevelStereotype="false" Class="false" Active="false">
      </ModelChildren>
```

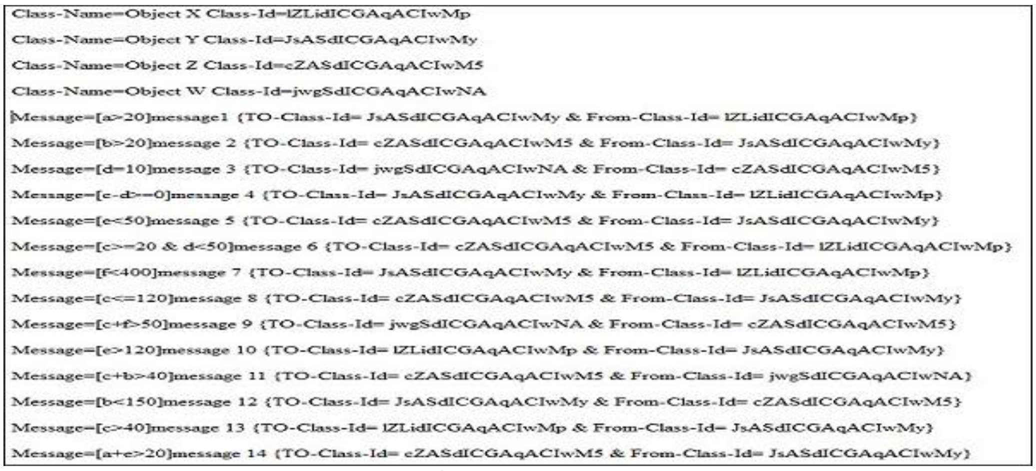
Fig 3: XML file of Sequence diagram

III. Document Object Model (DOM) parser parse the XML code and generate an output file (with .txt extension) having Object name, identifier, message name, message to & fro information. Figure 4 shows the DOM parser and Figure 5 depicts the output file generated by DOM parser after parsing the XML file.



```
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.*;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;

public class Read{

   public static void main (String argv []){
int type_id;
   try {

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

 DocumentBuilder db = dbf.newDocumentBuilder();
 Document doc = db.parse (new File("project.xml"));//.openStream());

 doc.getDocumentElement().normalize();
  System.setOut(new PrintStream(new FileOutputStream("Test.txt")));
  NodeList nodeList1 = doc.getElementsByTagName("InteractionLifeLine");
 System.out.println("over all object tag:"+nodeList1.getLength());
 if (nodeList1 != null && nodeList1.getLength() > 0)
 {
              for (int k = 0; k < nodeList1.getLength(); k++)
              {

    Element ell = (org.w3c.dom.Element) nodeList1.item(k);
    System.out.println("Object-Name="+ell.getAttribute("Name")+" "+"Object-Id="+ell.getAttribute("Id"));
    System.out.println("\n");
```

Fig 4: DOM parser



```
Class-Name=Object X Class-Id=lZLidICGAqACIwMp

Class-Name=Object Y Class-Id=JsASdICGAqACIwMy

Class-Name=Object Z Class-Id=cZASdICGAqACIwM5

Class-Name=Object W Class-Id=jwgSdICGAqACIwNA

Message=[a>20]message1 {TO-Class-Id= JsASdICGAqACIwMy & From-Class-Id= lZLidICGAqACIwMp}

Message=[b>20]message 2 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= JsASdICGAqACIwMy}

Message=[d=10]message 3 {TO-Class-Id= jwgSdICGAqACIwNA & From-Class-Id= cZASdICGAqACIwM5}

Message=[c-d>=0]message 4 {TO-Class-Id= JsASdICGAqACIwMy & From-Class-Id= lZLidICGAqACIwMp}

Message=[e<50]message 5 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= JsASdICGAqACIwMy}

Message=[c>=20 & d<50]message 6 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= lZLidICGAqACIwMp}

Message=[f<400]message 7 {TO-Class-Id= JsASdICGAqACIwMy & From-Class-Id= lZLidICGAqACIwMp}

Message=[c<=120]message 8 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= JsASdICGAqACIwMy}

Message=[c+f>50]message 9 {TO-Class-Id= jwgSdICGAqACIwNA & From-Class-Id= cZASdICGAqACIwM5}

Message=[e>120]message 10 {TO-Class-Id= lZLidICGAqACIwMp & From-Class-Id= JsASdICGAqACIwMy}

Message=[c+b>40]message 11 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= jwgSdICGAqACIwNA}

Message=[b<150]message 12 {TO-Class-Id= JsASdICGAqACIwMy & From-Class-Id= cZASdICGAqACIwM5}

Message=[c>40]message 13 {TO-Class-Id= lZLidICGAqACIwMp & From-Class-Id= JsASdICGAqACIwMy}

Message=[a+e>20]message 14 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= JsASdICGAqACIwMy}
```

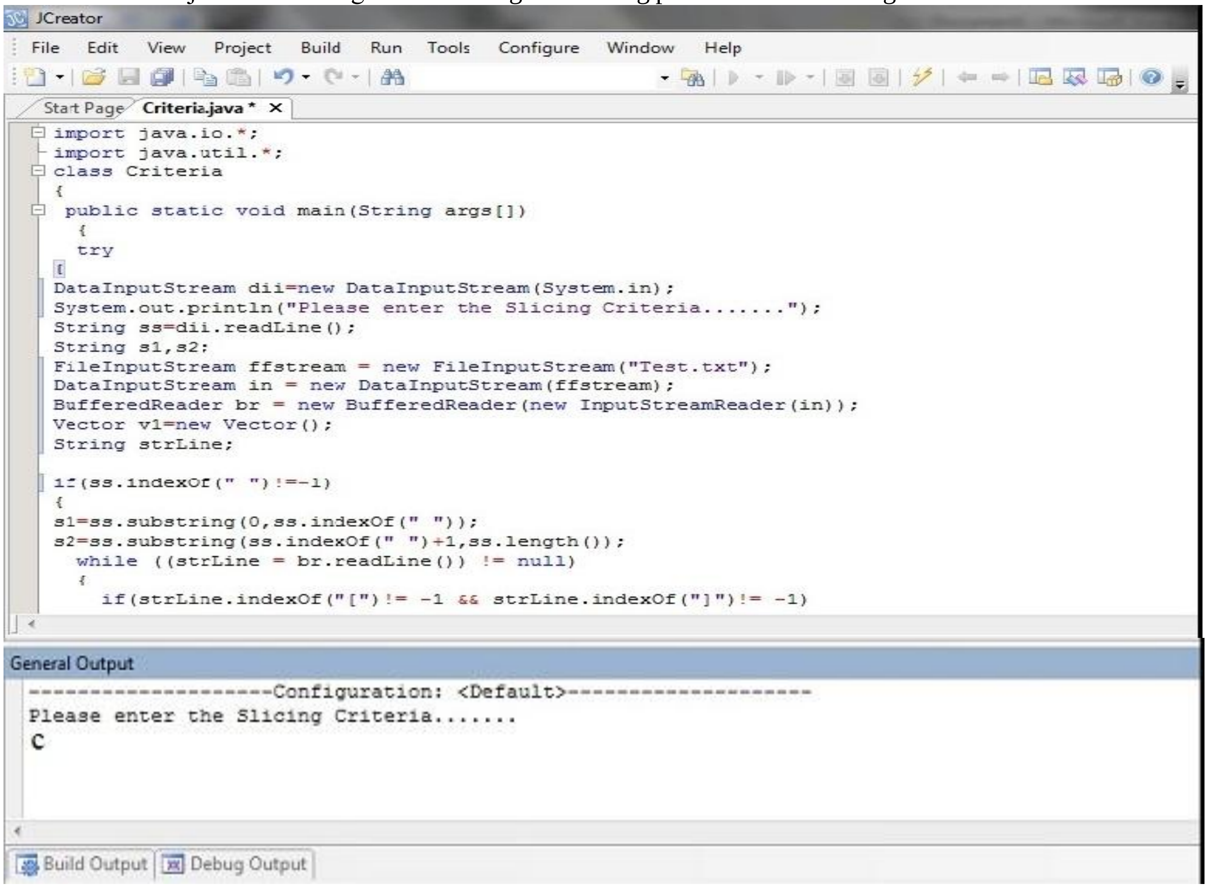
Fig 5: Output-file generated by DOM parser

IV. 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 as shown in figure 6. In this example user define the ‘**c**’ variable as slicing criteria.

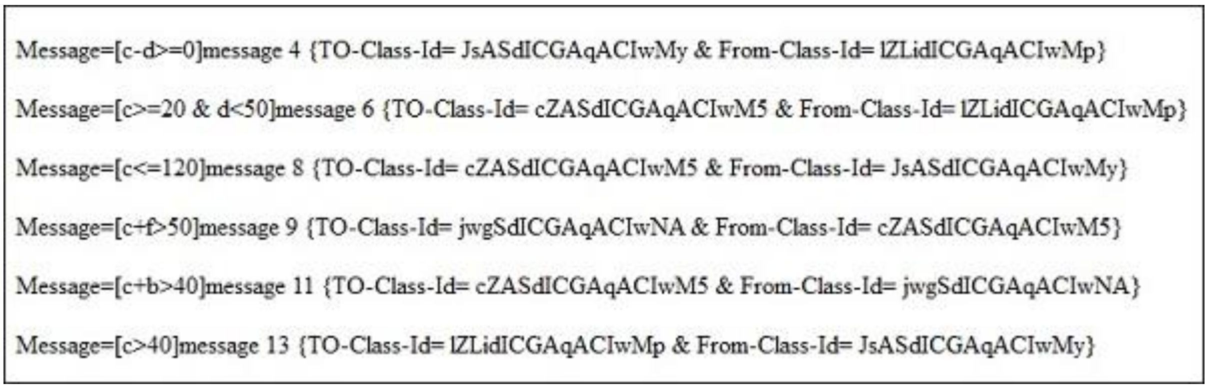
4.3. Computed slices will be store in separate .txt file which holds the information of messages, their guard condition and objects id’s among which messages are being passed as shown in figure 7.



```
import java.io.*;
import java.util.*;
class Criteria
{
 public static void main(String args[])
  {
  try
{
DataInputStream dii=new DataInputStream(System.in);
System.out.println("Please enter the Slicing Criteria.......");
String ss=dii.readLine();
String s1,s2;
FileInputStream ffstream = new FileInputStream("Test.txt");
DataInputStream in = new DataInputStream(ffstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
Vector v1=new Vector();
String strLine;

if(ss.indexOf(" ")!=-1)
{
s1=ss.substring(0,ss.indexOf(" "));
s2=ss.substring(ss.indexOf(" ")+1,ss.length());
  while ((strLine = br.readLine()) != null)
  {
    if(strLine.indexOf("[")!= -1 && strLine.indexOf("]")!= -1)
```

```
--------------------Configuration: <Default>--------------------
Please enter the Slicing Criteria.......
c
```

Fig 6: Java program for finding out the specified chunk



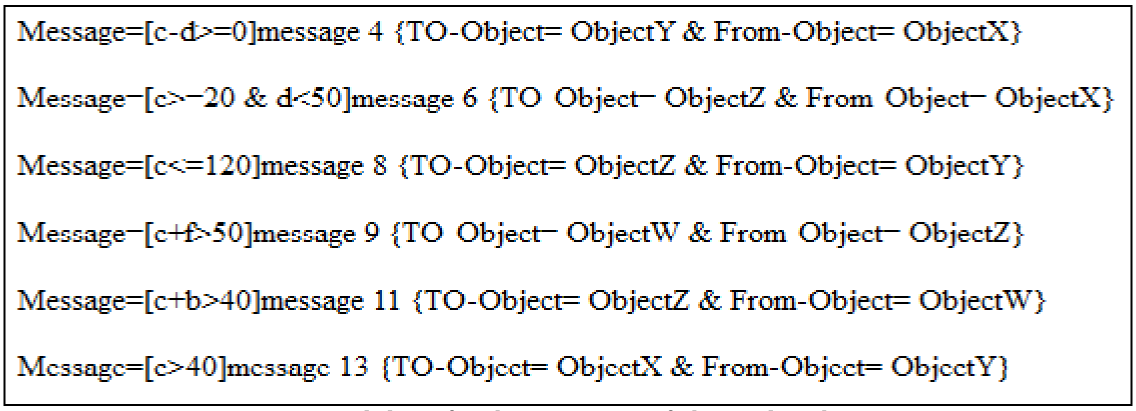
```
Message=[c-d>=0]message 4 {TO-Class-Id= JsASdICGAqACIwMy & From-Class-Id= lZLidICGAqACIwMp}

Message=[c>=20 & d<50]message 6 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= lZLidICGAqACIwMp}

Message=[c<=120]message 8 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= JsASdICGAqACIwMy}

Message=[c+f>50]message 9 {TO-Class-Id= jwgSdICGAqACIwNA & From-Class-Id= cZASdICGAqACIwM5}

Message=[c+b>40]message 11 {TO-Class-Id= cZASdICGAqACIwM5 & From-Class-Id= jwgSdICGAqACIwNA}

Message=[c>40]message 13 {TO-Class-Id= lZLidICGAqACIwMp & From-Class-Id= JsASdICGAqACIwMy}
```

Fig 7: output file generated after applying slicing

V. 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 retrieval of information, objects id’s will be replaced with their corresponding object name (in the file retrieved from step 4.3) (as shown in figure 8.)

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=[c-d>=0]message 4 {TO-Object= ObjectY & From-Object= ObjectX}

Message=[c>=20 & d<50]message 6 {TO-Object= ObjectZ & From-Object= ObjectX}

Message=[c<=120]message 8 {TO-Object= ObjectZ & From-Object= ObjectY}

Message=[c+f>50]message 9 {TO-Object= ObjectW & From-Object= ObjectZ}

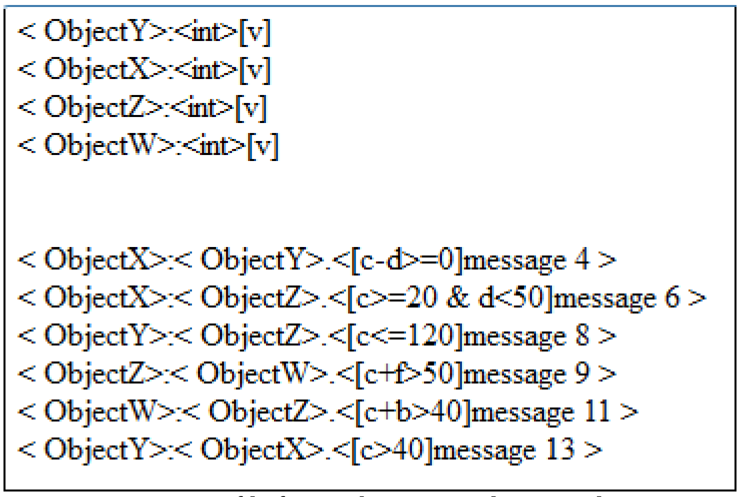
Message=[c+b>40]message 11 {TO-Object= ObjectZ & From-Object= ObjectW}

Message=[c>40]message 13 {TO-Object= ObjectX & From-Object= ObjectY}

Fig 8: computed slice after the conversion of object-id to object-name

VI. Passing txt file as obtained from step 5, to b .java program so that it can be converted into input file format

for 'Quick Sequence Diagram Editor' as shown in figure 9.



```
< ObjectY>:<int>[v]
< ObjectX>:<int>[v]
< ObjectZ>:<int>[v]
< ObjectW>:<int>[v]


< ObjectX>:< ObjectY>.<[c-d>=0]message 4 >
< ObjectX>:< ObjectZ>.<[c>=20 & d<50]message 6 >
< ObjectY>:< ObjectZ>.<[c<=120]message 8 >
< ObjectZ>:< ObjectW>.<[c+f>50]message 9 >
< ObjectW>:< ObjectZ>.<[c+b>40]message 11 >
< ObjectY>:< ObjectX>.<[c>40]message 13 >
```

Fig 9: Input file for quick sequence diagram editor

VII. 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 as shown in figure 10.

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.

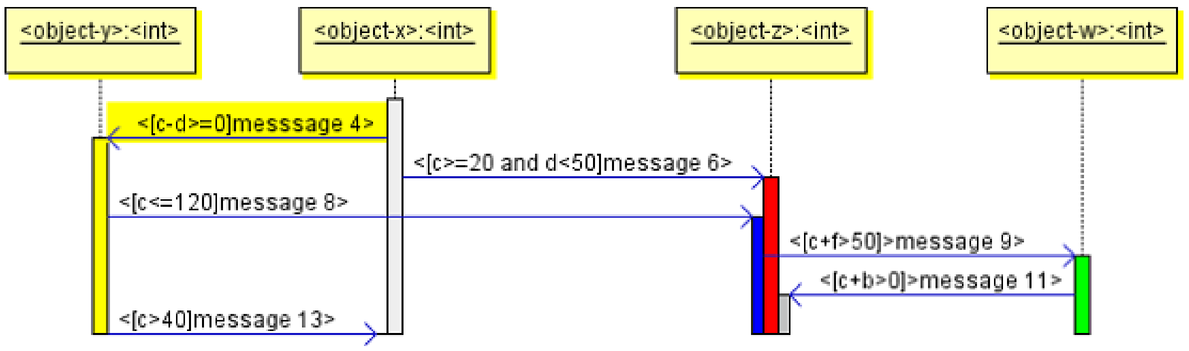


Fig 10: Computed sliced Sequence diagram

VIII. CONCLUSION

Practical implementation of technique that will extract the sub-model from architecture of software to ease the software visualization has been discussed. The key contribution of the technique is to generate the refined model slices related to slicing criteria using conditional predicate in sequence diagram. 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

REFERENCES

[1] IEEE Standard 1059-1993, IEEE Guide for Software Verification and Validation Plans, IEEE, 1993.

[2] IEEE Standard 829-1998, IEEE Standard for Software Test Documentation, IEEE, 1998.

[3] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, 2nd Edition, May 2005, Publisher. Addison Wesley.

[4] Jianjun Zhao, Slicing Software Architecture, Technical Report 97-SE-117, pp.85-92, Information Processing Society of Japan, Nov 1997.

[5] Jianjun Zhao, Applying slicing technique to software architectures, In Fourth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'98, pp 87 –98, 1998.

[6] J. Wang, Wei Dong, and Zhichang Qi, Slicing Hierarchical Automata for Model Checking UML Statechart,Proc. Fourth Int'l Conf. Formal Eng. Methods: Formal Methods and Software Eng., pp. 435-446, Oct. 2002.

[7] 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.

[8] S. Van Langehove, Internal Broadcasting to Slice UML State Charts: As Rich as Needed, Proc. Abstracts of the FNRS Contact Day: The Theory and Practice of Software Verification, Oct.2005.

[9] J. Lallchandani and R. Mall, Slicing UML Architectural Models, *ACM SIGSOFT, vol.33, no.3*, May 2008.

[10] Jaiprakash T. Lallchandani, R. Mall, Static Slicing of UML Architectural Models, *Journal of Object Technology, vol. 8, no. 1*, pp. 159-188, January-February 2009.

[11] J. Lallchandani and R. Mall, A Dynamic Slicing Technique for UML Architectural Models, IEEE Transaction on Software Engineering, Vol. 37, No. 6, NOV/DEC 2011.

[12] Philip Samuel, Rajib Mall, Slicing-Based Test Case Generation from UML Activity Diagrams, *ACM SIGSOFT Software Engineering Notes, Vol. 34 No. 6*, November 2009.

[13] Kunihiro Noda , Takashi Kobayashi, Kiyoshi Agusa, Shinichiro Yamamoto, Sequence Diagram Slicing, 16th Asia-Pacific Software Engineering Conference, IEEE, 2009.

[14] Ranjita Kumari Swain , Vikas Panthi, Prafulla Kumar Behera, Test Case Design Using Slicing of UML Interaction Diagram, 2nd International Conference on communication, computing and security, *vol.6* , pp.136-144, ELSEVIR, 2012.

[15] Hyeon-Jeong Kim , Doo-Hwan Bae, Vidroha Debroy, W. Eric Wong, Deriving Data Dependence from UML State Machine Diagrams, Fifth International Conference on Secure Software Integration and Reliability Improvement (IEEE), 2011.

[16] Nisansala Yatapanage, KirstenWinter, and Saad Zafar, Slicing behavior tree models for verification, In IFIP Advances in Information and Communication Technology, *Vol. 323*, pp. 125–139, 2010.

[17] B. Korel, I. Singh, L. Tahat, and B. Vaysburg, Slicing of State Based Models, Proc. Int'l Conf. Software Maintenance, pp. 34-43, 2003.

[18] Kevin Lano Crest, Slicing of UML State Machines, Proceedings of the 9th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '09), 2009.

[19] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France, Slicing feature models, In 26th IEEE/ACM International Conference On Automated Software Engineering (ASE'11), IEEE/ACM, 2011.

[20] J. Julliand, N. Stouls, P-C. Bue, P-A. Masson, B model slicing and predicate abstraction to generate tests, *Software Quality Journal, vol. 21*, pp.127-158, 2013.

[21] Asadullah Shaikh, Robert Clarisó, Uffe Kock Wiil, and Nasrullah Memon, Verification-driven slicing of UML/OCL models, In Proceedings of the IEEE/ACM international conference on Automated software engineering, pp. 185–194, ACM, 2010.

[22] T. Kim, Y.-T. Song, L. Chung, and D.T. Huynh, Dynamic Software Architecture Slicing, Proc. 23rd Int'l Computer Software and Applications Conf., pp. 61-66, 1999.

[23] T. Kim, Y.-T. Song, L. Chung, and D.T. Huynh, Software Architecture Analysis: A Dynamic Slicing Approach, *J. Computer and Information Science, vol. 1, no. 2*, pp. 91-103, 2000.

[24] Kevin Lano and Shekoufeh K. Rahimi, Slicing of UML Models Using Model Transformations, Model Driven Engineering Languages and Systems, 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Lecture Notes of Computer Science, *Vol. 6395*, pp. 228-242, Springer, 2010.

[25] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, Towards dynamic backward slicing of model transformations, In 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp.404–407, IEEE Computer Society, 2011.

[26] Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró, Dynamic Backward Slicing of Model Transformations, IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012.

[27] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, Modeling model slicers, Proceedings of the 14th international conference on Model driven engineering languages and systems, 2011.

[28] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, Kompren Modeling and Generating Model Slicers, Journal of Software and System Modeling, Springer, 2012.

[29] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Separation of Concerns in Feature Modeling: Support and Applications. In Aspect-Oriented Software Development(AOSD'12). ACM Press, 2012.

[30] Jung Ho Bae and Heung Seok Chae. UMLSlicer: A tool for modularizing the UML metamodel using slicing. In 8th IEEE International Conference on Computer and Information Technology (CIT), pages 772 –777, 2008.

[31] Miao Chunyu and Zhao Jianmin. Dynamic slicing of statechart specifications for reactive systems. In Intelligent Computation Technology and Automation (ICICTA),volume 1, pages 110–116, 2011.

[32] Davide Falessi, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Antonio Messina. SafeSlice: a model slicing and design safety inspection tool for SysML. In 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13). ACM, 2011.

[33] Asadullah Shaikh, Uffe Kock Wiil, and Nasrullah Memon, Evaluation of tools and slicing techniques for efficient verification of UML/OCL class diagrams, Advances in Software Engineering, vol.18, pp 173-192, 2011.

[34] Rupinder Singh, Vinay Arora, Literature Analysis on Model based Slicing, *International Journal of Computer Applications* vol. 70, issue 16, pp 45-51, May 2013.

[35] Vinay Arora, Rajesh Kumar Bhatia and Maninder Singh, Evaluation of Flow Graph and Dependence Graphs for Program Representation, *International Journal of Computer Applications* Vol. 56, Issue 14, page 18-23, October 2012.