ABSTRACTION OF UML DIAGRAMS FROM JAVA CODE

Prof. R.N Kulkarni
HOD,Dept Of CSE
BITM, Bellary

Shilpa Jain M
Dept Of CSE
BITM, Bellary
shiilpabaldota80@gmail.com

Bhavana S H
Dept Of CSE
BITM, Bellary
bhavanareddy.06@gmail.com

Nethravati K
Dept Of CSE
BITM,Bellary
nethra.kamagandi@gmail.com

Shalini S
Dept Of CSE
BITM
shaliniskhr@gmail.com

Abstract: During the last few decades, we can see tremendous growth in the software industry. Every organisation is developing either generic or customized applications for the customers. At the time of development, the developers collect the requirements from the customer and then they realize requirement into design and then design into code. The requirement and the design documents are available with the developers only. If any user wants to know the design information, then it is not available. There is no tool available where we can extract the design of an information system by giving the source code written in java programming language.In this paper, we are proposing a automated methodology to abstract the design information from the input java code, which comprises of three phases: Restructuring, Moulding and then Designing. The output design information is represented in the form of following UML diagrams (class diagrams, object diagrams, usecase diagrams, sequence diagrams and activity diagrams)Moulding, Restructuring and Designing are the three phases required to achieve our motive.

Keywords: Restructuring, Moulding, Designing, Reverse Engineering.

1. INTRODUCTION

Designing the structure of source code for example, as an UML class diagram, object diagram, use case diagram, sequence diagram and activity diagram. It is quite well understood Java code acts an input and corresponding types of UML diagrams emerge as an output. Simple implementation of Reverse engineering UML model format, by directly extracting the needed information from the given problem code. Reverse Engineering enables representations of the system in another form at a higher abstraction level.
IBM Rational Rose Enterprise software provides a modeling support for application development and works with a number of implementation technologies. But this tool, all the UML elements should be defined in prior for building UML diagrams. This drawback made us think about the limitation of Rational Rose and work on it to overcome. We try to develop a tool which abstracts UML diagrams from the problem code. It reduces complexity when compared to other existing tools. It is simple to implement as it directly extracts its needed information from the input java code, hence there is no need of defining the elements of UML in prior. In the paper [1], the author specifies Software requirements in natural language (NL). However, requirements specified in NL can often be ambiguous, incomplete, and inconsistent. Moreover, the interpretation and understanding of anything described in NL has the potential of being influenced by geographical, psychological and sociological factors. The proposed tool in [1] is referred to as Requirement analysis to Provide Instant Diagrams (RAPID). The RAPID tool assists analysts by providing an efficient and fast way to abstract only the class diagram from their requirements. But in our methodology, we are abstracting class, object, use case, sequence and activity diagrams from the source code.

In the paper [2], the authors proposed a novel approach for integrating OOAD and Task Modelling. By exploiting the common semantic ground between task models and system behaviour models, they describe generation of UML diagrams, such as use cases, use case diagrams and scenario diagrams, from the task models represented in a semi-formal notation.

In the paper [3], the authors try to use fuzzy pattern detection techniques for the recovery of UML collaboration diagrams from source code. The approach is based on a knowledge base of basic data types and of generic collection classes and of code clichés for Java beans and of fuzzy patterns for object structure look-up and modification clichés.

The authors Grady booch, James Rumbaugh in their book titled *"The Unified Modeling Use Guide"* [4] provides a reference to the use of specific UML features. The user guide describes a development process for use wth UML and speaks about the usage of UML effectively from the scratch. It is primarily directed to members of the development team who create UML models.

In the book [5] *"UML for Java Programmers"*, the author clearly illustrates the overview of UML for java programmers. It clearly explains the relationship of UML elements with the java code. It guides in the drawing of various UML diagrams.

In our methodology, we try to abstract UML diagrams from the java source code. Java code acts as an input and five types of UML diagram format emerge as an output. It reduces complexity when compared to other methodologies already available in the market.

2. TERMINOLOGY

2.1 Restructuring: Restructuring is the transformation from one representation form to another at the same abstraction level. The transformation preserves the external behavior of the system. Restructuring is

typically used in implementation stage to transform code from an unstructured form to a structured form.

2.2 Moulding: It is a method for decomposing programs by analyzing their data and control flow. The behavior of the classes and the control flow is analyzed that helps in determining relationships between them. It moulds the program according to the behavior and relationships between the elements of the program.

2.3 Designing: Designing represents the actual designing of the UML diagrams which includes class diagrams, object diagrams, use case diagrams, sequence diagrams and activity diagrams as an output. The UML diagrams are abstracted from the source code.

2.4 Reverse Engineering: The reverse engineering is the process of analyzing the subject system with two goals:
- To identify the system's components and their interrelationships
- To create representations of the system in another form at a higher abstraction level.

3. PROPOSED METHODOLOGY:
In this paper, we are proposing a methodology for the abstraction of design information in the form of class, object, use case and sequence diagram from the input java program. The methodology comprises the following steps:
3.1 Restructuring: Initially scan the program, restructure the code by placing the methods in call in sequences order and arranging the classes in inheritance sequence.
/* Algorithm for the Restructuring */ Input: Executable 'java' program
Output: Restructured java code stored in a file
1. [Restructuring of input 'java' program]
1.1 Scan the program, search for the function main.
1.2 Identify the method in the main, arrange the methods in calling sequence in calling-called hierarchy. 1.3 Arrange the classes in their inheritance sequence.
1.4 Store the obtained output to another file.

3.2 Moulding: Mould the entire code by removing the comments, blank lines, break the line such that only one statement is included in each line.
/* Algorithm for the Moulding */
Input: Restructured output file
Output: Moulded java code stored in a file
1. [Moulding of the java code]
1.1 Remove the comment lines, if any.
1.2 Remove the blank lines, if any.
1.3 Break the line, if multiple statements are present in a single line or if selection statements (if, while, do-while, for) are encountered. 1.4 Assign line number to each physical statement of the program. 1.5 Store the obtained output in a file.
3.3 Designing: This phase includes designing various UML diagrams- class diagrams, object diagrams, use case diagrams, sequence diagrams, activity diagrams.
/* Algorithm for Designing */

Input: The moulded output stored in a second file.
Output: Designed UML diagrams for the input source code which includes class, object, use case, sequence, activity diagram.

1. [Designing class diagrams]

1.1 Search for the keyword "class" in the code, and place the class name in the first compartment. Search for the data types of the variable in the code and place it in second compartment.
1.2 If a verb is encountered in the class, for example print(), place it in the third compartment of the respective class diagram as methods of the class.
1.3 Precede the attributes and functions in the class diagram with the visibility in the java code (+ public, - private, # protected, _ static). The return types and parameters of the method should also be specified in the class diagram.
1.4 Abstract class names should be in italics. Identify association, aggregation, generalization and multiplicity.
1.5 Repeat the steps and construct the class diagrams for every classes encountered in the code.

2. [Designing object diagrams] 2.1 Extract the object names in main, which can be one of the two format in the code: classname objectname;
(or) objectname = new classname(); 2.2 Object diagram is of two compartments, object name : class name, attributes : values respectively. 2.3 For every object instantiation in the code, object diagram is drawn with its attributes and values for each object. 2.4 The object diagrams of a class are connected in the same way as of class diagram.

3. [Designing Use case Diagrams] 3.1 One object represents one actor, which

is represented by a stickman. 3.2 The methods performed on objects which describe functionality should be included in an elongated ellipse, each representing one use case. 3.3 The use cases can be connected to objects (actors) as per the methods invocation. 3.4 The set of use cases formed can be presented in a rectangular set, with the use case name specified at the left end corner.

4. [Designing Sequence Diagrams] 4.1 The problem statements in the body of the function can be treated as a message which represents object communication. 4.2 The actors (objects) in the use case are written in a rectangular box preceded with colon(:). 4.3 The vertical line named *lifeline* is drawn from top to bottom representing the time span between the function call from constructor to destructor function of the object. 4.4 The period of time for an object execution is represented as a thin rectangle called *activation* or *focus of control*. 4.5 Methods called perform operation on objects. 4.6 Messages can be simple, asynchronous and synchronous. Solid arrow head represents synchronous calls , open arrow head represents asynchronous calls and a dashed line represents reply message. 4.7 When a destructor function is invoked for the object, the object is destroyed represented by 'X' mark on the lifeline.

5. [Designing Activity Diagram] 5.1 As the objects are manipulated, activity diagram is drawn according to the flow of control. 5.2 The start of flow of control is represented via a solid circle. 5.3 The elongated ovals shows activities and arrow shows sequencing. 5.4 If conditions are encountered in the code then it can be represented in a diamond with the successor activities represented by two arrows respectively for true or false. 5.5 Several activities which are performed concurrently are included via synchronization bar-a heavy line with one or more input arrows. 5.6 The end of the activity is represented with a bull's eye.

4. CASE STUDY

The proposed procedure is implemented for number of 'java' programs and the results we got are correct and complete. The sample 'java' program depicted in figure 2 is the output of the methodology implemented in our paper.

[A sample 'java' program]

```
class Rectangle
{
int length, width;
void getdata(int x, int y)
{
 length=x;
width=y;
 }
  int rectArea()
{
int area=length*width;
return(area);
 }
}
 class rectArea
{
 public static void main(String args[])
  {
 int area;
Rectangle rect=new Rectangle();
rect.getdata(x, y);
area=rect.rectArea();
 System.out.println("Area="+area);
  }
 }
```

Output:

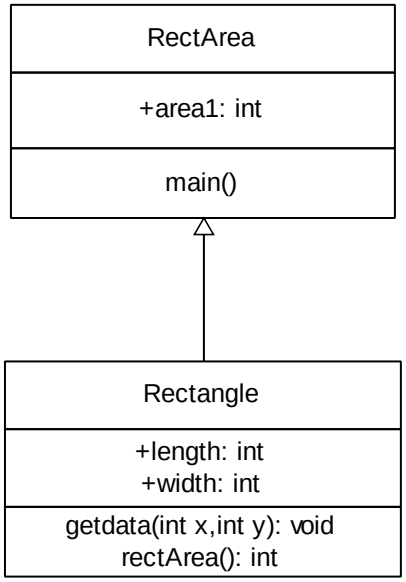

Figure1:class diagram.

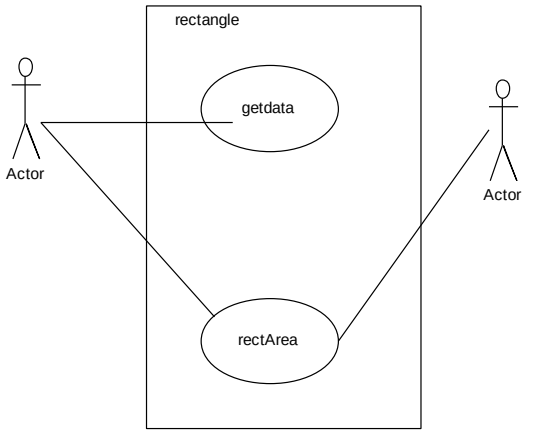

Figure2:usecase diagram.

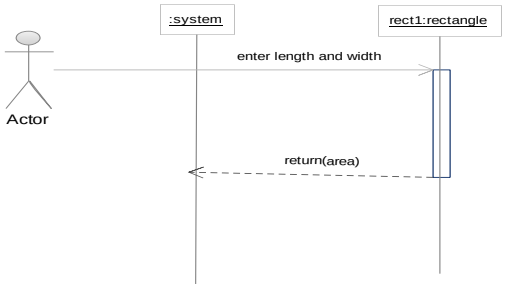

Figure3: Sequence diagram.

5. CONCLUSION

This paper presents an automatic tool that abstracts the required elements from the program and builds the UML diagram format. This procedure is very simple as the extraction is directly from the code itself. This tool describe the techniques to inspect java source code using annotations and knowledge base about the semantics of pre-defined container classes and their access operations. It concludes Reverse engineering support that analyzes java source code and tries to create the corresponding UML model format. Five basic structural and behavioral diagrams are retrieved. The constituents needed for the generation of UML model format are extracted directly from the code itself, overcoming the limitations of the few already implemented tools in the market, for instance, Rational Rose. Our proposed tool is very beneficial in Modeling and Designing the problem. Its simplicity and ease in implementation makes the tool quite interesting and more advantageous when compared to other existing tools in the market.

6. REFERENCES

[1] *Generating UML Diagrams from Natural Language Specifications* by Priyanka More and Rashmi at *International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 1– No.8, April 2012*.

[2] *Generating UML Diagrams from Task Models* by Shijian Lu, Cécile Paris, Keith Vander Linden and Nathalie Colineau. Department of Computer Science, Calvin College, Grand Rapids, MI 49546, USA. kvlinden@calvin.ed

[3] *Recovering UML Diagrams from Java Code using Patterns* by Jörg Niere, Jörg P. Wadsack Albert Zündorf.Department of Mathematics and Computer Science University of Paderborn Warburgerstraße 100 33098 Paderborn, Germany zuendorf@uni-paderborn.de

[4] *"The Unified Modeling Use Guide"* book by Grady booch, James Rumbaugh, 3rd ed published by Pearson Education in South Asia.

[5] *UML for Java Programmers*, book by Robert Cecil Martin, Object Mentor Inc.Prentice Hall, Englewood Cliffs, New Jersey 07632.

[6] *An Ameliorated Methodology for the design of Object Structures from legacy 'C' Program*, Dr. Shivanand M. Handigund and Rajkumar N. Kulkarni, BITM,Bellary.

[7] Michael R Blaha and Jamesh R Rumbaugh, *Object Oriented and Design with UML*, second ed. Addison-Wesley, 2005.

[8] *A Systematic Study of UML Class Diagram Constituents for their Abstract and Precise Recovery* by Yann-Gael Gu'eh'eneuc.

[9] http://www.uml-diagrams.org/. An official website of UML diagrams.

[10] *Formalizing class diagram in UML* published in Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on 15-17 July 2011