ISSN:2229-6093

Reverse Engineering: An Analysis of Dynamic Behavior of Object Oriented Programs by Extracting UML Interaction Diagram

Mrinal Kanti Sarkar
Dept. of Computer Science & Engineering
University of Engineering & Management, Jaipur

Trijit Chaterjee
Dept. of Computer Science & Engineering
University of Engineering & Management, Jaipur

Abstract

The Unified Modeling Language (UML) is widely used as a high level object oriented specification language. UML is a good target language for the reverse engineering models since it is largely used and offers different diagrams. In this paper we present a novel approach in which reverse engineering is performed using UML as the modeling language used to achieve a representation of the implemented system. In this work we have considered java programs. After a brief introduction to the subject we present some analyses which go beyond mere enumeration of methods and fields. We sketch our method of determining association multiplicities, being, in a sense, representative of our approach which produce a simple sequence diagram that can be understood by a programmer when inspecting the source code of a given java programs. To fully understand the behavior of a program, it is crucial to have efficient techniques to reverse dynamic views of the program. In this paper, we focus on the reverse engineering of UML sequence diagram from an object oriented programs and analysis of its dynamic behavior.

Keywords*-Reverse; Engineering; OOP; UML; Extraction; Instrumentation; Interaction Diagram; Sequence Diagram;*

1. INTRODUCTION

Reverse-engineering can help to understand a complex system by retrieving models and documentation from a program. UML is a good target language for the reverse engineering of models since it is largely used and offers different diagrams. In the present work, we outline a reverse engineering approach for UML specification in form of sequence diagram from Java programs. A glance onto some related work shows that there seems to be no solution for the reverse engineering of the more difficult sequence diagram.

Traditional software engineering research and development focuses on increasing the productivity and quality of systems under development or being planned. Without diminishing the importance of software engineering activities focusing on initial design and development, empirical evidence suggests that significant resources are devoted to reversing the effects of poorly designed or neglected software systems. In a perfect world, all software systems, past and present, would be developed and maintained with the benefit of well-structured software engineering guidelines. In the real world, many systems are not or have had their structured design negated, and there must be tools and methodologies to handle these cases. Reverse Engineering is a methodology that greatly reduces the time, effort and complexity involved in solving the program comprehension problem. Reverse Engineering is best defined by Chikofsky and Cross [1] as "the process of analyzing a subject system

- To identify the system's components and their Inter-relationships and
- To create representations of the system in another form or at a higher level of abstraction."

Reverse Engineering is a well-established practice in that there are numerous CASE tools available to map source code to good quality structural models. The reverse engineering of UML static diagram – has been studied and is now implemented in several tools. Static views of the system allow engineers to understand its structure but it does not show the behavior of the software. To fully understand its behavior, dynamic models are needed, such as sequence diagrams or state charts diagram. Many works has been done on reverse engineering of UML sequence diagrams. Automatic reverse engineering of UML dynamic models [11] are describe by Y-G Guhenneuc and T Ziadi. Another approach by using state vectors through trace analysis [13] can get the dynamic views of a system. These CASE tools were being used alongside sequential programming languages like COBOL to maintain the design of documentation. The concept of Reverse Engineering emerged from the hardware world where hardware circuits were reverse-engineered to create clones. When the software engineers adopted the same term to describe some software engineering practices, there was a dearth of well-defined terminology to use for both technical and market-place discussions. A Library Information System (LIS) application which is being used extensively in all university's library in the world was chosen as a case study for the purpose of this work. The *eLib* program is a small Java program that supports the main functions operated in a library. It is not so easy to understand how the classes are

organized, how they interact with each other to fulfill the main functions, how responsibilities are distributed among the classes, what is computed locally and what is delegated.

In this paper we extract the dynamic behavior of an object oriented system. We have chosen UML sequence diagram to understand the dynamic behavior of an OOP.

Our paper is organized as follows. In section 2, we present the concept of reverse engineering and analyze the difficulties of reverse engineering. Section 3 explores the sequence diagram and its perspective. In section 4, we address the design issues to extract sequence diagram from a java source code. We present our result in section 5. Finally, in section 6 we discuss the future work focusing on the dynamic behavior of any object oriented system.

2. REVERSE ENGINEERING

A. *Reverse Engineering Defined*

Chikofsky and Cross [1] made a very successful attempt at providing some precise and long standing definitions for much of the terminology used to this day in the field of Reverse Engineering. In this paper, they have started with a description of the ANSI definition of software maintenance and established the concept of software development life cycles. The following terms and definitions are adapted from the canonical taxonomy given in [1].

Forward Engineering: This term, forward engineering is opposite of reverse engineering and it distinguish the traditional software engineering process from reverse engineering. Forward engineering takes sequences from feasibility study through designing its implementation.

Reverse Engineering: This is the process for identify and analysis of software's system components, their inter-relationships and the representation of their entities at a higher level of abstraction. Reverse engineering by itself involves only analysis; it does not involve changing the subject system or not create any new system. It has been found that there are many sub-areas of reverse engineering. But re-documentation and design recovery are widely used in reverse engineering.

- *Re-documentation:* Perhaps the weakest form of reverse engineering. This involves the creation or revision of system documentation at the same level of abstraction. It is the simple and oldest form of reverse engineering and it gives you the alternate views of the system. The primary goals of these tools are to provide easier ways to visualize relationship among program component.
- *Design recovery:* Design recovery re-documents the system component, but uses domain knowledge and other external information where possible to create a model of the system at a higher level of abstraction.

According to Ted Bigger Staff "Design recovery recreates design abstraction from a combination of code, existing design documentation, personal experience, and general knowledge about problem and application domains. It must reproduce all of the information required for a person to fully understand what a program does, how does it, why does it, and so forth."

- *Restructuring: Restructuring* transforms the system within the same level of abstraction and also maintains same level of functionality and semantics. It improves the code's structure in the traditional sense of structured design.
- *Re-engineering:* Re-engineering changes the functionality and direction of the system. It involves a combination of reverse engineering for comprehension, and a re-application of forward engineering and maximizes which functionalities that needs to be retained, deleted or added. It includes some form of reverse engineering followed by some form of forward engineering or restructuring.

B. *Objectives of Reverse Engineering*

The primary purpose of reverse engineering a software system is to increase the overall comprehensibility of the system for both maintenance and new development. When we try to characterize reverse engineering in terms of its objectives, there are six key objectives (Chikofsky and Cross II 1990) in reverse engineering [2].

- *Cope with complexity:* We must develop method to better deal with the share volume and complexity of the system. A key to controlling these attributes is automated support. Reverse Engineering methods and tools, combined with case environments, will provide a way to extract relevant information so decision makers can control the process and the product in the system.
- *Generate alternate views:* Graphical representation has long accepted as comprehension aids. However, creating and maintaining them continuous to be a bottleneck in the process. Reverse-engineering tools facilitate the generation or regeneration of graphical representation from other forms. While many designers work from a single, primary perspective (like dataflow diagrams), reverse-engineering tools can generate additional views from other perspective (like control flow diagram, structure chart, and entity relationship diagrams) to aid the review and verification process.
- *Recover lost information:* The continuing evolution of large, long –lived systems leads to lost information about the system design. Modifications

are frequently not reflected in the documentation, particularly at a higher level than the code itself. While it is no substitute for preserving design history in the first place, reverse-engineering – particularly design recovery is our way to salvage whatever we can form the existing system.

- *Detect side effect:* Both haphazard initial design and successive modification can lead to unintended ramification and side effects that impede a system's performance in subtle ways.
- *Synthesize higher abstraction*: Reverse-engineering requires methods and techniques for creating alternate views that transcend to higher abstraction levels. There is a debate in the software community as to how completely process can be automated. Clearly, expert-system technology will play a major role in achieving the full potential of generating high-level abstraction.
- *Facilitate reuse:* A significant issue in the movement toward software reusability is the large body of existing software assets. Reverse-engineering can help detect candidates for reusable software components from present system.

C. *Difficulties in Reverse Engineering*

Michael L. Nelson [10] gives a detailed discussion of the practical difficulties involved in the reverse engineering. These difficulties include that of the choice of the level of abstraction needed, and that of the formal/cognitive distinction. Computers and programming languages are formal, while the human cognitive capabilities are non-formal. Therefore, the result of any reverse engineering work could be very subjective. Any program is "understood to the extent that the reverse engineer can build up correct high level chunks from the low level details available in the program."

A discussion of how these difficulties are manifested to the reverse engineer follows. The choice of methodology, representation and tools used will define the usefulness of the derived reverse engineering information. The work done on integrating the top-down and bottom-up approaches to understanding a program to develop an approach, called the synchronized refinement is described. This approach is based on the detection of design decisions in the source code and the organization of the information into an information structure suitable for browsing by software maintainers. However, the process suggested is labor intensive, though the paper suggests that automating the individual tasks in the process can reduce the rigor involved.

The author suggests that many of the activities described in the Synchronized Refinement methodology are automatable. He suggests that if a comprehensive information structure is populated with information about a program, different views of the system can be extracted from the database as required for understanding a particular part of the source code.

Currently, reverse engineering is currently heavily dependent on human interaction and memorization. While there are tools to assist the reverse engineer in program comprehension, it is not a fully automated process. The human element present in program comprehension is the subject of another field, *software psychology* [8]. Software psychology measures human performance while interacting with computer and information systems.

3. SEQUENCE DIAGRAM

A sequence diagram is kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development

The basic idea of architecture definition is to design software structure and object interaction (Behavior) before the detailed design phase. Behavior of software is really what our software does after it rolls out into the production environment. The behavior is defined of software system by defining interactions between structural elements.

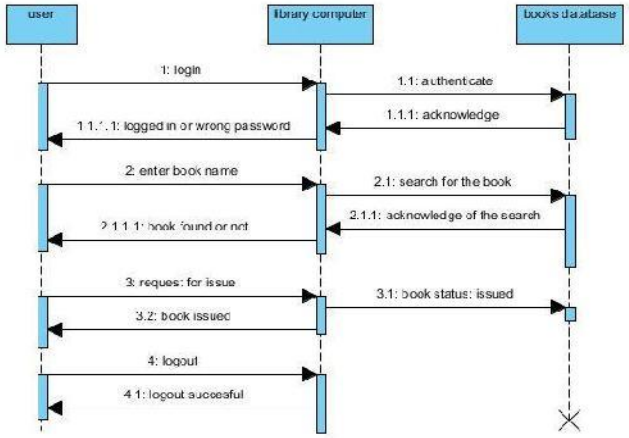

Figure 1: Sequence Diagram

Figure 1 shows a UML sequence diagram showing a number of interactions that, together, allow the system to support the user. Here we see three object interactions. First, a user object interacts with the library computer object to have time for login in library computer. The library computer instance then interacts with book database object to check authentication. Then book database instance returns an acknowledgement to library computer object. Then depending upon the correct information of the particular user gets logged in or wrong password.

4. EXTRACTING SEQUENCE DIAGRAM

In this chapter we will address the design issues to extract sequence diagram from a java source code. Some generic ideas imbibed from previous work and intuitive notation were decided upon initially to set up the environment for the reverse engineering effort. We will give the overview of the design of the tool and see how the design can be extended to extract sequence diagram from a object oriented program.

To extracting sequence diagram from a OOP programs, we concentrate on reverse engineering relies on dynamic analysis. Dynamic analysis is usually performed on the java programs in a form of execution i.e. at the running time of the programs. It is very difficult generate complete sequence at execution time. Our work take places three steps, to generate a sequence diagram from an OOP programs at their running time. Firstly, we instrument the java programs and get a new instrumented java code. Then we generate a picture file by message sending sequence when run the instrumented java code. Finally, we get the sequence diagram from the .pic file using the 'GNU' pic2plot tools. We stated below the process of generation of a sequence diagram from an OOP program.

- Instrumentation of the java programs
- Message sending sequence from instrumented code to create pic file.
- Generation of sequence diagram from .pic file using pic2plot.

A. *Approach to generate instrumented code*

For instrument of a java programs, we have used a 3-Pass algorithm.

- **In the first pass**, we have gathered all the class names and objects declared in the Java Source Code. This we have done by reading tokens and identifying the "class" keyword. Care has been taken to avoid token detection inside commented pieces of code and inside strings. Once the class keyword is detected, the next token available is definitely the class name. Again comments are ignored. The objects are detected by identifying the "new" keyword. Since all objects need to be created in order to be used. Objects are stored by noting the class in which they are, the scope level in which they are present and a constructor of what type is used to initialize this object. Scope levels are set by using second brackets ("{" and "}").
- **In the second pass**, we have used the data from the first pass to insert code in appropriate places in the Java Source Code. The insertion of code is not done here, in the 2nd pass. However, the position in the file where a particular insertion will be made is recorded along with the code to be inserted. Function calls and body is detected. The entire function name is read (along with object name, of the format: obj.func) and then the object name is separated. The object name is searched from the global store that was generated from the previous pass. An object of type *refactored* is created after every function body starts. This object calls the "begin" function and "end" functions after the beginning and ending of every function call. The code is marked for insertion at these points. The object of type *refactored* is declared in the file: refactored.java. The file does the appropriate processing required to generate the graph from these calls. Also the end of function main is detected and appropriate code is marked for insertion also.
- **In the third pass**, the output is generated to standard output. The source code program is read again and output character by character. Whenever a position that matched a code to be inserted is found, the corresponding code is inserted and output to stdout equations.

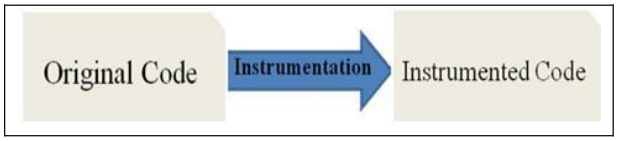

Figure 2: Instrumentation of OOP

B. *Message sending sequence from instrumented code to create .pic file.*

After instrumenting the java programs, we get the instrumented code. First we capture the message sequence and then generate a pic file (written in the pic language format). Now we will briefly describe the procedure implemented in the code refactored.java for generation of UML sequence diagram from message sequences sent by the instrumented java code to create a pic file.

1) *Capture message sequences from instrumented code*

When the instrumented java code is run two external method calls are made. First one is the method begin() which is called before any method call found in the original java program. Another one is end() which is called after that target method is returned. So, the calling function actually calls the following:

i. begin(<parameters>)
ii. <original method>
iii. end()

begin() and end() are implemented here in the refactored.java file. Since these are external methods, an object named r2109 of type "public class refactored" is created in the scope of the calling function. Its the parameters which convey the messages for sequence diagram generation. We need the called function name, its object name (null in the case of static method or where the object is unknown), the class to which the object belongs, the object for the calling function and the class of this object. Thus begin () is thus called with the following parameters: source class name <class1>, called function

<func>, destination object name <obj2>, destination class name <class2>. End is called without any parameters 'cause the parameters for begin () are already saved. There is another method endMain() which is called just before any return path of main.

Figure 3: Process of generating .pic file

begin() and end() internally calls another public method WritePicFile() implemented in refactored.java . It's the actual method which writes the required sequence_diagra.pic. It takes two parameters as its parameters. First one is "param" of type array of strings and last one is a calltype of type int. The meaning of these parameters is given bellowbegin(<parameters>)

param[0]: Source object name
param[1]: Source class name
param[2]: Destination object name
param[3]: Destination class name
param[4]: Called method name

calltype: INIT_PICFILE or INVOKE_MAIN or CREATE_OBJECT or METHOD_CALL or METHOD_RETURN or MAIN_RETURN .These constants are defined in the beginning of public class refactored as "public static final int" with values 10,1,2,3,4,0. calltype is determined by the begin() and end() method. When an object is created calltype is CREATE_OBJECT. When a method call is made calltypepe METHOD_CALL. For end() method its always METHOD_RETURN and for endMain() its always MAIN_RETURN. Also, there are three other methods defined in refactored.java namely ConvertFloatToString() , ConvertIntToString() and initpicobjects(). These are needed by WritePicFile() method.

C. Generation of sequence diagram from pic file using pic2plot GNU tool.

Sequence diagram is generated from a picture description file (written in the pic language format) which is converted to an image by a tool GNU pic2plot. GNU pic2plot is a part of the GNU plotutils package. Here the generated p i c file name is sequence_diagram.pic. The documentation f o r the pic language is available from http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/web/pic. html For drawing sequence diagram using pic, a handy macro definition is used called UML Graph. Ref. http://www.umlgraph.org

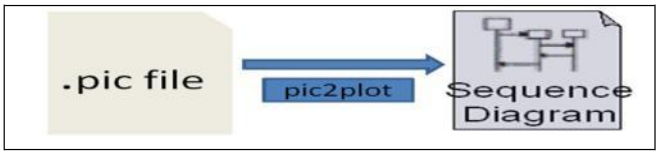

Figure 4: .pic fife to sequence diagram

5. IMPLEMENTATION RESULT OF SEQUENCE DIAGRAM

The design of the sequence diagram tool has been implemented in g++ and jdk 1.6.0_16 in LINUX platform. For the instrumentation, we write down a C programs names main.cpp in g++ and for sending message to generate pic file we implement a java programs names refactored.java. To generate picture we use a picture description file (written in the pic language format) which is converted to an image by a tool GNU pic2plot. GNU pic2plot is a part of the GNU plotutils package. Here the generated pic file name is sequence_diagram.pic.

A. Example of insrumented code

```
public class kool {
    public static void main( String a[] ) {
    refactored r2109 = new refactored();
    r2109.begin( "kool", "abc", "j", "abc" );
    abc j = new abc();
    r2109.end();
     r2109.begin( "kool", "println", "", "System.out" );
    System.out.println("I can be here too for checking!");
    r2109.end();
    r2109.begin( "kool", "msg", "j", "abc" );
    j.msg();
    r2109.end();
    r2109.begin( "kool", "m2", "j", "abc" );
    j.m2();
    r2109.end();
    r2109.endMain();
  }
}
class abc {
    void msg() {
        refactored r2109 = new refactored();
        r2109.begin( "abc", "println", "", "System.out" );
        System.out.println( "In abc" );
         r2109.end();
    }
    void m2() {
        refactored r2109 = new refactored();
        for( int i=0; i<3; i++ ) {
        r2109.begin( "abc", "println", "", "System.out" );
        System.out.println( "Num : " + i );
        r2109.end();
    }
}
```

B. Code snippts of Generated .pic file

```
.PS 0.000000 0.000000
copy "sd_macros.def"
actor(O0,"");
boxwid=0.5; object(O1,":kool");
step();
message(O0,O1,"main"); active(O1);
n=1; m=0;
for i=1 to n  do {
for j=1 to n do {
if `j'th box.c.x > `i'th box.c.x then {j=n+2}
}
if j == n+1 then {m=i; i=n+1;}
```

```
}
down; step(); move to (O1.c.x + awid/2,Here.y); right; arrow from Here to
(`m'th box.c.x + `m'th box.wid + movewid, Here.y)  "<<create>>" "";
boxwid=0.5 object(O2,"j:abc"); move left; down; move (spacing + boxht) /
2; active(O2);
rmessage(O2,O1,"<<return>>");inactive(O2);
step();
x=Here.x; y=Here.y; define svdir {right};
n=2; m=0;
for i=1 to n  do {
  for j=1 to n do {
  if `j'th box.c.x > `i'th box.c.x then {j=n+2}
  }
  if j == n+1 then {m=i; i=n+1;}
  }
  move to (`m'th box.c.x,O1.y); move right `m'th box.wid; move;
  boxwid=1.25; object(O3,":System.out");
  move to (x,y); svdir;
  step();
  message(O1,O3,"println"); active(O3); step();
  rmessage(O3,O1,"<<return>>");inactive(O3);
  step();
  step();
  message(O2,O3,"println"); active(O3); step();
  rmessage(O3,O2,"<<return>>");inactive(O3);
  message(O2,O3,"println"); active(O3);
  step();message(O2,O3,"println"); active(O3);
  step();
  rmessage(O3,O2,"<<return>>");inactive(O3);
  step();
  step();
  message(O2,O3,"println"); active(O3);
  step();
  rmessage(O3,O2,"<<return>>");inactive(O3);
  step();
  rmessage(O2,O1,"<<return>>");inactive(O2);
  step();
  dmessage(O1,O0,"")
  complete(O0);
  complete(O1);
  complete(O2);
  complete(O3);
  .PE
```

C. Generated sequence diagram

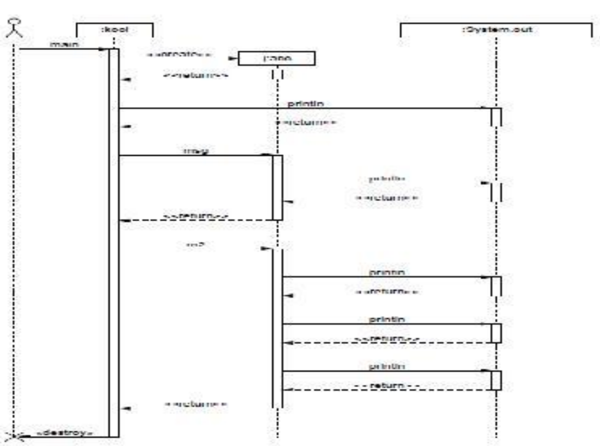

Figure 5: Sequence Diagram

6. CONCLUSION

Obviously, it is very hard to implement UML interaction diagram from an Object Oriented Programming Languages. Static analysis is not sufficient to analyze or understand an Object Oriented Programming properly. We have find out sequence diagram using dynamic analysis. Till now our works does not handle state diagram and other model of the UML diagram. Thus our reverse engineering tool can work only simple OOP. While implementing our design we have only handled a small subset of the languages able to find out sequence diagram. The more complicated features of the programs have to be implemented. A good graphical user interface is to needed to view the all diagrams. Here escape character are not recognized. Method calls within a method calls also ignored.

This work is one of the earliest works done in applying a reverse engineering process to Object Oriented Systems. The formal process described does not have any automated approaches involved at this point, but the emergence of such tools is expected from the tools industry soon. In future details such as loop or condition, which require more advanced instrumentation technique, may be captured and presented in the generated diagrams. The main key issue of reverse engineering is to discover the abstraction of java source code.

REFERENCES

[1] E. J. Chikofsky and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, pp. 13-17, January 1990.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Stan Jarzabek and Guosheng Wang, "Model Based Design of Reverse Engineering Tools," Software Maintennance: Research and Practice, J. Softw. Maint: Res. Pract.10, 353-380 (1998).K. Elissa, "Title of paper if known," unpublished.

[3] "Application Reengineering", Guide Pub, GPP-208, Guide Int'l Corp., Chicago, 1989.

[4] R. Mall, "Fundamentals of Software Engineering", Prentice-Hall, India, 2nd Edition, August, 2008.

[5] Nicola Howarth, "Abstract Syntax Tree Design" ANSA Phase III, 23rd August, 1995

[6] Romain Delamare, Benoit Baudry, Yves LeTraon, "Reverse engineering of UML 2.0 Sequence Diagrams from Execution Traces" " French national institute for research in computer science and control, April, 2006

[7] D. Kornack and P. Rakic, "Cell Proliferation without Neurogenesis in Adult Primate Neocortex," Science, vol. 294, Dec. 2001, pp. 2127-2130, doi:10.1126/science.1065467.

[8] *F.G. Pagan, "Partial Computation and the Construction of Language Processors", Prentice Hall, 1991*

[9] Ben Shneiderman , "Software Psychology: Human Factor in Computer and Information System", Little Brown and Co., Boston,Massachusetts, 1980

[10] Frank Tip, "A survey of Program Slicing Techniques", Journal of Programming languages, Sept. 1995

[11] Michael L. Nelson "A Survey of reverse Engineering and Program comprehension" April 19, 1996.

[12] Y.-G. Gu´eh´eneuc and T. Ziadi, "Automated reverse-engineering of UML 2.0 dynamic models", Proceedings of the 6th ECOOP Workshop on Object-Oriented Reengineering, 2005.

[13] Yves Le Traon, Benoit Baudry and Romain Delamare "Reverse Engineering of UML 2.0 Sequence Diagram from execution Traces" French national institute for research in computer science April, 2006

[14] Object Management Group website http://www.omg.org/uml

[15] Rational Corporation website http://www.rational.com