

1747

# CASETOOL FOR OBJECT ORIENTED MODELING

**JAWAHARLAL NEHRU UNIVERSITY**

*Dissertation Submitted to*

**JAWAHARLAL NEHRU UNIVERSITY**

*in partial fulfilment of requirements*

*for the award of the degree of*

**Master of Technology**

*in*

**Computer Science**



*by*

**ANKEM HANU**

86P+219



**SCHOOL OF COMPUTER & SYSTEMS SCIENCES**

**JAWAHARLAL NEHRU UNIVERSITY**

**NEW DELHI – 110 067**

**JANUARY 1999**

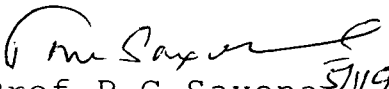


SCHOOL OF COMPUTER & SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY,  
NEW DELHI - 110 067

## CERTIFICATE

This is to certify that the dissertation entitled "CASETOOL FOR OBJECT ORIENTED MODELING" developed in VC++, submitted by Mr.A.Hanu to the School of computer and system sciences, Jawaharlal Nehru University, for the award of Master of Technology in Computer Science is a bonafied work carried out by him.

The results embodied in this project report have not been submitted to any other university or institute for the award of any degree.

  
Prof. P.C. Saxena 5/11/99

(Dean SC&SS)

  
Dr. N. Parimala

(Supervisor)

## DECLARATION

This is to certify that the dissertation entitled "Case tool for Object Oriented Modeling" which is being submitted to the school of Computer and systems sciences, Jawaharlal Nehru University, for the award of Master of Technology in Computer Science, is a record of bonafied work carried out by me.

This work has not been submitted in part or full to any university or institution for the award of any degree.

  
(A. HANU)

## **ACKNOWLEDGEMENTS**

I express my deep gratitude to my guide Dr.N.Parimala for her guidance and great patience at all stages of my project.

I would like to express my thanks to *Dean, Prof. P. C. Saxena*, School of Computer & Systems Sciences, JNU for providing the necessary facilities in the centre for the successful completion of the project.

I take this opportunity to thank my colleagues Nagamani and Shachi for their critical comments during course of the project.

Last but not the least I would like to thank my classmates and the entire SC&SS staff who helped me directly or indirectly in making my project a success.

**ANKEM HANU**

# CONTENTS

<b>Abstract</b>	
<b>1. Introduction</b>	<b>1</b>
1.1 Problem	1
1.2 Case tool	3
1.3 Organization of the report	3
<b>2. Object Oriented Concepts</b>	<b>5</b>
2.1 Object Oriented Methodology	5
2.2 The Object Model	6
2.3 The Dynamic Model	7
2.4 The Functional Model	7
2.5 Diagrams	8
<b>3. Problem Description</b>	<b>10</b>
3.1 University Admission System	10
3.2 Augmented State Transition Diagram	14
3.3 Integrated Diagram	16
3.4 Integrated Diagram for Univ Admission system	17
<b>4. Implementing Class Diagrams</b>	<b>21</b>
<b>5. Generating Event Trace Diagram</b>	<b>26</b>
<b>6. Implementation Issues</b>	<b>30</b>
6.1 Class Diagram	30
6.2 Inheritance Relationship	32
6.3 Deletion of Relations and Class Icons	34
6.4 Implementing drag and drop option	35
6.5 Event Trace Diagram	35
6.6 Programming Language Issues	37
<b>7. Annexes</b>	<b>39</b>
7.1 Source Code	39
7.2 Sample screens	83
<b>References</b>	<b>86</b>

## ABSTRACT

The case tool is a user-friendly application developed in VC++ which helps system analyst in object-oriented modeling and design. The case tool can handle modifications in the static and dynamic behaviour of the system.

To make design as flexible as possible, we have to take into account several requirements, specifications and interaction of the system in the real world. Current object oriented systems deal with static and dynamic behaviour of the objects. If we consider particularly the dynamic aspect several structures exist to represent this behaviour. However there is no method, which incorporates consistency in the dynamic behaviour when modifications are done. As dynamic model consists of multiple state transition diagrams, any change in the system implies a change in one or more state transition diagrams. Which may lead to set of state transition diagrams in an inconsistent state. The application introduces consistency checking through the help of integrated state transition diagrams. The current project includes implementing Class diagrams and generating Event trace diagram from State transition diagram.

The case tool accepts the static behaviour of the system through the help of a GUI. The system analyst will be able to input information about the class, attributes, methods, inheritance relationship among the classes directly without programming. It allows him to draw the Integrated State Transition diagram (ISTD) by accepting state and event information. Besides this the case tool draws event trace diagram from the ISTD and finally generates the c++ source code for the system.

# 1 . INTRODUCTION

---

The main purpose of object oriented development is to build the real world models, using an object oriented view of the world. The fundamental building blocks of object oriented development includes objects and classes. Object oriented analysis include understanding the structure of the classes, inheritance mechanism, individual behaviour of the objects and so on. It is very difficult to capture all details of the complex system in just one view. Thus we go along with the analysis in two dimensions:

1. Static structure of the system.
2. Dynamic behaviour of the system.

The static nature include identifying the classes, identifying the attributes, methods that belong to a class, identifying the inheritance relationship between the classes etc. The dynamic behaviour include the state transition that an object undergoes, the events that cause these transitions, the order in which the events are occurring etc.

## 1.1 Problem

It is possible that the requirements of the system may change during design stage of the software lifecycle. Object oriented systems are flexible enough to handle changes in the requirements. The impact of change is supposed to be easily

identified, bounded and assessed. The main issue is to identify the changes that may take place in the requirements and study how easily these changes can be incorporated in the object oriented analysis of the system. The change itself can be in the static or in the dynamic nature of the system.

The major aspects that are dealt with are the class diagrams, the state transition diagrams, the event trace diagrams. The state transition diagrams show the different states of objects of a given class, the events that cause the transition from one state to another and the actions that are to be performed.

Whenever there are multiple diagrams, then any change in the system may imply that many diagrams will have to undergo modifications, in such a situation there is a possibility that partial modifications may take place leaving the set of diagrams in an inconsistent state.

We postulate that when analyzing the dynamic behaviour of an object the different aspects of this behaviour must not be segregated and analyzed separately but all aspects must be considered together[Par 95]. The dynamic requirement analysis must consider the intra as well as inter object behaviour together. That is the analysis must deal with an integrated behaviour of objects covering state transitions that an object undergoes and the events that are sent and received by an object. If this were to be done, then any change to be made will be in one place. This would eliminate partial modifications, which leads to inconsistent diagrams.



## 1.2 Case tool

If we look back at the brief history of programming we notice that programming used to be done in machine language with minimum tools available, later on assemblers, compilers have been developed. Further advances brought editors, which displays the code in different colours, source level debuggers etc., which made programming easier. Trying to build a large software system with a minimal tool is a Herculean task. Traditional software development tools embody knowledge only about source code, but since object oriented analysis and design highlight key abstractions and mechanisms we need tools that can focus on richer semantics. Great designs come from great designers, not from great tools. Tools help the designer complete his job quickly and easily. The case tool developed helps the designer in drawing class diagrams, Integrated diagrams. It can handle consistency checking in the static and dynamic behaviour of the system. It also generates event trace diagram from the Integrated State transition diagram.

## 1.3 Organization of the report

Chapter 2 Object oriented concepts: The chapter explains in briefs the OO methodology, it discusses OO Models.

Chapter 3 Problem Description: It explains in detail the partial modification problem with a University Admission system example. The chapter also gives the approach to tackle the problem.

## *I. INTRODUCTION*

Chapter 4 Implementing Class Diagrams: The chapter gives the complete picture about how the end user has to use the casetool to draw class diagrams.

Chapter 5 Generating Event Trace Diagrams: This chapter explains how the end user can get the event trace diagram using the casetool. It also gives the advantages of object interaction diagram.

Chapter 6 Implementation Issues: This chapter explains in detail the data structures, functions, methodology used in the development of the casetool. Finally it gives the features of the programming language used.

Chapter 7 Annexes: Annexes include the source code for the case tool and sample screens.

## 2.OBJECT ORIENTED CONCEPTS

### 2.1 Object Oriented Methodology

Object oriented methodology consists of building a model of an application domain and then adding implementation details to it during the design of a system. We call this approach Object Modeling Technique (OMT).

The methodology includes:

1. Analysis: From the statement of the problem, the analyst builds a model of real world situation showing its important properties. The analysis model is a concise, precise abstraction of what the desired system must do, not how it will be done. The objects in the model must be application domain and not computer implementation concepts such as data structures. The analysis model should not contain any implementation decisions.
2. System Design: The system design makes high level decisions about the overall architecture. During system design, the target system is organized into subsystems based on analysis, structure and the proposed architecture. The system designer must decide what performance characteristics to optimise, choose a strategy of attacking the problem and make tentative resource allocations.

3. Object Design: The object designer builds the design model based on the analysis model but containing implementation details. The designer adds details to the design model in accordance with the strategy established during system design. The focus of object design is the data structures and algorithms needed to implement each class.
  
4. Implementation: The objects and their relationships developed during object design are finally translated into a particular programming language, database or hardware implementation. Programming should not be tedious; it must be more mechanical since major decisions are made in the design stage. The design should not depend upon the implementation details rather programming language must be selected which can support (Help the implementation of) the design property.

Object oriented concepts can be applied through out the system development life cycle, from analysis through design to implementation. The three OO Models include the object model, the dynamic model and the functional model.

## **2.2 The Object Model**

The object-oriented model is based on objects, which are structures that combine related code and data. The description of an object is contained in its class declaration. The basic properties of objects are called encapsulation, inheritance and polymorphism.

### 2.3 The Dynamic Model

The dynamic model describes those aspects of a system concerned with time and sequencing of operations. The dynamic model captures control. Control is that aspect of a system that describes the sequences of operations that occur in response to external stimuli, without consideration of what the operations do, what they operate on, or how they are implemented. The dynamic model consists of multiple state diagrams, one state diagram for each class with important dynamic behaviour and shows the pattern of activity for an entire system.

### 2.4 The Functional Model

The functional model describes those aspects of a system concerned with transformations of values, functions, mappings, constraints and functional dependencies. The functional model captures what a system does, without regard for how or when it is done. The functional model consists of multiple data flow diagrams, which show the flow of values from external inputs, through operations and internal data stores, to external outputs. The functional model specifies the meaning of the operations in the object model and the actions in the dynamic model, as well as any constraints in the object model.

#### Relationship among the models

Each model views the system in a different perspective to give the complete picture of the system. The object model describes data structure that the dynamic and functional models depend on. The operations in the object model

correspond to events in dynamic model and functions in the functional model. The functional model specifies what happens, the dynamic model specifies when it happens and the object model specifies what it happens to.

## 2.5 Diagrams

Class Diagrams: A class diagram is used to show the existence of classes and their relationships in the logical view of a system. A single class diagram represents a view of class structure of a system. The class diagram consists of a rectangle with a maximum of three horizontal divisions, the first division contains the name of the class, the second division contains the attributes and the third division contains methods that belong to the class. During analysis we use class diagrams to indicate the common roles and responsibilities of the entities that provide the systems behaviour. During design we use class diagrams to capture the structure of classes that form the systems architecture.

Object Diagram: The object diagram is similar to the class diagram with the ends of the rectangle rounded. There are however no partitions in the rectangle the rounded rectangle contains the values of the attributes.

The two diagrams we have introduced thus far are largely static. However, events happen dynamically in all s/w intensive systems, Objects are created and destroyed, objects send messages to one another in an orderly fashion, and in some systems external events trigger operations upon certain

objects. In object oriented development, we express the dynamic semantics of a problem or its implementation through two additional diagrams. They are State transition diagrams, Event trace diagrams.

State transition diagrams: A state transition diagram(STD) is used to show the static space of a given class, the events that cause a transition from one state to another, and the actions that result from a state change. A single state transition diagram represents a view of the dynamic model of a single class or of the entire system. STDs show the event ordered behaviour of the system as a whole. During analysis we use STDs to indicate the dynamic behaviour of the system. During design we use state transition diagrams to capture the dynamic behaviour of individual classes or of collaborations of classes.

Interaction Diagram: An interaction diagram is used to trace the execution of a scenario in the same context as an object diagram. Indeed to a large degree, an interaction diagram is simply another way of representing an object diagram.

### 3. PROBLEM DESCRIPTION

---

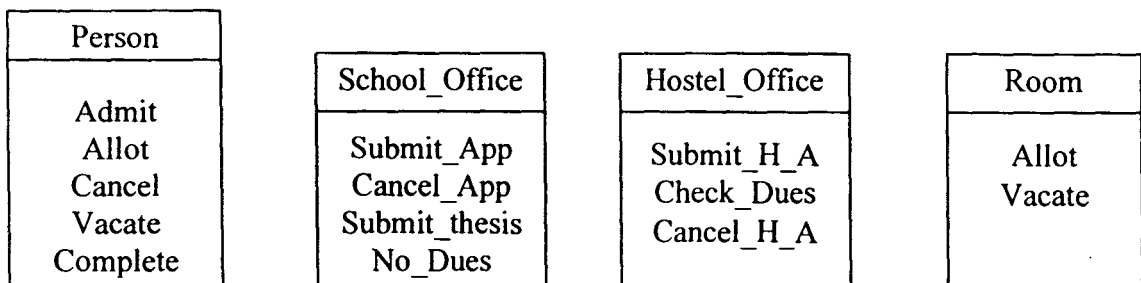
Whenever there are multiple state transition diagrams representing the dynamic behaviour of the system, then any change in the system may imply that many diagrams will have to undergo modifications. In such a situation, there is a possibility that partial modification may take place leaving the set of diagrams in an inconsistent state.

In this chapter we perform analysis of an example system and arrive at the different diagrams of the analysis document, later, we assume that requirements undergo a change and show how partial modifications can take place.

#### 3.1 University Admission System

Consider a University Admission system where a person takes admission into a school. If he is not a local student, he can apply for hostel accommodation. The student can be relieved from the university in two ways either by submitting the thesis or by applying for course cancellation. In either case if he is a resident he has to vacate the room to get his dues cleared. The class diagrams for the University Admission system can be shown in fig: 3.1.

Fig: 3.1.Class Diagrams





The state transition diagrams for the Person, School\_Office, Hostel\_Office, Room are given in the fig 3.2. Using the notation of Rumbaugh.

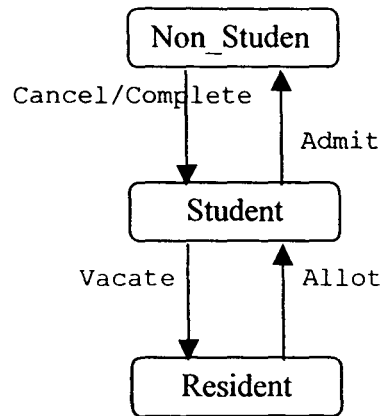


Fig:3.2(a) State Transition Diagram of Person.

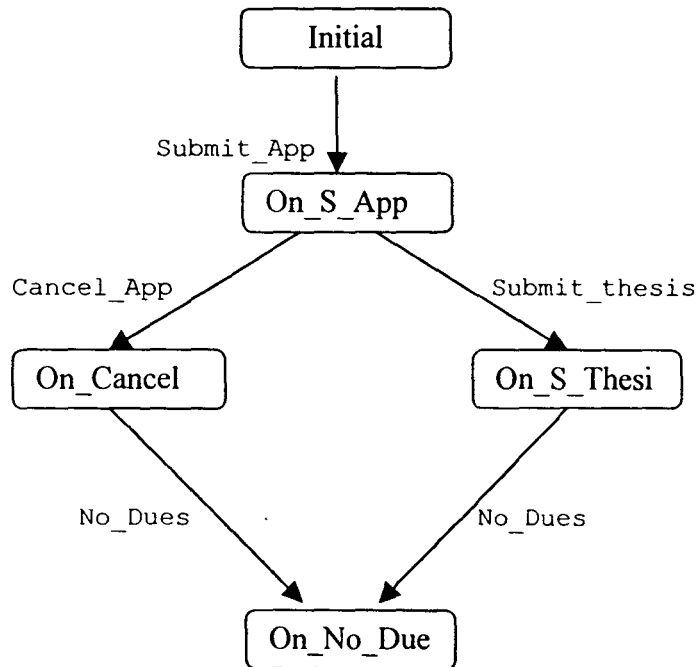


Fig:3.2(b) State Transition Diagram of School\_Office.

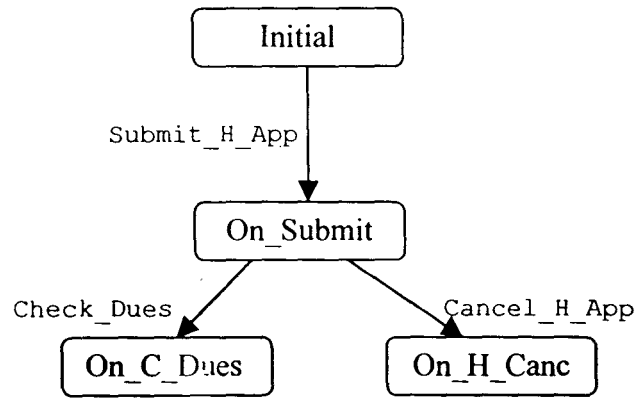


Fig:3.2(c) State Transition Diagram of Hostel\_Office.

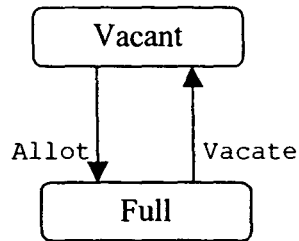


Fig:3.2(d) State Transition Diagram of Room.

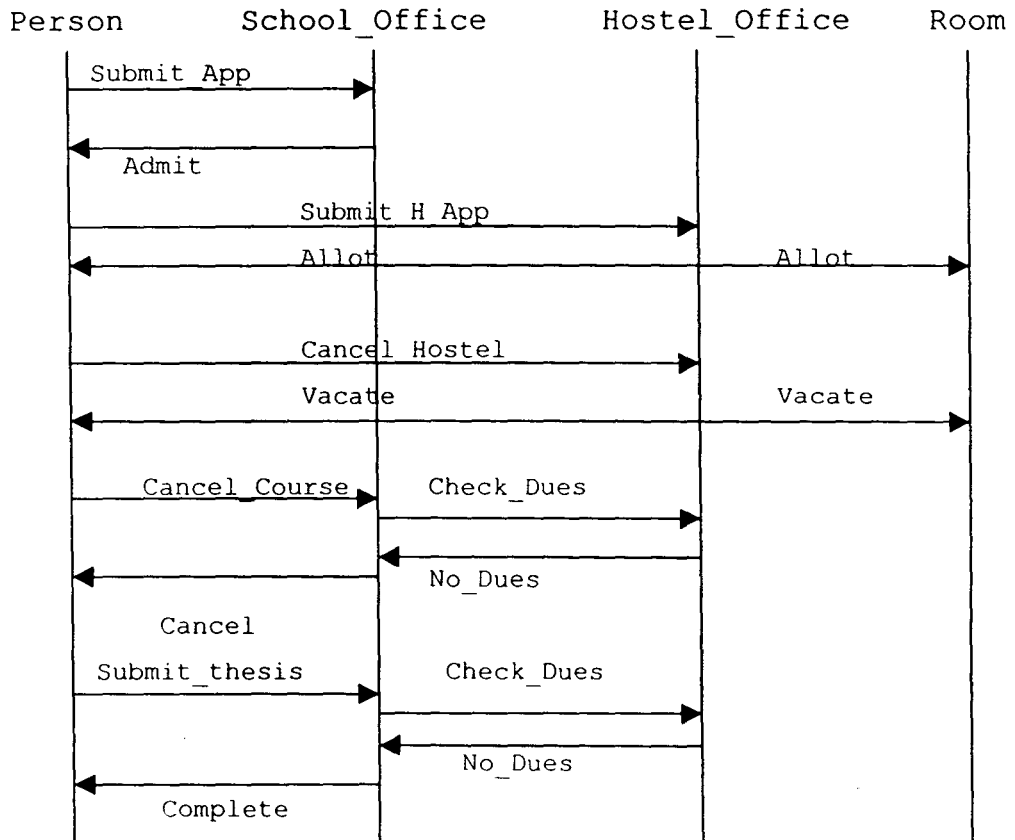


Fig: 3.3.Event Trace Diagram.

The event trace for the system is shown in the fig 3.3.

Let us now assume that there is a change in the policy of Admission policy. The new rule is that no cancellation is allowed. Notice that the change is to be incorporated then modifications to different diagrams has to be made to reflect the situation. In particular, the state transition diagram of School\_Office has to be modified. Here the On\_cancel State has to be absent. Similarly in the class Person transition from student to non\_student should be removed. The event trace for the scenario of cancel will have to be removed.

If it happens that only the state transition diagram of School\_Office is modified to the one in fig 3.4. Then we see that the set of state transition diagrams for the University Admission system will be in an inconsistent state. There is nothing in the system to prevent this situation. Clearly if the changes were to be made in just one place then there would be no inconsistency.

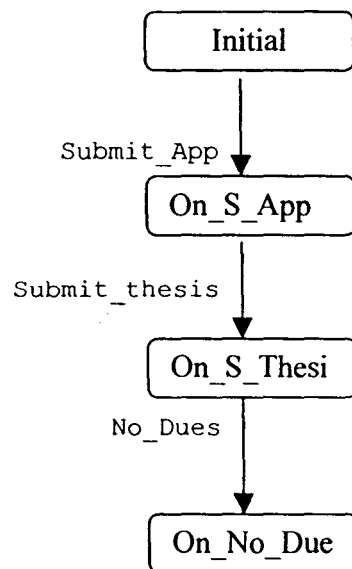


Fig: 3.4.State Transition of School\_Office

We propose to augment the state transition diagram to include not only the state transition of objects of a given class  $c_1$  but also the states of objects of class  $c_2 \dots c_n$  which cause the state transitions of objects of  $c_1$ . [Par 95] To achieve this, first we define the augmented state transition diagram (ASTD) then the Scenario State transition diagram (SSTD) and finally integrated diagram (ID).

### 3.2 Augmented State Transition Diagram

Let an object  $o_1$  belonging to class  $c_1$  in the state  $os_1$  make a transition to state  $os_2$  when the event  $ev_1$  takes place. Let  $ev_1$  be an event caused by an object  $o_2$  belonging to class  $c_2$  in state  $os$ . Then, a dotted line with the label  $ev_1$  is drawn from  $OS$  to  $O_2$  to the transition of  $O_1$  from  $OS_1$  to  $OS_2$  as shown in the fig 3.5.

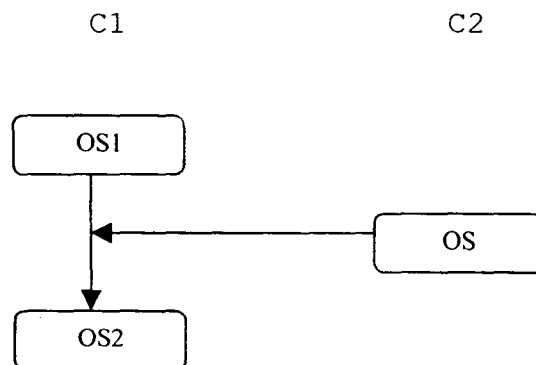


Fig: 3.5.

The above augmented state transition diagram specifies:

1. The state transition of an object of a class  $C_1$ .
2. The class  $C_2$ , an object of which causes the event and
3. The state in which it can cause the event.

It may be possible that objects belonging to more than one class can cause the event *ev1*. In such a situation the ASTD consists of

1. The state transition of an object of a class *cl*,
2. All the classes, an object of each of which cause the event and
3. The state in which each of these objects can cause the event.

Consider the University Admission example defined in earlier section, The ASTD for the Admission into the school is shown in fig 3.6. Person in state *Non\_Student* causes the event '*Submit\_App*' when the *School\_Office* is in initial state. The counter then moves to *On\_Submit*, this state causes the event '*Admit*' which transforms person from *Non\_Student* state to *Student* state.

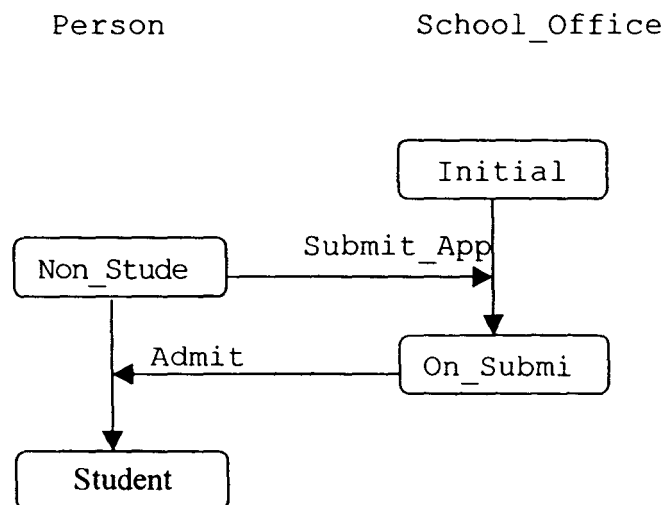


Fig: 3.6. Augmented State Transition Diagram

Scenario State Transition Diagram:

An event scenario is a sequence of events occurring during one particular execution of the system. It gives the event trace. The scenario state transition diagram is drawn as follows: Start with the first event of the event trace and draw the ASTD for the object, which goes through a state transition when this event occurred. Pick up the next event and extend the existing state transition diagrams with states, transitions and events to include the ASTD for the state transition caused by the new event. Continue till all the events in the event trace are exhausted.

For the library system, consider the event trace shown in fig 3.3. The first event is 'Submit\_App'. This causes a state transition in School\_Office. The ASTD for Admission will be shown in fig 3.6. The next event is Admit. This causes a state transition in Person from the state Non\_Student to Student. Similarly we can trace the sequence of events for the complete system.

**3.3 Integrated Diagram**

An Integrated Diagram consists of all the scenario state transition diagrams of a given system. That is, it includes the scenario state transition diagrams for all the event traces of the system.

Graphical Notation: A state is denoted by a rounded rectangle with its name in it. The class name to which the object belongs is written above the ASTD of the object. Each objects

state transition is in a vertical line. State transitions are denoted as solid lines with an arrowhead. The head points to the new state of the object. The events are denoted as dashed lines with an arrowhead. The line originates from the state of the object, which undergoes the particular transition as a result of the event.

Solution to the partial modification problem:

The integrated diagram gives the complete specification of the dynamic behaviour of the system, which includes the state transition of objects of various classes and the states of objects of other classes causing the event. When the system undergoes a change in its dynamic behaviour then all the changes to be made in the analysis diagrams is in one place. Thus any change can be reflected without giving rise to any partial modification.

### **3.4 Integrated Diagram for the University Admission System**

In this section we consider the University Admission example again draw an integrated diagram for the dynamic behaviour. The static analysis remains the same. Therefore, we have the object classes Person, School\_Office, Hostel\_Office, Room. The Integrated diagram is shown in Fig.3.7.

The School\_Office is in Initial State, which receives an event submit\_App from the person in the Non\_Student State. The School\_Office then moves to the next state On\_Submit. Which causes the event Admit. This event Admit causes a transition in person from Non\_Student State to Student State. The student may submit Hostel\_App, which causes a transition

in the Hostel\_Office from Initial to On\_Submit. The student and the room receive event Allot from the On\_Submit state of the hostel, with the event the Person moves from Vacant state to Full State. Rest of the events can be explained similarly.

If the new policy is introduced, then the On\_cancel State for the School\_Office will be removed and the events Cancel\_App will be deleted. Applying the changes to the fig.3.7 will give the fig.3.8.



Person                      School\_Office                      Hostel\_Office                      Room

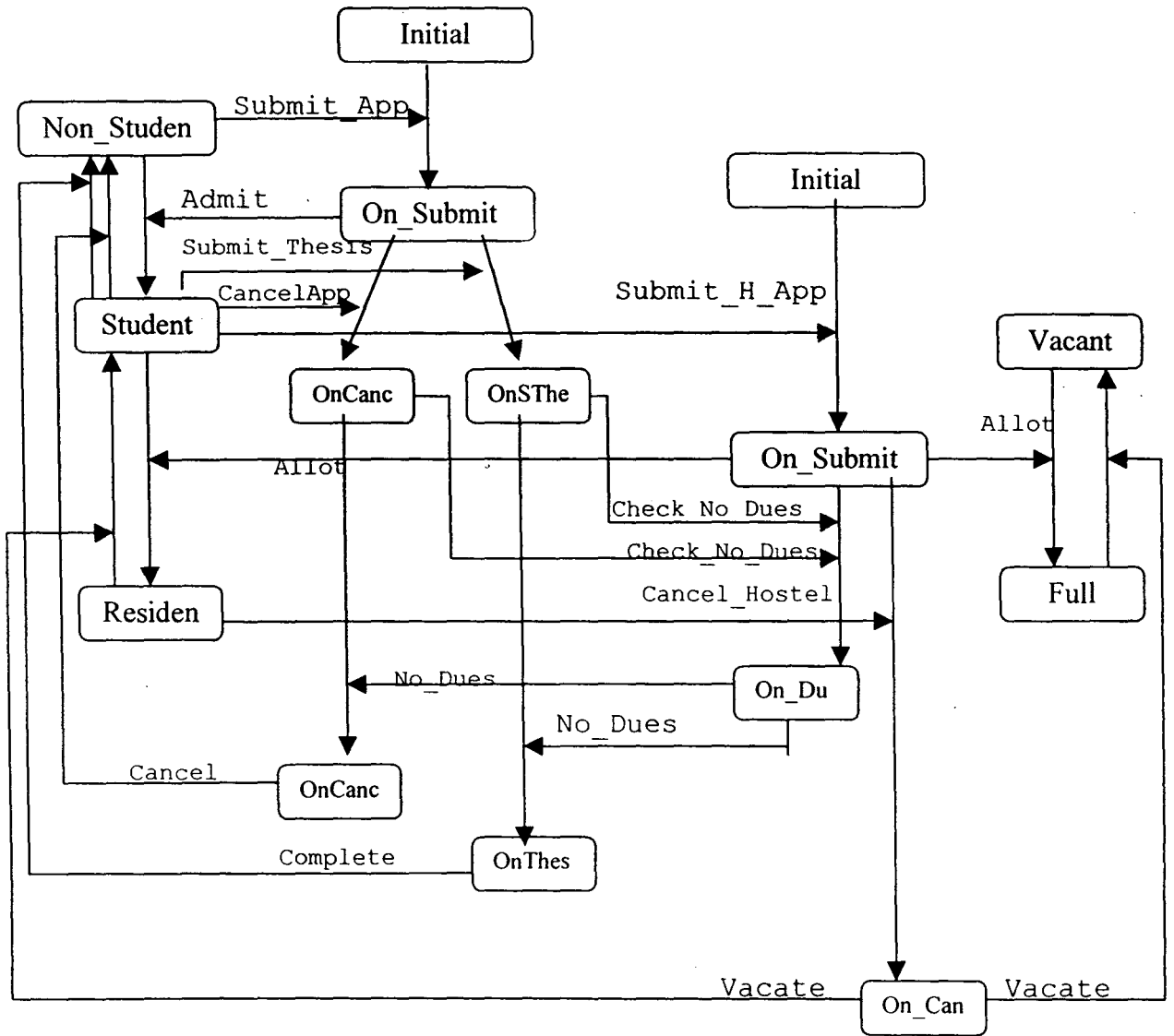


Fig 3.7

Person

School\_Office

Hostel\_Office

Room

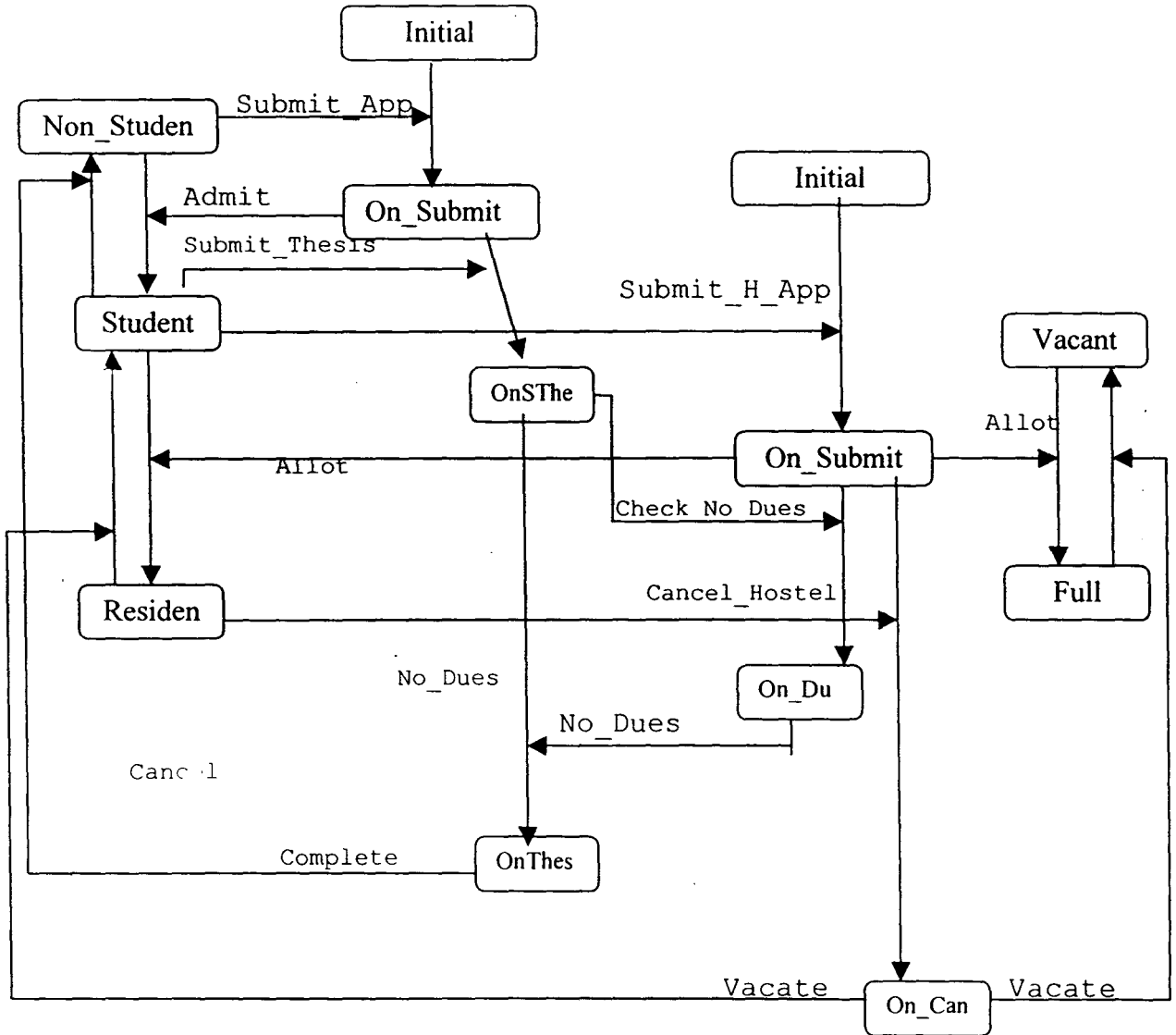


Fig 3.8

## 4. IMPLEMENTING CLASS DIAGRAMS

The first step in implementing an object-oriented design is to declare object classes. Each attribute and operation in an object diagram must be declared as part of its corresponding class. The classes and their relationships gives the top level logical architecture of the system.

In the current project the system analyst must first enter the model name to start his design and a class diagram window is opened for him. The tool is very easy to use, he has to just select class from the draw menu item or just click on the class icon of the tool bar, then he has to click on the client area to create a new class with the default class name. The default class names will be class1, class2... etc. The class icon has three portions in t he first portion the default class name is written. The other two portions are for entering variables, and methods, which belong to the class. These portions are initially empty and they have scroll bars attached to them. If the number of attributes or methods is more then the scroll bars can be used to view them. See fig 4.1. Inorder to enter the name of the class, the mouse cursor should be positioned on the class icon and its right button is clicked. The tool opens a dialog box which has a number of edit boxes which include class name, position (X0, Y0) the left corner co-ordinates of the icon, the height and width edit boxes gives the default size of the icon. Besides this it has two edit-boxes to enter the attribute names and the method names. See Fig 4.2.

TH-7645



The cursor is brought to the class name editbox and the required class name is typed.

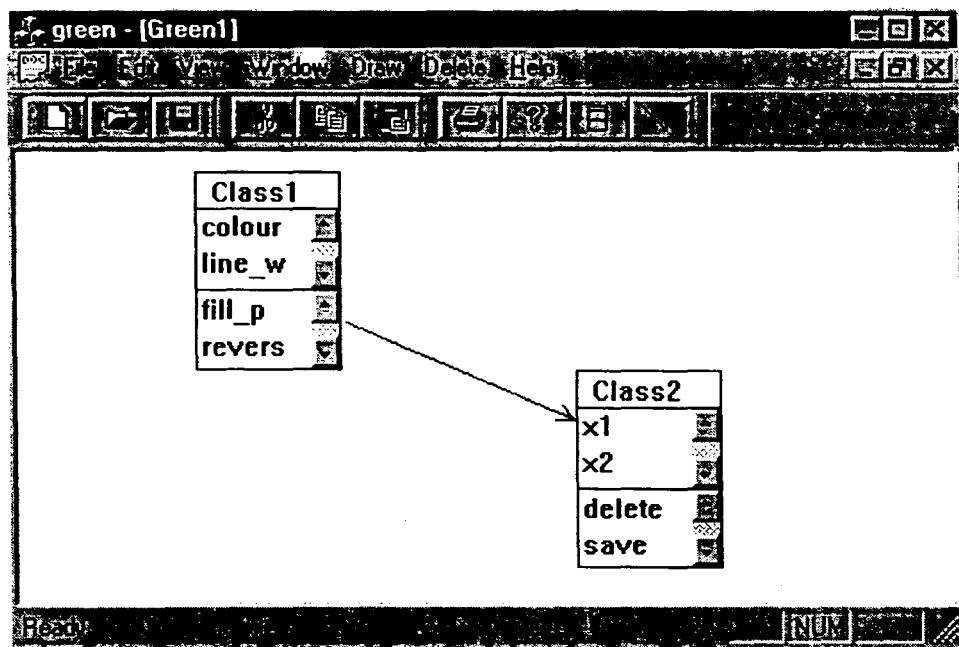


Fig 4.1 Class Diagrams

If there are more than one, class icons in the client area then the tool checks the current name with the name typed. It doesn't allow duplicate class names. Changing the XO, YO values can change the position of the class icon. The size of the icon can be increased or decreased by changing the height and width values. However any value smaller than 50 and any value greater than 400 will not be accepted. For every invalid input corresponding warning is messagebox is opened displaying the problem/valid input that must be entered. Next in the sequence is the attribute edit box. Here the attribute name along with the datatype is typed and the next attribute button is clicked the contents of the editbox will get cleared if the attribute name is not given earlier for that



class indicating the user to enter next attribute. If the same attribute name is typed then a messagebox indicating

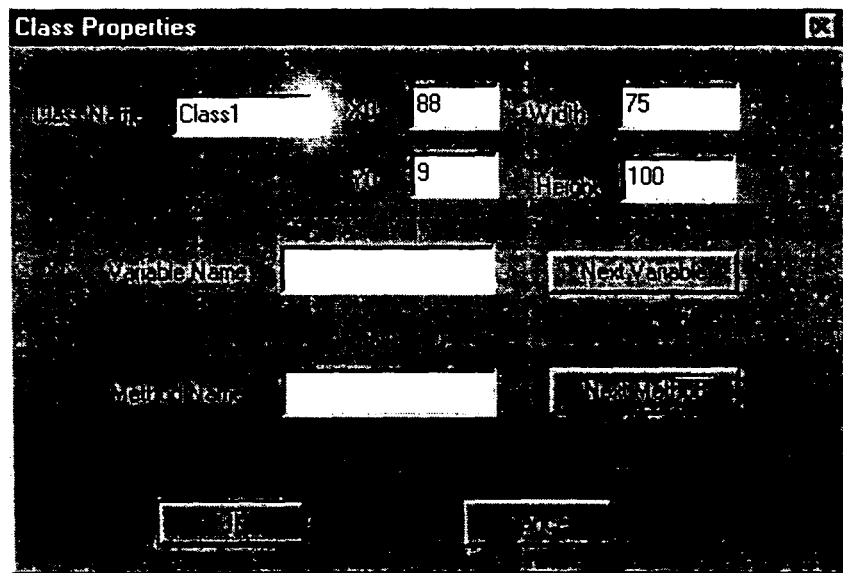


Fig 4.2 Class Properties Dialog Box

that duplicate attribute name is shown. After entering the last attribute next attribute button must be clicked to accept the value. Similarly method names are also entered in the corresponding edit box. Finally OK button is clicked to accept the values entered. If cancel button is clicked then the dialogbox is closed and no modifications are made to the classname, X0, Y0, height, width values. However the values entered for the attributes and methods will be accepted, thus care must be taken while entering these values. After the dialogbox is closed the class icon is available with the entire modifications name, position, size etc. The attributes and methods will be displayed in the second and third portions of the icon. If the numbers of attributes/methods

are more than the scroll bars can be used to view them. If the class name, attribute name or method names are long to fit into the class icon then these names are truncated and displayed in the icon. The actual names however will be available and can also be seen either by increasing the width of the icon or by right clicking the icon and viewing the names in the properties. Repeating the above procedure required class icons could be drawn in the client area of the window.

The icons can also be moved using drag and drop method. The mouse cursor is brought over the icon, its left button clicked without releasing and the mouse is moved to the required location and the button is released to place the icon over there. If the icon is dragged and dropped over another icon then the icon will move back to its previous position.

After drawing all the class icons, the next step in the design is to link the classes if there is any inheritance relationship between the classes. This can be done by selecting the inheritance menu item from the links menu from the menu bar or by selecting directly from the tool bar and clicking on the classes consecutively. The first click must be on the base class and the second must be on the derived class. An arrow is drawn from the base class to the derived class. However, if there is any class icon between the base class and the derived class then the arrow is drawn over the icon. Proper positioning of the icons can eliminate the links over a class. The casetool doesn't allow the user to enter an inheritance relationship if it is already existing between the classes. It also checks illegal inheritance relationships

between the classes and displays the same when the user tries to do so. If class1 is the base class of class2 and class2 is the base class of class3 then class3 is indirectly is a derived class of class1. If the user tries to add an inheritance relation from class3 to class1 then the casetool does not allow him, thus avoiding incorrect relationships entered accidentally. The drag and drop option still work after the inheritance relationships are added to the class icon. As the icons are moved the links move automatically in effect to the change. Deleting the class icons is the option, which is necessary if a class is created by mistake or which has no significance. This option is also available for the user in the casetool. The user has to get the mouse cursor over the class icon and double click over it to select the class. The class icon along with all the links to other class icons will turn blue showing the effect of selection. To delete the highlighted part cut toolbar option or delete class menu item is clicked and a Messagebox warning the user that the highlighted part will get deleted is displayed. The user can delete the highlighted part by clicking cancel. Deleting the links is also provided in a similar fashion. However no class icons will get deleted while deleting the links. The class properties can be modified at any time by right clicking the class icon and modifying the values in the edit box.

Finally class diagrams can be saved by clicking on save option of the toolbar or from the save menu item. The class icons along with links are saved in a file with the default name modelname.cls this name can be changed by the user. The user can get back the class diagram from such a file. After completing class diagram he can switch over to ISD part of the casetool through the menu.

## 5. GENERATING EVENT TRACE DIAGRAM

An interaction diagram also known as event trace diagram is used to trace the execution of a scenario in the same context as an object diagram. An interaction diagram is simply another way of representing an object diagram. The advantage of using an interaction diagram is that it is easier to read the passing of messages in the relative order. A useful Object oriented design methodology must provide the designer with the means to handle complex situations effectively. Inter object relationships are not always as simple as they are presented in some rather trivial examples of object oriented design. To handle complex situations effectively we use the concept of object interaction diagram.

In the interaction diagram there are no icons. The object names are written horizontally across the top of the diagram. A dashed vertical line is drawn below each object. Message or events are shown horizontally using horizontal arrows. The endpoints of the message icons connect with the vertical lines that connect with the entities at the top of the diagram and are drawn from the client to the supplier. Ordering is indicated by vertical position, thus the first message is at the top of the diagram and the last message is at the bottom of the diagram.

In the case tool after drawing the integrated diagram it is saved and in order to obtain the event trace click on the event trace menu item from the menu bar. The case tool opens



another document containing the event trace, which is obtained through the use of threads along each object in the integrated diagram. The designer can just view the event trace equivalent to the previous drawn integrated diagram no modifications to the event trace is allowed. The event trace for the example-Integrated diagram (Fig 5.1) will be shown in the fig 5.2.

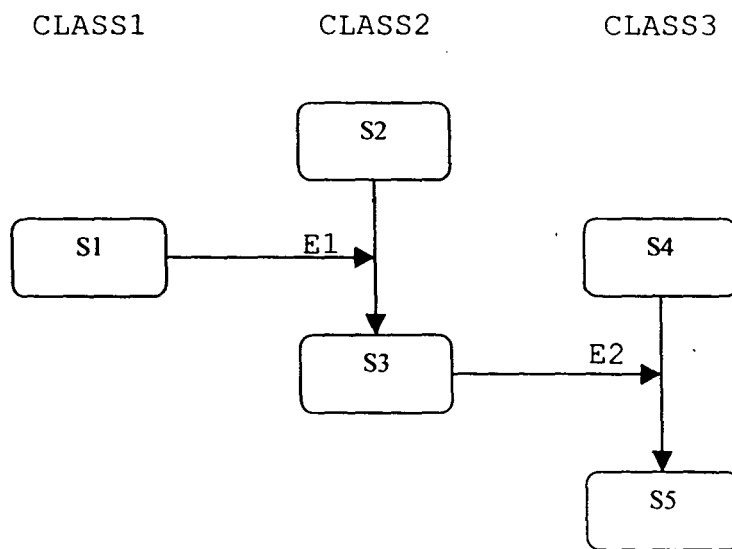


Fig 5.1

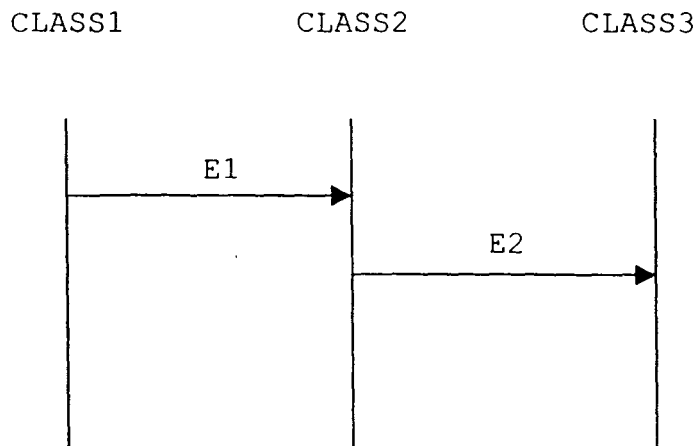


Fig 5.2

The implementation details will be dealt in the next chapter.

The advantages of the use of the object interaction model in system development are:

1. Consistency with the object model, since it describes the possible paths of interaction between objects that are defined in the object model.
2. Determination of dependencies among the objects. OIDs depict those services of cooperating objects that an object calls, in order to succeed in offering its services. This fact inherently creates dependencies among the objects that may be either strong or weak. In any case, we immediately know the list of cooperating objects and their specific relations.
- 1.3. The OID provides the whole set of object operations, enabling the easy design of state diagrams. The state diagram requires the whole set of object interactions, in order to be completely defined. We can take this information from the relevant OID. We must state at this point that using object expansion, we arrive at simple primitive objects that offer a very specific and constrained set of services and that usually have very simple state diagrams. A complex primitive object is usually one that has already been designed and is reused in the current design; consequently we are not interested in redesigning it again. Our interest shifts from the internal aspect of the object and how it exhibits its behaviour to its external interface and its cooperation with the other objects within the system.

3. Immediate view of what roles each object plays in the system according to the mechanism it participates in and of the surrounding context of the object.
4. In large real time systems things are not ideal and design cannot always be constrained by the general directions and tactics given by the Object oriented community. Sometimes, groups of objects that are related are partially overlapped. The created complex structures can be recognized through the use of OIDs and thus designed appropriately.
5. Documents the effects that an event can cause within the system.
6. If we consider the maintenance or extension of the system by people who have not been involved during the initial development, we find that OIDs facilitate the study of already developed systems.

## 6. IMPLEMENTATION ISSUES

### 6.1 Class diagram

As the number of classes in an application is a variable a linked list data structure is selected to store class information. Whenever a class icon is drawn in the client area a new node is created and added to the linked list. Each class has two scrollbars, hence two pointers to the scrollbar objects are declared. Each class has different number of attributes/methods to handle them two linked lists are defined in the class data structure. The overall data structure for the class icon is shown in fig 6.1.

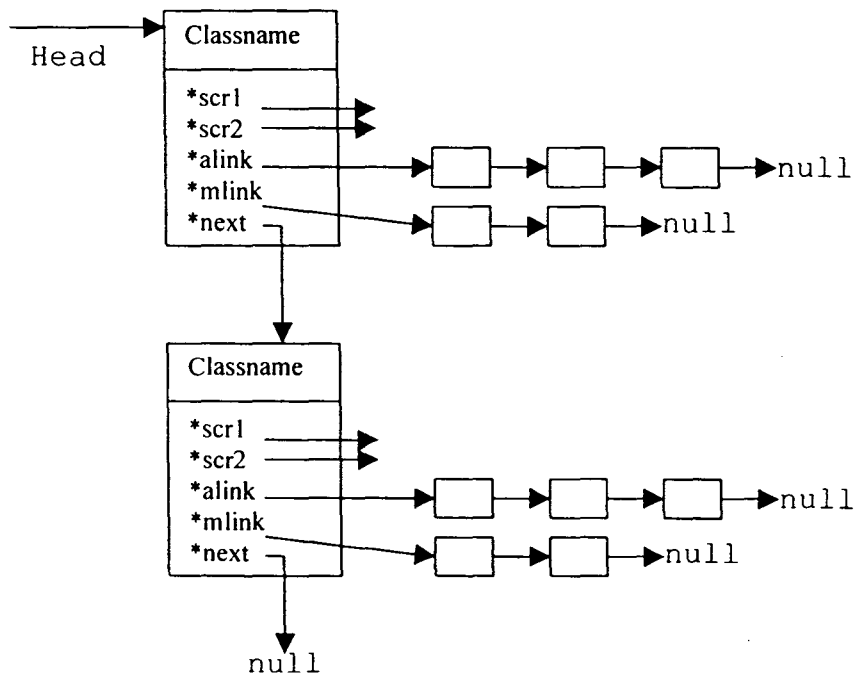


Fig: 6.1

The icon further has variables x0, y0 to store the top left co-ordinates of the icon and Width, height to store the size of the icon. Whenever a class icon is created on the client area the global count value is incremented (initially zero) and default name for the class icon is set with the count value. The default class names will be class1, class2... etc. As the attributes and methods are added to the class, new nodes are created and added to the linked list. As the scroll bars are created with their position fixed the movement of the icon requires deletion of the scrollbar objects and recreating them in the new position. Whenever the class name is changed the new class name is accepted if and only if it does not match with any of the other class names. Similar checking is done for the attributes and methods also.

Depending upon the size of the icon the number of characters that can fit into the icon is calculated. If the class name/attribute names/method names are longer then they are truncated and displayed over the icon. Whenever the user clicks on the arrows of the scrollbars the current contents are cleared and the new contents are printed depending on the button clicked. In the program Head is the pointer which always points to the starting node of the linked list, while the variables like move, temp are used to traverse the list.

In-order to change the properties of the class icon a dialogbox object is created whenever the user right clicks the icon. This dialogbox uses DDX (Do Data Exchange) function to accept the values from the user and DDV (Do Data Validation) function to do validation checking on the data.

## 6.2 Inheritance relationship

The number of inheritance links is a variable hence linked list data structure is used to store the information. Each node contains variables class1, class2, clpos, c2pos. Class1 corresponds to the base class while class2 corresponds to the derived class, clpos is the position value on the class1 from where the arrow has to be drawn and c2pos is the position value on the class2 where the arrow ends. The clpos, c2pos are calculated as follows:

1	8	7
2	c1	6
3	4	5

Fig: 6.2

Assume that class icon is present in the closed region of the fig 6.2. The rest of the region can be divided into eight regions with respect to the class icon. If the class2 is completely in the region1 then clpos will be 1 and the arrow is drawn as shown in fig 6.3.

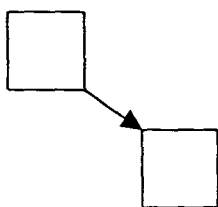


Fig 6.3

If the class2 is partially in region1 and partially in region2 then clpos will be 2 and the arrow is drawn as shown in the fig: 6.4.

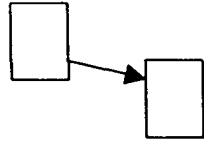


Fig 6.4

If the class is completely in the region2 then clpos will be 3 and the arrow is drawn as shown in fig 6.5.

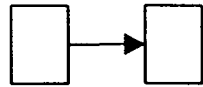


Fig 6.5.

Similarly the boundary of a class icon has 16 positions, which is assigned to clpos depending upon the relative position of the class2. The function coordinates () calculates the absolute co-ordinates from the class and the clpos.

Whenever a new inheritance relation is drawn a new node for the relation is created and added to the linked list. Before doing so the following validations are done.

1. If there is already a relation existing between the classes: This is verified by checking the class1 and class2 values in all the existing nodes.
2. If the new relation forms a circular link: This is verified by the checkrelation() function, which checks the existing, links in recursive fashion eliminating the links one by one.

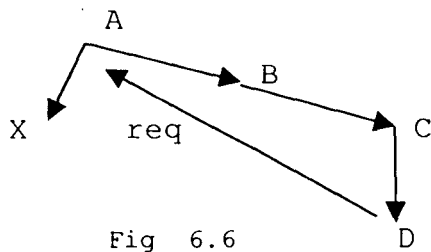


Fig 6.6

Consider the fig 6.6. Let A, B, C, D, X be the classes and the arrows represent the relations. If the user tried to draw relation from D to A then checkrelation function is called with A,D as parameters. From A there are paths to B and X, So the function recursively calls itself with B,D and X,D as parameters. From X there are no links so the function terminates. The function called with B,D will recursively call with C,D as parameters followed by call with D,D parameters which implies that this is a case of cycle formation hence the new link is not accepted.

### 6.3 Deletion of relations and Class icons

When the mouse button is double clicked over a link findrelation() is used to get the link number of the link. From the link number the link is deleted from the linked list and the arrow on the client area is erased. When the mouse button is double clicked on the class icon, the icon along with all relations towards the class and from the class are highlighted representing the selection. If delete option is selected then the highlighted part gets deleted.



#### 6.4 Implementing drag and drop option

As the mouse cursor is moved within the view window, the `OnMouseMove` function is called in frequent intervals when the drag operation is in progress (if `m_dragging=1`, mouse is clicked on the icon and moved) `OnMouseMove` first creates a device context object associated with the view window. It then calls the `CDC::SetROP2` function to create a drawing mode in which lines are drawn by inverting (reversing) the current colour on the screen. Under this mode, when a line is first drawn at a particular position, it's visible; however, when a line is drawn a second time at the same position, it becomes invisible. The mouse message handlers are thus able to easily draw and erase a series of temporary rectangles. The rectangles are drawn using `CDC::MoveTo` and `CDC::LineTo` functions. When the button is released `OnLButtonUp` function is called which ends the drag operation (`m_dragging` set to 0) and the new co-ordinates are noted. If the new position for the icon overlaps any other icon then the icon is not drawn at the new position. If it doesn't overlap any icons then the icon is drawn at the new location. If the icon is linked to other icons through inheritance relationships the `find_clear_relation` function finds all the links from & towards the icon and clears them on the screen. After moving the icon to new position the relations are redrawn.

#### 6.5 Event Trace Diagram

In the application after the Integrated diagram is drawn clicking corresponding menu item draws its equivalent event

trace. For this threads equivalent to the number of classes in the system are created. Consider the fig 6.7.

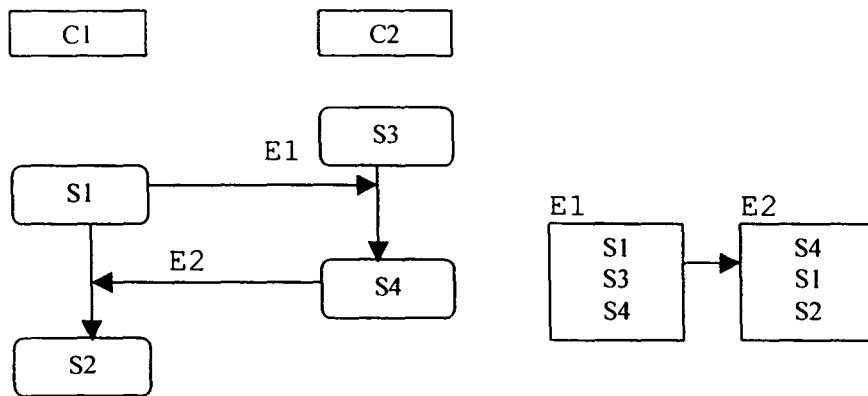


Fig 6.7

Here t1 and t2 are the threads created. Each thread calls the functions as follows:

```

starting_state=getstate();
While(starting_state!=Laststate)
{
    enable_event();
    wait_for_event();
    update(starting_state);
}

```

starting\_state is initially assigned to first state of the class through the getstate function. Here s1 is the starting state of t1 and s3 is the starting state of t2. Each event node has information about three states; the first is the state, which enables the event. The third state is reached from the second state as a result of the event. Now from each state enable\_event function is called which checks the event nodes to match the starting\_state with the first state of the event node, If it matches then corresponding event is enabled. In the example t1 enables e1 while t2 doesn't enable any event. Each thread next calls wait\_for\_event function, this function matches with the second state of the event nodes. If it matches then it checks whether the event has

been enabled. If it is enabled it moves to the next state else wait for the event. In the example there is a match for s1 in the second position of the e2 event, but e2 event is not enabled so t1 has starting\_state as s1. For s3 there is a match in the e1. E1 event is enabled earlier by the t1 thread so the start\_state variable is set to s4 for the thread t2. Now enable event is again called which enables e2 event, t1 thread finds this event and updates its starting state to s2. As there are no transitions from s2 and s4 the thread functions will stop execution. Through the use of threads the sequence in which the events occur is found and event trace diagram is drawn as show in fig: 6.8.

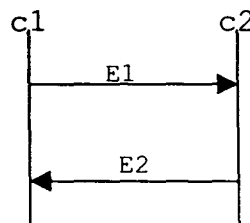


Fig 6.8

## 6.6 Programming Language Issues

### Main Features provided by VC++

1. It is object oriented programming language.
2. The user need not do low level windows programming. VC++ exploits API (Application Programming Interface) functions, MFC (Microsoft Foundation Classes), DLL (Dynamic Link Libraries) to achieve its effect.
3. It is available with Microsoft Developer Studio, which provides application wizards, class wizards that make programming easier.

4. It supports document-view architecture, where document handles the data and view handles visual display.
5. The applications developed using VC++ are device independent. All drawing calls are made through the device context object, which encapsulates the windows API to draw on the device.
6. It supports multi-threading.
7. Finally it is easy to switch over from C++ to VC++.

#### Microsoft Developer Studio for VC++:

The Developer studio is the core of the VC++ product. It's an integrated application that provides a complete set of programming tools. The Developer Studio includes a project manager for keeping track of your programs, source files and build options. A text editor for entering program source code, a set of resource editors for designing program resources such as menus, dialog boxes and icons. It also provides programming wizards (App Wizard and Class Wizard) which help in generating the basic source code, define C++ classes, handle window messages and perform other tasks. Programs can be build and executed from within the Developer Studio, which automatically runs the optimizing compiler, the incremental linker and any other required build tools. Debugging programs can be done using the integrated debugger. Finally VC++ online help can be had from the help menu of the Developer Studio.

### 7.1 Source Code

```
//classdecl.h

#include<afxwin.h>
#include<math.h>
#include "resource.h"

////////////////////////////////////////////////////////////////

// The relation class contain information about the
// inhertance relation between the classes
// ie. classname-1 , classname-2 , clpos,c2pos-position
// class1 & class2 to draw a relation
// count is a unique value distinguishing one relation
// from another
// status is used to reposition the links
// next link is used to implement linked list structure.

struct relation
{
    //CString name;
    int class1;
    int clpos;
    int class2;
    int c2pos;
    int count;
    int status;
    relation *next;
};

////////////////////////////////////////////////////////////////

// this structure stores information about the
// variable names & method names both variables &
// methods use the same attrib structure.
// next is used to implement a linked list data structure

struct attrib
{
    CString name;
    int type;
    struct attrib *next;
};
```

```
////////////////////////////////////  
// clasdiagramar stores information about each class icon  
// that is displayed on the screen. name-contains the  
// name of the class, cno contains the unique class number  
// x0,y0 contains the initial position(left corner) of  
// the diagram & width, height will give its corresponding  
// values alink is a link to the attribute linked list  
// mlink is a link to the method linked list  
// scr1 & scr2 will point to scrollbar objects of the  
// class. next is used to give it a linked list structure  
  
struct clasdiagramar  
{  
    CString name;        //Struct to store object information  
    int cno;  
    int x0,y0;  
    int width;  
    int height;  
    struct attrib *alink; // pointer to variables  
    struct attrib *mlink; // pointer to methods  
    CScrollBar *scr1,*scr2;  
    clasdiagramar *next;  
};
```

```
////////////////////////////////////  
  
struct classsave  
{  
    char name[20];  
    int cno;  
    int x0,y0;  
    int width;  
    int height;  
  
};
```

```
////////////////////////////////////  
  
// clasdiagramar stores information about each class icon  
// that is displayed on the screen. name-contains the  
// name of the class, cno contains the unique class number  
// x0,y0 contains the initial position(left corner) of  
// the diagram & width, height will give its corresponding  
// values alink is a link to the attribute linked list  
//mlink is a link to the method linked list  
// scr1 & scr2 will point to scrollbar objects of the  
// class. next is used to give it a linked list structure  
  
struct clasdiagramar  
{  
    CString name;        //Struct to store object information  
    int cno;  
    int x0,y0;  
    int width;  
    int height;  
    struct attrib *alink; // pointer to variables  
    struct attrib *mlink; // pointer to methods  
    CScrollBar *scr1,*scr2;  
    clasdiagramar *next;  
};
```

```
////////////////////////////////////  
  
struct classsave  
{  
    char name[20];  
    int cno;  
    int x0,y0;  
    int width;  
    int height;  
  
};
```





**// MainFrm.cpp : implementation of the CMainFrame class**

```

#include "stdafx.h"
#include "green.h"

#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)

//{{AFX_MSG_MAP(CMainFrame)

    // NOTE - the ClassWizard will add and remove mapping macros here.
    //      DO NOT EDIT what you see in these blocks of generated code !

    ON_WM_CREATE()

//}}AFX_MSG_MAP

END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)

```

```

        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;        // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizable toolbar

    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    // be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CMDIFrameWnd::PreCreateWindow(cs);
}

////////////////////////////////////
///
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif // _DEBUG

```

**// green.h : main header file for the GREEN application**

```

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CGreenApp:
// See green.cpp for the implementation of this class
//
void DrawArrow(CPoint &st,CPoint &en,CDC *pdc);
CRect GenerateRect(CPoint &p);
int checkside(CPoint &p1,CPoint &p2,CPoint &p3);

CString get_dir();
class CGreenApp : public CWinApp
{
public:
    CGreenApp();
    void temporaryfile();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGreenApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CGreenApp)
    afx_msg void OnAppAbout();
    afx_msg void OnNewmodel();
    afx_msg void OnIsd();
    afx_msg void OnEt();
    afx_msg void OnFileNew();
    afx_msg void OnFileOpenIsd();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

```

```
// green.cpp : Defines the class behaviors
```

```
#include "stdafx.h"
#include "green.h"
#include "newdoc.h"
#include "etdoc.h"
#include "etview.h"
#include "cmoddlg.h"
#include "newview.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "greenDoc.h"
#include "greenView.h"
#include "moddlg.h"
#include "direct.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CGreenApp
CString dire;
BEGIN_MESSAGE_MAP(CGGreenApp, CWinApp)
   //{{AFX_MSG_MAP(CGGreenApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_NEWMODEL, OnNewmodel)
    ON_COMMAND(ID_ISD, OnIsd)
    ON_COMMAND(ID_ET, OnEt)
    ON_COMMAND(ID_FILE_NEW, OnFileNew)
    ON_COMMAND(ID_FILE_NEW_ISD, OnIsd)
    ON_COMMAND(ID_FILE_OPEN_ISD, OnFileOpenIsd)
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
// CGreenApp construction

void CGreenApp::temporaryfile()
{
    CWinApp::OnFileNew();
}

CGreenApp::CGreenApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```

////////////////////////////////////
// The one and only CGreenApp object

CGreenApp theApp;

////////////////////////////////////
// CGreenApp initialization

BOOL CGreenApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls(); // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();
    // Call this when linking to MFC statically
#endif

    LoadStdProfileSettings();
    // Load std INI file options (including MRU)

    // Register the application's document templates. Documenttemplates
    // serve as the connection between documents, frame windows and views.

    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_GREENTYPE,
        RUNTIME_CLASS(CGreenDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CGreenView));
    AddDocTemplate(pDocTemplate);

    CMultiDocTemplate* pl;
    pl = new CMultiDocTemplate(
        IDR_MENU1,
        RUNTIME_CLASS(newdoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(newview));
    AddDocTemplate(pl);

    CMultiDocTemplate* pet;
    pet = new CMultiDocTemplate(
        IDR_MENU3,
        RUNTIME_CLASS(etdoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(etview));
    AddDocTemplate(pet);

```

```

// create main MDI Frame window

    CMainFrame* pMainFrame = new CMainFrame;

    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

// Parse command line for standard shell commands, DDE, file open
//     CCommandLineInfo cmdInfo;
//     ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line
//     if (!ProcessShellCommand(cmdInfo))
//         return FALSE;

// The main window has been initialized, so show and update it.

pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
return TRUE;

}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
// No message handlers
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

```

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CGreenApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////
// CGreenApp commands

void CGreenApp::OnNewmodel()
{
    moddlg d;
    if(d.DoModal()==IDOK)
    {
        CString name;
        CDocTemplate *p1;
        dire=d.m_model;
        POSITION pos=GetFirstDocTemplatePosition();
        p1=(CDocTemplate *)GetNextDocTemplate(pos);
        ASSERT(p1->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
        p1->GetDocString(name,CDocTemplate::docName);
        if(_mkdir(d.m_model)==0)
        {
            p1->OpenDocumentFile(NULL);
        }
        else
        {
            p1->OpenDocumentFile("test.cls");
        }
    }

    // TODO: Add your command handler code here
}

```

```
void CGreenApp::OnIsd()
{
    CString name;
    CDocTemplate *p1;
    POSITION pos=GetFirstDocTemplatePosition();
    GetNextDocTemplate(pos);
    p1=(CDocTemplate *)GetNextDocTemplate(pos);
    ASSERT(p1->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
    p1->GetDocString(name,CDocTemplate::docName);

    p1->OpenDocumentFile(NULL);
}

CRect GenerateRect(CPoint &p)
{
    CRect temp(p.x-35,p.y-15,p.x+35,p.y+15);
    return temp;
}

void DrawArrow(CPoint &st,CPoint &en,CDC *pdc)
{
    if(st.x==en.x)
    {
        if(st.y>en.y)
        {
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x-5,en.y+5);
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x+5,en.y+5);
        }
        else
        {
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x-5,en.y-5);
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x+5,en.y-5);
        }
        return;
    }
    if(st.y==en.y)
    {
        if(st.x>en.x)
        {
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x+5,en.y-5);
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x+5,en.y+5);
        }
        else
        {
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x-5,en.y-5);
            pdc->MoveTo(en.x,en.y);
            pdc->LineTo(en.x-5,en.y+5);
        }
    }
}
```



```

        return;
    }
}

int checkside(CPoint &p1,CPoint &p2,CPoint &p3)
{
    if(p1.x>p2.x && p1.x>p3.x)
        return 1;
    else
        return 0;
}

void CGreenApp::OnEt()
{
    CString name;
    CDocTemplate *p1;
    POSITION pos=GetFirstDocTemplatePosition();
    GetNextDocTemplate(pos);
    GetNextDocTemplate(pos);
    p1=(CDocTemplate *)GetNextDocTemplate(pos);
    ASSERT(p1->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
    p1->GetDocString(name,CDocTemplate::docName);

    p1->OpenDocumentFile(NULL);
    // TODO: Add your command handler code here

}

void CGreenApp::OnFileNew()
{
}

void CGreenApp::OnFileOpenIsd()
{
    char *str="AllFiles (*.*)|*..*|CFiles (*.isd*)|*.isd*|";
    CFileDialog dl(TRUE,"isd",0,0,str);
    if(dl.DoModal()==IDOK)
    {
        CString name;
        CDocTemplate *p1;
        POSITION pos=GetFirstDocTemplatePosition();
        GetNextDocTemplate(pos);
        p1=(CDocTemplate *)GetNextDocTemplate(pos);
        ASSERT(p1->IsKindOf(RUNTIME_CLASS(CDocTemplate)));
        p1->GetDocString(name,CDocTemplate::docName);
        p1->OpenDocumentFile(dl.GetPathName());
    }
}

CString get_dir()
{
    return dire;
}

```

**// greenDoc.h : interface of the CGreenDoc class**

```

class CGreenDoc : public CDocument
{
protected: // create from serialization only
    CGreenDoc();
    DECLARE_DYNCREATE(CGreenDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGreenDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CGreenDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CGreenDoc)
    // NOTE - the ClassWizard will add and remove member functions here.
    //      DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
///

```

**// greenDoc.cpp : implementation of the CGreenDoc class**

```

#include "stdafx.h"
#include "green.h"
#include "greenDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CGreenDoc

IMPLEMENT_DYNCREATE(CGreenDoc, CDocument)

BEGIN_MESSAGE_MAP(CGreenDoc, CDocument)

   //{{AFX_MSG_MAP(CGreenDoc)
    // NOTE - the ClassWizard will add and remove mapping macros here.
    //      DO NOT EDIT what you see in these blocks of generated code!
   //}}AFX_MSG_MAP

END_MESSAGE_MAP()

////////////////////////////////////
// CGreenDoc construction/destruction

CGreenDoc::CGreenDoc()
{
    // TODO: add one-time construction code here
}

CGreenDoc::~CGreenDoc()
{
}

BOOL CGreenDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}

```



**// greenView.h : interface of the CGreenView class**

```

class CGreenView : public CView
{
protected: // create from serialization only
    CGreenView();
    DECLARE_DYNCREATE(CGreenView)

private:

    int classcount;
    int linkcount;
    classdiagramar *head; //classdiagramar[20];
    relation *rel;
    int link,drawnew;
    int class1,class2;
    int delrelation;
    int delclass;
    int deltsel;
    CFile fpcls,fplnk,fpatb;

CPoint m_PointOrigin;
CPoint m_PointOld;
int m_Dragging,m_Dragging1,dx,dy,mmclass;
HCURSOR m_HCross;
public:

    void logout(void); //Exit option
    void newc(void);
    void newclass(CPoint pt); //For Creating a new object
    void OnPaint();
    void myroundrect(int x0,int y0,int width,int height,
                    int count,int clr);
    void clearDiagram(classdiagramar *prev);
    void redraw(int count);
    int findclass(CPoint pt);
    int findrelation(CPoint pt);

    void OnRButtonDown(UINT flag,CPoint pt);
    void redrawrelation(void);
    void find_clear_relation(int classcount,int whattodo);
    void help(void);
    void association(void);
    int iposition(int class1,int class2);
    int position(classdiagramar *p,classdiagramar *q);
    void seldelclass(void);
    void seldelrel(void);
    void deleteclass(int count);
    void deleterelationstatus2(void);
    void deleterelation(int lcount);
    void OnLButtonDown(UINT flag,CPoint pt);
    void OnMouseMove(UINT nFlags, CPoint point);
    void OnLButtonUp(UINT nFlags, CPoint point);
    void OnLButtonDblClk(UINT flag,CPoint pt);
    void newrelation(int class1,int class2);

```

```

    int checkrelation(int class1,int class2);
    int checkpath(int c1,int c2);
    void drawrelation(relation *r,int clr);
    void coordinates(int count,int pos,int *x,int *y);
    int findlenll(attrib *mov);
    void OnVScroll(UINT code,UINT pos,CScrollBar *scroll);
    void writedata();
    void clearall();
    void readdata();
    void OnDestroy();

// Attributes
public:
    CGreenDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGreenView)
    public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CGreenView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CGreenView)
    afx_msg void OnNewmodel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in greenView.cpp
inline CGreenDoc* CGreenView::GetDocument()
    { return (CGreenDoc*)m_pDocument; }
#endif

```

```
// greenView.cpp : implementation of the CGreenView class
```

```
#include "stdafx.h"
#include "green.h"
#include "cmoddlg.h"

#include "greenDoc.h"
#include "greenView.h"

#include "reldialog.h"
#include "classdialog.h"
#include "mydialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CGreenView

IMPLEMENT_DYNCREATE(CGreenView, CView)

BEGIN_MESSAGE_MAP(CGreenView, CView)

    ON_WM_VSCROLL()
    ON_COMMAND(103, readdata)
    ON_COMMAND(104, writedata)

    ON_WM_PAINT()
    ON_WM_RBUTTONDOWN()
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONUP()
    ON_WM_LBUTTONDBLCLK()

    ON_COMMAND(402, association)
    ON_COMMAND(501, help)
    ON_COMMAND(102, newc)
    ON_COMMAND(106, logout)
    ON_COMMAND(701, seldelrel)
    ON_COMMAND(601, seldelclass)
    ON_COMMAND(ID_FILE_NEW, OnNewmodel)

// Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

```

////////////////////////////////////
// CGreenView construction/destruction

CGreenView::CGreenView()
{
    head=NULL;
    rel=NULL;
    m_Dragging=0;
    m_Dragging1=0;
    mmclass=0; // for mouse move funtion
    delrelation=0;
    delclass=0;
    delcsel=0;
    linkcount=0;drawnew=0;
    link=class1=class2=0;
    classcount=0;
}

CGreenView::~CGreenView()
{
}

BOOL CGreenView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    return CView::PreCreateWindow(cs);
}

////////////////////////////////////
// CGreenView drawing

void CGreenView::OnDraw(CDC* pDC)
{
    CGreenDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: add draw code for native data here
}

////////////////////////////////////
// CGreenView printing

BOOL CGreenView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CGreenView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

```



```

void CGreenView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CGreenView diagnostics

#ifdef _DEBUG
void CGreenView::AssertValid() const
{
    CView::AssertValid();
}

void CGreenView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CGreenDoc* CGreenView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CGreenDoc)));
    return (CGreenDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CGreenView message handlers

void CGreenView::OnNewmodel()
{
    cmoddlg d;
    d.DoModal();
    // TODO: Add your command handler code here
}

void CGreenView::logout(void) //Exit option
{
    exit(0);
}

void CGreenView::newc(void)
{
    drawnew=1; //for drawing a new class
}

void CGreenView::newclass(CPoint pt) //For Creating a new object
{
    CClientDC d(this);
    classdiagramar *temp;
    temp=new classdiagramar;

    CString a;
    char b[20];

```

```

    classcount++;
    sprintf(b,"%d",classcount);
    a="Class";
    a+=b; //Assigning default class name.

    temp->name=a;
    temp->cno=classcount;

    if((temp->x0=pt.x-37)<0) temp->x0=0;

    if((temp->y0=pt.y-50)<0) temp->y0=0;

    temp->width=75;
    temp->height=100;

    temp->alink=NULL;
    temp->mlink=NULL;

    temp->next=head;

    head=temp;

    myroundrect (temp->x0,temp->y0,temp->width,temp->height,classcount,0);
}

void CGreenView::OnPaint()
{
    CPaintDC d(this);
    int clr=0;

    CString a;
    classdiagramar *temp=head;

    CBrush mybrush;
    CPen mypen;
    if(clr==0) // black colour
    mypen.CreatePen(PS_SOLID,1,RGB(0,0,0));
    if(clr==1) // white colour
        mypen.CreatePen(PS_SOLID,1,RGB(255,255,255));

    d.SelectObject(&mypen);
    mybrush.CreateSolidBrush(RGB(255,255,255));

    int xleft,xright,yleft,yright;
    int ymid;

    while(temp!=NULL)
    {
        xleft=temp->x0+1;
        yleft=temp->y0+1;
        xright=temp->x0+temp->width-1;
        yright=temp->y0+temp->height-1;

        ymid=int((yright+yleft+18)*0.5);
    }
}

```

```

d.Rectangle(xleft,yleft,xright,yright);

d.MoveTo(xleft,yleft+18);
d.LineTo(xright,yleft+18);

d.MoveTo(xleft,ymid);
d.LineTo(xright,ymid);

    unsigned int stringlength=(temp->width/8)-1;

// Truncating the strings to display on the icon
if(strlen(temp->name)>stringlength) stringlength=strlen(temp->name);
d.TextOut(xleft+8,yleft+2,temp->name,stringlength);
    temp=temp->next;

    }

}

void CGreenView::myroundrect(int x0,int y0,int width,
                             int height,int count,int clr)
{

    CString a;
    classdiagramar *temp=head;
    CClientDC d(this);

    CBrush mybrush;
    CPen mypen;
    if(clr==0) // black colour
    mypen.CreatePen(PS_SOLID,1,RGB(0,0,0));
    if(clr==1) // white colour
    mypen.CreatePen(PS_SOLID,1,RGB(255,255,255));

    if(clr==2)// blue colour
    mypen.CreatePen(PS_SOLID,1,RGB(0,0,255));

    d.SelectObject(&mypen);

    mybrush.CreateSolidBrush(RGB(255,255,255));

    int xleft,xright,yleft,yright;
    int ymid;

    xleft=x0+1;
    yleft=y0+1;
    xright=x0+width-1;
    yright=y0+height-1;
    ymid=int((yright+yleft+18)*0.5);

    d.Rectangle(xleft,yleft,xright,yright);

    d.MoveTo(xleft,yleft+18);
    d.LineTo(xright,yleft+18);

    d.MoveTo(xleft,ymid);
    d.LineTo(xright,ymid);
}

```

```

        if(clr==0 || clr==2)
        {
            while(temp!=NULL)
            {
                if(temp->cno==count) break;
                temp=temp->next;
            };

            temp->scr1=new CScrollBar;
            temp->scr1->Create(SBS_VERT|WS_CHILD|WS_VISIBLE,
                CRect(xright-15,yleft+19,xright-1,ymid),
                this,temp->cno);
            temp->scr1->SetScrollRange(0,3);
            temp->scr1->SetScrollPos(0);

            temp->scr2=new CScrollBar;

            temp->scr2->Create(SBS_VERT|WS_CHILD|WS_VISIBLE,
                CRect(xright-15,ymid+1,xright-1,yright-1),
                this,temp->cno);
            temp->scr2->SetScrollRange(0,3);
            temp->scr2->SetScrollPos(0);

            unsigned int stringlength=(width/8)-1;

            if(strlen(temp->name)<stringlength) stringlength=strlen(temp->name);
            d.TextOut(xleft+8,yleft+2,temp->name,stringlength);

            OnVScroll(SB_LINEUP,1,temp->scr1);

            OnVScroll(SB_LINEUP,1,temp->scr2);

        }
    }

//To Clear the diagram

void CGreenView::cleardiagram(classdiagramar *prev)
{
    myroundrect(prev->x0,prev->y0,prev->width,prev->height,prev->cno,1);
}

//For Changing the position of the Class
void CGreenView::redraw(int count)
{
    struct classdiagramar *temp=head;

```

```

while(temp!=NULL)
{
    if(temp->cno==count) break;
    temp=temp->next;
};

delete temp->scr1;
delete temp->scr2;

myroundrect(temp->x0,temp->y0,temp->width,
            temp->height,count,0);
}

//Function returns the object number from the coordinates of pt

int CGreenView::findclass(CPoint pt)
{
    classdiagramar *move;// head;
    int xleft,yleft,xright,yright;

    move=head;

    while(move!=NULL)
    {
        xleft=move->x0;
        yleft=move->y0;
        xright=xleft+move->width;
        yright=yleft+move->height;

        if(pt.x > xleft && pt.x < xright
           && pt.y >yleft && pt.y<yright)
        {
            return move->cno;
        }

        move=move->next;
    }
    return 0;
}

int CGreenView::findrelation(CPoint pt)
{
    int ss=4;
    for(relation *move=rel;move!=NULL;move=move->next)
    {
        int x1,y1,x2,y2;

        coordinates(move->class1,move->c1pos,&x1,&y1);
        coordinates(move->class2,move->c2pos,&x2,&y2);
    }
}

```

```

while(temp!=NULL)
{
    if(temp->cno==count) break;
    temp=temp->next;
}

mydialog d(IDX_DIALOG1,count,head,
    temp->name,
    temp->x0,
    temp->y0,
    temp->width,
    temp->height);

find_clear_relation(temp->cno,0); //0(whattodo)to redraw the rel

//To store prev values

prev->cno=temp->cno;
prev->x0=temp->x0;
prev->y0=temp->y0;
prev->width=temp->width;
prev->height=temp->height;
prev->name=temp->name;

d.DoModal(); // to display dialog box

if((temp->name=d.return_classname())!=prev->name)
    change=1;

if((temp->x0=d.return_x0())!=prev->x0)
    change=1;
if((temp->y0=d.return_y0())!=prev->y0)
    change=1;

if((temp->width=d.return_width())!=prev->width)
    change=1;

if((temp->height=d.return_height())!=prev->height)
    change=1;

if(change)
{
    // to clear prev rectangle

    clearDiagram(prev);

    // to redraw the rectangle (class)

    redraw(count);
}

```

```

else
{
    OnVScroll(SB_LINEUP,1,temp->scr1);
    OnVScroll(SB_LINEUP,1,temp->scr2);
}
// redrawing the relations
redrawrelation();

delete(prev);
}

// redraws the relations whose status is zero
// relations which are connected to the class moved
void CGreenView::redrawrelation(void)
{
    relation *move=rel;
    while(move!=NULL)
    {
        if(move->status==0)
        {
            move->c1pos=iposition(move->class1,move->class2);
            move->c2pos=iposition(move->class2,move->class1);
            drawrelation(move,1);
            move->status=1;
        }
        move=move->next;
    }
}

// this function just clears the relations that are linked to count class
void CGreenView::find_clear_relation(int classcount,int whattodo)
{
    relation *move=rel;
    while(move!=NULL)
    {
        if(move->class1==classcount||move->class2==classcount)
        {
            drawrelation(move,whattodo);
            move->status=whattodo;
        }
        move=move->next;
    }
}

```

```

void CGreenView::help(void)
{
    MessageBox("copy rights reserved ver 4.0", "OOD");
}

void CGreenView::association(void)
{
    link=1;
}

int CGreenView::iposition(int class1,int class2)
{
    classdiagramar *temp1,*temp2;

    temp1=head;
    temp2=head;

    while(temp1!=NULL)
    {
        if(temp1->cno==class1) break;
        temp1=temp1->next;
    };

    while(temp2!=NULL)
    {
        if(temp2->cno==class2) break;
        temp2=temp2->next;
    };

    return(position(temp1,temp2));
}

int CGreenView::position(classdiagramar *p,classdiagramar *q)
{
    int px0=p->x0,py0=p->y0,qx0=q->x0,qy0=q->y0;
    int px1=p->x0+p->width;
    int py1=p->y0+p->height;
    int qx1=q->x0+q->width;
    int qy1=q->y0+q->height;

    if(py0>qy1)
    {
        if(px0>qx1) return 1;
        if(px1<qx0) return 13;
        if(px1>qx1)
            if(px0<qx0) return 15;
            else return 16;
        else
            if(px0<qx0) return 14;
        return 15;
    }
}

```



```

if (py1 < qy0)
{
    if (px0 > qx1) return 5;
    if (px1 < qx0) return 9;
    if (px1 > qx1)
        if (px0 < qx0) return 7;
        else return 6;
    else
        if (px1 < qx1) return 8;
    return 7;
}

if (px0 > qx1)
{
    if (qy1 < py1)
        if (py0 < qy0) return 3;
        else return 2;
    else
        if (py0 < qy0) return 3;
    return 3;
}

if (px1 < qx0)
{
    if (qy1 < py1)
        if (py0 > qy0) return 12;
        else return 11;
    else
        if (py0 < qy0) return 10;
    return 11;
}

return 0;
}

//Function which calls a dialog box to verify class deletion
void CGreenView::selclass(void)
{
    if (delclass == 0) return;

    classdialog d2 (IDX_DIALOG2);

    if (d2.DoModal() == IDOK)
        deleteclass (delclass);
    else
    {
        redraw (delclass);
        find_clear_relation (delclass, 1);
        delclass = 0;
    }
}

```

```

void CGreenView::seldelrel(void)
{
    delrelation=1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// DELETE CLASS

void CGreenView::deleteclass(int count)
{
    classdiagramar *cmove=head;

    delcsel=0;

    while(cmove!=NULL)
    {
        if(cmove->cno==count)
        {
            cmove->scr1->ShowScrollBar(FALSE);
            delete cmove->scr1;

            cmove->scr2->ShowScrollBar(FALSE);
            delete cmove->scr2;

            cleardiagram(cmove);

            delclass=0;

            deleterelationstatus2();

            cmove->cno=0;

            return;
        }

        cmove=cmove->next;
    }

}

void CGreenView::deleterelationstatus2(void)
{
    if(rel==NULL) return;

    relation *rmove=rel->next;
    relation *prev=rel;

    while(rmove!=NULL)
    {
        if(rmove->status==2)
        {
            drawrelation(rmove,0); // clears relation
        }
    }
}

```

```

        prev->next=remove->next;
        delete remove;
        if(prev->next==NULL) break;
        remove=prev->next;
    }
    else
    {
        remove=remove->next;
        prev=prev->next;
    }
}
relation *start=rel;
if(start->status==2)
{
    drawrelation(start,0);
    rel=rel->next;
    delete start;
}
}

// DELETES THE RELATION from the linked list as well as
// clears the relation(erases from the screen)
// by accepting the link count

void CGreenView::deleterelation(int lcount)
{
    relation *move=rel;

    if(move->count==lcount)
    {
        drawrelation(move,0);
        rel=rel->next;
        delete move;
        return;
    }

    relation *prev=move;
    move=move->next;

    while(move!=NULL)
    {
        if(move->count==lcount)
        {
            drawrelation(move,0);
            prev->next=move->next;
            delete move;
            return;
        }
        move=move->next;
        prev=prev->next;
    }
}

```

```

////////////////////////////////////
// LEFT BUTTON DOWN

```

```

void CGreenView::OnLButtonDown(UINT flag,CPoint pt)
{
    //int x0,y0,width,height,change=0;

    if(drawnew==1)
    {
        newclass(pt);
        drawnew=0;
        return;
    }

    if(delrelation==1)
    {
        int lcount=findrelation(pt);

        if(lcount==0)
        {
            MessageBox("No Relation selected","Select Again");
            delrelation=0;
            return;
        }

        reldialog d3(IDX_DIALOG3);

        if(d3.DoModal()==IDOK)
            deleterelation(lcount);

        delrelation=0;
        return;
    }

    int count;

    count=findclass(pt);

    if(count!=0 && link==0)//Initialization for drag and drop option
    {
        mmclass=count;

        classdiagramar *mmc=head;
        while(mmc->cno!=mmclass)
            mmc=mmc->next;

        dx=pt.x-mmclass->x0;
        dy=pt.y-mmclass->y0;
    }
}

```

## 7. ANNEXES

```
m_PointOrigin=pt;
m_PointOld=pt;
SetCapture();

m_Dragging=1;

RECT Rect;
GetClientRect(&Rect);
ClientToScreen(&Rect);
::ClipCursor(&Rect);
}

if(link==0) return; //No link selected

if(count==0) //No class selected to draw link
{
    MessageBox("Select again","Error");
    class1=0;
    link=0;
    return;
}

if(class1==0) //First class selected for the link
{
    class1=count;
    return;
}

if(class1==count)
{
    MessageBox("Same Class Selected","Error");
    class1=0;
    link=0;
    return;
}

else // Link created
{
    class2=count;

    if(checkrelation(class1,class2))
        newrelation(class1,class2);

    link=0; // Giving them initial values
    class1=0; // Indicating no selection
    class2=0;
}
}
```

```

void CGreenView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    ::SetCursor(m_HCross);
    if(m_Dragging)
    {
        classdiagramar *mmc=head;
        while(mmc->cno!=mmclass)
            mmc=mmc->next;

        CClientDC ClientDC(this);
        ClientDC.SetROP2(R2_NOT);

        ClientDC.MoveTo(m_PointOld.x-dx+1,m_PointOld.y-dy+1);
        ClientDC.LineTo(m_PointOld.x-dx+1,m_PointOld.y-dy+mmc->height-2);
        ClientDC.MoveTo(m_PointOld.x-dx+1,m_PointOld.y-dy+mmc->height-2);
        ClientDC.LineTo(m_PointOld.x-dx+mmc->width-2,
            m_PointOld.y-dy+mmc->height-2);
        ClientDC.MoveTo(m_PointOld.x-dx+mmc->width-2,
            m_PointOld.y-dy+mmc->height-2);
        ClientDC.LineTo(m_PointOld.x-dx+mmc->width-2,m_PointOld.y-dy+1);
        ClientDC.MoveTo(m_PointOld.x-dx+mmc->width-2,m_PointOld.y-dy+1);
        ClientDC.LineTo(m_PointOld.x-dx+1,m_PointOld.y-dy+1);

        ClientDC.MoveTo(point.x-dx+1,point.y-dy+1);
        ClientDC.LineTo(point.x-dx+1,point.y-dy+mmc->height-2);
        ClientDC.MoveTo(point.x-dx+1,point.y-dy+mmc->height-2);
        ClientDC.LineTo(point.x-dx+mmc->width-2,
            point.y-dy+mmc->height-2);
        ClientDC.MoveTo(point.x-dx+mmc->width-2,
            point.y-dy+mmc->height-2);
        ClientDC.LineTo(point.x-dx+mmc->width-2,point.y-dy+1);
        ClientDC.MoveTo(point.x-dx+mmc->width-2,point.y-dy+1);
        ClientDC.LineTo(point.x-dx+1,point.y-dy+1);

        m_PointOld=point;
    }

    //CView::OnMouseMove(nFlags, point);
}

////////////////////////////////////

void CGreenView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(m_Dragging)
    {
        classdiagramar *mmc=head;
        while(mmc->cno!=mmclass)
            mmc=mmc->next;
    }
}

```

```

m_Dragging=0;
::ReleaseCapture();
::ClipCursor(NULL);
CClientDC ClientDC(this);
ClientDC.SetROP2(R2_NOT);

ClientDC.MoveTo(m_PointOld.x-dx+1,m_PointOld.y-dy+1);
ClientDC.LineTo(m_PointOld.x-dx+1,
m_PointOld.y-dy+mmc->height-2);
ClientDC.MoveTo(m_PointOld.x-dx+1,
m_PointOld.y-dy+mmc->height-2);
ClientDC.LineTo(m_PointOld.x-dx+mmc->width-2,
m_PointOld.y-dy+mmc->height-2);
ClientDC.MoveTo(m_PointOld.x-dx+mmc->width-2,
m_PointOld.y-dy+mmc->height-2);
ClientDC.LineTo(m_PointOld.x-dx+mmc->width-2,
m_PointOld.y-dy+1);
ClientDC.MoveTo(m_PointOld.x-dx+mmc->width-2,
m_PointOld.y-dy+1);
ClientDC.LineTo(m_PointOld.x-dx+1,m_PointOld.y-dy+1);

find_clear_relation(mmc->cno,0);//0(whattodo)to redraw the rel

cleardiagram(mmc);

// to redraw the rectangle (class)

mmc->x0=point.x-dx;
mmc->y0=point.y-dy;
if(mmc->x0<0) mmc->x0=0;
if(mmc->y0<0) mmc->y0=0;

int overlapstatus=0;

classdiagramar *overlap;

for(overlap=head;overlap!=NULL;overlap=overlap->next)
    if(overlap->cno!=mmc->cno && overlap->cno!=0)
    {
        if(mmc->x0>overlap->x0+overlap->width) continue;
        if(mmc->x0+mmc->width<overlap->x0) continue;

        if(mmc->y0>overlap->y0+overlap->height) continue;
        if(mmc->y0+mmc->height<overlap->y0) continue;

        overlapstatus=1;
        break;
    }

if(overlapstatus==1)
{
    mmc->x0=m_PointOrigin.x-dx;
    mmc->y0=m_PointOrigin.y-dy;
    overlapstatus=0;
}

```

```

        redraw(mmc->cno);
        redrawrelation();

//    ClientDC.SetROP2(R2_COPYPEN);

    }

//CView::OnLButtonUp(nFlags, point);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// LEFT BUTTON DOUBLE CLICK

void CGreenView::OnLButtonDblClk(UINT flag,CPoint pt)
{

    int count;
    count=findclass(pt);
    if(count==0)
        return;

    if(delcsel!=0)
    {
        redraw(delcsel);
        find_clear_relation(delcsel,1);
        delcsel=0;
    }

    classdiagramar *temp;
    temp=head;
    while(temp!=NULL)
    {
        if(temp->cno==count) break;
        temp=temp->next;
    };

    delclass=count;
    //highlighting the class

    delete temp->scr1;
    delete temp->scr2;

    delcsel=count;

    myroundrect(temp->x0,temp->y0,temp->width,temp->height,count,2);
    find_clear_relation(count,2);

}

```



```

//Creates relation by accepting two class nos
void CGreenView::newrelation(int class1,int class2)
{
    classdiagramar *temp1,*temp2;

    temp1=head;
    temp2=head;

    while(temp1!=NULL)
    {
        if(temp1->cno==class1) break;
        temp1=temp1->next;
    };

    while(temp2!=NULL)
    {
        if(temp2->cno==class2) break;
        temp2=temp2->next;
    };

    relation *temprel;
    temprel= new relation;

    temprel->class1=class1;
    temprel->class2=class2;

    temprel->c1pos = position(temp1,temp2);
    temprel->c2pos = position(temp2,temp1);

    temprel->count=++linkcount;
    temprel->status=1;

    temprel->next=rel;

    rel=temprel;

    drawrelation(temprel,1);
}

// returns 1 if there is an error in the link
int CGreenView::checkrelation(int class1,int class2)
{
    relation *move=rel;
    while(move!=NULL)
    {
        if(move->class1==class1 &&
           move->class2==class2)
        {
            MessageBox("Relation already
            present","Error");
            return 0;
        }
    }
}

```

```

        if(checkpath(class2,class1)==0)
        {
            MessageBox("Illegal inheritance","Error");
            return 0;
        }

        move=move->next;

    }
    return 1;
}

int CGreenView::checkpath(int c1,int c2)
{
    if(c1==c2) return 0;// return 0 if there is a loop
    int a=1;
    relation *move=rel;
    while(move!=NULL)
    {
        if(move->class1==c1)
            a*=checkpath(move->class2,c2);
        move=move->next;
    }
    return a;
}

//Draws the relation by taking the pointer to rel struct and clr

void CGreenView::drawrelation(relation *r,int clr)
{
    CClientDC d(this);
    int x1,y1,x2,y2;
    int dd=10,ss=5;
    int xr,yr;

    CBrush mybrush;
    CPen mypen;

    if(clr==1) // black colour
        mypen.CreatePen(PS_SOLID,1,RGB(0,0,0));

    if(clr==0) // white colour
        mypen.CreatePen(PS_SOLID,1,RGB(255,255,255));

    if(clr==2) // blue colour
        mypen.CreatePen(PS_SOLID,1,RGB(0,0,255));

    d.SelectObject(&mypen);

    mybrush.CreateSolidBrush(RGB(255,255,255));

    coordinates(r->class1,r->c1pos,&x1,&y1);
    coordinates(r->class2,r->c2pos,&x2,&y2);
}

```

```

double len=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));

xr=x2-(int)(((dd*(x2-x1)+ss*(y2-y1))/len)+0.5);
yr=y2-(int)(((dd*(y2-y1)-ss*(x2-x1))/len)+0.5);

d.MoveTo(x2,y2); // Drawing line
d.LineTo(xr,yr);

xr=x2-(int)(((dd*(x2-x1)-ss*(y2-y1))/len)+0.5);
yr=y2-(int)(((dd*(y2-y1)+ss*(x2-x1))/len)+0.5);

d.MoveTo(x2,y2); // Drawing arrow mark
d.LineTo(xr,yr);

d.MoveTo(x1,y1);
d.LineTo(x2,y2);

}

void CGreenView::coordinates(int count,int pos,int *x,int *y)
{
    classdiagramar *temp=head;
    while(temp!=NULL)
    {
        if(temp->cno==count) break;
        temp=temp->next;
    };

    *x=temp->x0;
    *y=temp->y0;
    int ht=temp->height;
    int wd=temp->width;

    int yinc=ht/4;
    int xinc=wd/4;

    switch(pos)
    {
        case 1: *x--;*y--;break;
        case 2: *x--;*y+=yinc; break;
        case 3: *x--;*y+=2*yinc; break;
        case 4: *x--;*y+=3*yinc;break;
        case 5: *x--;*y+=ht+1;break;
        case 6: (*y)+=ht+1;(*x)+=xinc; break;
        case 7: (*y)+=ht+1;(*x)+=2*xinc; break;
        case 8: (*y)+=ht+1;(*x)+=3*xinc; break;
        case 9: (*y)+=ht+1;(*x)+=wd+1; break;
        case 10: (*y)+=3*yinc;(*x)+=wd+1; break;
        case 11: (*y)+=2*yinc;(*x)+=wd+1; break;
        case 12: (*y)+=yinc;(*x)+=wd+1; break;
        case 13: *y--;(*x)+=wd+1; break;
        case 14: *y--;(*x)+=3*xinc;break;
        case 15: *y--;(*x)+=2*xinc;break;
        case 16: *y--;(*x)+=xinc;break;
    }
}

```

```

int CGreenView::findlenll(attrib *mov)
{
    for(int len=0;mov!=NULL;len++,mov=mov->next);
    return len;
}

void CGreenView::OnVScroll(UINT code,UINT pos,CScrollBar *scroll)
{
    classdiagramar *move=head;
    int charht=18; //ht of the characters

    static int red=0; static int green=0;
    //red and green variables give the positions of the scroll bars

    CClientDC d(this);

    CPen mypen;
    mypen.CreatePen(PS_SOLID,1,RGB(255,255,255));
    d.SelectObject(&mypen);

    while(move!=NULL)
    {

        int csize=(move->height-18)/40;
        attrib *amove=move->alink;
        attrib *mmove=move->mlink;

        int attlen=findlenll(amove);
        if(scroll==move->scrl)
        {

            if(code==SB_THUMBTRACK)
                red=(int)pos;

            if(code==SB_LINEDOWN)
                if(red+csize!=attlen) red++;

            if(code==SB_LINEUP)
                if(red!=0) red--;

            int xpos=move->x0+4,ypos=move->y0+2+charht;

            for(int imov=0;imov<red && amove!=NULL;imov++,
                amove=amove->next);

            d.Rectangle(xpos,ypos,xpos+move->width-20,
                move->y0+((move->height+18)/2));

            for(int pint=0;amove!=NULL && pint<csize;pint++,
                amove=amove->next,ypos+=charht)
            {

                unsigned int stringlength=(move->width/8)-3;

                if(strlen(amove->name)<stringlength)

```

```

        stringlength=strlen(amine->name);
        d.TextOut(xpos,ypos,amine->name,stringlength);
    }

    move->scr1->SetScrollPos(red,move->cno);
}

if(scroll==move->scr2)
{
    if(code==SB_THUMBTRACK)
        green=(int)pos;

    if(code==SB_LINEDOWN)
        if(green+csize!=attlen) green++;

    if(code==SB_LINEUP)
        if(green!=0) green--;

    int xpos=move->x0+4,
        ypos=move->y0+2+(int)((move->height+18)/2);

    for(int imov=0;imov<green && mmove!=NULL;
        imov++,mmove=mmove->next);

    CPen mypen;
    mypen.CreatePen(PS_SOLID,1,RGB(255,255,255));
    d.SelectObject(&mypen);
    d.Rectangle(xpos,ypos,xpos+move->width-20,
        move->y0+move->height-2);

    for(int pint=0;mmove!=NULL && pint<csize;pint++,
        mmove=mmove->next,ypos+=charht)
    {
        unsigned int stringlength=(move->width/8)-3;

        if(strlen(mmove->name)<stringlength)
            stringlength=strlen(mmove->name);
        d.TextOut(xpos,ypos,mmove->name,stringlength);
    }

    move->scr2->SetScrollPos(green,move->cno);
}

move=move->next;
}
}

```

```

void CGreenView::writedata()
{
    //Saving the classes

    CString fname;
    fname="c:\\green\\"+get_dir()+"\\test.cls";
    fpcls.Open(fname,CFile::modeCreate|CFile::modeReadWrite);
    fname="c:\\green\\"+get_dir()+"\\test.lnk";
    fplnk.Open(fname,CFile::modeCreate|CFile::modeReadWrite);

    classdiagramar *cmove=head;
    fpcls.SeekToBegin();
    classsave temp;

    while(cmove!=NULL)
    {
        if(cmove->cno!=0)
        {
            strcpy(temp.name,cmove->name);
            temp.cno=cmove->cno;
            temp.x0=cmove->x0;
            temp.y0=cmove->y0;
            temp.height=cmove->height;
            temp.width=cmove->width;
            fpcls.Write(&temp,sizeof(temp));
        }
        cmove=cmove->next;
    }

    // Saving the relations

    relation *rmove=rel,temps;

    fplnk.SeekToBegin();
    while(rmove!=NULL)
    {
        temps.class1=rmove->class1;
        temps.class2=rmove->class2;
        temps.clpos=rmove->clpos;
        temps.c2pos=rmove->c2pos;
        temps.count=rmove->count;
        temps.status=rmove->status;
        fplnk.Write(&temps,sizeof(temps));
        rmove->status=2;
        rmove=rmove->next;
    }

    MessageBox("Save done","writedata");

    deleterelationstatus2();
    clearall(); // clearing the contents of the screen
    fplnk.Close();
    fpcls.Close();
}

```

```

void CGreenView::clearall()
{
    classdiagramar *temp=head;

    while(temp!=NULL)
    {
        if(temp->cno!=0)
        {
            temp->scr2->ShowScrollBar(FALSE);
            delete(temp->scr2);

            temp->scr1->ShowScrollBar(FALSE);
            delete(temp->scr1);

            // to clear prev rectangle
            cleardiagram(temp);

        }
        temp=temp->next;
    }

    //Invalidate();
}

void CGreenView::readdata()
{
    classsave rtemp;
    classdiagramar *cmove;
    int maxcount=0;
    CString fname;
    fname="c:\\green\\"+get_dir()+"\\test.cls";
    fpcls.Open(fname,CFile::modeRead);
    fname="c:\\green\\"+get_dir()+"\\test.lnk";
    fplnk.Open(fname,CFile::modeRead);

    head=NULL;

    fpcls.SeekToBegin();

    if(fpcls.GetLength()==0)
        MessageBox("Class File is empty","Read Record...");

    fpcls.SeekToBegin();
    while(fpcls.Read(&rtemp,sizeof(classsave))!=0)
    {

        cmove=new classdiagramar;
        cmove->name=rtemp.name;
        cmove->cno=rtemp.cno;
        if(maxcount<rtemp.cno) maxcount=rtemp.cno;
        cmove->x0=rtemp.x0;
        cmove->y0=rtemp.y0;
    }
}

```

```

        cmove->height=rtemp.height;
        cmove->width=rtemp.width;
        cmove->alink=NULL;
        cmove->mlink=NULL;
        cmove->next=head;
        head=cmove;

myroundrect (rtemp.x0,rtemp.y0,rtemp.width,rtemp.height,rtemp.cno,0);

    }

    classcount=maxcount;//considering the count value

    maxcount=0;

    relation *rmove,newtemp;

    rel=NULL;

    if(fplnk.GetLength()==0)
        MessageBox("Link File is empty","Read Record...");

    fplnk.SeekToBegin();
    while(fplnk.Read(&newtemp,sizeof(newtemp))!=0)
    {
        rmove=new relation;

        rmove->class1=newtemp.class1;
        rmove->class2=newtemp.class2;
        rmove->clpos=newtemp.clpos;
        rmove->c2pos=newtemp.c2pos;
        rmove->count=newtemp.count;

        if(maxcount<rmove->count) maxcount=rmove->count;

        rmove->status=1;
        rmove->next=rel;
        rel=rmove;
    }

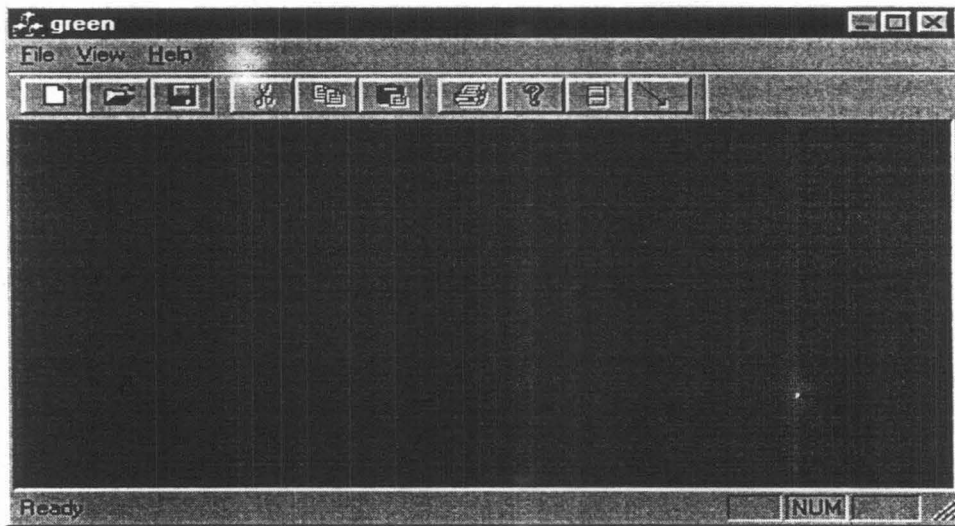
    linkcount=maxcount;

    relation *temps=rel;
    while(temps!=NULL)
    {
        drawrelation(temps,1);
        temps=temps->next;
    }
    fplnk.Close();
    fpcls.Close();
}

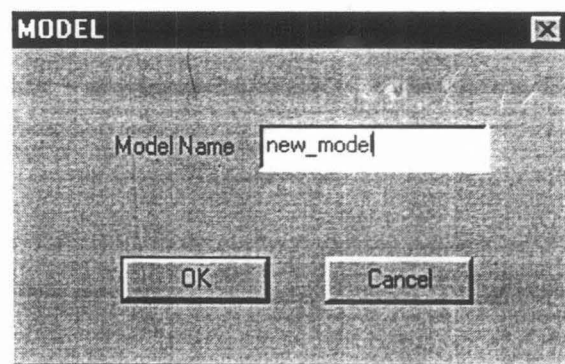
```



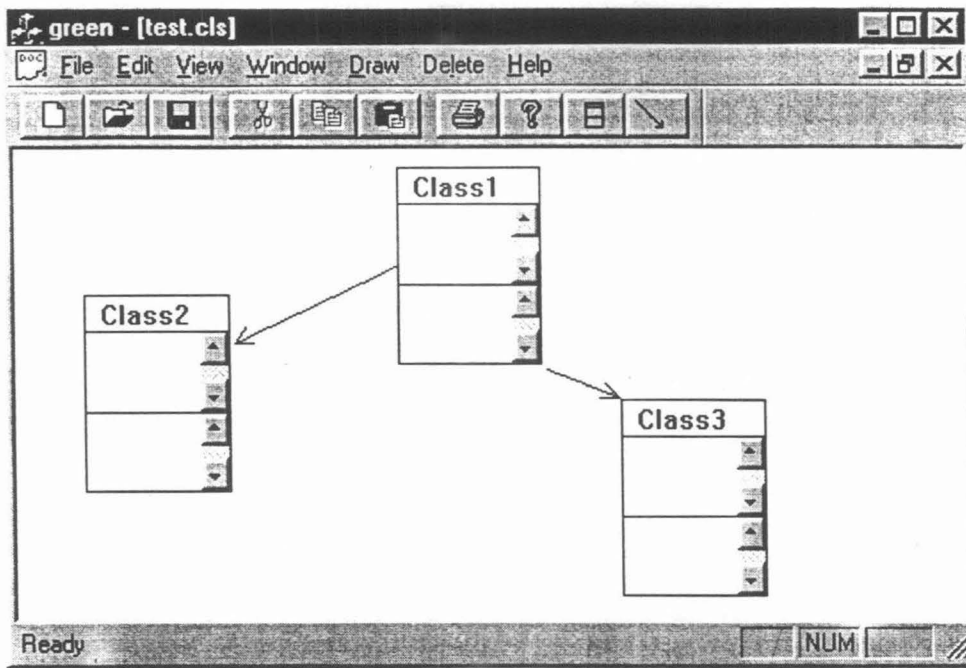
### 7.2 Sample Screens



Initial screen



Dialog Box to enter a model



Class Diagram

```

int CGreenView::position(classdiagramar *p,classdia
{
    int px0=p->x0,py0=p->y0,qx0=q->x0,qy0=q->y0;
    int px1=p->x0+p->width;
    int py1=p->y0+p->height;
    int qx1=q->x0+q->width;
    int qy1=q->y0+q->height;

    if(py0>qy1)
    {
        if(px0>qx1) return 1;
        if(px1<qx0) return 13;
        if(px1>qx1)
            if(px0<qx0) return 15;
    }
}
    
```

Microsoft Developer Studio

## **8. CONCLUSION**

---

In the project a GUI based casetool is developed which can handle modifications in the static and dynamic behaviour of the object oriented system. The casetool allows the system analyst to draw the Class diagram, Integrated State transition diagram by accepting state and event information. Besides this the casetool draws event trace diagram from the ISTD and finally generates the C++ code.

Indeed, the casetool can be used throughout the lifecycle, as the design evolves into a production implementation. It can also be used during systems maintenance.

## REFERENCES

---

- [Boo 94] Booch G., Object Oriented analysis and design with applications. Addison-Wesley 2<sup>nd</sup> edition.
- [Rum 91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenson, W.1997.Object Oriented Modeling and Design, Prentice Hall.
- [Par 95] Dr.N.Parimala, Handling Changes in Dynamic Specifications in Object Oriented Systems. 1995.
- [Bja 97] Bjarne Stroustrup.The C++ Programming Language.1997. Addison-Wesley 3rd edition.
- [Yas 98] Yashavant.P.Kanetkar. VC++ Programming.1998, 1st Edition. BPB publications.
- [Krugli] David.J.Kruglinski. Inside VC++. 4th Edition. Microsoft Press.
- [Rog 97] Roger.S.Pressman Software Engineering, A Practitioners approach.1997, 4th Edition. McGraw Hill International Edition.