International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:5, 2007

Translator Design to Model Cpp Files

Er. Satwinder Singh, Dr. K.S. Kahlon, Rakesh Kumar, Er. Gurjeet Singh

***Abstract*—** The most reliable and accurate description of the actual behavior of a software system is its source code. However, not all questions about the system can be answered directly by resorting to this repository of information. What the reverse engineering methodology aims at is the extraction of abstract, goal-oriented "views" of the system, able to summarize relevant properties of the computation performed by the program. While concentrating on reverse engineering we had modeled the C++ files by designing the translator.

***Keywords*:-** Translator, Modeling, UML, DYNO, ISVis, TED

I. Introduction

WE all want to build software that makes things better, avoiding the bad things that lurk in the shadow of failed efforts. To succeed we need discipline when software is designed and built. We need an engineered approach.

The usefulness of an abstract system model was already recognized in the 1970s, when *structured methods* were proposed as software development methods. These methods offered Entity-Relationship diagrams [1] to model the data aspect of a system, and data flow diagrams or functional decomposition techniques to model the functional, behavioral aspect of a system. The main drawbacks of these structured approaches were the often missing horizontal consistency between the data and behavior part within the overall system model, and the vertical mismatch of concepts between the real world domain and the model as well as between the model and the implementation. As a solution to these drawbacks, the concept of an *abstract data type*, where data and behavior of objects are closely coupled, became popular within the 1980s. This concept then formed the base for the *object-oriented paradigm*. In particular, object-oriented languages like C++ or Java have become the de facto standard for programming [10]. The same holds for the analysis and design phases within a software development process, where object-oriented modeling approaches are becoming more and more the standard ones. The success of object-oriented modeling approaches was hindered in the beginning of the 90s due to the fact that surely more than fifty object-oriented modeling approaches claimed to be the right one. This so-called *method war* came to an end by an industrial initiative, which pushed the development of the meanwhile standardized object-oriented modeling language UML (Unified Modeling Language) [8]. But, despite the fact that it has been standardized, UML is still a moving target.

II. Related Work

TIMO RAITALAAKSO in Dynamic Visualization of C++ Programs with UML sequence Diagrams introduced DYNO [6] system. DYNO collects runtime data of a subject non-concurrent C++ program. Information about classes and methods has to be collected and certain objects must be added into C++ code to collect the data about the dynamic behavior of a program.

Dyno operates only on WINDOWS NT 4.0 environment because TED Support only to it. It does not operate in other environment. It uses the TED software engineering environment to visualize the sequence diagram. It also does not support more than one source file and slicing before instrumentation. The exception is not handled by the it. When an exception is thrown in the C++ program the execution is moved to the exception handler. DYNO can not produce information from the executed exception and therefore loses track of the position where the program code is been presently executed.

Robert J. Walker *et al,* [3] have developed an off-line, flexible approach for visualizing the operation of an object-oriented system at the architectural level. This approach complements and extends existing visualization approaches available to engineers attempting to utilize dynamic information. There approach abstracts two fundamental pieces of dynamic information: the number of objects involved in the execution, and the interactions between the objects. They visualize these two pieces of information in terms of a high-level view of the system that is selected by the engineer as useful for the task being performed. Their approach allows an

1. Unfamiliar system to be studied without alteration of source code,
2. Permits lightweight changes to the abstraction used for condensing the dynamic information,
3. Supplies a visualization independent of the speed of execution of the system being studied,
4. Allows a user to investigate the abstracted information in a detailed manner by supporting both forwards and backwards navigation across the visualizations.

ISVis [5] uses the static information of a program. It depend on user that in which part of program he is interested in. user can choose single part of his interest. ISVis produces the whole scenario so that the chosen participants interact into the right part of window. By selecting a part from the larger trace it is possible to zoom a smaller part of a trace in the main window. By zooming more actual method names called are drawn in the view. In this dynamic information is missing. And in this the interface is not the GUI which make it difficult to understand.

Koskimies Kai, Mössenböck Hanspeter [2]. Scene develop a Scene which is a reverse engineering tool that uses scenarios to visualize object-oriented program execution. It uses source code instrumentation technique to produce event traces. With Scene, it is possible to browse not only scenarios but several different kinds of documents like class diagrams, source code, class interfaces and call matrices. Scene also allows the user to

investigate the inner state of an object in a specific point of execution.

III. Translator Design

A. LOGICAL DESIGN

Logical design gave the conceptual view of the solution. In this paper the conceptual view of the translator was described. As described in the problem analysis to design a translator there were four components which were to be designed. These include:

1. Packages
2. Variables
3. Methods
4. Classes

1.Packages: Translator should give the list of in build and user build packages or header files used in source CPP file.

2.Variables: Variables declared in the CPP file should be listed. Data type of each variable will be represented in the notated form. A special and unique symbol should be used for notation of each variables data type.

3.Methods: Translator should list the user defined functions which were declared in source file. It should also list the return type of methods in notated form. Notations used for each return type key word should be same as in declaring variables. Definition of each method should also be output. This definition includes the variable name declared in it including their data type in notated form. If translator found any data type from the following list then it was notated with special symbol as marked against each data type.

TABLE I
NOTATION FOR VARIABLES AND FUNCTIONS

S.No	Data Type	Notation
1.	Int	$
2	Int*	$*
3	Int*	$*
4	Long	$
5	Short	$
6	Float	@
7	Float*	@*
8	Float**	@*
9	Double	@
10	Char	&
11	Char*	&*
12	Char**	&*
13	Void	0

4.Class: In this translator listed those item which are encapsulated in the class e.g. data member, member functions and inherited classes. It should also define the scope of each data members and member functions. It should list the data type or return type of data member or member functions respectively with their name. The notation used for specifying the data type and return type would be same as in Table I. Notations for the scope of variables and methods were followed.

TABLE II
NOTATIONS FOR SCOPE OF VARIABLES AND FUNCTIONS

S.No	Scope	Notation
1.	Public	-
2	Private	+
3	Protected	=

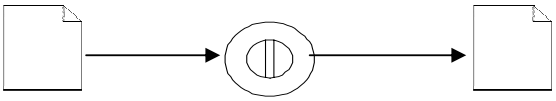
B. OUTPUT

Any amount of work which has been done can be mirrored in the output. So if the output was not well designed or required results were not produced then all work can be damaged. Out put of translator was designed very carefully so that it can be understood by the beginner or layman. The output of translator will be in text form. In the output first it display the file use as source file in the box with its path and current directory.

Out put screen list the packages used in the source CPP file. List was headed by the message "*Packages used Are:"*. After this message list of packages will be appeared. There will be one header file or Package in one line. Each header file was ended with extension *".h"*.

Then if global variables and methods were declared after the packages in the source file then these will be shown with their data type and return type respectively. Data type and return type will be notated as described in the Table I. These variables and methods were globally declared in the source CPP file. Variables will be listed in the same order as in the source file. If there will be any initialization of variables this would also be displayed as it will be. If there will be more than one variable of same data type declared in one line in source file then all these variables will be notated in front of variables name as per Table I. Global variables or methods can be declared any where in the source file. So these are displayed in the design where it was defined.

Below figure shows that how the translator works on inputting the CPP file. After input of CPP file translator will model the output in text form and store the modeled design in new text file. From where we can again read it to analyze for further work.



CPP File **Translator**
Text(Modeled) File

Fig. 1 Translator working

Output model of class was designed in a manner such that it should be easy to understand. After the message of *"Class Started"* ,name of the class will be displayed. After that if there will be any inherited classes in the current class then the same will be listed with their scope. After the message of "*Inherited Classes"* name of the classes will be appeared. Name of the class will be prefixed with their scope which will be notated as per notations in Table II. After this screen will be divided into three columns *"Scope, Data type , Variables and Methods"* . Under the scope column, scope of each variable or method will

be listed. Scope will be notated as per Table II. Every time scope will not be notated for each variable or Method until new scope will be encountered.

FOR EXAMPLE:

```
Class A
{Public:
        Int a;
Int box();
                        Private :
                                Int b;
                                Int square();
 };
```

Scope of variable *'a'* and method *'box()' is public* and this will be notated once with sign '+' . Variables/methods after '+' sign will be public until new scope will be encountered. As in above example when translator reads *"Private:"* it notate them as "-" and variables and methods under this sign will be private until end of class is encountered. Output model of above class will be as follow:

```
       Class  A
             Declaration Of Variables And
Methods
       Scope Data Type     Variables/Methods
       +
          $                a
          $                box()
       -
                     $                    b
                     $
       square()
       End Class
```

In the column of Data type, data type or return type of Variables and Methods respectively will be notated as per Table I. Data type or return type will be notated every time under this column e.g. as in above model of class A, under column Data type, data type of variable *'a'* is notated with special symbol $(dollar) and return type of method *'box()'* is again notated with symbol (dollar).

At the last in the column of "V*ariable/Method"* name of variable and method will be displayed as it will be in source file. If variable had some initialized value then that will also be displayed as it will be. It will not be necessary that each line had one variable in one line under column "*Variables/Methods"*. If there will be more than one variable in one line then all these variables will be of same data type as notated in the column "*Data Type"*. This depends on the number of variable in one line in the source file.

When ever the function definition will be defined in the source file at same place it will be displayed in the design. In the function definition the list of variables declared will be displayed with their data type notations as per Table I. Function definition will not show the what ever coding or concept running in it e.g. any loop, control statement, any inbuild function etc. definition will only shows the list of variables declared.

A complete Example of one CPP (C++) file is shown in Table III. Where first column contain the code of CPP file and second column contains the model of the corresponding file. This also shows the inherited class "a" in "Class b" with public represented by the plus (+) sign. This also shows the definition of each function. At the end of model column. Empty definition shows that there no new variable is define in the functions (getdata(), putdata(), due(),display())

TABLE III
COMPLETE EXAMPLE

Cpp file	Model

#include<conio.h>
#include<iostream.h>
>
class a
{ private:
int i;
public:
void getdata();
void putdata(); };
class b : public a
{ int j;
public:
void due();
void display(); };
void a::getdata()
{ cout<<"enter roll
no.";
cin>>i; }
void a::putdata()
{ cout<<" roll no.
is"<<i<<endl; }
void b::due()
{ a::getdata();
cout<<"enter due
amount.";
cin>>j; }
void b::display()
{ a::putdata();
cout<<" Due is"<<j;
}
void main()
{ clrscr();
b derived;
derived.due();
derived.display();
getch(); }
	Packages in use are:
	1 conio.h
	2 iostream.h
	Class Started a
	Declaration of variables and methods
	Scope DataType Variables/Methods
	-
	$ i
	+
	0 getdata()
	0 putdata()
	End of Class
	Class Started b
	Inherited Classes +a
	Declaration of variables and methods
	Scope DataType Variables/Methods
	-
	$ j
	+
	0 due()
	0 display()
	End of Class
	FUNCTION DEFINATION OF
	getdata()
	FUNCTION DEFINATION OF
	putdata()
	FUNCTION DEFINATION OF
	due()
	FUNCTION DEFINATION OF
	display()

IV. CONCLUSION

Translator proposed in this paper gives the design of underlying system which is coded in C++ library. This will model the CPP files and analyze object oriented nature of the file. The main items which were abstracted from the code were classes and their relationship, header files used in source file, variables declared global and the definition of the function used

global or encapsulated in the classes. Notations were devised in translator where the main consideration about this was that the notation used was as near as UML.

There is lot to be done which makes the translator stronger and more active. Suggestions are translator should support more than one source file which should make it more active. The exceptions are not handled by the translator. Try and catch blocks are not instrumented in translator. Ways to produce information that includes exception handling should be investigated and implemented into translator. Work is going on to make this translator suitable for the any object oriented language. A part from C++.

Interface can also help to make it more understandable and easy to use. Interface can be designed with the existing visualization tools like jGRASP and TED. Some Interface can also be designed in tree form where output can be expand or decompress by a single mouse click.

REFERENCES

[1] P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems, 1(1), 1976, 9-36.
[2] Koskimies Kai, Mössenböck Hanspeter. Scene: Using Scenario Diagrams and Active Test for Illustrating Object-Oriented Programs, Proceedings of the 18th International Conference on Software Engineering (ICSE –96), ACM Press, 1996, pp.366-375.
[3] Walker Robert J., Murphy Gail C., Bjorn Freeman-Benson, Wright Darin, Swanson Darin, and Isaak Jeremy. Visualizing Dynamic Software System Information through High-level Models,Proceedings of the Conference on Object-Oriented Programing, Systems, Languages, and Applications (Vancouver, British Columbia, Canada; 18–22 October 1998), ACM SIGPLAN, pp. 271–283, 1998. Published as ACM SIGPLAN Notices, 33(10), October 1998.
[4] Kazman Rick, Carriere Jeromy, View Extraction and View Fusion in Architectural Understanding, Proceedings of the Fifth International Conference on Software Reuse (ICSR5), 1998.
[5] MORALE project, ISVis tool, Available: http://www.cc.gatech.edu/morale/tools/ .
[6] TIMO RAITALAAKSO, Dynamic Visualization of C++ Programs with UML Sequence Diagrams, Master of Science Thesis.
[7] Müller Hausi A., Understanding Software Systems Using Reverse Engineering Technologies Research and Practice, Available:http://www.rigi.csc.uvic.ca/UVicRevTut/F4rev.html, Department of Computer Science, University of Victoria, 2000.
[8] Booch Grady,Rumbagh James and Jacobson Ivar. The Unified Modeling User Guide-5th Indian Reprint,Addison Wesley Longman (Singapore) Pte.Ltd,2001.
[9] Deitel H.M. and Deitel P.J. .C++ How To Program 4th edition. Pearson Education (Singapore) Pte. Ltd. 2004.
[10] Gregor Engels, Object-Oriented Modeling: A Roadmap, Available : www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalengels.pdf

Er. Satwinder Singh born in Amritsar (India) on 08 july 1980. Author is M-tech(Information Tecchnology) from guru nanak dev University Amritsar, Punjab (India) in July 2002. Major field of study is modeling of object oriented language and compiler design. Working in this field from last two years.

He has work experience of two year teaching . He also has one year research experience during the post graduation. He had participated in various national conferences. His work has been published in various national proceedings. Currently he is working as a LECTURER in Rayat Institute of Engg. and Information technology, Rail Majra, near Ropar, Nawan Shahar, Punjab(India) E-mail : satwinder_man@rediffmail.com

Dr. K.S. Kahlon born in Patiala . Author is Doctorate in Computer Science. His major field Of study is Parallel Computing. He had seven year of work experience in this field and Currently he is doing research in object oriented modeling.

He has a thirteen year of teaching and research experience. He had participated in various national and international conferences. His dozen of paper has been published in various journals and proceedings. Currently he is working as ASSOCIATE PROFESSOR in CSE Dept in Guru Nanak Dev University, Amritsar, Punjab(India)

Rakesh Kumar born in Gurdaspur(India) on 23 June 1980. Author is M-tech in Information Technology. His major field of study Object Orinted Analysis & Design and Data security techniques. Working in this field from last two years

He has work experience of one year research in Data security techniques during his M-tech. has two year teaching experience in Post Graduate degree college. Currently he is working as LECTURER in Computer Science Dept in S.S.M College Dinanagar , Distt. Gurdaspur, Punjab, INDIA. He has participated in various National level conferences in India.

Gurjeet Singh born in Amritsar (India) in August 1977. Author did Masters in M-tech in information Technology from G.N.D.U. Major field of study is Modeling of languages and Parallel processing and Scheduling algorithms. Working in these fields from last three years

He has four year work experience of teaching to post graduate classes. He has participated in three National level conferences in India. Currently he is lecturer Post graduate Computer Science Dept of Tri Shatabdi College near Gurdwara Ramsar , Amritsar , Punjab, India. E-mail gurjit_singh_1977@yahoomail.com