Sequence Diagram Extractor for Better Software Visualization

Arya Bharti
M. Tech. Scholar
Department of Computer Science & Engineering.
Om Institute of Technology & Management, Juglan Hisar.

Surender Singh
Asstt. Professor
Department of Computer Science & Engineering.
Om Institute of Technology & Management, Juglan Hisar.

Abstract - The attempt has been made to generate a smaller sequence diagram from a larger one. The Sequence Diagram is generated with the help of the Visual Paradigm for UML Version 10.0. This tool comes with the inbuilt functionality to generate the image, excel as well as XML for the sequence diagram. The XML has been then parsed for transforming the data objects in a format acceptable for programming across the other platforms. This has been done with the help of JAVA DOM parser. The parsed file has been stored as the .txt file which is again modified to extract the valid program slice as per the slicing criteria applied by the programmer. The output of this step has also been modified by replacing the Object Ids with their names for better representation and understanding. This has been achieved by the tool JCreator LE. The result of this step has also been stored in the form of a .txt file. Finally, the text file is in the desired format which is accepted by another tool known as Quick Sequence Diagram Editor which generates the sequence diagram on the basis of the last .txt file.

Keywords- Sequence Diagram, XML, Software Visualization, Slicing, Quick Sequence Diagram Editor.

I. INTRODUCTION

The software visualization and testing are some of the most costly and time consuming activities in the Software Development Life Cycle. Both of the activities involve the analysis of the software architecture which is mediated by the Sequence Diagrams. The sequence diagrams are a type of the interaction diagram which keep a track of the messages shared in a software system on the basis of the time stamps and maintains all the information regarding the system states, the objects, their lifespan, the messages shared between various objects and all other aspects. That is why, the sequence diagrams are used for the software visualization and slicing.

***Slicing*:** Slicing can be thought of as a technique for decomposing the software into parts based on a particular criteria which is decided by the testes or the programmer to test whether the program satisfies the required expectations or not. The slicing is also done on the basis of UML Sequence Diagram these days. Hence, if we are able to reduce the size of the UML Sequence Diagrams, we shall be able to make the process of slicing as well as software visualization, test case generations, and other such activities related to the software in a better manner.

UML Sequence Diagram: UML Sequence Diagram are of various types which represent the system model or architecture in an abstract manner to provide the testers and other programmer a better view of the software dependencies, timings, messages, guard conditions, control flow, message flow, lifetimes, and other such details. There is but one problem with the sequence diagrams that they tend to become so huge with the size and complexity of the software system that it becomes a herculean task to extract the relevant slice or chunk of the program from the software to study or test it.

***Example1*:** An example sequence diagram for the ATM system with minimal functionality and no details given to the security and other parameters is mentioned in the following section.

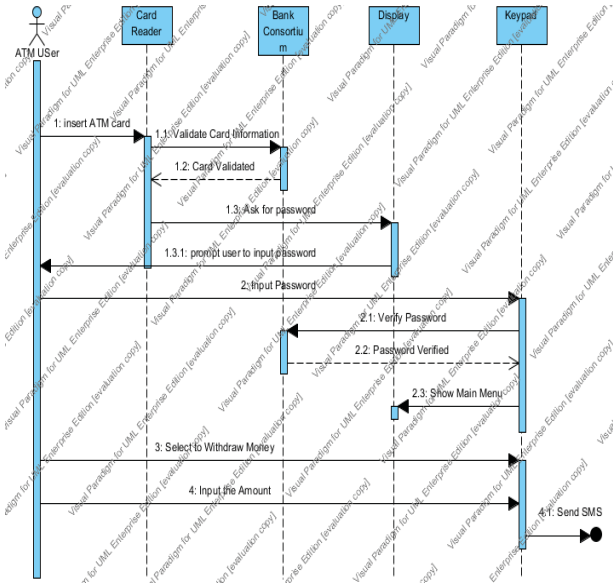

Figure 1. Sequence Diagram of an ATM System for Money Withdrawal.

Example 2: Sequence Diagram at a random system time for some random objects interacting and sharing messages across the system with some guard variables is another example. Find how the size of the sequence diagram fluctuates from the one given above to the one given below.

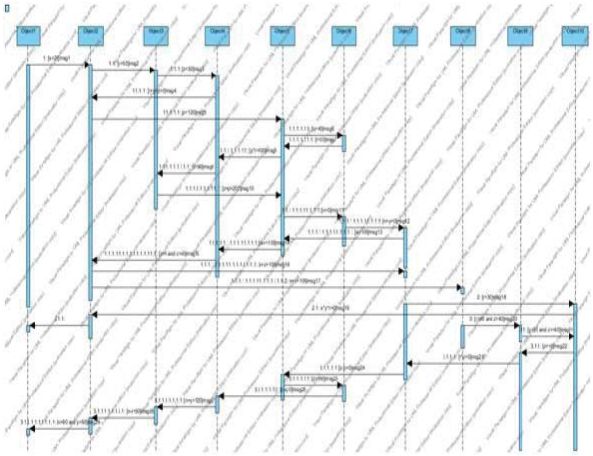

Figure 2. Sequence Diagram at a random time in a random system scenario

Now, as it is evident that the size of the sequence diagrams tends to be huge, we have focused on reducing this size by employing the following:

- The unique property of the XML to enable the interchange of the data in between devices with different platforms.
- The ability of the extraction of the program slice by passing a slicing criteria to a .java program via simple java environments such as command prompt and JCreator.
- The ability of formatting the text files as per the desired format which is acceptable to the tool Quick Sequence Diagram Generator.
- The ability of the above mentioned tool to generate the Sequence Diagrams directly from the .txt file in a proper format.
- The ability of the java programs to run fast, on a very low system requirement and cross platform nature.

II. IMPLEMENTATION OF THE TECHNIQUE

***Step 1*:** Download, Install and create the JRE an JDK environment, set the environment variables and specify the path.

Step 2: Download and Install the tool Visual Paradigm for UML version 10.0.

Step 3: Create a sample sequence diagram in the tool and export the diagram as XML. Find the screenshot of the XML file in the following figure:

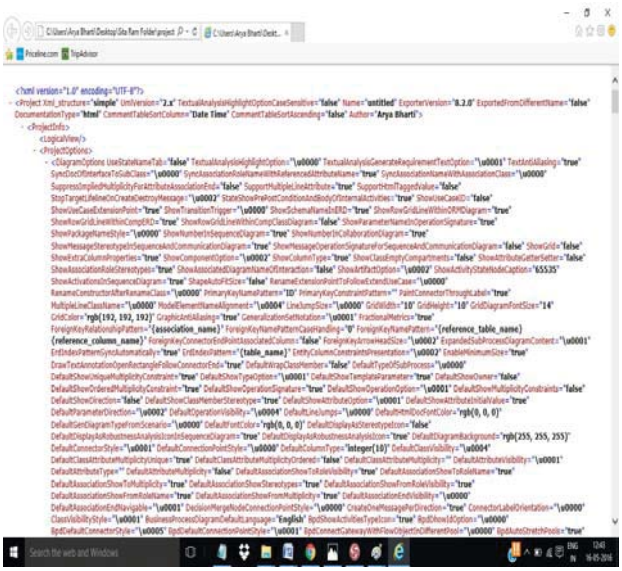

Figure 3. XML File for Figure 2

Step 4: Generate a JAVA DOM parser with the DocumentBuilderFactory; method and parse the XML file generated in Step 3.

```
import java.io.File;
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"));
doc.getDocumentElement().normalize();
System.setOut(new PrintStream(new FileOutputStream("Test.txt")));
NodeList nodeList1 = doc.getElementsByTagName("InteractionLifeLine");
System.out.println("over all class 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("Class-Name="+ell.getAttribute("Name")+" "+"Class-Id="+ell.getAttribute("Id"));
    System.out.println("\n");
```

Figure 4. DOM Parser screenshot

Step 5: The output of the DOM Parser will be stored in a .txt file which is then fed again to JCreator for compilation.

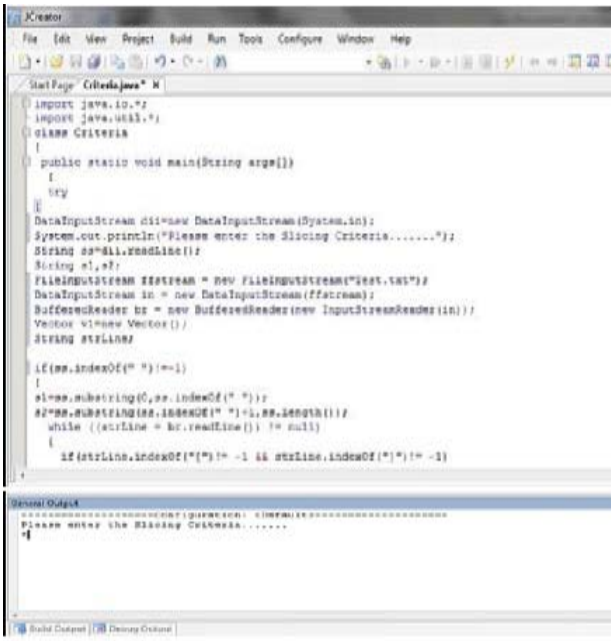

Figure 5. JCreator Screen Shot for txt file

***Step 6*:** Now, the Jcreator will ask for the slicing criteria which has been fixed as 'z' for the diagram in Figure 2. Finally, the output will also be generated in the form of a text file.

Message=[z<50]msg3 {TO-Object-Id= dOJL4XyFYHySDwM7 & From-Object-Id= jHxL4XyFYHySDwM0}

Message=[x+y+z>0]msg4 {TO-Object-Id= y0xL4XyFYHySDwMt & From-Object-Id= dOJL4XyFYHySDwM7}

Message=[z=90]msg9 {TO-Object-Id= jHxL4XyFYHySDwM0 & From-Object-Id= dOJL4XyFYHySDwM7}

Message=[z+p>200]msg10 {TO-Object-Id= 4lpL4XyFYHySDwNC & From-Object-Id= jHxL4XyFYHySDwM0}

Message=[x+z>100]msg16 {TO-Object-Id= ZSZL4XyFYHySDwNQ & From-Object-Id= y0xL4XyFYHySDwMt}

Message=[x+z<100]msg17 {TO-Object-Id= rA5L4XyFYHySDwNX & From-Object-Id= y0xL4XyFYHySDwMt}

Message=[x=40 and z=40]msg20 {TO-Object-Id= rIFL4XyFYHySDwNe & From-Object-Id= rA5L4XyFYHySDwNX}

Message=[y=50 and z>=40]msg21 {TO-Object-Id= IblL4XyFYHySDwNl & From-Object-Id= rIFL4XyFYHySDwNe}

Message=[z>=0]msg22 {TO-Object-Id= rIFL4XyFYHySDwNe & From-Object-Id= IblL4XyFYHySDwNl}

Message=[y*z>0]msg23 {TO-Object-Id= ZSZL4XyFYHySDwNQ & From-Object-Id= rIFL4XyFYHySDwNe}

Message=[z-p>0]msg24 {TO-Object-Id= 4lpL4XyFYHySDwNC & From-Object-Id= ZSZL4XyFYHySDwNQ}

Message=[z>100]msg25 {TO-Object-Id= SlpL4XyFYHySDwNJ & From-Object-Id= 4lpL4XyFYHySDwNC}

Message=[z-q>0]msg26 {TO-Object-Id= dOJL4XyFYHySDwM7 & From-Object-Id= SlpL4XyFYHySDwNJ}

Figure 6. Output File after the slicing criteria is applied

Step 7: This file is input to another java program to replace the Object Ids with the Object Names which will make the addressing and representation easy and better resp.

```
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 7. Program to Replace the Ids with Names

Step 8: Finally, the output file is again compiled by a java program to generate the txt file which is in proper format to be compiled by the Quick Sequence Diagram Editor tool which takes inputs in a fixed format.

```
import java.io.*;
import java.util.*;
class FileRead
{
 public static void main(String args[])
  {
  try
{FileInputStream ffstream = new FileInputStream("rupi.txt");
 DataInputStream in = new DataInputStream(ffstream);
 BufferedReader br = new BufferedReader(new InputStreamReader(in));
  String strLine;
  Vector v=new Vector();
  Vector v2=new Vector();
  while ((strLine = br.readLine()) != null)
  { String s ;
if (strLine.indexOf("TO-Object") != -1)
 { int i=strLine.indexOf("TO-Object")+10;
    s=strLine.substring(i,strLine.indexOf(" ",(i+2)));
     v.addElement(s);
     if(strLine.indexOf("From-Object") != -1)
      {
        //v2.addElement(s);
        int ii=strLine.indexOf("From-Object")+12;
        String ss=strLine.substring(ii,strLine.indexOf("}",(ii+2)));
        v2.addElement(ss);
        v2.addElement(s);
        ii=strLine.indexOf("Message=")+8;
        ss=strLine.substring(ii,strLine.indexOf("{",(ii+2)));
        v2.addElement(ss);
        }
  }
if(strLine.indexOf("From-Object") != -1 )
{
int i=strLine.indexOf("From-Object")+12;
    s=strLine.substring(i,strLine.indexOf("}",(i+2)));
      v.addElement(s);
}}
  Vector v1=new Vector();
```

Figure 8. Program for the conversion of the format of input to the Quick Sequence Diagram Editor

Step 9: Download and Install the Quick Sequence Diagram Editor tool which is available as open source. It is an executable JAR File which will take inputs in the form of text (in proper format) and generate the sequence diagram corresponding to the same in a few seconds.

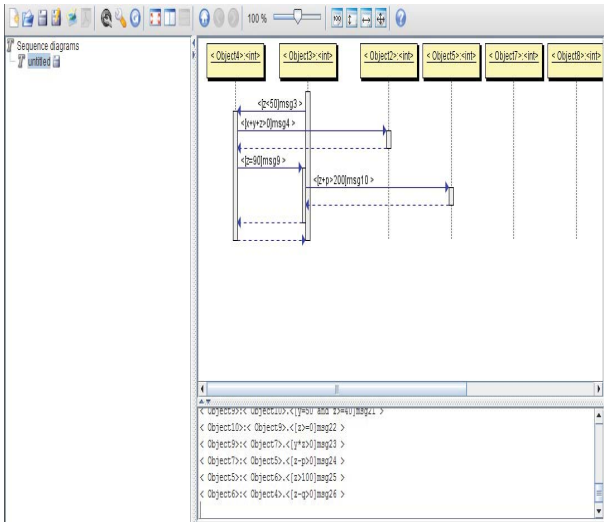

Figure 9. Quick Sequence Diagram Editor with the generated sequence diagram on the top and code of the txt file in the bottom section

III. RESULT AND ANALYSIS

The result has been released in the form of a much smaller sequence diagram. From the diagram shown in the Figure 2, we have extracted the diagram in the Figure 9. This extraction has been done on the basis of the slicing criteria 'z' which means that all the objects with the messages having the variable 'z' in it will be extracted from the complete diagram and a refined slice will be generated for the programmer or the tester.

IV.CONCLUSION

There are a lot of activities in the software development lifecycle which depend on the UML Sequence Diagram and if we can reduce the size of this diagram, we shall be able to make all those phases easy and less costly.
This technique can be applied for better software visualization, refined slice generation, minute examination and testing of the software slices and better test case generation from the sequence diagram.

REFERENCES

[1] 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.
[2] H. Kagdi, J.I. Maletic, and A. Sutton, "Context-Free Slicing of UML Class Models", In Proceeding of 21st IEEE International Conference on Software Maintenance, pp. 635-638, 2005.
[3] Kevin Lano Crest, "Slicing of UML State Machines", In Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications, pp.63-69, 2009.
[4] P. Mathur, "Foundation of Software Testing", Pearson/Addison Wesley, 2008.
[5] IEEE Standard 1059-1993, "IEEE Guide for Software Verification and Validation Plans", pp.1-87, Computer Society, 1993.
[6] Richard Torkar, "Towards Automated Software Testing Techniques, Classifications and Frameworks", Doctoral Dissertation, Department of Systems and Software Engineering, Blekinge Institute of Technology, pp.1-235, Series No 2006:04, Sweden, 2006.
[7] R. D. Craig, S. P. Jaskiel, "Systematic Software Testing", Artech House Publishers, Boston-London, 2002.
[8] S. R. Rakitin, "Software Verification and Validation for Practitioners and Managers", Artech House Publishers, Boston-London, 2001.
[9] 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.
[10] Kevin Lano Crest, "Slicing of UML State Machines", In Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications, pp.63-69, 2009.
[11] H. Kagdi, J.I. Maletic, and A. Sutton, "Context-Free Slicing of UML Class Models", In Proceeding of 21st IEEE International Conference on Software Maintenance, pp. 635-638, 2005.
[12] J.H. Bae, K.M. Lee, and H.S. Chae., "Modularization of the UML metamodel using model slicing", IEEE 5th International Conference on Information Technology: New Generation., pp. 1253–1254, 2008.
[13] 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.

[14] Mahesh Shirole, Rajeev Kumar, "Testing for Concurrency in UML Diagrams", ACM SIGSOFT Software Engineering Notes, vol. 37, No. 5, pp.1-8, 2012.
[15] Davide Falessi, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Antonio zessina, "SafeSlice: A model slicing and design safety inspection tool for SysML", In Proceeding 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference, ACM, 2011.
[16] B. Korel, I. Singh, L. Tahat, and B. Vaysburg, "Slicing of State Based Models", In Proceeding of International Conference of Software Maintenance, pp.34-43, 2003.