REVERSE ENGINEERING OF C++ CODE

Dissertation submitted in partial fulfillment
of the requirements for the
award of the degree of

Master of Technology

in

Computer Science

by

S. Ramanarayana Reddy

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067

December 2001

REVERSE ENGINEERING OF C++ CODE

Dissertation submitted in partial fulfillment
of the requirements for the
award of the degree of

Master of Technology

in

Computer Science

by

S. Ramanarayana Reddy

SCHOOL OF COMPUTER & SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110067

December 2001

CONTENTS

CERTIFICATE

This is to certify that the project entitled "Reverse Engineering of C++ code" being submitted by S. Ramanarayana Reddy to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science is a bonafide work carried by me under the guidance and supervision of Prof. Parimala N.

The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.

[illegible]

(S. Ramanarayana Reddy)

Professor
School of Computer & Systems Sciences
Jawaharlal Nehru University,
New Delhi

[illegible]

Prof. Parimala N. 24/12/01

School of Computers and Systems Sciences,

Jawaharlal Nehru University,

New Delhi-110067.

[illegible]

24-12-01

Prof. K.K. Bharadwaj

Dean,

School of Computers and Systems Sciences,

Jawaharlal Nehru University,

New Delhi-110067.

ACKNOWLEDGEMENTS

I am thankful to my guide, Prof. Parimala N. for suggesting me to do the work in Reverse Engineering and being there to direct me all the time.

I would like to thank Prof. K.K. Bharadwaj, Dean, School of Computers and Systems Sciences (SC&SS), JNU for providing excellent lab facilities.

I would also like to thank Mr. T.V. Vijay Kumar, research scholar, SC&SS, Mr.G. Srikanth, my classmate and all my batch mates for their cooperation and advice.

[illegible]

(S. Ramanarayana Reddy)

ABSTRACT

Reverse engineering (RE) is a process of analyzing a system to identify the system components and their interrelationships and recreate the design of the system. RE generally involves extracting design artifacts. We can perform reverse engineering at any stage of project life cycle.

The primary purpose of reverse engineering a software system is to help in understanding of the system for both maintenance and further development. The other objectives are to generate an alternative views, design recovery, facilitate reuse and re-documentation etc.

The aim of this project is to arrive at the design of C++ program in UML notation. UML is used to represent the object-oriented design of the system. There are two views in any system, the static and the dynamic. Static view shows the different artifacts of the system like classes, variables and function signatures etc. Dynamic view shows the run time behavior of the system.

To achieve the static view of the system, class names, and variables of all types and function signatures in each class are identified. With this information, it generates the class diagram which shows the relationships between the different classes in the source program.

In this project, to achieve the dynamic view of the system the interaction between the objects of different classes at run time is identified. Objects interact by passing messages. This system generates the sequence diagram. It shows the sequence of messages representing the interaction between objects. In order to generate the dynamic view the source code is modified and the modified source code is executed to get an output file. This is then parsed to get sequence diagram. This project is implemented in C++.

CHAPTER-1

INTRODUCTION

1.1 Role of Reverse Engineering

The need for maintaining and improving software systems has increased dramatically over the last 15 years. Dealing with old software systems consumes billion-dollar assets to organizations, governments and educational institutions etc. It is a critical problem for those institutions and organizations. Studies indicated that 50-80% software development efforts are spent on maintaining existing systems rather than developing new systems.

The greater part of software maintenance process is devoted to understanding the system being maintained. It involves reading documentation, scanning source code and understanding the changes to be made. If we want to improve the software development, we should look at maintenance and if we want to improve the maintenance we should facilitate the process of easy understanding of the existing programs. This can be achieved through Reverse Engineering.

1.2 Definition of reverse engineering

Reverse Engineering (RE) is a process of analyzing a system to identify the system components and their interrelationships and recreate the representation of the system in another form.

RE is a part of the maintenance process that helps you understand the system so that you can make appropriate changes.

RE is an inverse process of forward engineering.

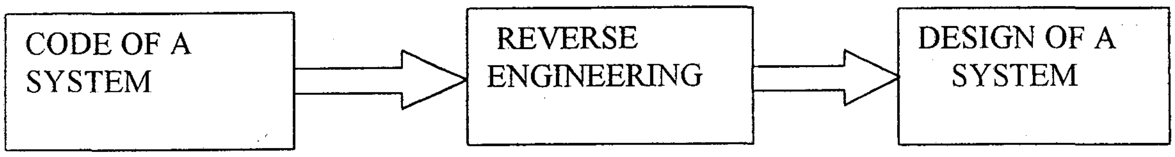


Figure 1.1 Reverse Engineering process.

1.3 Overview of RE

RE generally involves extracting design artifacts. We can perform reverse engineering at any stage of project life cycle. There are many sub systems in reverse engineering like redocumentation and design recovery. RE is a process of EXAMINATION, not a change or replication. RE is regularly applied to improve your own products, as well as to analyze a competitor product.

Many RE tools have been built over the last 15 years to help understand the S/W systems. These tools aid the extraction of S/W artifacts and their dependencies and the synthesis of high level concepts.

Many of the currently available RE tools emphasize STATIC RE i.e. analyzing the static structure of a subject S/W system. Such environments have been successfully used to support the understanding of applications written in procedural programming languages. However the change of programming languages and styles also affects the evaluation of RE tools and methods.

The adoption of object oriented paradigm has changed programming styles significantly. Dynamic nature of object oriented programs emphasizes the importance of dynamic RE that is analyzing the runtime behavior of a system. To understand the architecture of object oriented system both static and dynamic analysis is needed.

1.4 Sub-Systems in RE

RE can be performed at any stage of project life cycle. In the spanning lifecycle stages RE covers a broad range starting from the existing implementation, recreating the design etc. The main sub-systems in RE are

- Redocumentation.
- Design recovery.

Redocumentation:

It is the simplest and oldest form of RE. It is the creation or revision of a semantically equivalent representation with in the same relative abstraction level. Some of the tools used to perform the redocumentation are

- Pretty printers – which display code listing in an improved form.
- Diagram generators – which creates diagrams directly from the code.
- Cross-reference listing generators.

The key goal of this tool is to provide easier ways to visualize relationships among program components so you can recognize and follow paths clearly.

Design recovery:

It is a subset of RE in which domain knowledge, external information and reduction or fuzzy reasoning are added to the observations of the subject system to identify the meaningful higher level of abstractions beyond those obtained directly by examining the system itself. Design recovery recreates design abstractions from a combination of code, existing design documentation, personal experience, and general knowledge about problem and application domain. Design recovery must reproduce all of the information required for a person to fully understand

- What a program does?
- How it does?
- Why it does it?, and so forth.

1.5 Objectives of RE

We do RE of a S/W system to achieve

- Design recovery.
- Generate alternative views.
- Recover lost information.
- Cope with complexity.
- Detect side effects.
- Personal education.
- To make it compatible with other products.
- To learn the principles that guides a competitor design.
- Determine whether a product is capable of living up to its advertised climes.
- Reprogramming in different languages.

- Determine whether another company had stolen and reused some of his companies source code.

Generate alternative views:

RE tools facilitate the generation or regeneration of graphical representations from other forms. While many designers work from a single, primary perspective (like data flow diagrams), RE tools can generate additional views from other perspectives (like control flow diagrams, structure charts [6], and entity relationship diagrams) to aid the review and verification process.

Detect side effects:

Initial design and successive modifications can lead to unintended ramifications and side effects that impede a system's performance in subtle ways. RE can provide observations beyond those that can be obtained with a forward-engineering perspective and it can help detect anomalies and problems before users report them as bugs.

Recover lost information:

The continuing evaluation of large, long-lived systems leads to the recovery of the information lost about the system design. Modifications are frequently not reflected in documentation. By RE a software system, we can achieve the complete documentation.

Cope with complexity:

We must develop methods to better deal with the share volume and complexity of system. A key to controlling these attributes is automated support. RE methods and tools will provide a way to extract relevant information so that decision-makers can control the process and the product in the systems evolution.

Facilitate reuse:

RE can help to identify reusable software components from present systems for creation of new systems. This facilitates the development of the new systems easily and saves lot of resources.

1.5 Model of reverse engineering tool

Most tools of RE, restructuring and reengineering use the same basic architecture. This as shown in Figure 1.2, consists of Parser, Semantic analyzer, Information base and View composer(s).

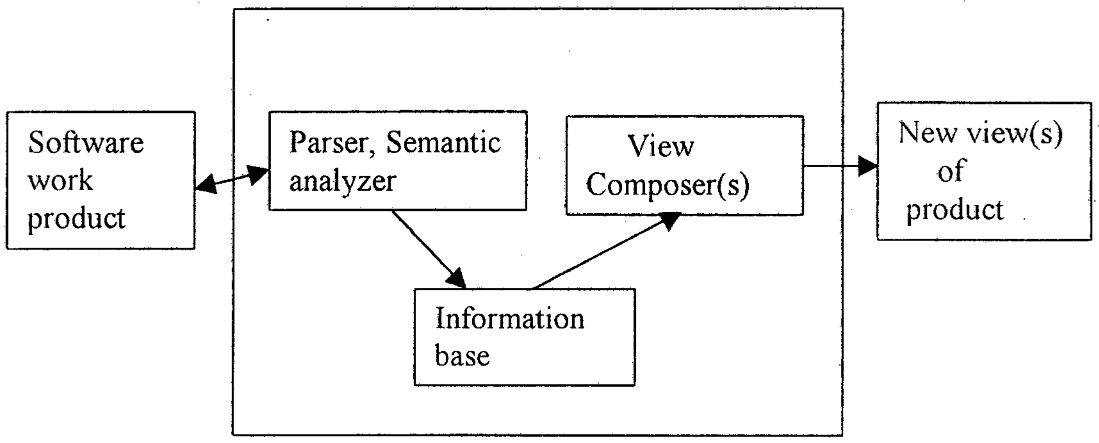


Figure 1.2 Model of RE tool.

1.6 Related terms in RE and their relationships

Reengineering:

Reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. It generally includes some form of RE followed by some form of forward engineering or restructuring. Reengineering is a process of transforming a system to the same level of abstraction. That is transforming assembly language program of one machine to another machine is also reengineering process.

Restructuring:

It is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior such as functional and semantic behavior. An essential aspect of restructuring is that the semantic behavior of the original system and the new system remains the same. For example, altering code to improve its structure is restructuring.

Forward engineering:

Forward engineering is a process of transforming the system from higher level abstractions and logical, implementation independent designs to the physical implementation of the system. Forward engineering follows a sequence of going from requirements through designing its implementations. Figure 1.3 shown bellow depicts the relation between forward engineering and reverse engineering.

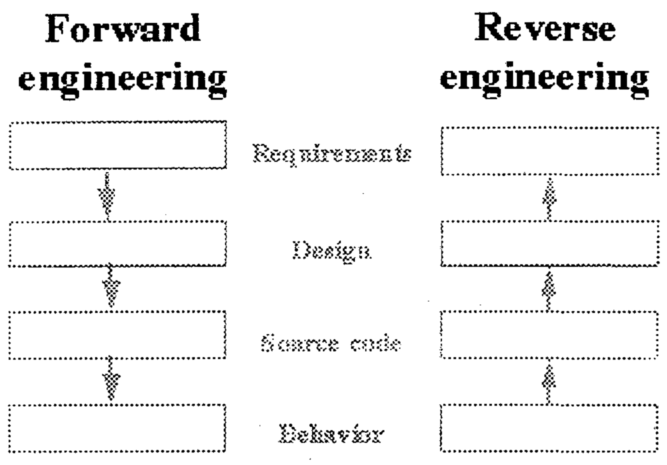


Figure1.3 Forward and Reverse engineering process

Relationships

The relation between forward engineering, reverse engineering, design recovery, reengineering and restructuring is depicted in Figure 1.6.

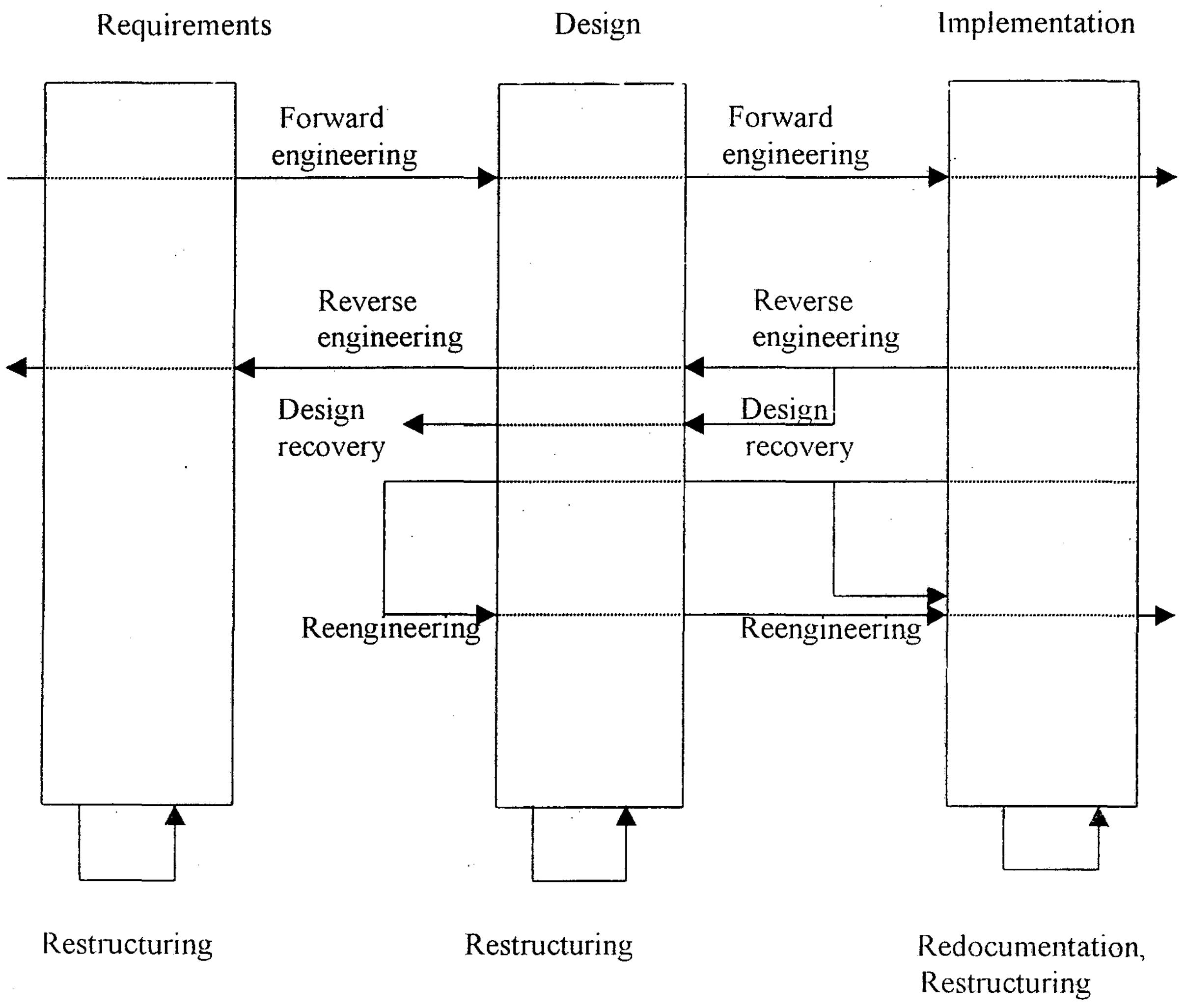


Figure 1.6 Relation between different terms.

1.7 Survey of RE Tools

Static RE (using RIGI)

RIGI [4] is an interactive and visual RE system. A semi-automatic RE approach of RIGI Consists of two phases.

1. Identification of S/W artifacts and their relations.
2. The extraction of design information and system abstractions.

RIGI includes several parsers to extract artifacts from a source code of a target S/W. The extracted static information can be visualized and analyzed using RIGI visualization engine-RIGIEDIT. The different artifacts are

- Classes
- Methods
- Interfaces
- Constructors
- Variables
- Initialization blocks

In addition some attribute values are attached to those above S/W artifacts. They contain information about

- Return types
- Visibility
- Access modifiers.

JExtractor tool in SHIMBA [1] extracts the following relationships among those artifacts.

- Extension - Class extends another class.
- Call - Method calls another method.
- Implementation -A class implements an interface.
- Containment - A class contains a method.
- Assignment – Method assigns a value to variable.
- Access – Method access a variable.

The extracted information is visualized using RIGI dependency graphs. RIGI uses directed graphs to view such static information.

- The S/W artifacts nodes.
- Relationship between artifacts as edges.
- Colors signify node and edge types.
- Attribute values are attached to nodes and edges.

SHIMBA provides some metrics. The metrics measures the properties of the classes, inheritance hierarchy and the interaction among classes of a subject system.

Dynamic RE (Shimba & SCED)

To collect the dynamic information using Shimba, we should select the classes and/or methods for which the event trace information is to be collected. We can also choose exceptions to be traced. Event trace information is generated that is new elements are added to SCED [1] sequence diagrams when methods are called or exceptions are thrown. To capture the event trace information, breakpoints are set with the JDebugger tool. The dynamic control flow information is also generated using breakpoints at control statements. Event trace information is generated when breakpoint is hit. To capture the dynamic information, state boxes, and assertion boxes are added to the SCED sequence diagrams [7] corresponding to the conditions to be evaluated and the resulting values, respectively.

C++ Reverse Engineering Overview

In this section the issues of GDPro 5.0 [8] for reverse engineering of C++ code are discussed. *GDPro* provides detailed and comprehensive support for reverse engineering source code written in any language. Both header files and body files can be reverse-engineered by running in GDPro. What is produced from Reverse Engineering of C++ code with the help of GDPro tool is shown bellow

1. UML [7] Class Diagram [7] which contains:

    Classes
    Class data members and methods
    Inheritance
    Class associations

2. Implementation Diagram which contains:

    Files processed
    File "Include Tree"

3. Repository completely populated with model data extracted from source code

When you perform reverse engineering of C++ code using *GDPro* a new system is created with two views, the view describing the classes is an Class model view and the other is the Implementation model view. When *GDPro* reverses C++ classes, the information about each class attribute is complete and there is no loss of any class information. Embedded information within the C++ class is maintained in the class object on the diagram. In this manner, all implementation details, those details that are not supported by a methodology, are stored for later use by code generation.

The Implementation Model is used to provide the second view that is generated when some C++ code is reverse engineered. It is a model used to represent all the files used in a source code, which is reverse engineered. This includes both user and systems include files. This model is used to provide support for seamless generation of C++ code from the system, minimizing adverse changes to the reversed code. The Implementation model represents all the files for a system in a hierarchical, left to right layout.

1.8 Approach in this thesis

In this implementation, reverse engineering is performed by depicting the static and dynamic views. Static view is represented by the class diagram and dynamic view by the sequence diagram. With the help of these diagrams one can understand the static and dynamic behavior of the system easily and quickly and can perform any modifications to adopt the legacy systems for his current use.

For drawing the class diagram, the following design artifacts are extracted from the source code.

All the class names.

All types of variables in each class.

All function signatures with different return types in each class.

Inheritance relationships between the classes.

The above information is achieved by parsing the source program.

To depict the runtime behavior of the source code this implementation follows the different approach in which it creates a new source program by inserting additional statements at appropriate places in the existing source program. The execution of the modified source program will create a new file, which contains the sequence of messages that are passed between the objects of different classes along with the class name of message passing object. By using these messages and class names the sequence diagram will be drawn. The advantage with this approach is that this application doesn't need a debugger to depict the run time environment. Using the modified version of the source program generated by this application, the user, while using the application, can simultaneously have a picture of the run time environment.

CHAPTER-2

UNIFIED MODELING LANGUAGE

2.1 UML

The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction.

The UML was developed jointly by Grady Booch, Ivar Jacobson, and Jim Rumbaugh at Rational Software Corporation, with contributions from other leading methodologists, software vendors, and many users. The UML provides the application modeling language for:

- ♦ Business processes modeling with use cases.
- ♦ Class and object modeling.
- ♦ Component modeling.
- ♦ Distribution and deployment modeling.

First and foremost, the Unified Modeling Language fuses the concepts of Booch, OMT, and OOSE. The result is a single, common, and wjdely usable modeling language for users of these and other methods.

Second, the Unified Modeling Language pushes the envelope of what can be done with existing methods. As an example, the UML authors targeted the modeling of concurrent, distributed systems to assure that UML adequately addresses these domains.
Third, the UML focuses on a standard modeling language, not a standard process. Although the UML must be applied in the context of a process, it is our experience that different organizations and problem domains require different processes.

The UML specifies a modeling language that incorporates the object-oriented community's consensus on core modeling concepts. It allows deviations to be expressed in terms of its extension mechanisms.

1.7.2 Class diagram

A class diagram shows static structure of the system. A class diagram is a diagram that shows a set of classes, interfaces, collaborations and their relationships. A class diagram commonly contains

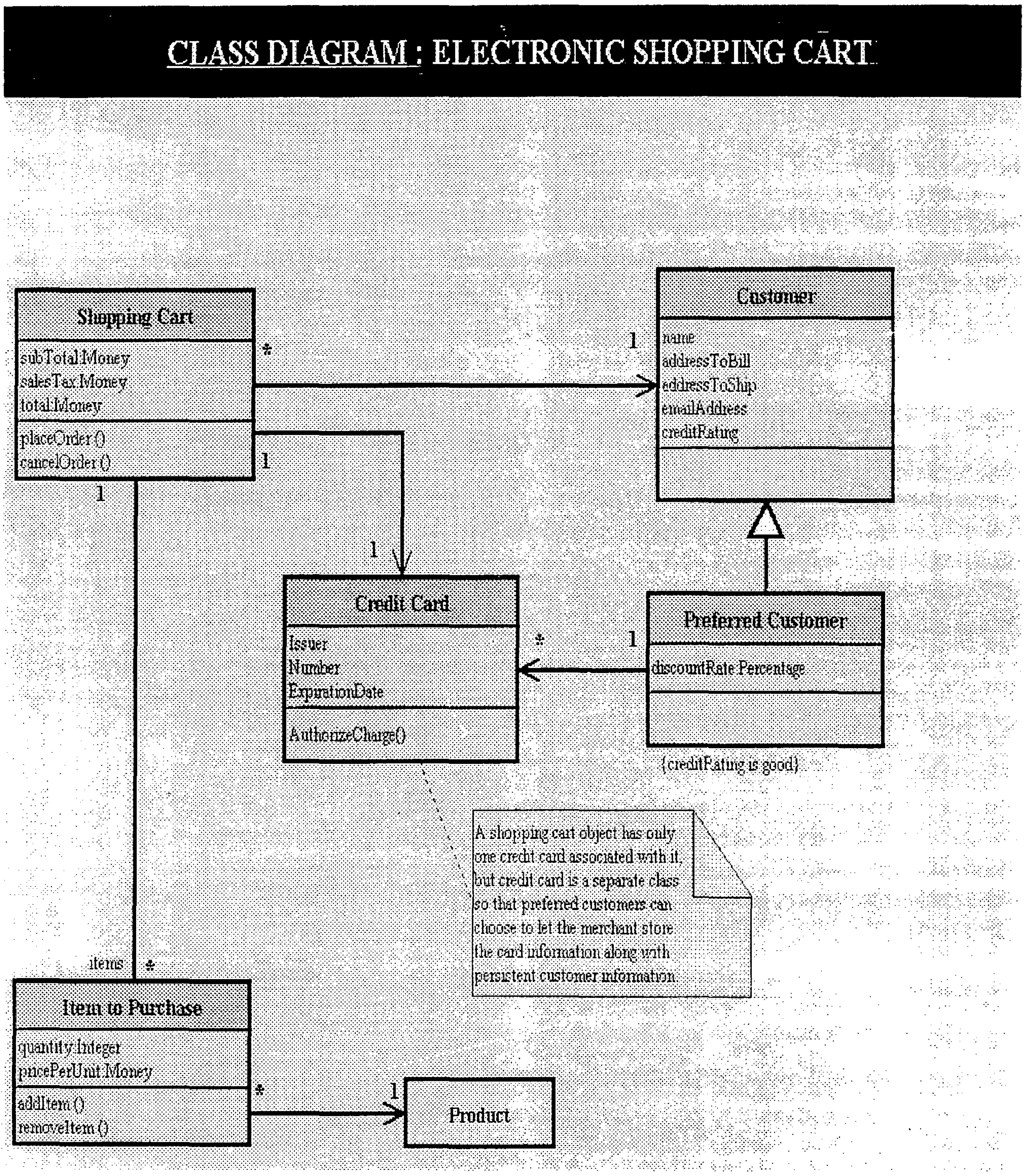
- ♦ Classes.
- ♦ Interfaces and
- ♦ Different relationships.

A class diagram is used to model the static design of the system. This supports the following requirements of a system that is the services the system should provide to its end users. A well-structured class diagram is focused on:

- ♦ Communicating one aspect of a system's static view.
- ♦ Contains only elements that are essential to understanding that aspect.
- ♦ It should not misinform the reader about important semantics.

A class diagram is drawn as follows:

- ♦ Draw a class symbol-rectangle for each class.
- ♦ Name the classes.
- ♦ Enter the class attributes.
- ♦ Enter the class operations.
- ♦ Add links and associations.
- ♦ Add notations.



A shopping cart object has only one credit card associated with it, but credit card is a separate class so that preferred customers can choose to let the merchant store the card information along with persistent customer information.

Figure 2.1 Class Diagram

1.7.2 Sequence diagram

Sequence diagram is an interaction diagram used in the UML for modeling the dynamic nature of the systems. It emphasizes the time ordering of messages. A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Interaction diagrams are important for

- ♦ Modeling the dynamic aspects of the system.
- ♦ Constructing executable systems through forward and reverse engineering.

A sequence diagram displays an interaction as a two dimensional chart.

Vertical dimension:

- ♦ It is the time axis.
- ♦ It starts from the top and proceeds down the page.

Horizontal dimension:

- ♦ It shows the classifier roles that represent individual objects in the collaboration.
- ♦ A vertical column-the life line represents each classifier role.

A message is shown as an arrow from the lifeline of one object to that of another object. The arrows are arranged in time sequence down the diagram. The lifeline is double when the objective is active and is doted if the objective is inactive. Sequence diagram is shown in Figure 2.2.

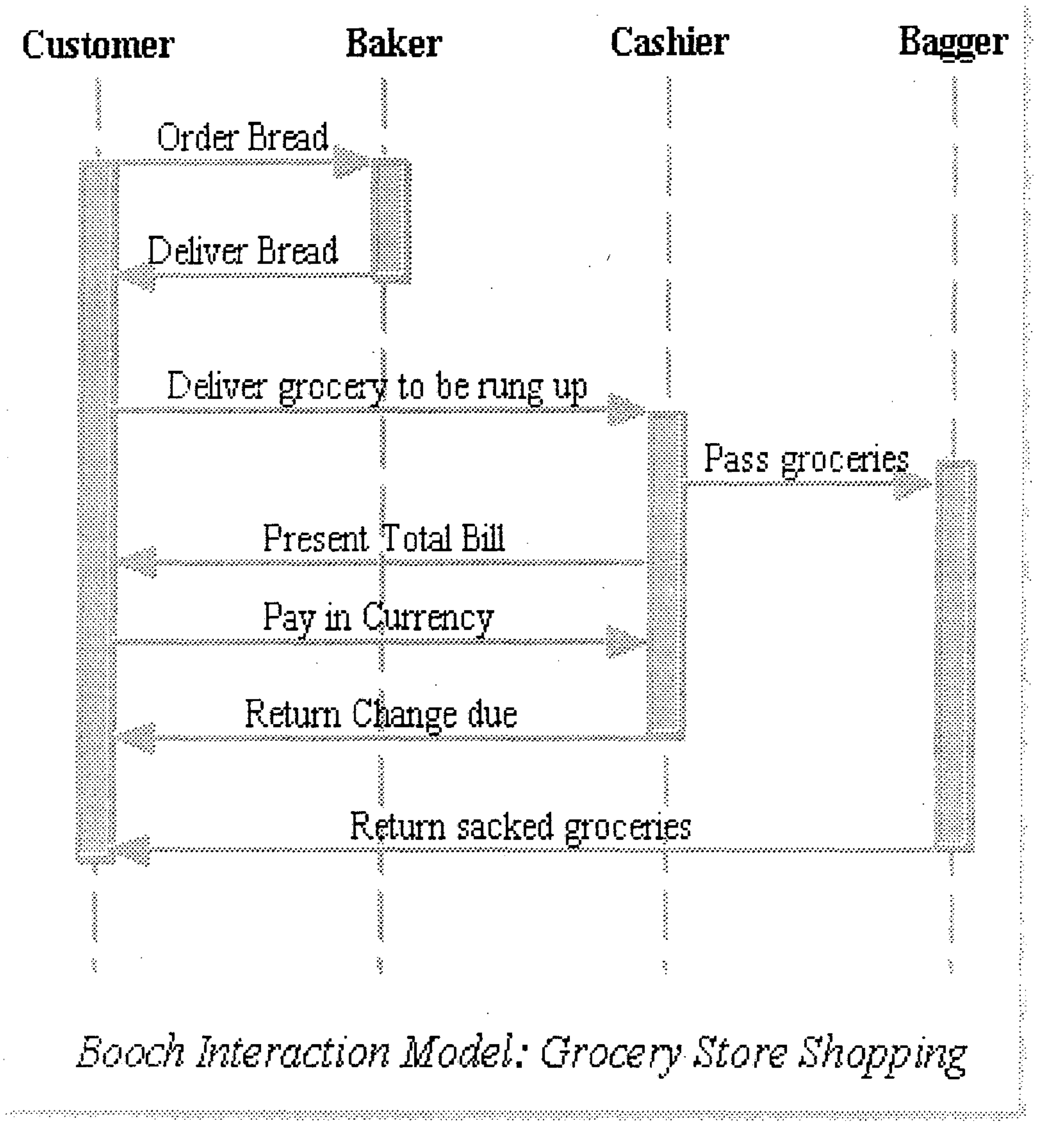


Figure 2.2 Sequence Diagram

CHAPTER-3

DESIGN

3.1 The Design

The design of this system is represented with the help of 'Structured charts'. Along with the overview of the system these structure charts depict the processes of

1. Drawing the Class diagram.
2. Modifying the Source code
3. Drawing the Sequence diagram.

3.2 Structure Chart for the complete system

This structured chart show in Figure 3.1 depicts the main design components of the system. They are:

Drawing the Class Diagram.
Creation of the modified source code.
Drawing Sequence Diagram.

These design components are described in the following sections.

The data items and process names of Figure 3.1 are explained below.

fname: Name of the source file for which Class diagram/ Sequence diagram is drawn.
cdig: Class diagram.
mfname: Name of the temporary file which is created by the module 'CreateTemp'.
sdig: Sequence diagram.

CDGenerator: This module will generate the class diagram.

CreateTemp: This will create a temporary file, which contains the modified source program for which sequence diagram is drawn by executing it.

SDGenerator: This module will generate the sequence diagram.

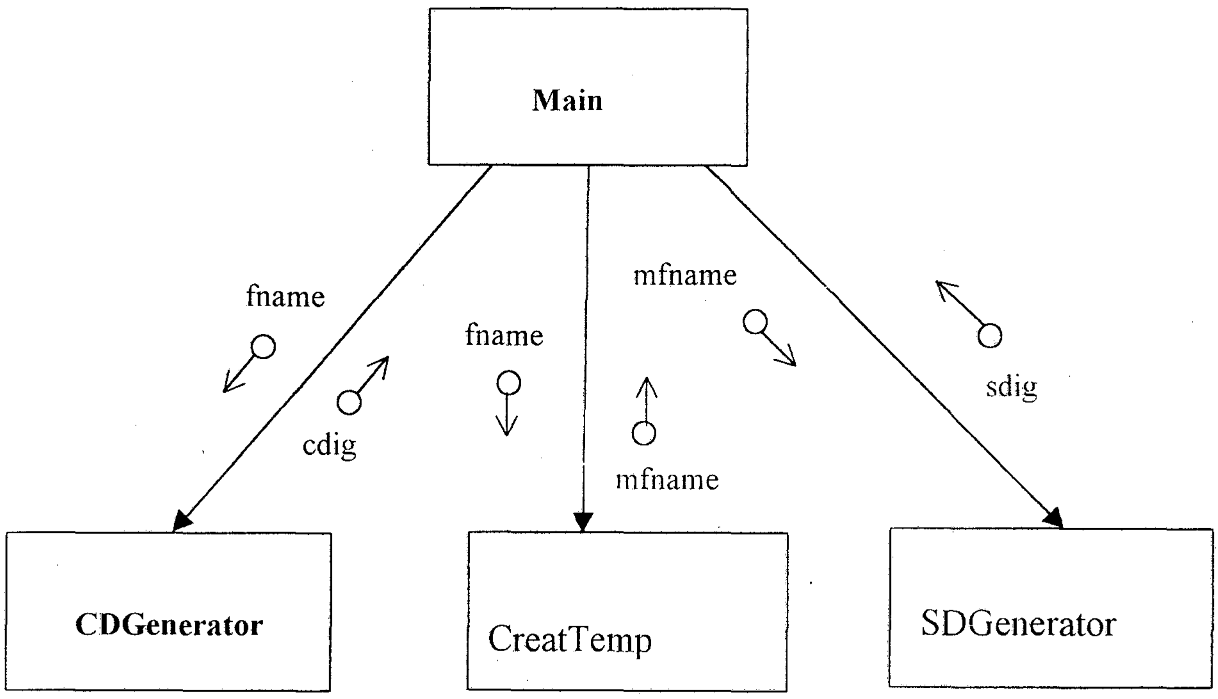


Figure 3.1 Structure chart for the complete system

TH- 9477

3.3 Structure chart for drawing class diagram

The structured chart for drawing the class diagram is shown in Figure 3.2. Each module in the Figure 3.2 is described below.

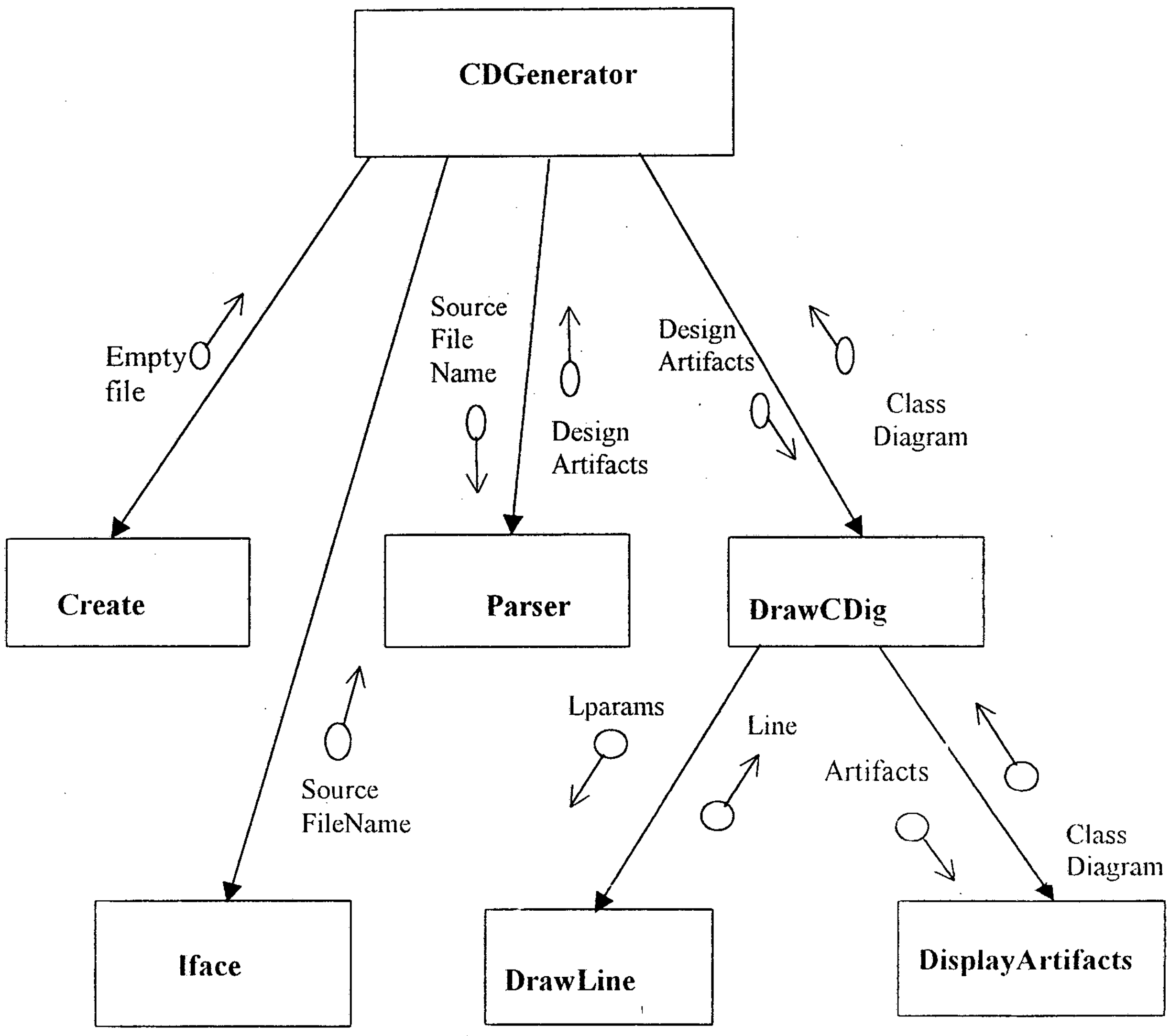


Figure 3.2 Structure chart for drawing class diagram

Create:

This module will create an empty file to draw the class diagram and return to CDGenerator module with specified dimensions.

Iface:

This module is an interface between user and the CDGenerator. It requests the user to enter the name of the file containing the source code for parsing and returns that file name to CDGenerator.

Parser:

This module receives the source file name from CDGenerator and extracts all design artifacts from the source file and returns those artifacts to CDGenerator.

DrawCDig:

This module receives the design artifacts and calls Drawline and DisplayArtifacts modules to draw the class diagram. It returns the class diagram to CDGenerator.

Drawline:

This module receives the line parameters from DrawCDig module and draws the line.

DisplayArtifacts:

This module receives the artifacts as input and displays the same in class diagram.

CDGenerator:

This is the coordinating module. It invokes the Create, Iface, Parser, and DrawCDig modules to generate the class diagram. This gives the static view of the source program.

3.3 Modifying the source code

The structure chart for creation of modified source code is shown in Figure 3.3. The functionality of each module is described below.

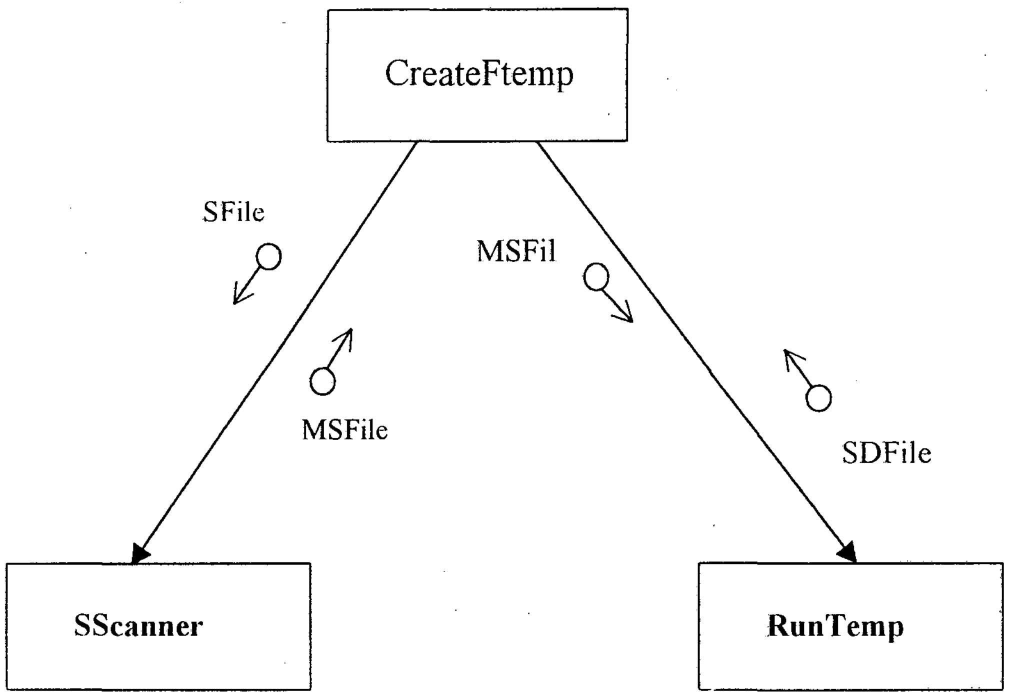


Figure3.3 Structure Chart for the module that create an intermediate file

The data items and process names of Figure 3.3 are explained below.

Sfile: Name of the file containing the source code for which runtime environment has to be depicted.
MSFile: Modified source file.
SDFile: This file contains sequence of messages.

CreateFtemp:

This module calls SScanner and RunTemp modules to generate the runtime environment of the source program and is saved in a file for later processing.

SScanner:

This module receives the source file and creates the temporary file, which contains the modified source program to capture its runtime environment. The way the source code is modified is explained in chapter no 4.

RunTemp:

This module receives the modified source file and requests the user to RUN it. Executions of this file will results in creation of another file that contains the sequence of messages that represent the interaction between different objects.

3.3 Drawing the sequence diagram

Figure 3.4 shows the Structure chart for drawing the sequence diagram. About each module in the Figure 3.4 are described as follows.

GetCnames:

This module receives the file name from SDGenerator and extracts the names of the classes contained in the file and returns them to the SDGenerator.

PrintCnames:

This module receives the class names from SDGenerator and prints them in the sequence diagram. It returns the partially drawn sequence diagram to SDGenerator.

DrawSD:

This module receives filename and S1diagram from SDGenerator and it calls GenerateMess and PrintMessages modules to label the interactions between objects with appropriate messages. This module returns the sequence diagram to SDGenerator.

GenerateMess:

This module extracts the messages passed between the objects and returns them to the module DrawSD.

PrintMessages:

This module receives the messages from DrawSD module and prints these messages in a sequence diagram.

SDGenerator:

This is the coordinating module to draw the sequence diagram by invoking the GetCnames, PrintCnames, and DrawSD modules. The dynamic behavior of the source program is depicted with the help of sequence diagram.

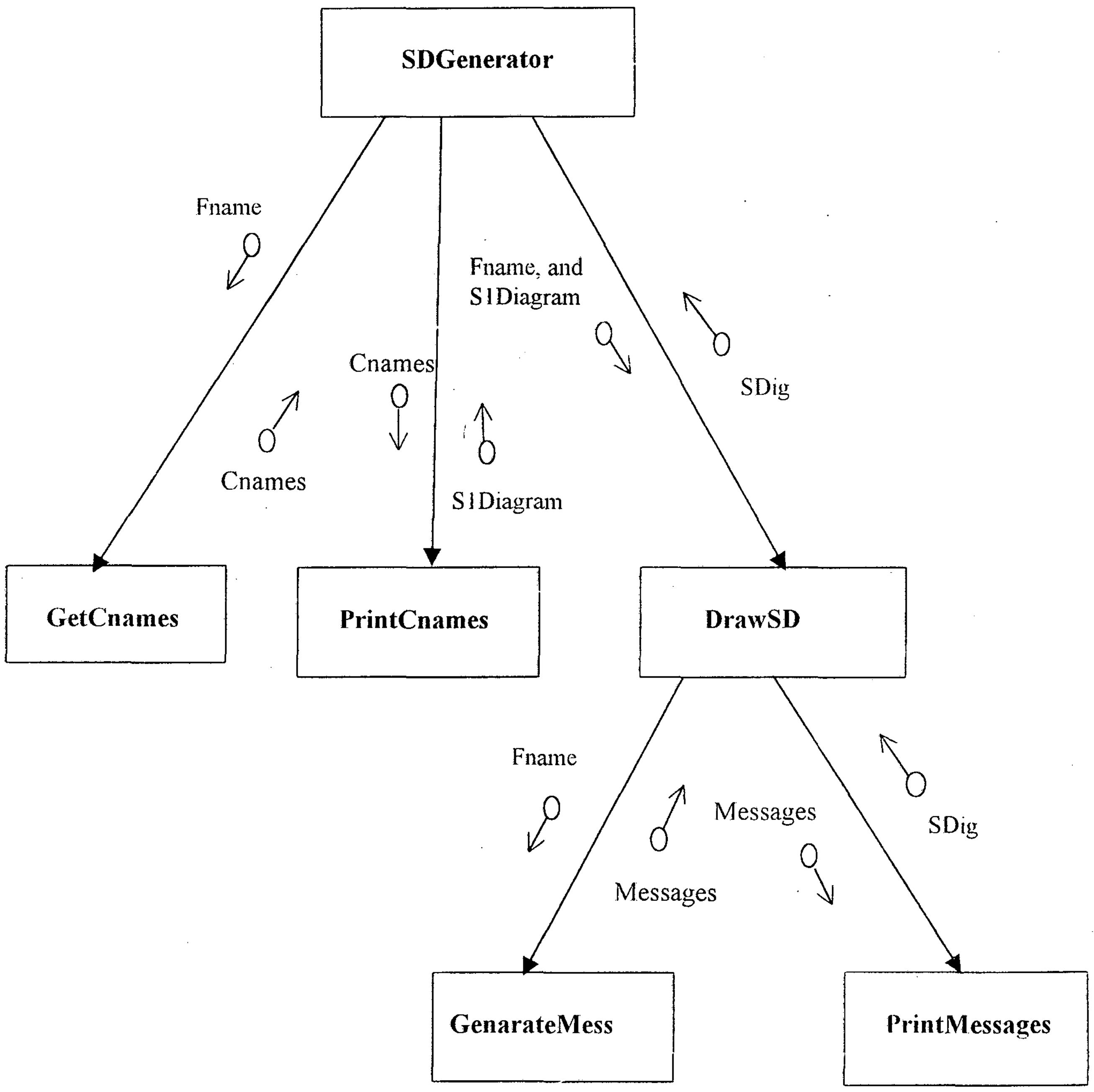


Figure3.4 Structured chart for drawing the Sequence diagram

IMPLEMENTATION

4.1 Introduction

Class diagram and the Sequence diagram have been implemented in this project in C++. The Class diagram shows the static behavior and Sequence diagram shows dynamic behavior of the system.

4.2 Implementation of the class diagram

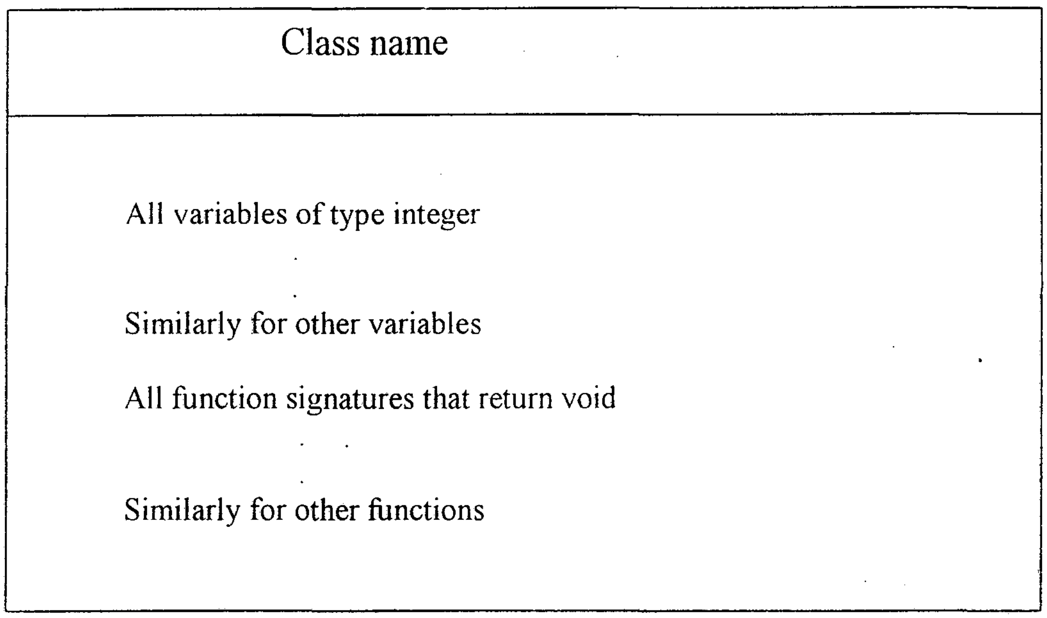
The parser will scan the source program written in C++ and it does the following functions:

Step1: Iface will request the user to enter the filename containing the source whose static view is to be depicted by drawing the class diagram.

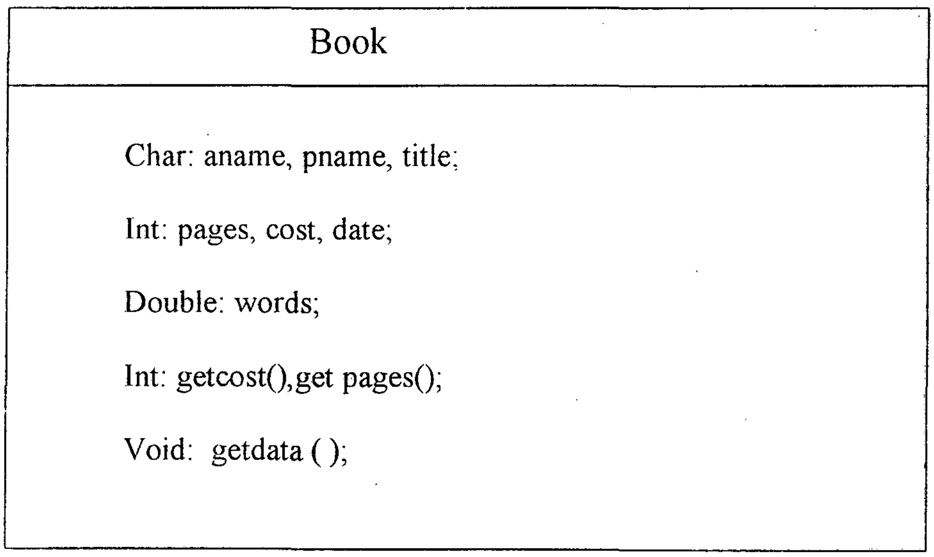
Step2: The Parser will search for a class declarations in the source program, extract their names and all types of variables and function signatures within each class. It also extracts the relationships between the classes if they exist. At the end of this step all the design artifacts of the system such as class names, all types of variables and function signatures in each class etc are extracted.

Step3: DrawCDig module receives all the design artifacts to draw the class diagram and it repetededly calls the 'DrawLine' and 'DisplayArtifacts' modules in the process of the drawing of class diagram.

The structure of the class in class diagram is shown in Figure below.


Class name
All variables of type integer
Similarly for other variables
All function signatures that return void
Similarly for other functions

Structure of a class



Actual structure of the class "BOOK".

4.3 Implementation of Sequence diagram

Sequence diagram shows the interaction between objects of the classes. To implement the sequence diagram the following sequence of steps are performed.

Step1: SScanner module will requests the user to enter the filename containing the source code whose run time behavior is to be depicted by drawing the Sequence diagram. This module will create a temporary file that contains the source code with the following modifications.

A. The statement given below is inserted in order to create a global file pointer. This file is used latter to depict the runtime environment.

```
FILE *Se ;
```

B. When SScanner module encounters the beginning of the main program, the following statement is inserted.

```
Se=fopen( "fsec.cpp", "w" ) ;
```

The contents of the file **fsec.cpp** give information about the interaction between different objects such as the classes to which those objects belong, the messages passed by those objects etc. This file is created when the user runs the modified source code.

C. In every class definition, SScanner module will insert the following function definition. This function will be executed whenever any method of the object is called. This results in printing the corresponding class name of the object.

```
virtual void   pnt( )
{
fprintf (se,  "%s\n",  Cname );
}
```

In the above function declaration:

Se: file pointer.
Cname: It is a variable, which assumes the name of the class in which the definition is inserted.

D. Whenever the SScanner module encounters the function definition in a class, it inserts the following statement within the body of that function.

```
this->pnt();
```

The above statement will invoke the function **'pnt()'**, and the execution of this function results in printing the class name corresponding to the object by which a method is called.

E. Whenever the SScanner module encounters a statement invoking a function of another object, it inserts 'fprintf' statement that contains the above invocation statement.

This module will return the modified source file to 'CreateFtemp' module.

Step2: CreateFtemp module will call the RunTemp module. RunTemp module will request the User to run the modified source file. The execution of this file will result in the creation of another file containing the sequences of messages (function calls). For each message, the class of the object that passes the message is also stored in the file. The sequence of messages reflects the interaction between the objects.

Step3: GetCname module will extract the class names and return to the SDGenerator module. The interactions occur between objects of these classes. SDGenerator invokes the PrintCname module to print the class names in the Sequence diagram.

Step4: SDGenerator will invoke the DrawSD module, which receives the messages extracted by the GetMess module. These messages are passed to PrintMessagess module. This module prints the messages between the interacting objects in the sequence diagram. This module is responsible to identify the object that passes the message and the one that receives. This module completes the printing of messages in sequence diagram.

Step5: Finally DrawSD module will return the completed Sequence diagram to the SDGenerator module.

Role of *this* pointer

C++ uses a unique keyword called **this** to represent an object that invokes a member function. **this** is a pointer that points to the object for which this function was called. This logic has been used in this implementation to depict the runtime environment by extracting interactions that takes place between the objects of the different classes during the execution of modified source program. This is explained in the following section with the help of example.

Example:

Consider the source program shown in Figure 5.1. This program is modified by the SScanner module and creates a new file, which contains the modified source code. This modified source code is used to depict the runtime environment of the original source code by drawing the Sequence diagram. The modified source code is shown in Figure 5.2

Source program:

```
#include<iostream.h>
#include<conio.h>
class X
{
 int x;
 public : int getx()
  {
  x=5;
  return x;
  }
};

class Y : public X
{
 int y;
 public : int gety()
 {
  y=10;
  return y;
 }
};
```

```
void main()
{
 int a,b,c;
 clrscr();
 X x0;  Y y0;
 a= x0.getx();
 b= y0.getx();
 c= y0.gety();
 cout<<a<<'\n'<<b<<c;
};
```

Figur5.1 Sample Source code.

Modified Source program:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
FILE *se;
class X
{
virtual void pnt()
{
fprintf(se,"%s","class X");
}
int x;
public : int getx()
{
this->pnt();
x=5;
return x;
}
};

class Y : public X
{
    virtual void pnt()
   {
    fprintf(se,"%s","class Y : public X");
   }
int y;
public : int gety()
{
this->pnt();
y=10;
return y;
}
};


void main()
{
se=fopen("fsec.cpp","w") ;
int a,b,c;
clrscr();
X x0;
Y y0;
fprintf(se,"%s","a= x0.getx();");
a= x0.getx();
fprintf(se,"%s","b= y0.getx();");
```

```
b= y0.getx();
fprintf(se,"%s","c= y0.gety();");
c= y0.gety();
cout<<a<<'\n'<<b<<c;
}
```

Figure 5.2 Modified Source program.

In the above program, the bold statements are the additional statements that are inserted by the SScanner module to depict the runtime behavior of the source program. The execution of modified source program that is shown in Figure 5.2 will result in the creation of another file shown in Figure 5.3. This file contains the sequence of messages that are passed between different objects, classes to which these objects belong etc.

```
a= x0.getx();class X
b= y0.getx();class Y : public X
c= y0.gety();class Y : public X
```

Figure 5.3: Sequence of messages and their class names.

The messages and class names are extracted from the above Figure to draw the Sequence diagram so that the runtime environment of the source program is depicted. Sequence diagram for some sample code is shown in the chapter-6.

The platform

This project has been implemented in C++ and Windows 98 is the operating system on which this project is developed. Bjarne Stroustrup at AT&T Bell Laboratories developed C++.

C++ is an object oriented programming language and hence carries the advantages of object oriented concepts such as Encapsulation, Inheritance, and Polymorphism. C++ programs are easily expandable and maintainable. It is expected that C++ will replace C as a general-purpose language in the near feature.

As can be seen from previous sections, a C++ compiler is a must in this implementation for reverse engineering the given source code. Using the C++ language to develop this application eliminates the need for another compiler and hence it is an added advantage.

CHAPTER-5

SAMPLE RESULTS

5.1 Sample Results

The following sections show few sample programs and the resulting Sequence diagram and Class diagram. Sequence diagram shows the dynamic behavior and class diagram shows the static behavior of the system. Examples depicting each of them are given below:

Sample source code-1

```
Class  CC
{
    int  s1,s2;
    public:  int getsum()
             {
                  s1=10;
                  s2=20;
                  return s1+s2;
               }
             void display()
             {
               AA  oba;
               oba.showsum();
             }
};
```

```
          Class  BB
          {
                    float f1;int  ii;
                    public: float fgetsum()
                            {
                                CC ob1;
                                 f1=32.6;
                                 ii=Ob1.getsum();
                                 return f1-ii;
                             }
           };

class AA
{
          float sum;
          public:void show()
                    {
                      BB ob2;
                      Sum=Ob2.fgetsum();
                    }
                  void showsum()
                    {
                      cout<<sum;
                    }
};
void main()
{
         AA  ob3;  ob3.show();

         CC  obc;

         obc.display();
 }
```

Sequence Diagram

Sequence diagram shows the Dynamic view of the system. Figure 6.1 shows the corresponding sequence diagram for the samplecode-1. In the Figure 6.1, AA, BB and CC are the class names. The objects of one class interact with the objects of other classes by passing messages. The message (function calls) along with the object name is shown in Figure.

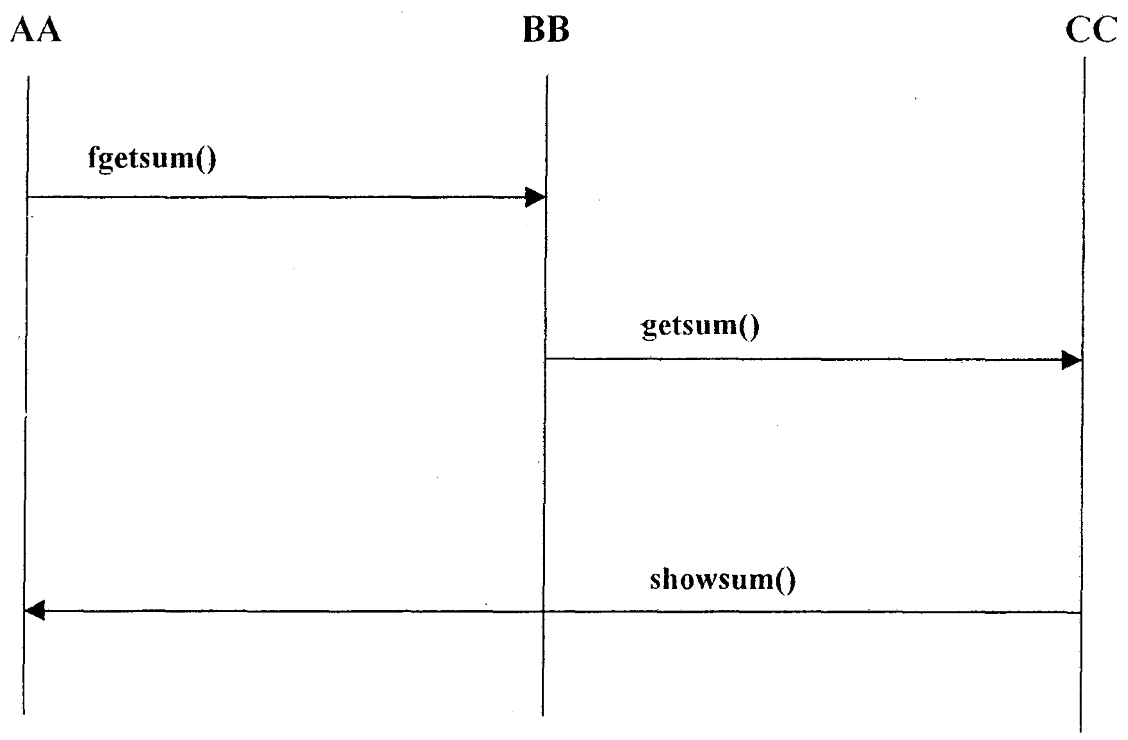


Figure 6.1 Sequence Diagram for the above sample code

Sample source code-2

```
class student
{
protected: int roll-numb,age;
char name[],class [];
public: void getnumper(int num)
{ .........   };
void  printnumer(void)
{.........   };
}
class test: public student
{
protected: int sub1,sub2;
char sname;
public: void  getmarks(int, int)
{   ............};
void  printmarks(void)
{   ...........};
};
class address: public student
{
protected: char fname,vname,state;
int pincode;
public: void getadd( char, char ,char ,int)
{   ..........};
void printadd(void )
{............};
};
```

Figure 6.2 Sample code for drawing class diagram

Class Diagram

Class Diagram shows the Static view of the system. Figure 6.3 shows the Class Diagram for the sample code shown in Figure 6.2. Student is a base class and address, test are inherited from base class. Variables and functions in each class are shown in the Figure.

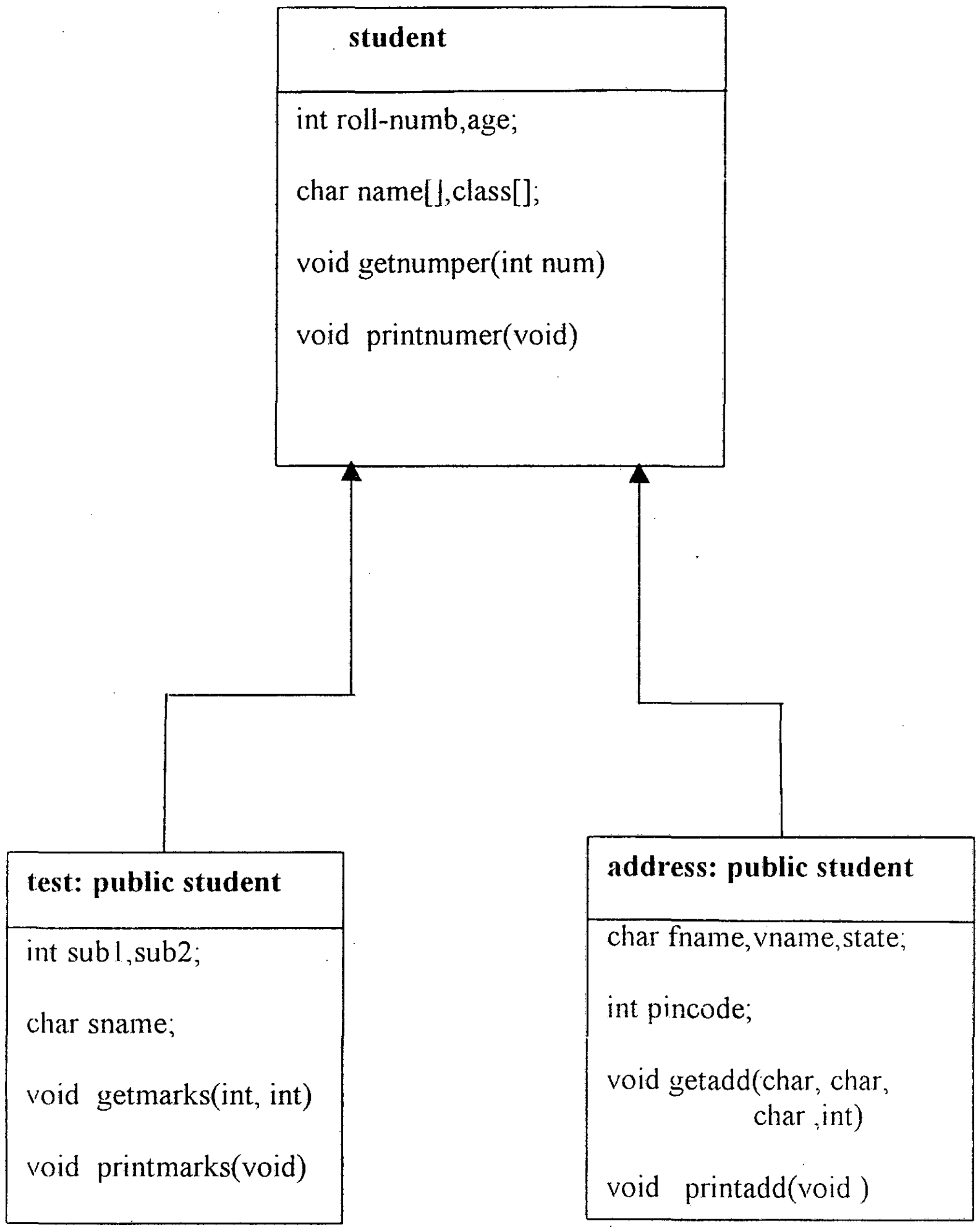


Figure 6.3 Class Diagram

Sample code-3

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
# include<string.h>
class zz;
class mm
{
 private:char s[20];
               int age;
 public: void get_mname();
};
class nn:public mm
{
 float amount;
 public: float getamount(float uu)
            {
             amount=uu;
             return amount;
            }
};

class zz
{
  public: char *name;
  public:void pntname()
     {
      float numb;
      mm objm;
      objm.get_mname();
```

```
    cout<<name;
    nn objn;
    numb=objn.getamount(45683.87);
    cout<<'\n'<<numb;
    objn.get_mname();
    cout<<'\n'<<name;
    }
    void setname(char* s )
    {
        name=s;
        cout<<name;
    }
};

void mm::get_mname()
{
  scanf("%s",s);
  zz x1;
  x1.setname(s);
}
void main()
{
  clrscr();
  zz objz;
  objz.pntname();
}
```

Figure 6.4 Sample code for depicting static and dynamic view.

For the sample code –3 shown in Figure 6.4, resulting class diagram is shown in Figure 6.5 and sequence diagram is shown in the Figure 6.6. In both the figures MM, NN and ZZ are the class names.

Class diagram for the sample code-3

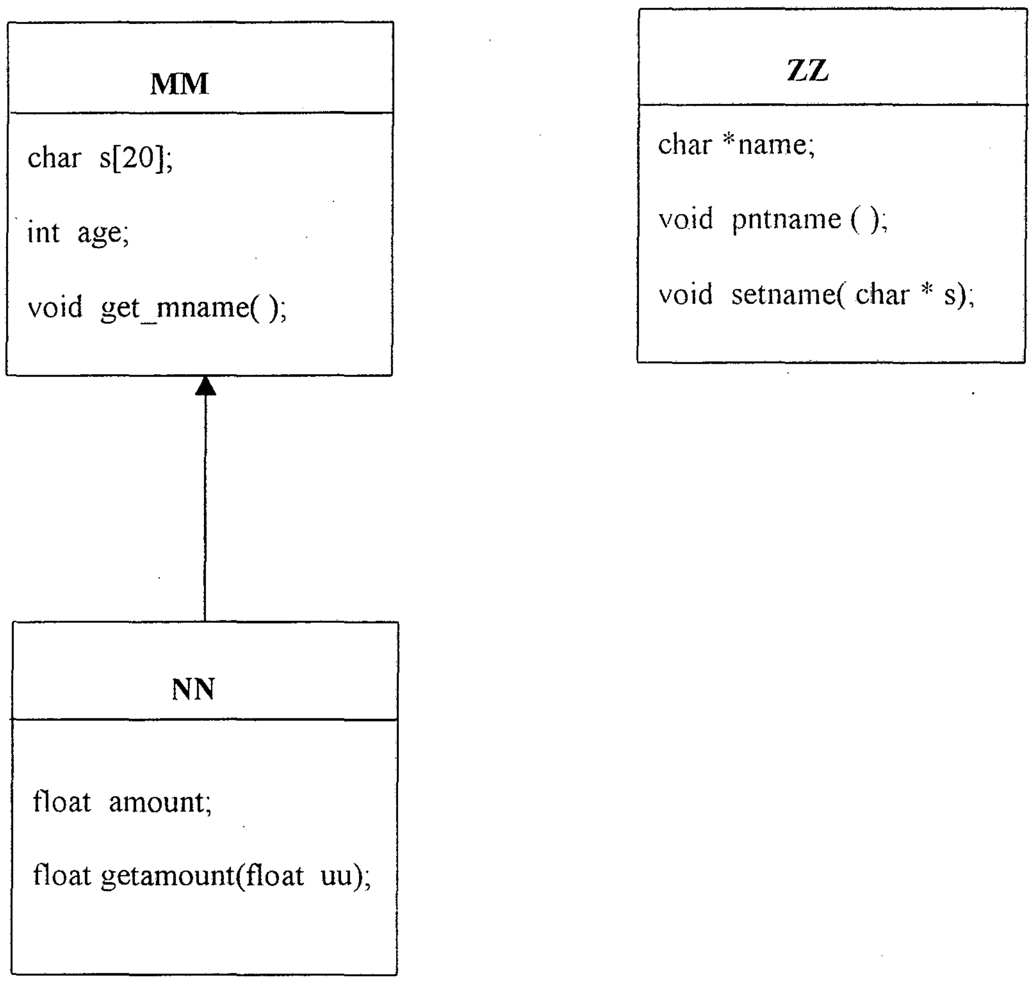


Figure 6.5 Class diagram for the sample code –3.

Sequence diagram for sample code- 3

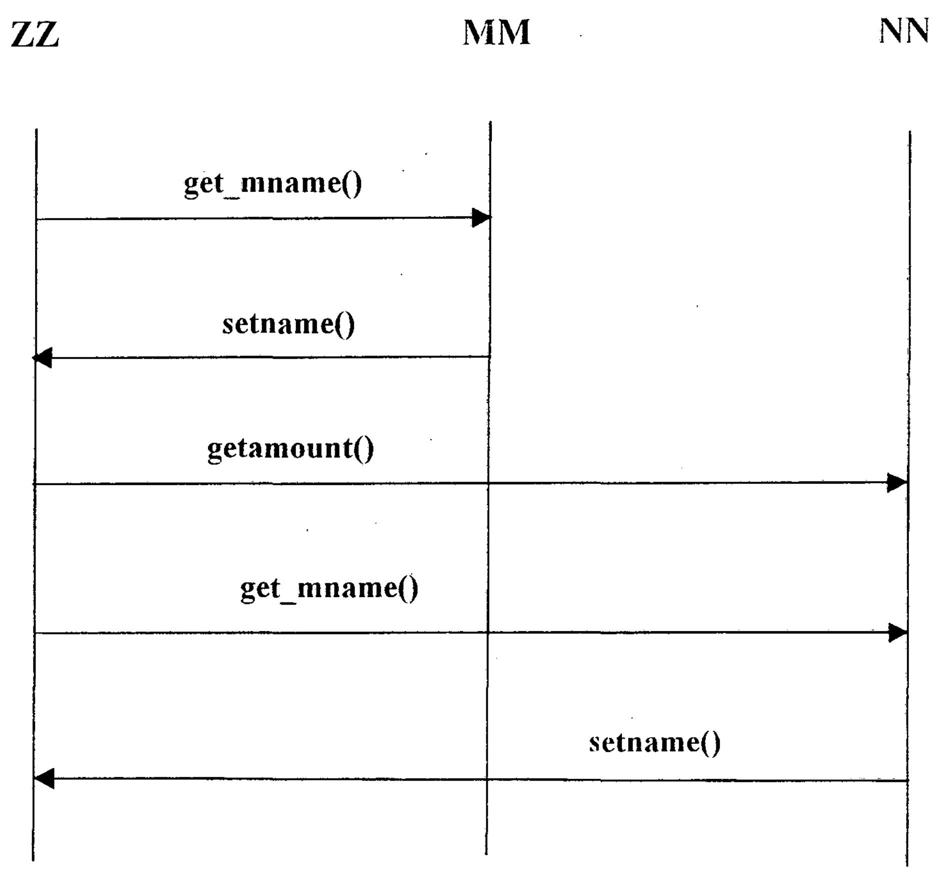


Figure 6.6 Sequence diagram for the sample code –3.

CHAPTER-6

CONCLUSIONS

Software development productivity and quality is improved if programs can be enhanced instead of rebuilding them. They are also improved if major pieces of existing systems can be reused with least possible effort. These activities require the software engineers to have the detailed knowledge of existing programs. Reverse engineering helps in achieving the above mentioned tasks easily and effectively.

This application doesn't need a debugger to depict the run time environment. Using the modified version of the source program generated by this application, the user, while using the application, can simultaneously have a picture of the run time environment.

This implementation has its limitations. It is language dependent i.e. it extracts the design of programs written in C++ only. At present many software organizations are developing products in C++. There are some tools that can convert the code written in C or Pascal into C++.

REFERENCES

1. Shimba – an environment for reverse engineering java software system by Tarja Systa,Koskimies,and Hausi Muller –SP&E,2001, 31:371-394.

2. Reverse engineering 4.7 million lines of code by P.Tonella, G.Antoniol, R. Finutem and F.Calzolari –SP&E, 2000,30:129-150.

3. Practical data exchange for reverses engineering frame works by Michael W.Godfrey – Software engineering Notes, volume 26 no 1,January 2001.

4. Rigi User's Manual -Version 5.4.1 by Kenny Wong ,July 10, 1996

5. Software Engineering, a practitioner's approach, 4th edition by Roger S. Pressmen – Mc GRAW-HILL publications.

6. An integrated approach to software engineering, second edition, by Pankaj Jalote – Narosa publications.

7. OMG UML Tutorials: http://www.celigent.com/omg/umlrtf/tutorials.htm

8. GDPro: http://www.advancedsw.com/products/products.html

KEY WORDS: reverse engineering, reengineering, maintenance, WCRE (working conference on reverse engineering)