IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Knowledge Based Visualization Tool for Object Oriented Concepts with UML

Kamna Singh Computer Science & Engineering
Ajay Kumar Garg Engineering College, Ghaziabad, India

Abstract

In the previous research in visualization of object-oriented concepts determined and emphasis on both adaptive and non-adaptive approach for programs. Visualization tool helps to learner to understand concepts easily and effectively. This paper discusses the technical issues and how the visualization tool can be used in teaching and learning ofprogramming. Our approach is to explore/adopt innovative ways of teaching and assessing which engage students in effective learning and acquisition of programming skills. Proposed approach describes that if user enters the source code into the visualization tool that it will be converted into theclass diagram and sequence diagram as per user requirement. We also describe algorithm to generateappropriate class diagram and sequence diagram to the object-oriented language source code which is entered by the user. We propose several algorithms to convert object-oriented source code into their equivalent UML diagrams (sequence and class diagram) and then achieve a reverse engineering goal that is source code is converted into the design documents.

Keywords: *Class diagram Generator(CDG), Sequence Diagram Generator(SDG), T2IG(Text to image generator, Sequence Diagram Instrumenter (SDI)*

1. INTRODUCTION

At present policy to generate class diagram and sequence diagram when the user enters the source codein object oriented programming language(c++) as input [1], In our approach to generate the class diagrams and sequence diagram, static analysis to record the classes and their relationships between classes are implemented in C++ through colon (with class name is recorded. Currently there are several modelling tools that include code generation options. In addition, a variety of programming languages such as Java, C#, C++, Ruby, Delphi, Perl, Python, Objective-C, Ada and PHP can be chosenas a transformation target platform.[6]. In this paper, we examine the transformation of source code into a class diagram and sequence diagram.

2. LITERATURE REVIEW

As a result of increasing technological diversity, more attention is being focused on model driven architecture (MDA), and its standard – UnifiedModelling Language (UML). UML class diagrams require correct diagram notation mapping to targetprogramming language syntax under the framework of MDA. [6]. UML is a modelling language thatmost developers employed during design phase. UML provides various types of diagrams used forspecifying both the structure and the behavior of systems. During the development process, modelsspecified by these diagrams are eventually transformed into corresponding code.[8]. Reverse Engineering. Reverse engineering is a method that transforms source code into a model. ForUML builds upon and expands some existing reverse engineering work.[9]

The reverse engineering allows addition of further plans, realizing patterns, and possibly 4862nnotatr intertwining rules to enable a plan recognizer to detect the actual realization that can be found in the code. Thus, the reverse engineer not only has to provide the plan pattern for the actual realization of aplan but also the transformations and the justifications for applying them. As an alternative approach,the same effect can be achieved by using small transformations backwards and abstract the code step by step. So that the ends result of which is an abstract plan corresponding to the given code. In essence,the transformations applied backwards "jitter" the code into a form in which canonical plans are moreeasily found. Such an abstract plan may already exist in the system or may be new domain knowledgedeveloped during the reverse engineering process. Doing so then avoids the need for repeatedly reverseengineering the system over its lifetime, caused by the continuous modification of its code.[7]
The need of automated tools to generate source code from visual models has been increasing over timeand a lot of work has been done on the automatic transformation of UML models to source code (Thongmak & Muenchaisri, 2002). Taking this into account,[10]

3. PROPOSED MODEL:

In proposed algorithm there is a reverse engineering tool which extracts design documents from the object oriented language based source code. Here we took design documents as a class diagram and sequence diagram. This visualization tool gives a pictorial representation from the given source code.

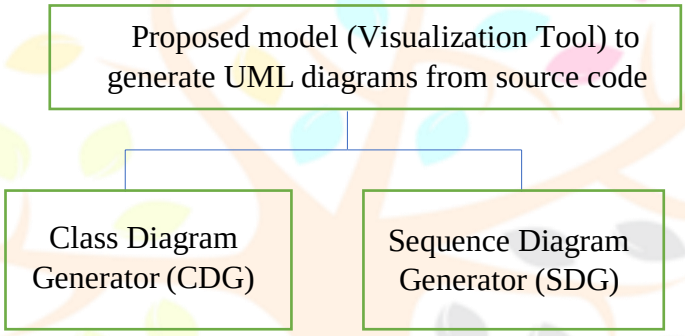

Figure 1: Visualization tool Components

Basically, this visualization tool is designed to reverse engineer the UML 2.0 representative viz class diagram and sequence diagram automatically from the object oriented language(c++) source code.it mainly consists of two main components:

a) Class diagram Generator (CDG)

b) Sequence Diagram Generator (SDG)

Features of Visualization Tool:

1. Visualization tool provide a diagrammatical approach to get design documents.
2. It improves teaching methodology.
3. It transforms source code which is written in any of the object-oriented language to UML diagrams.
4. It analyses program code and give ease of understanding.
5. It can implements blooms taxonomy of teaching methods with analyse, apply, understand andremember attributes.

3.1 Generation of Class Diagram

The **class diagram** is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models

into programming code. **Class diagrams** can also be used for data modeling. We proposed an algorithmto direct generation of class diagram from the given source code.

Steps to generate class diagram is given below:

Step 1: Extract the classes and their data members. The different classes and their data members fromthe input object-oriented language (c++) source code files are recorded.

Step 2: Eliminates those data members which are not classes. We keep only those data members whichare indeed a class. This removes false pairs from the recorded classes.

Step 3:

Convert the representation into UMLGraph5.1.

The remaining pairs give the relationships of the classes existing in the given input c++ files. This fileis converted into the representation of UMLGraph 5.1

Step 4: Pictorial representation of the class

diagram.The written file in UMLGraph5.1 syntax when executed gives a pictorialrepresentation of the class diagram.

3.1.1 Class Diagram Generator

Class Diagram Generator (CDG) component of the tool contains the following subcomponents.

The subcomponents process , the input files into intermediate representations and finally generate theclass diagram representations.

Pseudo code for CDG is as follows:

Step 1. Get the source code written in object-oriented language.

Step 2. Extracts the tokens from the source codeWith the help of parsing technique.

Step 3. Derived the relationship between base class and derived class by Class Tree Generator (CTG).**Step 4.** Class box pruner (CBP) prunes all the lines which are not belongs to class such as comments etc.

Step 5. Class structure generator (CSG) generates the class with data members and member function.

Step 6. T2IG(text to image converter) combines CBP and CSG(i.e. step 4 and step 5)

Step 7. Class diagram generated by UMLgraph 1.5With the T2IG.

The CDG is viewed as a collection of subcomponents as listed below(figure 2).

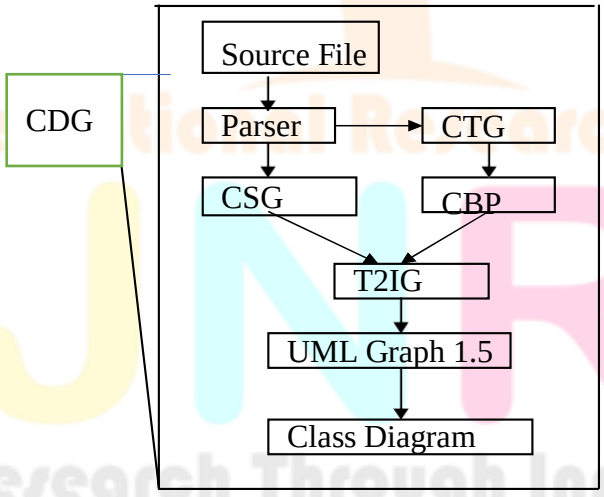

Figure 2.1: Class Diagram Generator

3.1.1 Working Details of CDG Components

Each component takes a c++ program file or an intermediate representation file and process further tofinally generate the class diagram. The description of the subcomponents in the CDG as follows: **Parser**: we are using C++ source files. It generates a parse tree for the given c++ source code.

Parsing. The object-oriented code is parsed by the
parser including grammars and libraries for performing translation actions.it generates parse language specifications like syntax, a notation for formally describing programming language syntax, and generate the library to parse the specified language. It distinguishes three compilation phases: lexical analysis, parsing, and tree walking.[9]

The parser query is used to parse c++ source programs into an internal tree structure. The following example explains how the parse tree matches the grammer.

The following is the grammer rule for "if Statement" ifStatement():

{

"If""(Expression()")"Statement()["else"Statement()]

}

This rule says that an if Statement is either an "if" keyword followed by a statement and expression,and/or it's followed by a "else" keyword followed by a statement.the parse tree will look like the following figure 2.2

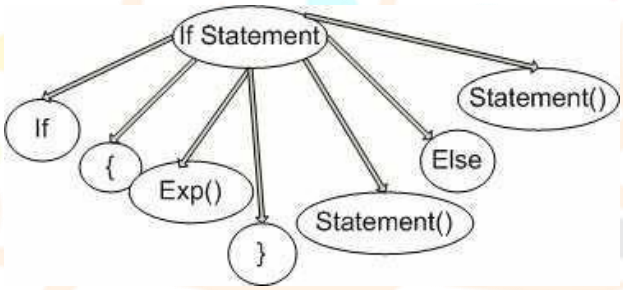

Figure 2.2: Example of parse tree structure

It denotes that parser generates tokens after analysing source file. Tokens can be statements, identifier, keywords, functions etc.

3.1.2 Working Details of CBG Components

The working of the CBG component is to extract the classes from the generated parse tree.the input isthe parse tree generated by the parser.the output generated is intermediate class Box which is two tuple<class x><class y>. Where class X and Class Y are the given classes in the input C++ source files.The relation between the two classes is that, class X is the base class and class y is the derived class isthe implementing colon followed by class name.

3.1.3 Class Structure Generator(CSG)

It gathers the structural information of the different classes present in the parse tree. The output is threetuples i.e. class name ,data member, member functions.it stores the individual class structure information.

The CBG(Class Box Generator) traverses the parse tree and registers the class name and its data members in the class box. The class box depicts the associations between the different classes. The box

contains two fields , the entries comes from multiple parse tree generated by the parser after readingthe source files. The structure of the CBG is
<Class X><Class Y>,
The algorithm for filling the entries as follows.For each class X do
For each data member M of X doInsert an Entry<X,M> in CB
For each class X do
If class x followed by colon andanother class with modifier
Insert an entry<X,M> in CB

3.1.4 Class Box Pruner(CBP)

The intermediate class table generated from CBG has some entries of data members that are not classes.Such entries have to removed. The output is the Class Box.
The Entries of <Class y> field in the class box are cross verified with the entries of <class X> field in the Class box. Those entries of <Class Y> which do not have a reference in <class X> are removed as they are not classes but other program elements. The Algorithm for pruning the Intermediate class Boxas follows:
For each value in <Class Y>do isClass:=false
for each value in <Class X>do
If (<ClassX>=<Class Y>)
isClass:=trueif(isClass<>true)
delete the entry<classX><ClassY>from the class box
isClass:=false

3.1.5 T2IC(Text to Image Converter)

This component reads the class Box and class structure information and writes code in the syntax of UML Graph5.1.The output is class diagram. It is the representation of class diagram.UML Graph 5.1:This is a open source software that generates the pictorial representations which is the class Diagram.

3.2 Sequence Diagram Generator

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.
The sequence diagram is drawn on the timeline. The static analysis is therefore not sufficient to record the message passing occurring between the objects of different classes. We use here dynamic analysisto record the live message passing, several runs are required to get a comprehensive set of message passes. It ensures nearly every message call between the objects has been exploited.

3.2.1 Generation of Sequence Diagram

Our approach to generation of sequence diagram:

Step 1: Instrument the Source codeStep 2: Generate the execution tree Step 3: Prune the execution trace
Step 4: Consolidate the pruned execution tracesStep 5: Convert the representation into UMLGraph5.1
Step 6: Pictorial representation of sequence diagram.
The SDG(Sequence Diagram Generator) is a collection of sub-components as shown below (Figure 3)

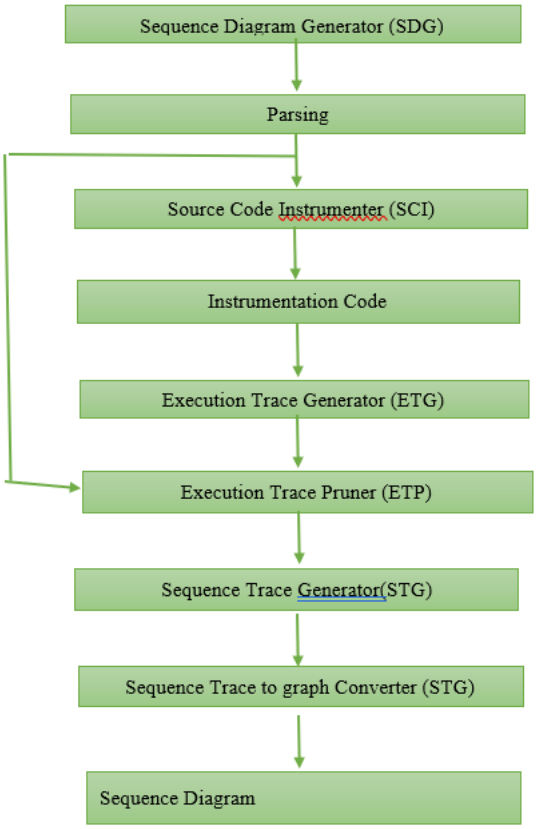

Figure 3: Sequence Diagram Generator

Execution Trace Pruner (ETP)

The description of the sub-components in the SDG(Sequence Diagram generator) is as follows- **Parser:** We are using a third party object oriented language package Query for parsing the C++ source code files. It generates a parse tree for the given source code.

SCI(Source Code Instrumentor): It annotates the nodes a in the parse tree and generates an instrumented code as output.

ETG(Execution Trace Generator): This component takes the input set for executing the program ina text file and for each input set generates a corresponding execution trace file.

ETP(Execution trace Pruner): The execution traces generated may contain some control statementsthat don't invoke any class methods. In other words, these control statements don't guard any methodcall. The ETP components prunes such irrelevant statements.

STG(Sequence Trace Generator): The individual traces are combined to form a single trace which is known as the sequence trace.

ST2GC (Sequence Trace 2 Graph Converter):This component reads the sequence trace and writes a code in the syntax of UML Graph1

5.1 .The output is a diagram.It is the representation of the sequence diagram.

3.2.2 Working details of SDG components:

3.2.2.1 SCI (Source code Instrumenter)

The instrumentation of the code is done in order to extract the following information: The method name, calle object and class,caller object and class.The entry and exit of the method calls. The entry and exit of control statementsif, if-else, while, for The condition part of the control statements The probable operator (ALT,OPT,LOOP)for the control statements. The control statements and method calls are numbered inorder to resolve their sequence in the final sequence trace. The format of the instrumentation:

For control:

<StatementNo><probableOperator><ConditionType>

<ConditionStatement><StatementNo</Probabl eOperat or></Conditiontype> For method calls:

<methodname><calleeClass><CallerClass>
</methodname>
Algorithm to instrument the source code: The source code is 4867nnotated using 4867nnotated methosReturn type :void
Input Parameters:root of the parse tree orsubtree .

Source code Instrumenter Algorithm:

Step1: Get the Type of the node

Step2: Case I: if the node is forstatement call handleif()

Case II:If node is forStatement handlefor()

CaseIII:If node is whilestatement call handleWhile()

CaseIV:if node statementexpression call handleMethod()

3.2.2.2 ETG Execution Trace generator)

This component will read line by line the input file and execute the instrumented program to generatethe execution traces interms of
files.the algorithm to generate the execution traces in terms of files.the algorithm to generate theexecution traces from the instrumented source code file as follows: Read the input file line by line and
do For each input line execute instrumented source code files.For each execution, execution trace fileswill be generated.

3.2.2.3 ETP (Execution trace Pruning):

The execution traces must be pruned to remove unnecessary statement not relevant to the sequencediagram. For example, consider the following traces:

<1><ALT<If><(If-condition 1)>
<3><LOOP><WHILE><(while-condition 1)>
<function1()><Class 1> <class2>
</\function1()>
<3><LOOPS><WHILE>
<1></ALT><If>

Figure :4a

<1><ALT<If><(If-condition 1)>
<3><LOOP><WHILE><(while-condition 1)>
<function1()><Class 1> <class2>
</\function1()>
<3><LOOPS><WHILE>
<4></ALT><(If-Condition 1)>
<4></ALT></If>
<3></Loop><While>
<1></ALT></If>

Figure :4b

in the figure 4a all the control statements are guarding some or the other method calls .i.e. they influencethe calling of the methods. These statements are relevant to the sequence diagram. In the Fig 4(b) , thetrace number<4> does not influence any method. In other words, it does not guard any method calls. Such statements are removed or pruned.

3.2.2.4 Sequence Trace Generator (STG)

The individual traces obtained present a scenario. We need to consolidate them for the completesoftware system. During the consolidation the probable operator mentioned in the trace can change mainly for ifelse statements.For example ,consider the following sample code and its corresponding traces.

If(cond1){
Function1(); } else{ a=b; }

Figure 5(a)

<1><ALT><If><(Cond 1)>
<function 1()><Class 1><Class 2>

</Function 1()>
<1></ALT><If>
Figure 5(b)
<1><ALT><ELSE><(!Cond 1)>
<1></ALT><ELSE>
Figure 5 c

The trace in fig 5 does not guard any method. so it has to be removed. Once it is removed, during theconsolidation we find that the if statement does not have else part any longer. So, the probable operatorfor the if Statement is modified to OPT.

4. CONCLUSION

Here we discuss several proposed pseudocode and algorithm to generate sequence and class diagram from the given source code to produce an experimental model.
So far, we have addressed reverse engineering of UML representations. We have implemented these representations and analyse them. The analysis mainly concerns with the integrity of the representationswill are able to generate. The main goal of this paper is to achieve reverse engineering concepts that issource code written in any of the object-oriented language is converted into the design documents. Here we are mainly concern in sequence and class diagram as a design document. In this paper we verify and validate for the reverse engineering tool as a knowledge base visualization tool with the helpof jsp and tomcat server. The proposed model reduces amount of time that maintainers need to spend understanding the programs that are being maintained.

5. FUTURE SCOPE

The future scope is the area is to take design patterns as well, as they deliver some more information regarding the software system which will helpful in testing the entire system thoroughly looking into its various views. We can also explore to implement the other uml diagrams such as activity diagram,state chart diagram, Use Case, component and deployment diagrams.so that we can transform object-oriented language source code into abovesaid uml diagrams.

REFERENCES

[1] Jin-Sung Kim,Chun-Sik Yoo,Mi-Kyung Lee,and Yong-Sung Kim,"Object Modeling of RDF Schema for converting UML classDiagram" proceedings ICCS Conferences 2005,LNC 348,p31-41 2005 Springer-Verlag Berlin Heidelberg 2005

[2] Maximo Lopez S ,Armando Alfonzo G,Joaquin Perez O,Juan Gabriel Gonzaalez S,Azucena Monster R"A meta model to carry out reverse engineering of c++ code into UML sequence diarams", Proceedings IEEE Mechanics Conference on Electronics Robotics and Automative Mechanic Conference(CERMA '06) 0-7695- 2569-5/06 IEEE 2006.

[3] David j Pearce ,james Noble," Relationship Aspects"Proceedings AOSD 2006, Mar 20-24Conference in Bonn,Germany,2006

[4] Niko Myller ,Roman Bednarik and Andres Moreno "Integrating Dynamic Program Visualization into BlueJ: the Jeliot 3 extension", Proceedings seventh IEEE International Conference on Advanced Learning Technologies(ICALT 2007),0-76952916-X/07 2007

[5] G.P. Nikishkov," object Oriented Design of a finite element code in java" Computer Modelling inEngineering and Sciences, 2006,p-11,81-90.

[6] Janis Sejans, Oksana Nikiforova, "Problems and Perspectives of Code Generation fromUML Class Diagram", Scientific Journal of Riga Technical University, vol 47

[7] Ramandeep Singh ,"A Review of Reverse Engineering Theories and Tools", International Journal of Engineering Science Invention ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726 www.ijesi.org Volume 2 Issue 1, January. 2013 p-35-38.

[8] Mathupayas Thongmak, Pornsiri Muenchaisri," Design of Rules for Transforming UML'Sequence Diagrams into Java code", Conference Paper · February 2002
DOI: 10.1109/APSEC.2002.1183052 · Source: IEEE Xplore

[9] Maryam . Mukhtar, Bashir S. Galadanci,"Automatic Code Generation From Uml Diagrams: The State-Of-

The-Art",Science World JournalVol 13(No4) 2018 ISSN

[11] Ra'Fat AL-msie'deen ,"Visualizing Object-oriented Software for Understanding and Documentation", (IJCSIS) International Journal of Computer Science and Information Security, Vol. 13, No. 5, 2015

[12] Ahid Yaseen , Hamed Fawareh , "Visualization and Synchronization of Object-oriented Programs using Re-engineering Approach", International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 8 (2019), pp. 1239-1246