

Lib. Copy

A CASE TOOL
FOR
HANDLING MODIFICATIONS
IN THE DYNAMIC BEHAVIOUR OF
AN OBJECT ORIENTED SYSTEM

Dissertation Submitted to
JAWAHARLAL NEHRU UNIVERSITY
In partial fulfillment of requirements for the award of the degree of

Master of Technology
In
Computer Science & Technology

By
NUNE NAGAMANI

SUPERVISOR
Dr. PARIMALA N.



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

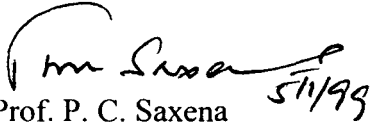
CERTIFICATE

It is certified that the Dissertation entitled **A Case Tool For Handling Modifications In The Dynamic Behaviour Of An Object Oriented System**, being submitted by Nune Nagamani, for the award of degree of Master of Technology in Computer Science & Technology, School Of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi, is a record of her own work carried out by her under my supervision and guidance.

I recommend her dissertation for the above degree.


Dr. Parimala N.

School of Computer & Systems Sciences,
Jawaharlal Nehru University,
New Delhi – 67


Prof. P. C. Saxena

Dean,
School of Computer & Systems Sciences,
Jawaharlal Nehru University,
New Delhi - 67

DECLARATION

I hereby declare that the dissertation entitled **A Case Tool For Handling Modifications In The Dynamic Behaviour Of An Object Oriented System**, is my original work and has not been submitted elsewhere towards the award of any degree, diploma or certificate.

I take complete responsibility for the authenticity of the work.

Dated : 5/1/99

Place : NEW DELHI

Nagamani
Nune Nagamani

ACKNOWLEDGEMENT

It is my immense pleasure to express thanks and obligations to all those who have been sine qua non to this dissertation.

I wish to begin this note of thanks by expressing my sincere gratitude to Dr. Parimala N, School of Computer and Systems Sciences, J.N.U., for her valuable guidance and inspiration given to me, without which the completion of this dissertation would have been impossible.

I am also deeply indebted to Dr. P. C. Saxena, Dean, School of Computer and Systems Sciences, for permitting me to undertake this project and helped me with his numerous valuable suggestions with which I acquired a wider perspective.

I am also extremely thankful to all the faculty members for their kind cooperation and needful help.

Last but not the least, I thank all my colleagues for their kind assistance.

Nagamani
Nune Nagamani

CONTENTS

Acknowledgement	iii
Abstract	vii
Chapter 1. Introduction	1-5
1.1 Object Oriented Modeling	1
1.2 Modeling Concept, Not Implementation	2
1.3 Object Oriented Methodology	2
1.4 Problem Definition	3
1.5 Case Tool	4
1.6 Organization of Dissertation	5
Chapter 2. Integrated Method	6-18
2.1 Modeling Concepts	6
2.1.1 Static Model	6
2.1.2 Dynamic Model	9
2.2 Partial Modification Problem	13
2.2.1 Augmented State Transition Diagram	14
2.2.2 Scenario State Transition Diagram	15
2.2.3 Integrated Diagram	16

Chapter 3	Consistency	19-22
	3.1 Duplication of Names	20
	3.2 Transitions and Events	20
	3.3 Handling Modifications	20
Chapter 4	Software Selection	23-25
	4.1 Windows Programming	23
	4.2 Microsoft Developer Studio	23
	4.3 Visual C++	23
Chapter 5	Project Designing	26-31
	5.1 Data Structures Used	26
	5.2 Steps to Maintain Consistency	27
	5.2.1 Duplication of Names	27
	5.2.2 State Check	29
	5.2.3 Transition Check	29
	5.2.4 Event Check	30
	5.3 Modifications	30
	5.4 Deletion	31
Chapter 6	Generating Code	33-34
	6.1 Design Description	33
	6.1.1 Code for Static Part	33
	6.1.2 Reasons for choosing VC++	33
	6.1.3 Code for Dynamic Part	34

Chapter 7	The Case Tool	35-41
7.1	How to use Case Tool	35
7.1.1	Drawing Class Diagram	36
7.1.2	Drawing Integrated Diagrams	37
7.1.3	Handling Modifications	39
7.1.4	Generating Code	41
Chapter 8	Conclusion	42
Annexes	Source Code	
References		

ABSTRACT

Object-oriented technology is nowadays mature enough to provide valuable solutions to real life problems. The case tool is a user-friendly application developed in VC++, helps system analyst in object-oriented modeling and design. It can also handle modifications in the static and dynamic behavior of the system.

Designing an Object oriented system includes modeling static and dynamic behavior of the objects in the system. Static part remains unchanged over time in contrast with dynamic part. If we consider particularly the dynamic aspect several structures exist to represent this behavior. However there is no method, which incorporates consistency over changes in the requirements of the system. The current project is an attempt at introducing concepts in behavioral model so that certain amount of consistency can be ensured. Further, when changes are made to the design the proposed project tries to build rules/checks by which every modification is consistent.

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. The current project is an attempt at introducing consistency checking through the help of integrated state transition diagrams.

The case tool models the static and dynamic behavior of the system through the help of GUI. The system analyst can input information about the class, attributes, methods, and inheritance relationship among the classes and draw class diagram. It allows him to draw the Integrated State Transition diagram (ISTD) by accepting state and event information. Besides this, the case tool checks consistency of the system at each step of development. It even allows the analyst to modify his system as requires. Finally, It draws event trace diagram from the ISTD and finally obtains the Visual C++ source code for the implementation of the system.

Chapter 1

Introduction

Object oriented modeling and design is a new a way of thinking about problems using models organized around real world concepts .The fundamental construct is the object, which combines both data structure and behavior in single entity. Object oriented models are useful for understanding problems, communicating with application experts, modeling enterprises, preparing documentation, designing programs and databases.

This report provides a methodology for object-oriented modeling and design. We also describe a graphical notation for representing this method. A case tool had been provided with detail explanation of how it helps in developing a consistent object oriented model over changes in the requirements of the system and source code of the case tool is also being presented.

1.1 Object Oriented Modeling

The fundamental idea behind Object-Oriented Modeling, is to combine into single unit the data and the operations on that data. This single unit is called an Object. Object oriented approach generally includes four major aspects, identity, classification, and inheritance.

Identity means that data is quantified into discrete, distinguishable entities called objects. Paragraph in a document, the white queen in a chessboard are some of the examples of object. Objects can be concrete such as a file in file system or conceptual such as scheduling policy in a multiprocessing system.

Classification means that objects with the same data structure and behavior are grouped into a class. Paragraph, Chess are examples of classes. A class is an abstraction that describes properties important to an application and ignores the rest. Each object is an instance of its class.

Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be broadly defined and refined successively into finer sub classes. Each subclass incorporates or inherits all of the properties of its super class and adds its own unique properties. For example, Scrolling window and Fixed window are sub classes of window.

1.2 Modeling Concept, Not Implementation

The current emphasis in the literature is on implementation rather than analysis and design. The real pay off comes from addressing front-end conceptual issues rather than back end implementation issues. The design flaws that surface during implementation are more costly to fix than those that are found earlier are. And focusing on implementation issues too early restricts design choices and often leads to an inferior product.

Object oriented development is a conceptual process independent of programming languages until the final stages. Object oriented development is fundamentally a new way of thinking, is not a programming technique. It serves as a medium for specification, analysis, documentation and identifying as well as for programming.

1.3 Object Oriented Methodology

This approach has the following stages:

1. Analysis

Analysis, the first step of the OMT methodology, is concerned with devising a precise, concise, understandable, and correct model of the real world. The purpose of the object-oriented analysis is to model the real world system so that it can be understood. To do this, one must examine the requirements, analyze their implications, and restate them rigorously. The successful analysis model states what must be done, without restricting how it is done, and avoids implementation decisions. The result of analysis should understand problem as preparation for design.

2. System Design

During analysis, the focus is on what to be done, independent of how it is done. During design, decisions are made about how the problem will be solved, first at a high level then increasingly detailed levels. System design is the first stage in which the basic approach to solving the problem is selected. During system design, the overall structure and style of the model will be decided.

3. Object Design

The object design phase determines the full definitions of the classes and associations used in the implementation, as well as the interfaces and algorithms of the methods used to implement operations. Thus the object design phase adds internal objects for implementation and optimizes data structures and algorithms.

4. Implementation

In this phase the classes and the relationships among them developed during object design are finally translated into a programming language, database, or hardware implementation. During implementation, it is important to follow good software engineering practice so that tractability to the design is straightforward and so that the implemented system remains flexible and extensible.

1.4 Problem Definition

For designing an object-oriented system, the analyst must analyze both static and dynamic behavior of the system. The static part of the system includes the identification of the classes, their structural properties and hierarchy relationship among classes. The dynamic analysis deals with the behavioral aspects of the objects, i.e. this analysis involves state transitions of the objects and events that are sent and received by objects. These major aspects are dealt with state transition diagrams and event trace diagrams. These different aspects are modeled using different models. But there is no method.

which can handle changes in the requirements of this dynamic behavior. 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.

To have a flexible system over changes, we adopted a method of dealing with integrated behavior of objects, covering state transitions that the object undergoes and the events that are sent and received by an object and called this method as integrated method. An integrated diagram is drawn to express this integrated behavior. Then the changes would be localized and also eliminates the partial modification problem.

1.5 Case tool

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. Tools help the designer complete his job quickly and easily. The case tool developed here, helps the designer in drawing class diagrams, Integrated diagrams. It can handle both the static and dynamic behavior of the system. It checks the consistency of the system at each step of the development of the system and also whenever the system undergoes any changes. It can also generate event trace diagram from the integrated diagram and finally, it generates source code for the system designed from the integrated diagram and helps the analyst in implementing his system.

This case tool is a combined effort. My main part of the design in the case tool includes implementing necessary steps to maintain consistency in the system and allowing analyst to modify his system at any time and then implementing necessary modifications into the system to retain consistency in the system. My next part is to supply source code in Visual C++ to the analyst for his system implementation.

1.6 Organization of the Report

Chapter two addresses object oriented concepts, integrated method and presents a graphical notation for expressing them.

Chapter three explains about steps to maintain consistency in an object-oriented system.

Chapter four explains System's life cycle.

Chapter five explains the selection of software for the case tool.

Chapter six addresses project designing.

Chapter seven explains about how to generate source code for the system and design issue.

Chapter eight explains about how to use the case tool.

Chapter 2

Integrated Method

A model is an abstraction of something for the purpose of understanding it before building it. Because it omits non-essential details, and it is easier to manipulate than the original entity. This chapter describes the concepts and notations involved in developing a flexible object oriented system.

2.1 Modeling Concepts

Generally, a model is designed from two different points of view, each capturing important aspects of the system, but both required for a complete description. They are static model and dynamic model. The static model presents the static, structural and data aspects of a system and the dynamic model represents the temporal, behavioral, and control aspects of a system. For the basic definitions and principles, we referred Rambaug method.

2.1.1 Static Model

The static model describes the structure of classes in the system - inheritance relationships among them, their attributes and their methods. The static model provides the essential framework into which the dynamic model can be placed. It captures those concepts from the real world that are important to an application. The major aspect of static model is classes.

- ***Objects***

We define an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. Object serves two purposes: promotes understanding of the real world and a practical basis for computer implementation. All objects have identity and are distinguished by their inherent existence and not by descriptive properties that may have.

- ***Classes***

A class describes a group of objects with similar properties, and common behavior. Person, Window are examples of classes. An object is an instance of a class. Objects in a class will have same attributes and behavior patterns.

- ***Inheritance***

Inheritance is a powerful abstraction for sharing similarities among classes, while preserving their differences. Inheritance is a relationship between a class and refined version of it. The class being refined is called super class and the refined class is called subclass. Inheritance is transitive across an arbitrary number of levels. The term ancestor and descendent refer to inheritance of classes across multiple levels. An instance of a subclass is simultaneously an instance of all its ancestor classes. The state of an instance includes a value for every attribute of every ancestor class. Any operation on any ancestor class can be applied to an instance. Each subclass not only inherits all the features of ancestors, but adds its own specific attributes and operators as well.

- ***Attribute***

An attribute of a class is a common property shared by all its objects. For example, Name, Age are the attributes of Person class. Each attribute has a value for each object instance. The attribute Age has the value 24 in object John. Different Object instances may have the same or different values for a given attribute. Each attribute name is unique within a class.

- ***Methods and Operations***

An operation is a function or transformation that may be applied to or by an object. For example, open, close, hide are operations on a class Window. All objects in a class share same operations. A method is an implementation of an operation for a class. All objects in the system interact with each other through these methods. An operation may apply to many different classes. Such an operation is polymorphic; that is, the same operation behaves differently on different classes.

- ***Class Diagrams***

Class diagrams provides a formal notation for modeling classes and their relations to one another. These diagrams are concise, easy to understand and are useful for abstract modeling. In a class diagram, a box, which may have as many as three regions, represents a class. The regions contain from top to bottom: class name, a list of attributes, and a list methods. Each operation name may be followed by optional details such as argument list and result type and the attribute name may be followed by the optional details such as type and default value. Attributes and methods may not be shown and it depends on level of detail desired. Each inheritance relationship is represented in the diagram by an arrow from the super class to subclass.

For example, consider the Bank system where money can be drawn and deposited. The object classes are Ledger, Counter, and Customer. Every Customer must have some minimum amount in his account always. For every Customer of the bank, bank maintains a ledger. He can draw, or deposit cash in the bank. The customer can close his account, as he desires. Let us assume that the requirements analysis has been performed and the analysis document consists of the class diagram which is as shown below:

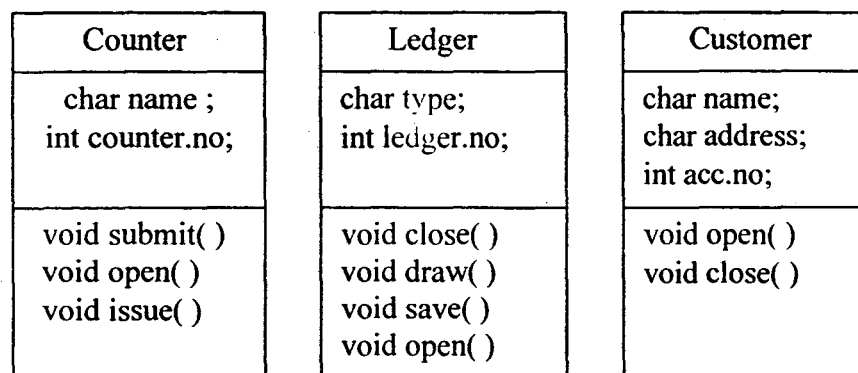


Fig.2.1 Class Diagram

Ledger, Customer, and Counter are the classes in the Banking System. The methods of the class Ledger are draw, save, close, and open and the attributes are type and ledger no. Counter class has the attributes count no., and name and the methods are issue, open. Similarly, the class Customer has the attributes name, address and account no., and the methods are close, and open.

2.1.2 Dynamic Model

The aspects of the system that are concerned with time and changes are dynamic model, in contrast with the static model. The major aspects of dynamic model are states, and events.

- ***Events***

An event is something that happens at a point in time, such as User depresses left button or Flight 123 departs from Delhi. An event has no duration. It is instantaneous. One event may logically precede or follow another. An event is one way of transmission of information from one object to another. That is it causes a transition

- ***Transition***

State transition is defined as change of a state, which is caused by an event. Even though the event causing the transition is specified, it is usual to leave unspecified the object causing the event and the state in which it can cause the event. As suggested by Rumbaugh, in the later stages the various diagrams have to be combined to arrive at such information.

- ***States***

A state is an abstraction of the attribute values of an object. Set of values is grouped together into a state according to properties that affect the gross behavior of the object.

For example, the state of a Bank is either solvent or insolvent depending on whether its assets exceed its liabilities. A state specifies the response of the objects to input events. The response of an object to an event may include an action or a change of state by the object. A state corresponds to the interval between two events received by an object.

Event represents points in time and state represents intervals of time. Both events and states depend on the level of abstraction used.

- ***Scenario and Event Trace***

A scenario is a sequence of events that occur during on particular execution of a system. The scope of scenario can vary, it may include all the events in the system or it may include only those events that are generated by certain objects in the system. For the Banking system, the scenario for drawing cash from the bank is,

Customer submits the Form

Counter verifies the Form

Ledger debits from the Account

Counter issues the Cash

The sequence of events and the objects exchanging events can be shown in an augmented scenario called an Event trace diagram. This diagram shows each object as a vertical line and each event as a horizontal arrow from sender to receiver object. Time increases from top to down. The following figure shows the event trace diagram for drawing cash in the banking system

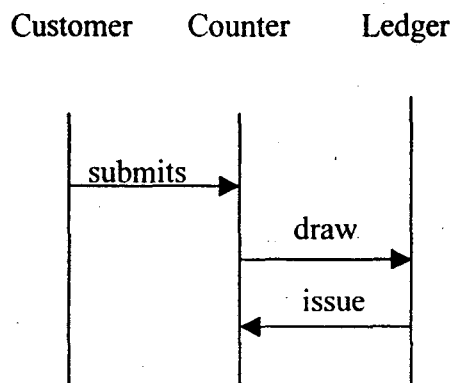


Fig.2.2 Event Trace Diagram

Similar event traces can be drawn for closing and depositing.

- ***State Transition Diagrams***

A state transition diagram relates to states and events. When an event is received, the next state depends on the current state and as well as the event. A state diagram is a graph whose nodes are states and whose directed arcs are transitions labeled by event names. A state is drawn as a rounded rectangle containing the optional name. A transition is drawn as an arrow from the receiving state to the target state; the label on the transition is the same of the event causing the transition. The state transition diagrams for each of the classes in the library system are follows:

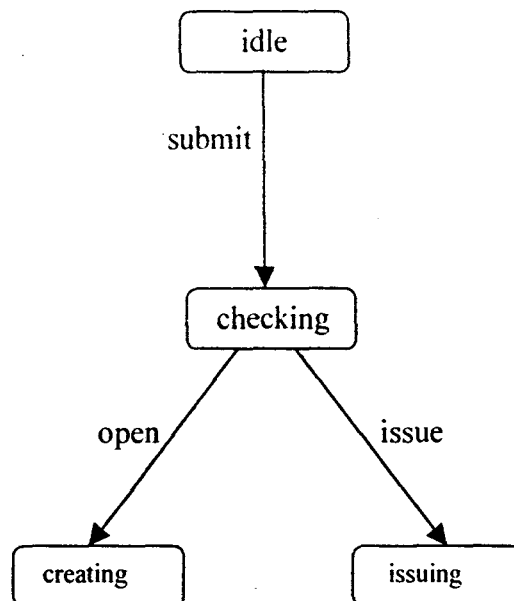


Fig.2.3 State Transition Diagram for class Counter

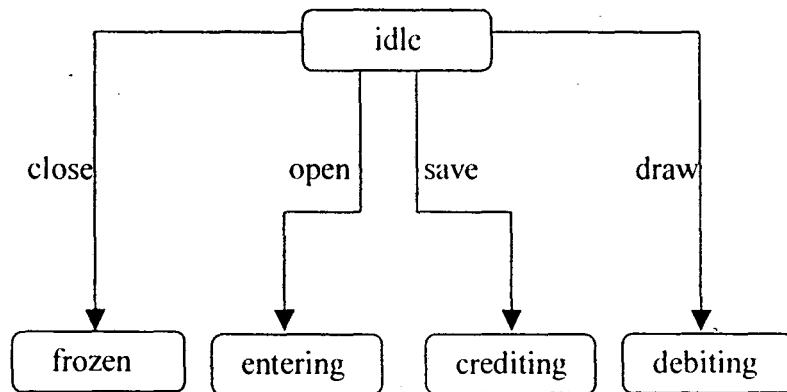


Fig.2.4 *State Transition Diagram for class Ledger*

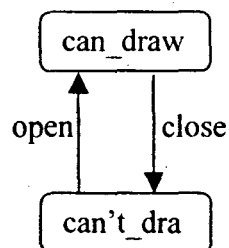


Fig.2.5 *State Transition Diagram for class Customer*

Combining all these state transition diagrams of each class in the system and the event trace diagrams gives the complete description of dynamic behavior of the system. But there is a problem of handling these multiple diagrams over changes.

2.2 Partial Modification Problem

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

For example, in the above banking system, if the analyst wants to change the policy of the accepting deposits from the customer. Suppose the new rule is that no customer is allowed to deposit less than 10,000 rupees. To reflect this change in the system, the state transition diagram of Ledger is to be modified, and in the class Counter a new transition 'deposit' and a state 'counting' has to be present.

If it happens that only the state transition diagram of Ledger is modified to the one as shown below, then we see that the set of state transition diagrams are in inconsistent state. Clearly if the changes were to be made at one place then there would be no inconsistency.

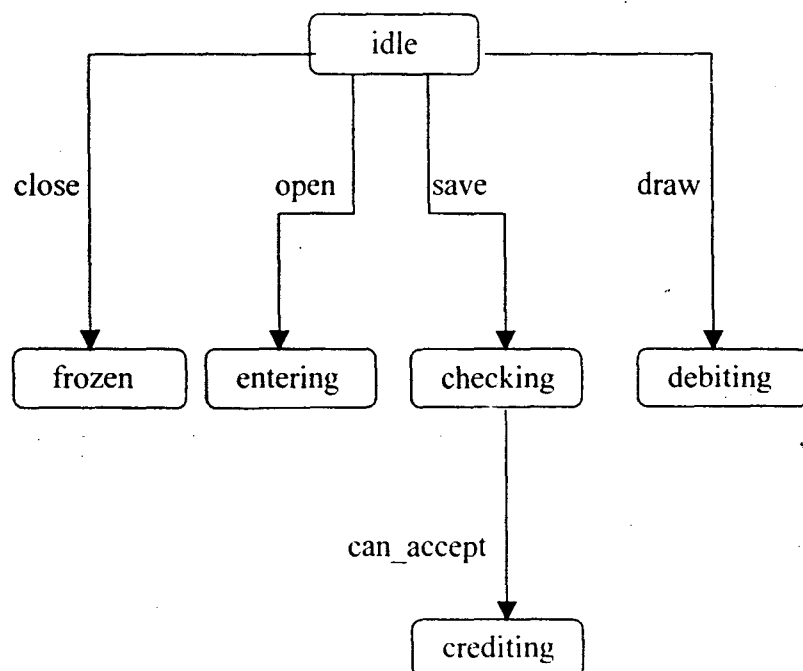


Fig.2.6 Modified State Transition Diagram for class Ledger

Hence, we postulate that the different aspects of dynamic behavior must not be segregated and analyzed separately but all the aspects must be considered together. That is the analysis must deal with an integrated behavior of objects covering the state transitions that an object undergoes and the events that are sent and received by objects. If this were to be done, then any change to be effected would be in one place and also eliminates the partial modification problem.

To achieve this, the state transition diagram should include not only the states under going transition but also the states of objects that are causing the state transitions. For this purpose we first define the augmented state transition diagram (ASTD), then the scenario state transition diagram (SSTD), and lastly the Integrated diagram (ID). We have called this method as integrated method.

2.2.1 Augmented State Transition Diagram

This diagram specifies

state transition of an object

class, an object of which causes the event

state in which it can cause the event

If the event is caused by object belonging to more than one class, than ASTD consists of

state transition of an object

all the classes, an object of which cause the event

state in which each of these objects cause the event

For example, in the Banking system, the ASTD for the state transition of Counter is as shown below. Customer in state `can_draw` state causes the event `submit` when the Counter is in idle state. The Counter then moves to checking state.

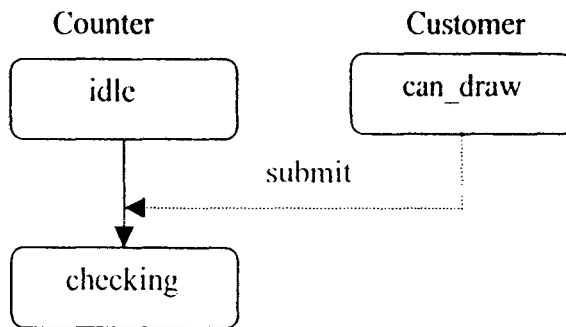


Fig.2.7 *Augmented State Transition Diagram*

2.2.2 Scenario State Transition Diagram

This diagram is drawn as follows

Start with the first event of the event trace and draw the ASTD for the object for which goes through a state transition when this event taken place. Pick up the next event and extend the existing state transition diagram with states, transitions and events to include the ASTD for the state transition caused by the new event. Continue till the events in the event trace is exhausted.

For the banking system, consider the event trace for drawing cash as shown in figure 8. The corresponding SSTD is as shown below:

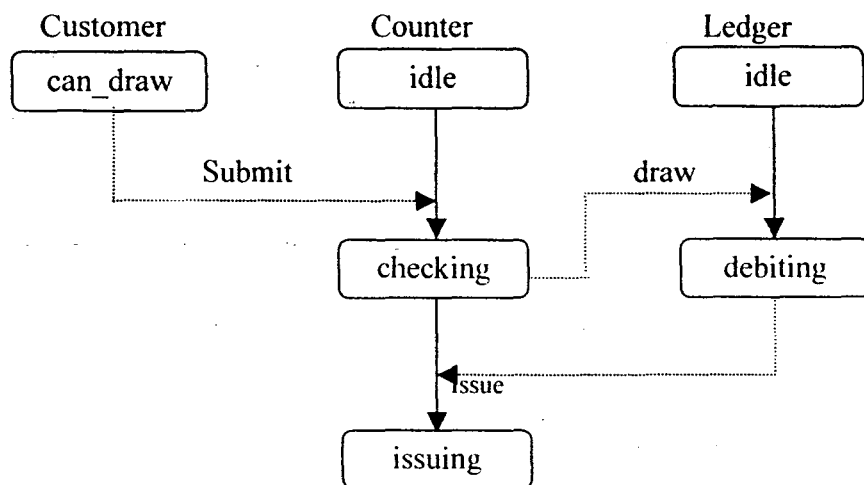


Fig.2.8 *Scenario State Transition Diagram*

For the banking system, consider the event trace for drawing cash from the bank shown in fig.2.2. The first event is *submit*. This causes a transition in Counter. The ASTD for Counter will be as shown in fig.2.7 above. The next event is *checking*. This causes a transition in Ledger from *idle* state to *debiting* state. This implies that a new class Counter has to be added and the ASTD for Counter has to be included in our earlier diagram. The SSTD is as shown in fig.2.8.

2.2.3 Integrated Diagram

An integrated diagram consists of all the Scenario State Transition Diagram of a given system. That is, it includes the scenario state transition diagrams for all the event traces of the system. In this diagram, the states and the transitions are represented as they are represented in state transition diagrams. The class name to which the object belongs is written above the ASTD of the object. The events are denoted as dotted lines with an arrowhead from the state, which causes the event, to the transition, caused by the event. For the library system, the integrated diagram is as shown in the fig.2.9. Assume now, as before that there is a change in depositing procedure. Here, we trace the sequence of events and the state changes for deposit. All events in this sequence are dropped. And the modified integrated diagram is as shown in fig.2.10.

Thus, the integrated diagram gives complete specification of the dynamic behavior of the system, which includes all different behaviors of all objects of all classes in the system. When the system undergoes a change in its dynamic behavior then the changes to be done will be at only one place, the integrated diagram. Thus, any change can be reflected without giving rise to any partial modification.

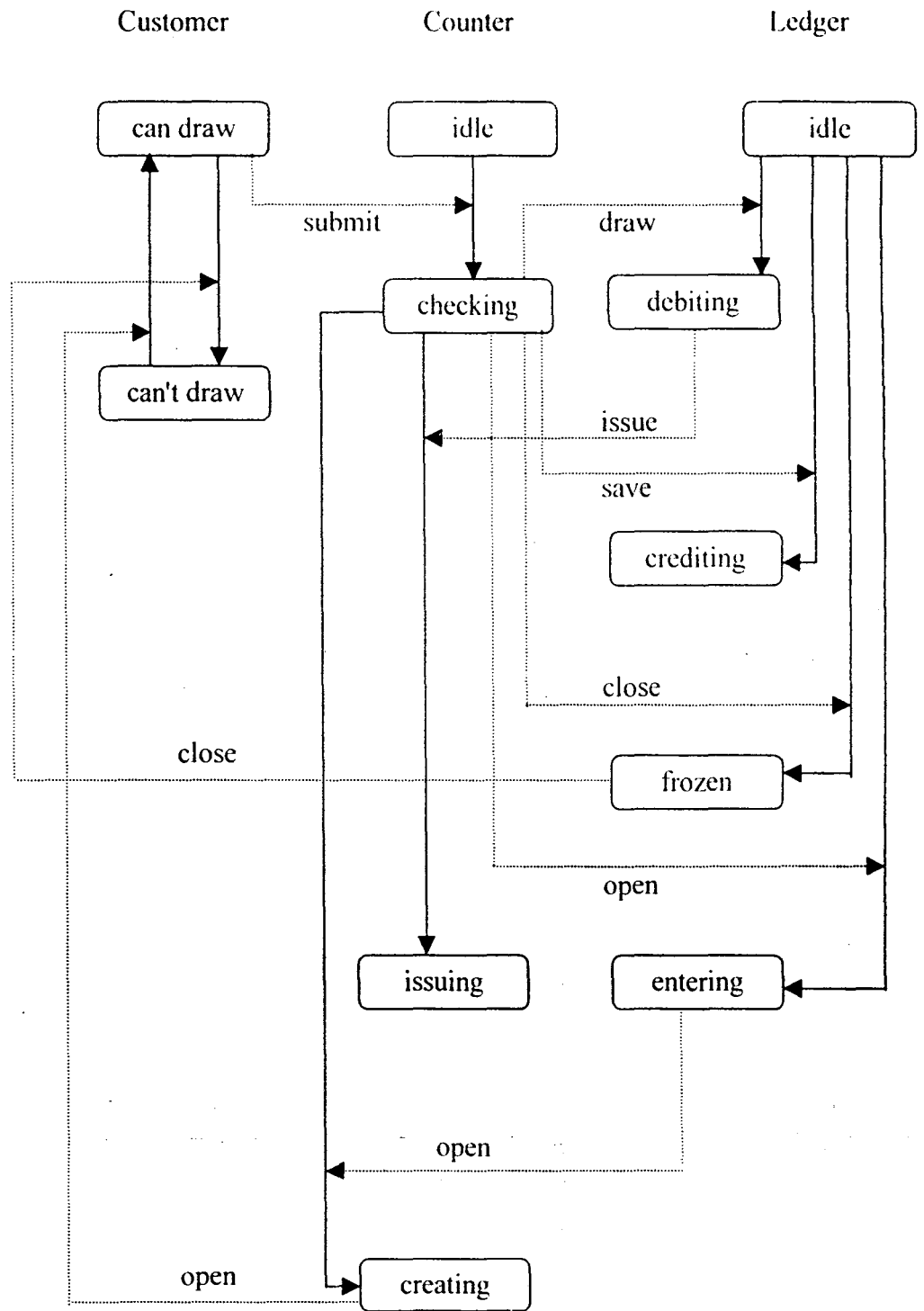


Fig.2.9 *Integrated Diagram for Bank System*

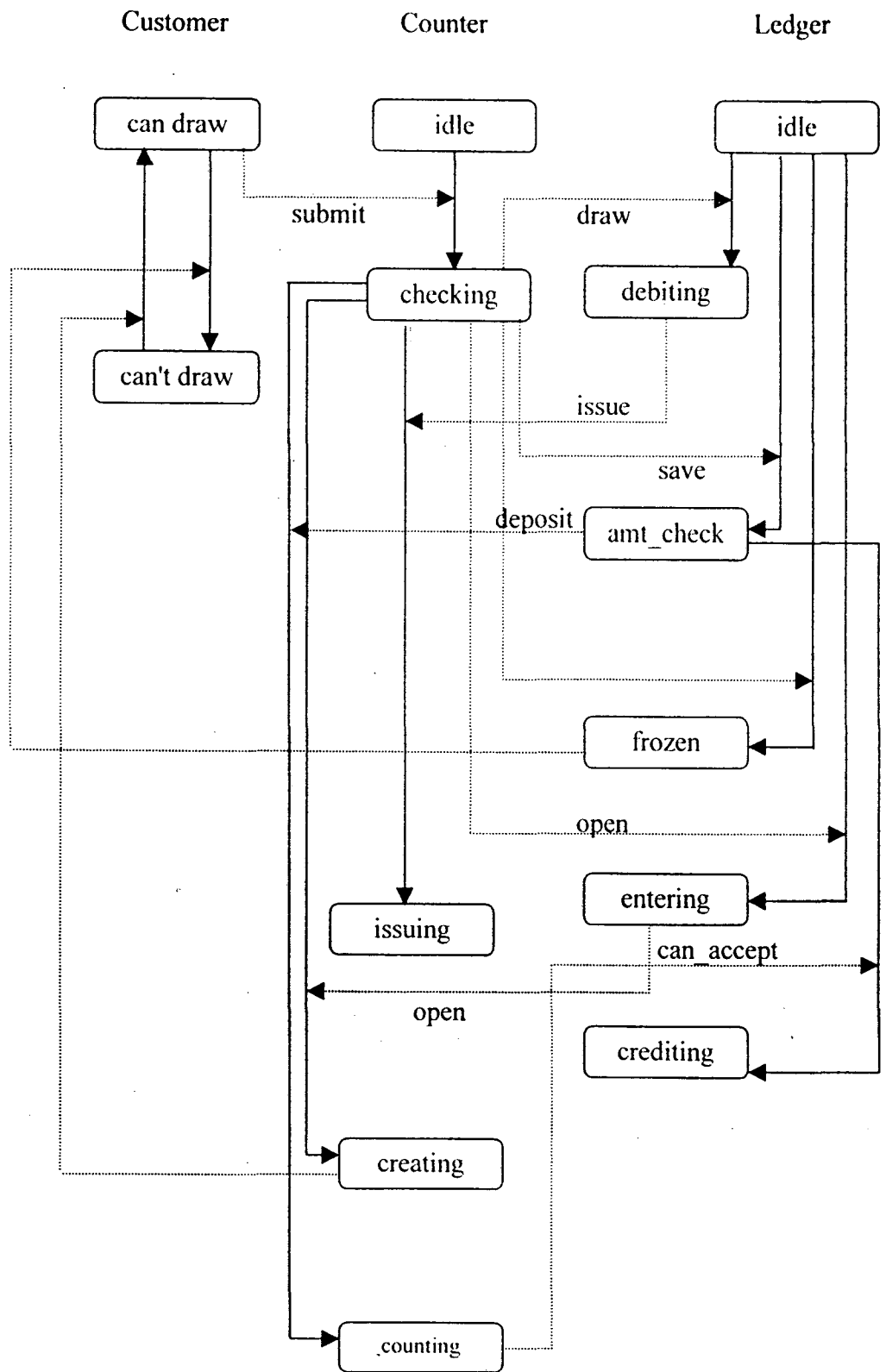


Fig.2.10 *Modified Integrated Diagram for Bank System*

Chapter 3

Consistency

Consistency is one of the main aspect to be considered in Object oriented analysis of a system. Without consistency, how well the system is designed, it becomes irrelevant. Consistency should be maintained while developing the system and as well as when the system undergoes some modifications. All systems are susceptible to change, and object oriented systems are no exception. However, object oriented systems are to provide stability over changes in the requirements. The impact of change is supposed to be easily identified, bounded and assessed. Even though different methods of object oriented analysis has been studied and compared, the study does not include the efficiency with which these methods handle changes in the requirements. The main work has been to identify the changes that may take place in the requirements and study how easily these changes can be incorporated in the system. The change can be in static part or in the dynamic part. However, this chapter explains what are the main steps to be taken while developing an object oriented system in the integrated method described in the previous chapter.

The consistency of object-oriented models can be judged by considering relations among entities in the model. An inconsistent model has the representation in one part that is not correctly reflected in the other portions of the model. To assess the consistency in the static part of the model, each class and its inheritance relation with other classes should be examined. The class diagram can be used to facilitate this activity.

To represent the dynamic behavior of a system, integrated method proposes an integrated diagram, which combines all different behavioral aspects of all objects at one place, which eliminated partial modification problem and changes are localized. To assess consistency in the dynamic part of the system, this diagram must be examined.

To design a flexible and consistent system in this integrated method, the main aspects to be considered in designing are the following:

3.1 Duplication of Names

For clear understanding and for well maintenance of the system, the duplication of class names, state names, and event names should be avoided.

3.2 Transitions and Events

As the analyst is given freedom to consider as many states as he desires, it may happen that he might consider an irrelevant event or transition. To avoid this we restricted the transitions and the events that the analyst draws in the integrated diagram. The analyst can include in the diagram only the transitions that come from a state, which is either starting state of the class to which it belongs, or a member of some transition, which is already included in the diagram. Similarly, the analyst can include only those events into the diagram whose causing state is either starting state of the class to which it belongs, or a member of some transition which is already included in the diagram. As the analyst knows well the real world problem for which he is modeling, this way of implementing the design is always possible. This way of designing helps in considering the transitions and events in systematic way and eliminates many invalid attempts. Transition included in the diagram must be a valid transition, i.e. it must take place between two states belonging to same class and there should be only one transition in each direction. Similarly, an event should be such that, causing state of the event must belong to a different class from the class the transition states of the event belongs to. Every event in the diagram must be a possible event to occur; i.e. it must be one of the methods in the class to which the transition states of the event belong.

3.3 Handling Modifications

- ***Change of Names***

For the convenience, the analyst may change state name, class name and event name. In such a case, before changing the name, the new name to which the object name has to change must be checked for duplication of names and if event name has to be changed, then the modified event also should be possible event to occur.

- **Deletion**

For neat design and reduce complexity in the system, analyst may delete some state or an event or a transition. Greater care should be taken to retain the consistency in the system after deletion.

Whenever, a state is deleted

All those events and transitions that are having this state as a member should be deleted

All those states that are effected by the deletion of events and transitions in the previous step

Repeat the same for each state deleted in previous step

For example in the following system the part in the rectangle should be deleted on deletion of state S2 in class C3, i.e. the states S2 in C2, S2 in C3, the event E2, and the transition from S3 to S2 in c3 must be deleted.

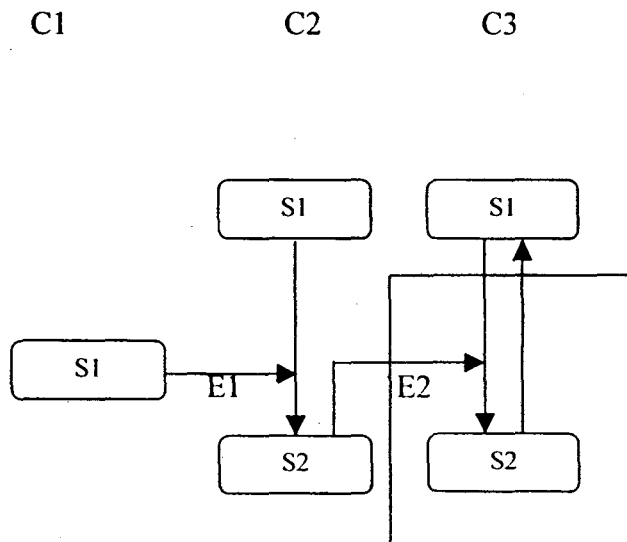


Fig.3.1 Integrated Diagram of an Arbitrary System

Whenever, an event or a transition is deleted,

If the direction of the transition caused by the event is not towards the starting state of the class it belongs, resulting state of the transition is deleted

All the steps of deleting state should be repeated for the state deleted

Thus, the integrated diagram and class diagrams can be used to examine the consistency in the system.

Chapter 4

Software Selection

The software world has changed a lot in recent years. The programming in windows has brought a revolution in the software field. Visual C++ is one such tool for programming in Windows. Microsoft Visual C++ is now an established mainstream product.

4.1 Windows Programming

Programming for Windows is different from old-style batch-oriented or transaction-oriented programming. It has many new features such as better message passing method, effective memory management. It also focuses on Resource Based Programming. The program in Windows, processes user input via messages from the operating system and not like in DOS where the program calls the operating system to get the user input.

4.2 Microsoft Developer Studio

The developer studio is a Windows-hosted integrated development environment that's shared by Visual C++, Microsoft FORTRAN, and some other products. In this a project is a collection of interrelated source files that are compiled and linked to make up an executable Windows based program or a DLL.

4.3 Visual C++

Microsoft Visual C++ is two complete Windows application development in one product. It helps to add today's technology to the applications. Many features are there in Visual C++ that makes programming much easier.

Visual C++ version 4.0 adds extensive debug support to the C run-time library, letting you step directly into run-time functions when debugging an application. The library also

provides a variety of tools to keep track of heap allocations, locate memory leaks, and track down other memory-related problems. This new architecture:

Offers run-time developers full access to the debugging environment.

Provides MFC developers with a smooth transition to both new and existing C++/MFC-specific features.

The various features are:

1. AppWizard

AppWizard is a code generator that creates a working skeleton of Windows application with features, class names, and source code filenames that you specify through dialog boxes. AppWizard code is minimalist code; the functionality is inside the application framework base classes. Its purpose is to get started quickly with a new application. It is capable of generating an OLE control project.

2. ClassWizard

Class Wizard is a program that's accessible from the Developer Studio's View Menu. ClassWizard takes the drudgery out of maintaining Visual C++ class code. ClassWizard writes the prototypes, the function bodies, and the code for new class, new virtual function, or a new message handler function to link the Windows message to the function. It can update the class code that you write, so that maintenance problem is avoided.

3. Component Gallery

The component gallery is a new feature that lets you share software components among different projects. It manages three types of modules. OLE Control, C++ Source modules, Wizard modules. All component gallery items can be imported from and exported to OGX files. These files are new distribution and sharing medium for Visual C++ components.

4. OLE Control

MFC now integrates the OLE Controls Development Kit (CDK) with the rest of MFC and supplies complete OLE control container support.

With the new OLE controls container, you need not understand all the details of using OLE controls in container applications. Support is now based on the CWnd class, which lets you create both the container and the control sides. In MFC version 4.0, an OLE control in a window is seen as a special kind of child window with CWnd functions, including CWnd::CreateControl, which dynamically creates an OLE control rather than an ordinary window.

Other OLE controls support includes creation from a dialog template, preloaded OLE control files for better performance, and transparent keyboard translation in IsDialogMessage.

The Visual C++ dialog editor supports placing OLE controls in a dialog template resource.

OLE controls now use the same run-time library and debug heap as the core of MFC.

5. Microsoft Foundation Class (MFC)

With the latest version of MFC:

You can create full-featured Windows 95 and Windows NT applications that meet the Windows 95 logo requirements.

You get Windows Sockets and MAPI support for connectivity to networking and e-mail.

Fast-integrated data access lets you create powerful database applications and components.

Designing encompasses the disciplined approach to invent a solution for some problem, thus, providing a path from requirements to implementation. The requirements of the problem here are to develop a case tool, which helps in designing a flexible and consistent object-oriented system. This chapter explains how the case tool does the consistency checks while developing the system and how it retains consistency when system under goes a change.

5.1 Data Structures Used

In developing this case tool, mainly four classes had been defined. They are CClass, CState, CSevent, and CEventx. The class CClass gives the class information. i.e., this gives class name, starting state name of that class, attributes and methods. The class CState contains state information i.e., state name, class name to which the state belongs to, and the rectangle drawn for the state. The class CEventx contains transition information, i.e. event id, name of the state which is under going transition, name of resulting state, and class name to which these states belongs to. The class CSevent contains event information i.e. event name, name of the state causing an event and name of the states under going transition.

We have maintained four lists one for each of the above classes.

Consider an example of a system of three classes c1, c2, and c3, contains two states say s1, and s2 belonging to class c1, one state say s1 belonging to class c2, and one state say s1 belonging to class c3. Suppose that in this system, the state s1 in c2 causing an event e1 which intern causing a transition from s1 to s2 in c1. The integrated diagram of this system is given below. This example will be considered in the further sections of this chapter.

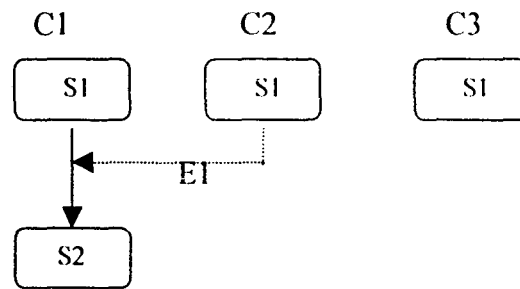


Fig.5.1 *An Integrated System*

5.2 *Steps to Maintain Consistency*

A system becomes meaning less if there is no consistency in the system. Hence consistency is one of the important factor to be considered while designing.

5.2.1 *Duplication of Names*

State name must be unique with in a class. Hence, whenever user attempts to draw a state and enters state information, the case tool checks whether that name is duplicated in the class chosen or not through a function calls. If the name is repeated in that class, the user will be prompted with an error message through a message box without drawing the state. Otherwise, a rounded rectangle is drawn with state name at the center of the rectangle.

Similarly, as the user enters event name and clicks on the dialog box which accepts the event information, again an input of duplicated name will be warned through a message box without drawing the event. Otherwise, the event will be drawn as described above.

For example, the following figure shows how the error message is displayed when the user tries to create state s1 in the class c1, in which already such state exist.

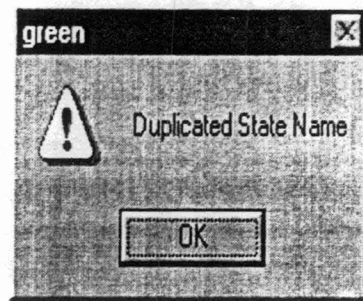


Fig.5.2 *An error message for Duplication of State Name*

The following are functions used to implement this check:

1. CheckSname:

This function takes a CString, and a CPoint members as arguments and returns null or CState node. The first argument is the name of the state about which information is required and the second argument is a point in the where the state supposed to be located in the client area. This function checks the state list and returns null if there is no state with that name at that point on the client area else returns the particular node whose information matches with the given arguments. Since a point in the client area is unique, when it returns null, it means that there is no state with that name in that class. If it returns a node, it means that the name is a repeated one.

2. CheckCname

This function takes a CString members as an argument and returns null or CClass node. The argument is the name of the class for which the test has to be done. It checks the class list and returns null if there is no class with that name and returns the particular node whose name matches with the given name. When it returns null, it means that there is no class with that name. If it returns a node, it means that the name is a repeated one.

3. CheckEname

This function takes a CString members as an argument and returns null or CEvent node. The argument is the name of the event about which the test has to be done. It checks the event list and returns null if there is no event with that name and returns the particular

node whose name matches with the given name. When it returns null, it means that there is no event with that name. If it returns a node, it means that the name is a repeated one.

5.2.2 State Check

This is done whenever the analyst tries to draw a transition or an event. This check checks whether a state is a starting state of the class to which it belongs, or a member of some transition.

This is implemented in this case tool through a function `transcheck`, which takes `CState` node as an argument and returns one or zero. This function first checks the class list to verify for the starting state and if it fails then searches the transition list and returns one if the state is a member of some transition or else returns zero.

5.2.3 Transition Check

Transition must take place between two states belonging to same class. So, when the analyst attempts to draw an automatic transition and chooses two states, the case tool checks the following:

- . Does state check for the state first clicked through the function `transcheck`
- . checks whether the two states selected belongs to same class or not
- . checks whether there already exist a transition or not between those two states

On failing of any of the above checks, the case tool alerts the user with an error message through a message box while canceling the option chosen. Otherwise, transition is drawn as described above. For example, the following figure shows the error message box, which appears when user tries to draw a transition between `s1` of `c1` and `s1`

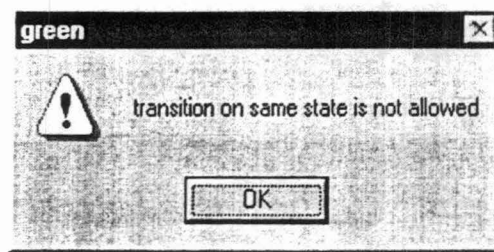


Fig.5.3 An error message when an Invalid Transition is attempted

5.2.4 Event Check

In this, a check is done whether an event is valid or not. That is, it checks whether the class of the state causing the event is different from the class of the states undergoing transitions, and state check is done to the causing state. This check is done when the user chooses event option and clicks on three states. This is implemented through a function call *transcheck* with the causing state as an argument. Then it checks whether the event is possible to occur or not which is done through the function call *possibleevent*. This function takes event name as argument and then checks the methods of the class to which the transition belongs. Depending on the result of this check either it gives an error message or draws an event.

Thus, this case tool avoids invalid states, events, and transitions at the first stage it self and makes the analyst to develop a consistent system.

5.3 Modification

To modify a state name, the analyst has to select a state and then select the option modify from the edit menu of the integrated diagram. Then, the case tool takes the new name and does duplicated name check for that name and depending on the result of the check either it modifies the system and redraws whole system or shows unmodified original system by giving an error message box. If it modifies the state, then the case tool checks the event

list and modifies all those events and transitions' information, which contain this state as a member. To keep track each class's starting state, I have considered one member in the class which contains starting state name. If the state being modified is the starting state of some class, Then simultaneously, this information is also modified.

Whenever a class modified, the case tool automatically changes the class information stored in states and events to make the system consistent.

5.4 Deletion

Whenever, an entity is selected for deletion from the integrated diagram, then the case tool creates three lists, one for states, one for events, and one for transitions. Then it examines all the events, states, and transitions in the diagram through the old lists and inserts all effected states, events and transition into the corresponding new lists. Then it highlights all those members that are inserted into the new lists and gives a dialog box which, as shown below, warns that all the highlighted part of the integrated diagram will be deleted. Then, depending on user's interest case tool responds. Whenever a class is deleted from class diagram, then the related information, i.e. related states, events, and the transitions will be deleted automatically and displays the modified system. The function calls for deleting state, transition, event, and class are *deletestate*, *deletetrans*, *deleteevent*, and *deleteclass* respectively.

Insertion of state or event or transition is same as drawing that particular object.

Thus, the case tool allows analyst to modify his system according to his interest and also takes required steps to maintain consistency in the system.

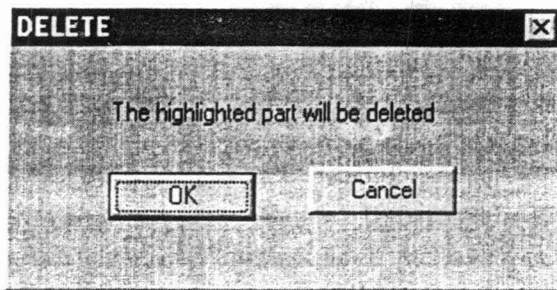


Fig.5.4 A warning message before Deletion

Chapter 6

Generating Code

Writing code is an extension of the design process. Writing code should be straight forward, almost mechanical because all the difficult decisions should already have been made during design. The code should be a simple translation of the design decision into peculiarities of a particular language. Decisions do have to be made while writing code. but each one should effect only a small part of the program so they can be changed easily. Nevertheless the program code is the ultimate embodiment of the solution to the problem. so the way in which it is written is important for maintainability and extensibility. The case tool developed also generates source code in VC++ for the system designed. This helps the analyst in implementing the system and in verifying the correctness of the system, so that he can modify his system design accordingly.

The code generated must represent both the static part as well as the dynamic part of the system.

6.1 Design Description

6.1.1 Code for Static Part

For generating code for the static part of the system, the class list created while developing class diagram is used. Each node in the list is processed and the class information is coded, i.e. class name, attributes and methods of each class, which is same as C++ code for a class definition.

6.1.2 Reasons for choosing VC++

To implement the dynamic behavior, the sequence of events taking place in the system and states causing these events should be considered and represented sequentially. As the each event is nothing but a method in some class, there should be a way. to make an object of one class to signal an event or method of some other object and also to make some object to wait for an event to occur. But there is neither a suitable a data structure

nor a direct/indirect method to implement this using C++. But in VC++ there are two predefined data structure called CWinThread and CEvent classes makes our work easy. Each instance of CWinThread class is called thread, which is nothing but an execution path. The objects of CEvent can be used to make two threads communicate with each other. In addition to this the case tool is itself developed in VC++ so the obvious choice has become VC++.

6.1.3 Code for Dynamic Part

For each class in the system, a thread is created and a list of CEvent objects is also maintained. For each event in the event list created while developing integrated diagram, one corresponding CEvent member is created and inserted into the list. The threads created are used to make the objects communicate each other. The first event drawn in integrated diagram is taken as the first event to take place in the system. The first thread is made to wait for this first event by making it to wait for the corresponding CEvent object to signal. Then from the integrated diagram the sequence of events is taken into consideration and the threads are made to wait and signal to execute using corresponding CEvent objects. As execution of these threads describes the dynamic behavior of the system, the code of these threads is also generated. Generating source code using case tool is explained in the next chapter.

Chapter 7

The Case Tool

The case tool is a graphical user interface; helps in analyst in object oriented modeling a design. This can handle both static and dynamic behavior of the system. It designs static model of the system by allowing system analyst to draw class diagram by specifying the class attributes, and class methods. It designs the dynamic behavior of the system by allowing system analyst to draw integrated diagram of the system by giving states, events, and transitions information. At each step of the development of an object oriented system necessary checks have been done to maintain the consistency of the system. Whenever the system is modified, this case tool does all required consistency checks and retains the consistency of the system.

7.1 How to use the Case Tool

This section explains how an analyst avail this case tool for drawing class diagram and integrated diagrams by designing the same Bank System described in chapter 2. When an analyst runs this application, a menu appears which is as shown below, has the options to create a new model or open an existing model. On choosing *open model* option, it opens an existing model and prompts the analyst to open either class diagram or integrated diagram by showing a dialog box. On choosing class diagram or integrated diagram, it opens corresponding diagrams. Then according to his interest the system can be modified. On choosing *new model* option, the case tool prompts the user to enter model name by showing a dialog box which is as shown below. As he enters the model name, and clicks ok, the case tool creates a directory on that name and opens menu for class diagram.

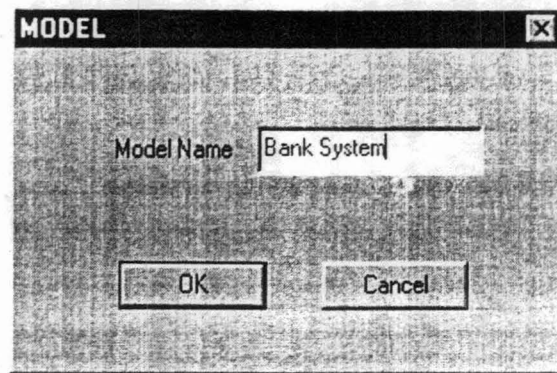


Fig.7.1 *Dialog Box used to create a model*

7.1.1 *Drawing Class Diagram*

- **Class**

To draw a class, first

Choose *class* option from draw menu

1. Click left mouse button on the desired place of the client area
2. Right click on the rectangle appeared when the left mouse button is clicked
Enter the class information in the dialog box displayed when right mouse button is clicked
3. Click on ok of dialog box

This way all the classes participating in the bank system can be drawn.

- **Inheritance**

For drawing an inheritance relationship between two classes

1. choose *inheritance* option from links menu
2. Click on the super class and then on the subclass

For every invalid input and every attempt of creating an invalid inheritance relation, the case tool warns the user through an error message displayed on a message box. The analyst is given an option for deleting or modifying a class in his class diagram. He is also provided with an option of saving his class diagram to a file.

7.1.2 *Drawing Integrated Diagram*

After creating and saving the class diagram, menu for integrated diagram can be opened choosing the ISD option from the window menu of the class diagram. Following the procedures given below can draw all states, events and the required transitions for representing dynamic behavior of the system:

- *State*

To draw a state:

- 1 choose *state* option from draw menu
- 2 click left mouse button on any where on the client area
enter the state name and select class name in the dialog box displayed on left mouse button click
- 3 Click on ok button of dialog box

Then, a rounded rectangle will be drawn with the state name at the center.

- *Event*

To draw an event:

- 1 choose *event* option from draw menu
- 2 click on the state which is causing the event
- 3 click on the state which is undergoing transition
- 4 click on the resulting state of the transition

Enter the event information in the dialog box displayed that appears after clicking on the resulting state, and click on ok button of dialog box.

Then, first transition is drawn and an arrow from state causing event to the transition is drawn.

- *Transition*

To draw a transition:

Choose *automatic transition* from draw menu

click on the state, which is undergoing transition

Click on the resulting state of the transition

Then, an arrow is drawn from first state to second state chosen.

For example, the SSTD for issuing money is drawn using case tool is as follows:

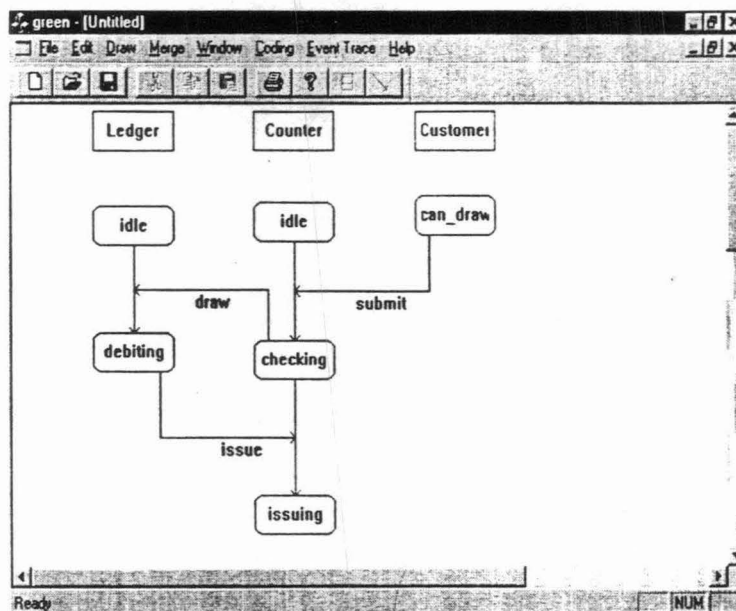


Fig.7.2 *SSTD for Bank System using Case Tool*

Thus, class diagram and integrated diagram can be drawn using the case tool.

7.1.3 Handling Modifications

After the analyst designs his system, it may always happen that he may modify his system. In which case, the system should be flexible enough to adapt these changes. This case tool even allows the analyst to modify his system at all times.

1. Modification

The analyst can modify or change any state/event name.

To change state/event name:

double click left mouse button on the state/event to be deleted, which highlights (draws with red color) the state/event selected which is shown below when the state s2 is chosen.

1 Choose *modify* option from edit menu,

Enter the name to which the state/event name has to be changed in the dialog box appeared on choosing modify option.

2 Click on ok of dialog box.

Then the case tool takes the new name and does duplicated name check for that name and depending on the result of the check either it modifies the system and redraws whole system or shows unmodified original system while giving an error message box.

For example. the following shows the modified system when the state can_draw of customer is changed to draw

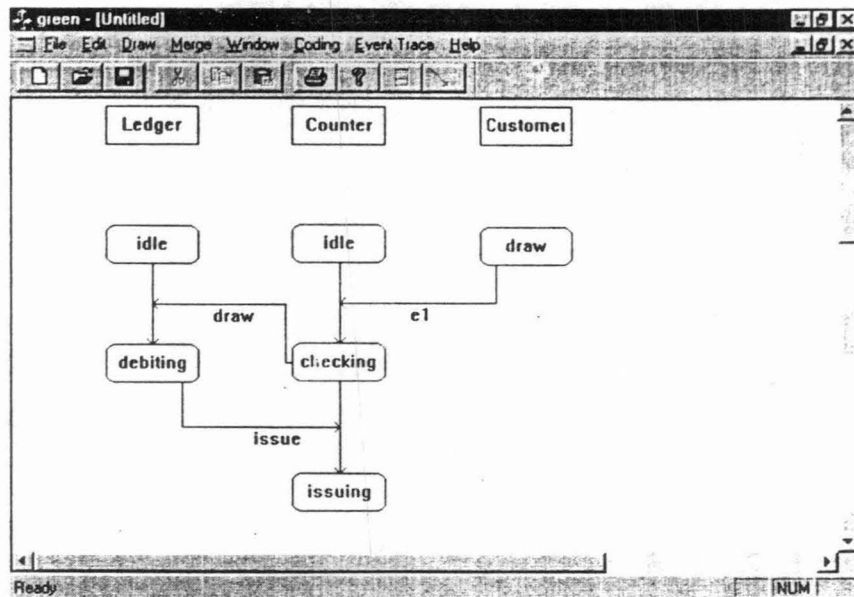


Fig.7.3 Modified SSTD for the Bank System

2. Deletion

A state or an event, or a transition in the system can be deleted by following the steps given below.

double click left mouse button on the state/event/transition to be deleted, which highlights (draws with red color) the state/event selected

Choose *delete* option from edit menu,

Then all the events, states, and transitions that get effected due to the deletion of state/event/transition chosen will be highlighted and prompts the analyst that deletion of state/event/transition chosen will imply that all the highlighted part also will be deleted to maintain the consistency in the system. On clicking on ok of dialog box, deletes all the highlighted part shown in the above step and shows the modified system. On clicking on cancel, it shows the original system without any modifications.

Whenever a class is deleted from class diagram, then the related information, i.e. states, events, and the transitions will be deleted automatically and displays the modified system.

For example, the following figure shows the modified system after deleting the state issuing in the class Counter

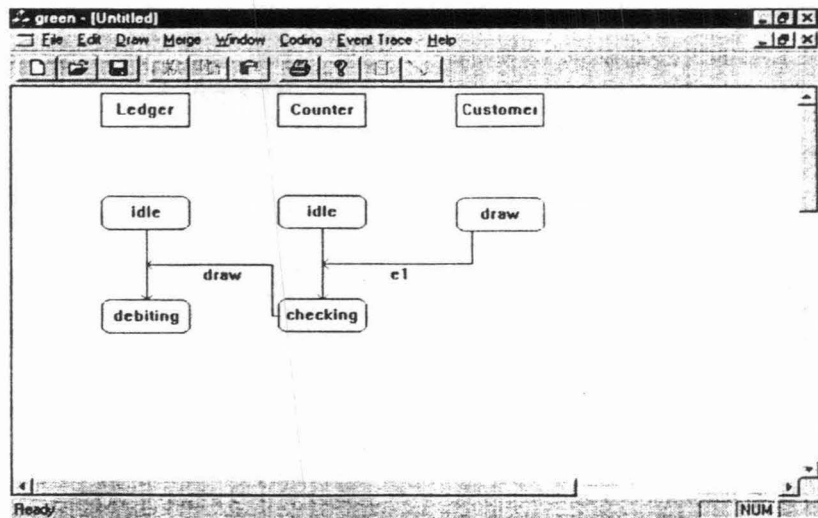


Fig.7.4 SSTD for Bank System after Deletion

Insertion of state or event or transition is same as drawing that particular object. Thus, the case tool allows analyst to modify his system according to his interest.

7.1.4 Generating Code

To generate source code for a system designed

choose the option *generate code* from the coding menu of integrated diagram

Then, you can see the whole source code on the client area. The analyst can also save to a file the code generated by selecting the option *save to file* from coding menu.

Chapter 8

Conclusion

As aforesaid in the previous chapter, integrated method provides a way to combines all different aspects of the dynamic behavior of the system. Thus, helps the analyst to incorporate changes in the requirements of the system by modifying only the integrated diagram.

The case tool presented is a GUI, helps in developing object-oriented system in the integrated method. It helps in representing static and dynamic behavior of the system by enabling an analyst to draw class diagram and integrated diagram. It develops an efficient system by checking consistency at every step. It also allows analyst to modify system and then again retains consistency in the system. It generates source code for the system designed and hence provides a way to implement the system. It gives a simple way of designing by providing an option of merging two systems designed using the case tool.

Finally, it also provides an option of generating event trace diagrams from integrated diagram.

SOURCE CODE

classes used For Modification and deletion

1. Class For State

Class Definition

```
#include "state.h"
#include "stdafx.h"
#include "project.h
class Definition
class CState : public COBJECT
{
protected:
    char sname[20];
    CState();
    CRect rc;
public:
    char cname[20];
    int ar, al, br, bl, l, r, tag ;
    CState ( CString n1, CString n2, CRect &r1 );
    CString GetName();
    Virtual ~CState();
    void Draw ( CDC *pdc );
    CState();
    void SetName( CString &n );
    CRect GetRect( );
};
```

Methods Declaration

```
CState :: CState ( )
{ }
CState :: CState (CString n1, CString n2, CRect &r1).
{
    strcpy(sname, n1);
    strcpy(cname, n2);
    rc=r1;
    ar=0;
    al=0;
    tag=0;
    br=0;
    bl=0;
    l=0;
    r=0;
}
```

```

CState :: ~CState()
{}

CString CState :: GetName()
{
    return sname;
}
CString CState :: GetName()
{
    return sname;
}
void CState :: Draw(CDC *pdc)
{
    CPoint pr, pr2;
    pr.x=16;
    pr.y=12;
    pdc->RoundRect(rc, pr);
    pr=rc.TopLeft();
    pr.x=pr.x+6;
    pr.y=pr.y+6;
    pr2=rc.BottomRight();
    pr2.x=pr2.x-6;
    pr2.y=pr2.y-6;
    pdc->DrawText(sname,-1,&CRect(pr.x,pr.y,pr2.x,pr2.y),DT_CENTER);
}
CRect CState :: GetRect()
{
    return rc;
}
void CState :: SetName(CString &n)
{
    strcpy(sname,n);
}

```

2. *Class For Event*

Class Definition

```

class CSevent : public CObject
{
private:
    char sename[20];
public:
    char s1[20],s2[20],s3[20];
    int e1,p;
    CPoint start, end, bet1, bet2;
    CSevent();
    CSevent( CString &n, CString n1, CString n3, CString n4, int n2, CPoint &st, CPoint
        &en, CPoint &a, CPoint &b);
    CString GetName();
}

```

```

void SetName(CString &n);
~CSevent();
void Draw(CDC *pdc);

```

```
};
```

Methods Declaration

```

Csevent :: CSevent()
{}
Csevent :: CSevent(CString &n, CString n1, CString n3, CString n4, int n2, CPoint &st, CPoint
&en, CPoint &a, CPoint &b)
{
    strcpy(sename,n);
    strcpy(s1,n1);
    e1=n2;
    strcpy(s2,n3);
    strcpy(s3,n4);
    start=st;
    end=en;
    bet1=a;
    bet2=b;
    if((bet1.x==0 && bet1.y==0)&&(bet2.x==0 && bet2.y==0))
    { P=0;
    return;
    }
    if((bet1.x!=0||bet1.y!=0) && (bet2.x==0 && bet2.y==0))
    {
        p=1;
        return;
    }
    if((bet1.x!=0||bet1.y!=0) && (bet2.x!=0||bet2.y!=0))
        p=2;
}
Csevent :: ~CSevent()
{}
CString Csevent :: GetName()
{
    return sename;
}

```

3. *Class For Transition*

Class Definition

```

class CEventx : public CObject
{
private:
    int ename;
public:
    char s1[20], s2[20];
    CPoint start, end, bet1, bet2;
    BOOL tag;
}

```

```

    CString cname;
    int l,r,dir;
    CEventx();
    CEventx(int &n,CString &n1,CString &n2,CPoint &st, Cpoint &en, CPoint &a1,CPoint
        &a2);
    CEventx(int &n,CString &n1,CString &n2,CPoint &st, CPoint &en);
    int GetName();
    void SetName(int &n);
    void Draw(CDC *pdc);
    ~CEventx();
};

```

Methods Declaration

```

Ceventx :: CEventx()
{ }
Ceventx :: CEventx(int &n,CString &n1,CString &n2,CPoint &st,CPoint &en)
{
    ename=n;
    strcpy(s1,n1);
    strcpy(s2,n2);
    start=st;
    end=en;
    tag=FALSE;
    l=0;
    r=0;
}
Ceventx :: CEventx(int &n,CString &n1,CString &n2, CPoint &st,CPoint &en,CPoint &a1,
    Point &a2)
{
    ename=n;
    strcpy(s1,n1);
    strcpy(s2,n2);
    start=st;
    end=en;
    bet1=a1;
    bet2=a2;
    tag=TRUE;
    l=0;
    r=0;
}
void Ceventx :: Draw(CDC *pdc)
{
    if(tag==FALSE)
    {
        pdc->MoveTo(start.x,start.y);
        pdc->LineTo(end.x,end.y);
        DrawArrow(start,end,pdc);
    }
    else
    {

```

```

        pdc->MoveTo(start.x,start.y);
        pdc->LineTo(bet1.x,bet1.y);
        pdc->MoveTo(bet1.x,bet1.y);
        pdc->LineTo(bet2.x,bet2.y);
        pdc->MoveTo(bet2.x,bet2.y);
        pdc->LineTo(end.x,end.y);
        DrawArrow(bet2,end,pdc);
    }
    return;
}

int Ceventx :: GetName()
{
    return ename;
}
void Ceventx :: SetName(int &n)
{
    ename=n;
}
Ceventx :: ~Ceventx()
{}

```

Classes Used For Generating Code

1. Classes Used For Coding Dynamic Behavior

Classes Definition

```

class Event : public CObject
{
public:
    CString c1, c2, s1, s2, s3; CEvent e;
    Event()
    { };
};

```

```

class startstate : public CObject
{
public:
    CString cname,starts;
};

```

Lists Used For Implementation

```

CTypedPtrList < CObList, CClass*> m_class;
CTypedPtrList < CObList, CState*> m_state;
CTypedPtrList < CObList, CState*> m_st;
CTypedPtrList < CObList, CEvent*> m_event;
CTypedPtrList < CObList, CSevent*> m_sevent;
CTypedPtrList < CObList, CSevent*> m_sevt;

```

```

CTypedPtrList < CObList, CClass*> m_class2;
CTypedPtrList < CObList, CState*> m_state2;
CTypedPtrList < CObList, CEventx*> m_event2;
CTypedPtrList < CObList, CEventx*> m_evt;
CTypedPtrList < CObList, CSevent*> m_sevent2
ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONDOWNBLCLK()
    ON_COMMAND(ID_MODIFY, OnModify)
    ON_COMMAND(ID_DELETE, OnDelete)
    ON_COMMAND(IDC_GENERATECODE, OnGeneratecode)
    ON_COMMAND(ID_SAVECODE, OnSavecode)

```

Functions Used For Consistency

1. Functions Used For Duplication Of Names

```

CState* CheckSname(CString n,CPoint pt)
{
    POSITION pos;
    CState *temp;
    CClass *ct;
    for(pos = m_class.GetHeadPosition();pos!=NULL;m_class.GetNext(pos))
    {
        ct=m_class.GetAt(pos);
        if(pt.x>=(ct->GetRect()).TopLeft().x
            && pt.x<=(ct->GetRect()).BottomRight().x)
            break;
    }
    for(pos=m_state.GetHeadPosition();pos!=NULL;m_state.GetNext(pos))
    {
        temp=m_state.GetAt(pos);
        if((n.CompareNoCase(temp->GetName())==0)
            &&(strcmp(temp->cname,ct->GetName())==0))
            return temp;
    }
    temp=NULL;
    return temp;
}

```

```

CState* CheckSname(CString n,CString nc)
{
    POSITION pos;
    CState *temp;
    for(pos=m_state.GetHeadPosition();pos!=NULL;m_state.GetNext(pos))
    {
        temp=m_state.GetAt(pos);
        if((n.CompareNoCase(temp->GetName())==0)
            &&(strcmp(temp->cname,nc)==0))
            return temp;
    }
    temp=NULL;
}

```



```

        return temp;
    }
    CEventx* CheckEname(int &n)
    {
        CEventx *temp;
        POSITION pos;
        for(pos=m_event.GetHeadPosition();pos!=NULL;)
        {
            temp=m_event.GetNext(pos);
            if(n==temp->GetName())
                return temp;
        }
        temp=NULL;
        return temp;
    }

    int startstate(CState* state)
    {
        int t=1;
        CState* st;
        POSITION pos;
        for(pos=m_state.GetHeadPosition();pos!=NULL;)
        {
            st=m_state.GetNext(pos);
            if(strcmp(state->cname,st->cname)==0)
            {
                t=0;
                break;
            }
        }
        return t;
    }

    CClass* CheckCname(CString n)
    {
        CClass *temp;
        POSITION pos;
        for(pos=m_class.GetHeadPosition();pos!=NULL;)
        {
            temp=m_class.GetNext(pos);
            CString str(temp->GetName());
            if(str.CompareNoCase(n)==0)
                return temp;
        }
        temp=NULL;
        return temp;
    }

    CEventx* CheckStevent(CState* s1,CState* s2)
    {
        POSITION pos;
        CEventx *temp;
        for(pos=m_event.GetHeadPosition();pos!=NULL;)
        {
            temp=m_event.GetNext(pos);

```

```

        if((strcmp(temp->s1,s1->GetName())==0)&&
           (strcmp(temp->cname,s1->cname)==0) &&
           (strcmp(temp->s2,s2->GetName())==0))
        return temp;
    }

    return NULL;
}

```

```

int transcheck(CState* s1)

```

```

{
    CClass* cl;
    int t=0;
    cl=CheckCname(s1->cname);
    if(strcmp(cl->st_state,s1->GetName())==0)
        t=1;
    lse
    {
        CEventx* et;
        POSITION pos;
        for(pos=m_event.GetHeadPosition();pos!=NULL;)
            { et=m_event.GetNext(pos);
              if(strcmp(et->s2,s1->GetName())==0 &&
                 strcmp(et->cname,s1->cname)==0)
                  { t=1;break;}
            }
    }
    return t;
}

```

```

int eventcheck(CState* s1,CState* s2)

```

```

{
    int t=0,i=0;
    t=transcheck(s2);
    if(t==0)
        i=0;
    else
        {
            CClass* cl;
            cl=CheckCname(s1->cname);
            if(strcmp(cl->st_state,s1->GetName())==0)
                i=1;
        }
    else
        {
            POSITION pos;
            for(pos=m_event.GetHeadPosition();pos!=NULL;)
                {
                    CEventx* et;
                    et=m_event.GetNext(pos);
                    if((strcmp(et->s1,s1->GetName())==0 &&
                       strcmp(et->cname,s1->cname)==0)

```

```

        || (strcmp(et->s2,s1->GetName())==0 &&
           strcmp(et->cname,s1->cname)==0))
        {i=1;break;}
    }
}
return i;
}

```

To keep track of direction of transition

```

int dir_event(CEventx* event)
{
    CState *st1,*st2;
    CClass* cl;int i;
    st1=CheckSname(event->s1,event->start);
    cl=CheckCname(st1->cname);
    if(strcmp(event->s1,cl->st_state)==0)
        return 1;
    if(strcmp(event->s2,cl->st_state)==0)
        return 0;
    CEventx* et;
    et=CheckStevent(event->s2,event->s1);
    if(et!=NULL)
        return 0;
    if(strcmp(event->s1,cl->st_state)!=0)
    {
        while( (strcmp(st1->GetName(),cl->st_state)!=0)&&(st1!=NULL) )
            {
                POSITION pos;
                for(pos=m_event.GetHeadPosition();pos!=NULL;)
                {
                    et=m_event.GetNext(pos);
                    if((strcmp(et->s2,st1->GetName())==0) && (et->dir == 1))
                    {
                        st2=CheckSname(et->s1,et->start);
                        st1=st2;
                        break;
                    }
                }
            }
        if(strcmp(st1->GetName(),cl->st_state)==0)
            i=1;
        else i=0;
    }
    return i;
}

```

Functions of view class

```
void OnLButtonDown(UINT nFlags, CPoint point)
{
    set=0;
    if(flag3==TRUE)
        OnDeselect();
    CClientDC d(this);
    CDC *pdc=(CDC *)&d;
    OnPrepareDC(pdc);
    d.DPtoLP(&point);
    switch(flag1)
    {
        case STATE :if(CheckState(point)==NULL)
            {
                s=NewState(point);
                if(s!=NULL)
                {
                    CRect re=s->GetRect();
                    mymessage(re);
                    s->Draw(pdc);
                }
            }
            break;

        case EVENT : count=count+1;
            switch(count)
            {
                case 1:
                    if((temp=CheckState(point))!=NULL)
                        pl=point;
                    else
                    {
                        AfxMessageBox("Click on the state undergoing transition");
                        count=0;
                    }
                    break;

                case 2:
                    if((temp1=CheckState(point))!=NULL)
                    { if(temp->GetName()==temp1->GetName())
                        {
                            AfxMessageBox("transition on same state is not allowed");
                            count=0;
                            break;
                        }
                    }
                    if(strcmp(temp->cname,temp1->cname)!=0)
                    {
                        AfxMessageBox("Transition is not allowed from a state of
                            same class to a state of same class");
                    }
            }
    }
}
```

```

count=1;
break;
} //end of same class
if(CheckStevent(temp,temp1)!=NULL)
{
    AfxMessageBox("Transition already exist");
    break;
} //end of existing transition
if(pdock-transcheck(temp)==1)
{
    e=NewTran(temp,temp1);
    Invalidate();
}
else
{
    AfxMessageBox("Invalid Transition");
    count=1;
}
}
else
{
    AfxMessageBox("Click on the resulting state");
    count=1;
}
count=0;
break;
}
break;
case SEVENT :count=count+1;
switch(count)
{
case 1:
    if((temp=CheckState(point))!=NULL)
    p1=point;
    else
    {
        AfxMessageBox("Click On A State First");
        count=0;
    }
    break;

case 2:
    if((temp1=CheckState(point))!=NULL)
    { p2=point;
      if(strcmp(temp->GetName(),
temp1->GetName())==0)
        if(strcmp(temp->cname,temp1->cname)==0)
        {
            AfxMessageBox("Different State Should Be
Selected");
            count=1;
        }
    }
}
}

```

```

    }
    if(strcmp(temp->cname,temp1->cname)==0)
    {
        AfxMessageBox("This event is not allowed");
        count=1;
        break;
    }
}
else
{
    AfxMessageBox("Click On Second State");
    count=1;
}
break;

```

case 3:

```

CState* temp2;
if((temp2=CheckState(point))!=NULL)
{
    p3=point;
    if(strcmp(temp1->GetName(),
temp2->GetName())==0)
    if(strcmp(temp1->cname,temp2->cname)==0) .
    {
        AfxMessageBox("Different state should be chosen");
        count=2;
        break;
    }
    if(strcmp(temp2->GetName(),temp->GetName())==0)
    if(strcmp(temp2->cname,temp->cname)==0)
    {
        AfxMessageBox("Different state should be chosen");
        count=2;
        break;
    }
    if(strcmp(temp1->cname,temp2->cname)!=0)
    {
        AfxMessageBox("Transition States Must Belong
        To Same Class");
        count=2;
        break;
    }
}
else
{
    if(CheckStevent(temp1,temp2)!=NULL)
    { AfxMessageBox("Event Already Exist");
    break;
    }
}
if(eventcheck(temp,temp1)==1)
{
    if(m_sevent.IsEmpty())

```

```

        {
        if(se1.DoModal()==IDOK)
        {
            e=NewEvent(p1,p2,p3);
            se=NewSevent (p1,p2,p3,e,se1.m_sename);
            e->Draw(pdc);
            se->Draw(pdc);
            Invalidate();
        }
        }//end of if
    else
    {
        if(se1.DoModal()==IDOK)
        {
            if(CheckSename(se1.m_sename)!=NULL)
                AfxMessageBox("Duplicated Event Name");
            else
            {
                e=NewEvent(p1,p2,p3);
                se=NewSevent(p1,p2,p3,e,se1.m_sename);
                e->Draw(pdc);
                se->Draw (pdc);
                Invalidate();
            }
        }
    }//end of else
} //end of eventcheck if
else
{ AfxMessageBox("Invalid Event");
  count=0;
  break;
}
count=0;
} //end of else
} //end of if
else
{
    AfxMessageBox("Click On Third State");
    count=2;
    break;
}
} //end of sevent's switch
} //end of flag
CScrollView::OnLButtonDown(nFlags, point);
}

```

Function for Selecting an object

```

void OnLButtonDbIClk(UINT nFlags, CPoint point)
{
    set=0;

```

```

CRect rc;
int ss=4;
float m,xr,yr,len;
CPoint t1,b1,b2,e1;
CPen *rpen,*oldpen,*bpen;
CClientDC d(this);
CDC *pdc=(CDC *)&d;
OnPrepareDC(pdc);
d.DPtoLP(&point);
if(flag3==TRUE)
OnDeselect();
for(pos=m_class.GetHeadPosition();pos!=NULL;
m_class.GetNext(pos))
{
    c=m_class.GetAt(pos);
    rc=c->GetRect();
    t1=rc.TopLeft();
    b1=rc.BottomRight();
    if((point.x>=t1.x && point.x<=b1.x)&&
        (point.y>=t1.y && point.y<=b1.y))
    {
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=pdc->SelectObject(rpen);
        pdc->SetTextColor(RGB(255,0,0));
        pdc->SetBkColor(RGB(255,255,255));
        c->Draw(pdc);
        rpen=pdc->SelectObject(oldpen);
        delete rpen;
        pdc->SetTextColor(RGB(0,0,0));
        flag2=CLASS;
        flag3=TRUE;
        return;
    }
}
for(pos=m_state.GetHeadPosition();pos!=NULL;
    m_state.GetNext(pos))
{
    s=m_state.GetAt(pos);
    rc=s->GetRect();
    t1=rc.TopLeft();
    b1=rc.BottomRight();
    if((point.x>=t1.x && point.x<=b1.x)&&(point.y>=t1.y && point.y<=b1.y))
    {
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=pdc->SelectObject(rpen);
        pdc->SetTextColor(RGB(255,0,0));
        pdc->SetBkColor(RGB(255,255,255));
        s->Draw(pdc);
        rpen=pdc->SelectObject(oldpen);
        delete rpen;
        pdc->SetTextColor(RGB(0,0,0));
    }
}

```



```

        flag2=STATE;
        flag3=TRUE;
        return;
    }
}
for(pos=m_event.GetHeadPosition();pos!=NULL;
    m_event.GetNext(pos))
{
    e=m_event.GetAt(pos);
    t1=e->start;
    e1=e->end;
    st1=CheckSname(e->s1,e->start);
    st2=CheckSname(e->s2,e->end);
    if(e->tag==FALSE)
    {
        if(t1.y>e1.y)
            if(point.y>t1.y||point.y<e1.y) continue;
        if(t1.y<e1.y)
            if(point.y<t1.y||point.y>e1.y) continue;
        if(abs(point.x-e1.x)>ss) continue;
        if(CheckSevent(e)==0)
        {
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
            oldpen=cdc->SelectObject(rpen);
            cdc->SetTextColor(RGB(255,0,0));
            e->Draw(cdc);
            rpen=cdc->SelectObject(bpen);
            cdc->SetTextColor(RGB(0,0,255));
            cdc->SetBkColor(RGB(255,255,255));
            st1->Draw(cdc);
            st2->Draw(cdc);
            bpen=cdc->SelectObject(oldpen);
            cdc->SetTextColor(RGB(0,0,0));
            delete(rpen);
            delete(bpen);
            flag2=EVENT;
            flag3=TRUE;
            return;
        }
    }
else
{
    b1=e->bet1;
    b2=e->bet2;
    if((t1.x>b1.x && (point.x<=t1.x && point.x>=b1.x))
        ||(t1.x<b1.x && (point.x>=t1.x && point.x<=b1.x)))
    {
        if(abs(b1.y-point.y)<=ss && CheckSevent(e)==0)
        {
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));

```

```

bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
oldpen=cdc->SelectObject(rpen);
cdc->SetTextColor(RGB(255,0,0));
e->Draw(cdc);
rpen=cdc->SelectObject(bpen);
cdc->SetBkColor(RGB(255,255,255));
cdc->SetTextColor(RGB(0,0,255));
st1->Draw(cdc);
st2->Draw(cdc);
bpen=cdc->SelectObject(oldpen);
cdc->SetTextColor(RGB(0,0,0));
delete bpen;
delete rpen;
flag2=EVENT;
flag3=TRUE;
return;
}
}
if((b1.y>b2.y && (point.y<=b1.y && point.y>=b2.y))
    ||(b1.y<b2.y && (point.y>=b1.y && point.y<=b2.y)))
{
if(abs(b2.x-point.x)<=ss && CheckSevent(e)==0)
{
rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
oldpen=cdc->SelectObject(rpen);
cdc->SetTextColor(RGB(255,0,0));
e->Draw(cdc);
rpen=cdc->SelectObject(bpen);
cdc->SetBkColor(RGB(255,255,255));
cdc->SetTextColor(RGB(0,0,255));
st1->Draw(cdc);
st2->Draw(cdc);
bpen=cdc->SelectObject(oldpen);
cdc->SetTextColor(RGB(0,0,0));
delete bpen;
delete rpen;
flag2=EVENT;
flag3=TRUE;
return;
}
}
if((b2.x<e1.x && (point.x<=e1.x && point.x>=b2.x))
    ||(b2.x>e1.x && (point.x>=e1.x && point.x<=b2.x)))
{
if(abs(e1.y-point.y)>ss || CheckSevent(e)==1)
continue;
rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
oldpen=cdc->SelectObject(rpen);
cdc->SetTextColor(RGB(255,0,0));

```

```

e->Draw(pdc);
rpen=pdc->SelectObject(bpen);
pdc->SetTextColor(RGB(0,0,255));
pdc->SetBkColor(RGB(255,255,255));
st1->Draw(pdc);
st2->Draw(pdc);
bpen=pdc->SelectObject(oldpen);
delete rpen;
delete bpen;
pdc->SetTextColor(RGB(0,0,0));
flag2=EVENT;
flag3=TRUE;
return;
}
}
}
for(pos=m_sevent.GetHeadPosition();pos!=NULL;
    m_sevent.GetNext(pos))
{
se=m_sevent.GetAt(pos);
st1=CheckSname(se->s1,se->start);
st2=CheckSname(se->s2,se->end);
st3=CheckSname(se->s3,se->end);
switch(se->p)
{
case 0: t1=se->start;
        b1=se->end;
        m=float(b1.y-t1.y)/float(b1.x-t1.x);
        if(b1.x!=t1.x)
        {
xr=(m*(point.y-b1.y)+m*m*b1.x+point.x)/(1+m*m);
yr=(m*(point.x-b1.x)+m*m*point.y+b1.y)/(1+m*m);
if(t1.x>b1.x)
if(xr<b1.x || xr>t1.x) break;
if(t1.x<b1.x)
if(xr<t1.x || xr>b1.x) break;
}
else
{
if(t1.y>b1.y)
if(point.y<b1.y || point.y>t1.y) break;
if(t1.y<b1.y)
if(point.y<t1.y || point.y>b1.y) break;
}
len=(float)(sqrt((b1.x-t1.x)*(b1.x-t1.x)+
(b1.y-t1.y)*(b1.y-t1.y)));
if(abs((point.y*(b1.x-t1.x)-point.x*(b1.y-t1.y)-
(b1.x*t1.y-b1.y*t1.x))>((int)(ss * len)))
break;
e=CheckEname(se->e1);
rpen=new CPen(PS_SOLID,1,RGB(255,0,0));

```

```

bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
oldpen=cdc->SelectObject(rpen);
cdc->SetTextColor(RGB(255,0,0));
se->Draw(pdc);
e->Draw(pdc);
rpen=cdc->SelectObject(bpen);
cdc->SetTextColor(RGB(0,0,255));
cdc->SetBkColor(RGB(255,255,255));
st1->Draw(pdc);
st2->Draw(pdc);
st3->Draw(pdc);
bpen=cdc->SelectObject(oldpen);
delete rpen;
delete bpen;
cdc->SetTextColor(RGB(0,0,0));
flag2=SEVENT;
flag3=TRUE;
return;
break;
case 1:
    t1=se->start;
    b1=se->bet1;
    e1=se->end;
    if((t1.y>b1.y && (point.y<=t1.y && point.y>=b1.y))
        ||(t1.y<b1.y && (point.y>=t1.y && point.y<=b1.y)))
        {
            if(abs(b1.x-point.x)<=ss)
            {
                e=CheckEname(se->e1);
                rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
                bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
                oldpen=cdc->SelectObject(rpen);
                cdc->SetTextColor(RGB(255,0,0));
                se->Draw(pdc);
                e->Draw(pdc);
                rpen=cdc->SelectObject(bpen);
                cdc->SetTextColor(RGB(0,0,255));
                cdc->SetBkColor(RGB(255,255,255));
                st1->Draw(pdc);
                st2->Draw(pdc);
                st3->Draw(pdc);
                bpen=cdc->SelectObject(oldpen);
                delete rpen;
            delete bpen;
                cdc->SetTextColor(RGB(0,0,0));
                flag2=SEVENT;
                flag3=TRUE;
                return;
            }
        }
    if((e1.x>b1.x && (point.x<=e1.x && point.x>=b1.x))

```

```

        ||(e1.x<b1.x && (point.x>=e1.x && point.x<=b1.x)))
    {
        if(abs(e1.y-point.y)>ss)
            break;
        e=CheckEname(se->e1);
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
        oldpen=cdc->SelectObject(rpen);
        cdc->SetTextColor(RGB(255,0,0));
        se->Draw(cdc);
        e->Draw(cdc);
        rpen=cdc->SelectObject(bpen);
        cdc->SetBkColor(RGB(255,255,255));
        cdc->SetTextColor(RGB(0,0,255));
        st1->Draw(cdc);
        st2->Draw(cdc);
        st3->Draw(cdc);
        bpen=cdc->SelectObject(oldpen);
        cdc->SetTextColor(RGB(0,0,0));
        delete bpen;
        delete rpen;
        flag2=SEVENT;
        flag3=TRUE;
        return;
    }
    break;

```

case 2:

```

    t1=se->start;
    b1=se->bet1;
    b2=se->bet2;
    e1=se->end;
    if((t1.x>b1.x && (point.x<=t1.x && point.x>=b1.x))
        ||(t1.x<b1.x && (point.x>=t1.x && point.x<=b1.x)))
    {
        if(abs(b1.y-point.y)<=ss)
        {
            e=CheckEname(se->e1);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
            oldpen=cdc->SelectObject(rpen);
            cdc->SetTextColor(RGB(255,0,0));
            se->Draw(cdc);
            e->Draw(cdc);
            rpen=cdc->SelectObject(bpen);
            cdc->SetBkColor(RGB(255,255,255));
            cdc->SetTextColor(RGB(0,0,255));
            st1->Draw(cdc);
            st2->Draw(cdc);
            st3->Draw(cdc);
            bpen=cdc->SelectObject(oldpen);

```

```

        pdc->SetTextColor(RGB(0,0,0));
        delete bpen;
        delete rpen;
        flag2=SEVENT;
        flag3=TRUE;
return;
    }
}
if((b1.y>b2.y && (point.y<=b1.y && point.y>=b2.y))
    ||(b1.y<b2.y && (point.y>=b1.y && point.y<=b2.y)))
{
    if(abs(b2.x-point.x)<=ss)
    {
        e=CheckEname(se->e1);
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
        oldpen=pdc->SelectObject(rpen);
        pdc->SetTextColor(RGB(255,0,0));
        se->Draw(pdc);
        e->Draw(pdc);
        rpen=pdc->SelectObject(bpen);
        pdc->SetBkColor(RGB(255,255,255));
        pdc->SetTextColor(RGB(0,0,255));
        st1->Draw(pdc);
        st2->Draw(pdc);
        st3->Draw(pdc);
        bpen=pdc->SelectObject(oldpen);
        pdc->SetTextColor(RGB(0,0,0));
        delete bpen;
        delete rpen;
        flag2=SEVENT;
        flag3=TRUE;
        return;
    }
}
if((b2.x<e1.x && (point.x<=e1.x && point.x>=b2.x))
    ||(b2.x>e1.x && (point.x>=e1.x && point.x<=b2.x)))
{
    if(abs(e1.y-point.y)>ss)
        break;
    e=CheckEname(se->e1);
    rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
    bpen=new CPen(PS_SOLID,1,RGB(0,0,255));
    oldpen=pdc->SelectObject(rpen);
    pdc->SetTextColor(RGB(255,0,0));
    se->Draw(pdc);
    e->Draw(pdc);
    rpen=pdc->SelectObject(bpen);
    pdc->SetTextColor(RGB(0,0,255));
    pdc->SetBkColor(RGB(255,255,255));
    st1->Draw(pdc);

```

```

        st2->Draw(pdc);
        st3->Draw(pdc);
        bpen=pdc->SelectObject(oldpen);
        delete rpen;
        delete bpen;
        pdc->SetTextColor(RGB(0,0,0));
        flag2=SEVENT;
        flag3=TRUE;
        return;
    }
    break;
}
}
}

CScrollView::OnLButtonDbClick(nFlags, point):
}

```

Function for modifying state name or event name

```

void OnModify()
{
    set=0;
    cmoddlg mod;
    switch(flag2)
    {
        case STATE:
            CState* st;
            if(mod.DoModal()==IDOK)
            { st=CheckSname(mod.m_stname,
            (s->GetRect()).CenterPoint());
            if(st==NULL)
            { CString name;
            name=s->GetName();
            s->SetName(mod.m_stname);
            POSITION pos;
            for(pos=m_class.GetHeadPosition();pos!=NULL;)
            { CClass* cl=m_class.GetNext(pos);
            if(strcmp(cl->st_state,name)==0 &&
            strcmp(cl->GetName(),s->cname)==0)
            {
                strcpy(cl->st_state,s->GetName());
                break;
            }
            }
            }
            for(pos=m_sevent.GetHeadPosition();pos!=NULL;)
            {
                CSevent *temp=m_sevent.GetNext(pos);
                CState* stt=CheckSname(temp->s1, temp->start);
                CEventx* et=CheckEname(temp->e1);
                if(strcmp(name,temp->s1)==0 &&
                strcmp(stt->cname,s->cname)==0)
            }
            }
    }
}

```

```

        strcpy(temp->s1,s->GetName());
        if(strcmp(name,temp->s2)==0 &&
        strcmp(et->cname,s->cname)==0)
        strcpy(temp->s2,s->GetName());
        if(strcmp(name,temp->s3)==0 &&
        strcmp(et->cname,s->cname)==0)
        strcpy(temp->s3,s->GetName());
    }
    for(pos=m_event.GetHeadPosition();pos!=NULL;)
    {
        CEventx* et=m_event.GetNext(pos);
        if(name.CompareNoCase(et->s1)==0 &&
        strcmp(s->cname,et->cname)==0)
        strcpy(et->s1,s->GetName());
        if(name.CompareNoCase(et->s2)==0 &&
        strcmp(et->cname,s->cname)==0)
        strcpy(et->s2,s->GetName());
    }
    SetModifiedFlag(TRUE);
}
else AfxMessageBox("Name Is Duplicated");
}
Invalidate();
flag2=-1;
flag1=-1;
flag3=FALSE;
break;

case SEVENT:
    CSevent* set;
    if(mod.DoModal()==IDOK)
    { set=CheckSevent(mod.m_stname);
    if(set==NULL)
        {
            se->SetName(mod.m_stname);
            e=CheckEname(se->e1);
            SetModifiedFlag(TRUE);
        }
        else AfxMessageBox("Name Is Duplicated");
        SetModifiedFlag(TRUE);
    }
    Invalidate();
    flag2=-1;
    flag1=-1;
    flag3=FALSE;
    break;
default:AfxMessageBox("Invalid flag2 value");
break;
}
}

```


Function for deletion

```
void newview::OnDelete()
{
    set=0;
    switch(flag2)
    {
        case STATE:delestate(s);
            SetModifiedFlag(TRUE);
            break;
        case EVENT:deletetrans(e);
            SetModifiedFlag(TRUE);
            break;
        case SEVENT:deleteevent(se);
            SetModifiedFlag(TRUE);
            break;
        case CLASS:deleteclass(c);
            SetModifiedFlag(TRUE);
            break;
        default:AfxMessageBox("Wrong flag2 value");
    }
    flag2=-1;flag1=-1;
    flag3=FALSE;OnDeselect();
}

void insert(CState* state)
{
    int t=0;
    POSITION pos;
    for(pos=m_st.GetHeadPosition();pos!=NULL;)
    {
        temp=m_st.GetNext(pos);
        if(strcmp(state->GetName(),temp->GetName())==0
            && strcmp(state->cname,temp->cname)==0 )
            t=1;
    }
    if(t==0)
        m_st.AddTail(state);
}

void insertevent(CSevent* sevent)
{
    int t=0;
    POSITION pos;
    CSevent* set;
    for(pos=m_sevt.GetHeadPosition();pos!=NULL;)
    {
        set=m_sevt.GetNext(pos);
        if(strcmp(sevent->GetName(),set->GetName())==0)
            t=1;
    }
}
```

```

    }
    if(t==0)
    {m_sevt.AddTail(sevent);
    CEventx* et;
    et=CheckEname(sevent->e1);
    inserttrans(et);
    }
}

```

```

void inserttrans(CEventx* event)

```

```

{
    int t=0;
    POSITION pos;
    for(pos=m_evt.GetHeadPosition();pos!=NULL;)
    {
    CEventx* et1=m_evt.GetNext(pos);
    if(event->GetName()==et1->GetName())
    t=1;
    }
    if(t==0)
    m_evt.AddTail(event);
}

```

```

void decrementevent(CSevent* set)

```

```

{
    CState* st;
    st=CheckSname(set->s1,set->start);
    CRect r;
    CPoint pt1,pt2,pt3;
    if(st!=NULL)
    { r=st->GetRect();
    pt1=r.TopLeft();
    pt3=r.CenterPoint();
    pt2=r.BottomRight();
    if((set->start.x >= pt1.x) && (set->start.x <= pt3.x-10))
    {
        if(set->start.y==pt1.y)
        (st->al)--;
        if(set->start.y==pt2.y)
        (st->bl)--;
    }
    if((set->start.x >= pt3.x+10) && (set->start.x <= pt2.x))
    {
        if(set->start.y==pt1.y)
        (st->ar)--;
        if(set->start.y==pt2.y)
        (st->br)--;
    }
    if((set->start.y > pt1.y) && (set->start.y < pt2.y))
    {
        if(set->start.x==pt2.x)

```

```

        (st->r)--;
        if(set->start.x==pt1.x)
            (st->l)--;
    }
}

void delstate(CState* state)
{
    CState* st;
    CSevent* set;
    CEventx* et;
    int t,i,j;
    POSITION pos;
    CString name;
    name=state->GetName();
    for(pos=m_sevent.GetHeadPosition();pos!=NULL;)
    { t=0;i=j=0;
    set=m_sevent.GetNext(pos);
    et=CheckEname(set->e1);
    st=CheckSname(set->s1,set->start);
    if(strcmp(name,set->s1)==0 && strcmp(state->cname,st->cname)==0)
        t=1;
    if(strcmp(name,set->s2)==0 && strcmp(state->cname.et->cname)==0)
        t=2;
    if(strcmp(name,set->s3)==0 && strcmp(state->cname.et->cname)==0)
        t=3;
    switch(t)
    {
    case 1: insertevent(set);
            if(et->dir==1)
            { st=CheckSname(set->s3,set->end);
              insert(st);
              st1=CheckSname(set->s2,set->end);
              CClass* cl;
              cl=CheckCname(st1->cname);
              if(strcmp(st1->GetName(),cl->st_state)==0)
                  break;
              POSITION pos1;
              CString name1=st1->GetName();
              for(pos1=m_sevent.GetHeadPosition();pos1!=NULL;)
              {
                  se=m_sevent.GetNext(pos1);
                  CEventx* et1=CheckEname(se->e1);
                  st=CheckSname(set->s1,set->start);
                  if((strcmp(se->s1,name1)==0 &&
                  strcmp(st1->cname,st->cname)==0)
                  ||(strcmp(se->s2,name1)==0 &&
                  strcmp(et1->cname,st1->cname)==0)
                  ||(strcmp(et1->cname,st1->cname)==0 &&
                  strcmp(se->s3,name1)==0 ))
                  { POSITION pos2;

```

```

        i++;
        for(pos2=m_sevt.GetHeadPosition();pos2!=NULL;)
        {
            CSevent* se1;
            se1=m_sevt.GetNext(pos2);
            if(strcmp(se1->GetName(),se->GetName())==0)
            {
                j++;
                break;
            }
        }
        }//end of if
    }//end of pos1
    if(i==j)
        insert(st1);
}
break;
case 2: insertevent(set);
        if(et->dir==1)
        {
            st=CheckSname(set->s3,set->end);
            insert(st);
        }
        break;
case 3: insertevent(set);
        break;
} //end of switch
} //end of for

for(pos=m_event.GetHeadPosition();pos!=NULL;)
{
    t=0;
    CEventx *et;
    et=m_event.GetNext(pos);
    if(strcmp(name,et->s1)==0 && strcmp(state->cname,et->cname)==0)
        t=1;
    if(strcmp(name,et->s2)==0 && strcmp(state->cname,et->cname)==0)
        t=2;
    switch(t)
    {
        case 1: inserttrans(et);
                if(et->dir==1)
                {
                    st=CheckSname(et->s2,et->end);
                    insert(st);
                }
                break;
        case 2: inserttrans(et);
                break;
    } //end of switch
} //end of for

```

```

}

int checkstate(CState* state)
{
    POSITION pos1;
    CState* st;
    int i=0;
    for(pos1=m_st.GetHeadPosition();pos1!=NULL;)
    {
        st=m_st.GetNext(pos1);
        if(strcmp(st->GetName(),state->GetName())==0 &&
        strcmp(state->cname,st->cname)==0 )
        {i=1; break; }
    }
    return i;
}

```

```

int checksevent(CSevent* sevent)
{
    POSITION pos1;
    CSevent* set;
    int i=0;
    for(pos1=m_sevt.GetHeadPosition();pos1!=NULL;)
    {
        set=m_sevt.GetNext(pos1);
        if(strcmp(set->GetName(),sevent->GetName())==0)
        { i=1; break;}
    }
    return i;
}

```

```

int checkevent(CEventx* event)
{
    POSITION pos1;
    CEventx* et;
    int i=0;
    for(pos1=m_evt.GetHeadPosition();pos1!=NULL;)
    {
        et=m_evt.GetNext(pos1);
        if(et->GetName()==event->GetName())
        { i=1; break;}
    }
    return i;
}

```

```

int checkstevent(CState* s1,CState* s2)
{
    int i=0;
    CEventx* et1;
    et1=CheckStevent(s1,s2);
    if(et1!=NULL)

```

```

    { i++;
      et1=CheckStevent(s2,s1);
      if(et1!=NULL)
        i++;
    }
    else
    {
      et1=CheckStevent(s2,s1);
      if(et1!=NULL)
        i++;
    }
    return i;
}

void deleteevent(CSevent* sevent)
{
    POSITION pos;
    CSevent* setemp;
    CPen *rpen,*oldpen;
    CClientDC d(this);
    CDC *pdc=(CDC *)&d;
    OnPrepareDC(pdc);
    insertevent(sevent);
    int t;
    t=checkstevent(st2,st3);
    deldlg dd;
    rpen=new CPen(PS_SOLID,1,RGB(0,0,0));
    oldpen=pdc->SelectObject(rpen);
    st1->Draw(pdc);
    st2->Draw(pdc);
    st3->Draw(pdc);
    rpen=pdc->SelectObject(oldpen);
    delete rpen;
    switch(t)
    {
    case 0: break;
    case 1: CEventx* et;
           et=CheckEname(sevent->e1);
           if(et->dir==1)
             insert(st3);
           break;
    case 2: et=CheckEname(sevent->e1);
           if(et->dir==1)
             insert(st3);
           break;
    }
    //end of switch
    if(!m_st.IsEmpty())
    {
    for(pos=m_st.GetHeadPosition();pos!=NULL;m_st.GetNext(pos))
    {
    temp=m_st.GetAt(pos);

```

```

rpen=new CPen(PS_SOLID,1,RGB(255.0,0));
oldpen=pdc->SelectObject(rpen);
temp->Draw(pdc);
rpen=pdc->SelectObject(oldpen);
delstate(temp);
delete rpen;
}
}
if(!m_sevt.IsEmpty())
{
for(pos=m_sevt.GetHeadPosition();pos!=NULL;)
{ setemp=m_sevt.GetNext(pos);
rpen=new CPen(PS_SOLID,1,RGB(255.0,0));
oldpen=pdc->SelectObject(rpen);
setemp->Draw(pdc);
rpen=pdc->SelectObject(oldpen);
delete rpen;
}
}
if(!m_evt.IsEmpty())
{
for(pos=m_evt.GetHeadPosition();pos!=NULL;)
{
rpen=new CPen(PS_SOLID,1,RGB(255.0,0));
oldpen=pdc->SelectObject(rpen);
CEventx* et=m_evt.GetNext(pos);
et->Draw(pdc);
rpen=pdc->SelectObject(oldpen);
delete rpen;
}
}
if(dd.DoModal()==IDOK)
{ POSITION pos1;
for(pos1=pos=m_sevent.GetHeadPosition();pos!=NULL;)
{
pos1=pos;
setemp=m_sevent.GetAt(pos1);
m_sevent.GetNext(pos);
if(checksevent(setemp)==1)
{ decrementevent(setemp);
m_sevent.RemoveAt(pos1);
}
}
}
}
}
for(pos1=pos=m_state.GetHeadPosition();pos!=NULL;)
{
pos1=pos;
templ=m_state.GetAt(pos1);
m_state.GetNext(pos);
if(checkstate(templ)==1)
m_state.RemoveAt(pos1);
}
}
}

```

```

for(pos1=pos=m_event.GetHeadPosition();pos!=NULL;)
{
pos1=pos;
CEventx* et=m_event.GetAt(pos1);
m_event.GetNext(pos);
if(checkevent(et)==1)
m_event.RemoveAt(pos1);
}
} //end of if
m_st.RemoveAll();
m_sevt.RemoveAll();
m_evt.RemoveAll();
Invalidate();
flag1=-1;flag2=-1;
flag3=FALSE;
}

void deletetrans(CEventx* et)
{
POSITION pos;
CPen *rpen,*oldpen;
CClientDC d(this);
CDC *pdc=(CDC *)&d;
CSevent* setemp;
OnPrepareDC(pdc);
inserttrans(et);
st1=CheckSname(et->s1,et->start);
st2=CheckSname(et->s2,et->end);
int t;
t=checkstevent(st1,st2);
deldlg dd;
rpen=new CPen(PS_SOLID,1,RGB(0,0,0));
oldpen=pdc->SelectObject(rpen);
st1->Draw(pdc);
st2->Draw(pdc);
rpen=pdc->SelectObject(oldpen);
delete rpen;
switch(t)
{
case 0: break;
case 1: if(et->dir==1)
insert(st2);
break;
case 2: if(et->dir==1)
insert(st2);
break;
} //end of switch
if(!m_st.IsEmpty())
{
for(pos=m_st.GetHeadPosition();pos!=NULL;m_st.GetNext(pos))
}
}

```



```

        temp=m_st.GetAt(pos);
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=pdc->SelectObject(rpen);
        temp->Draw(pdc);
        rpen=pdc->SelectObject(oldpen);
        delstate(temp);
        delete rpen;
    }
}
if(!m_sevt.IsEmpty())
{
    for(pos=m_sevt.GetHeadPosition();pos!=NULL;)
    {
        setemp=m_sevt.GetNext(pos);
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=pdc->SelectObject(rpen);
        setemp->Draw(pdc);
        rpen=pdc->SelectObject(oldpen);
        delete rpen;
    }
}

if(!m_evt.IsEmpty())
{
    for(pos=m_evt.GetHeadPosition();pos!=NULL;)
    {
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=pdc->SelectObject(rpen);
        CEventx* et=m_evt.GetNext(pos);
        et->Draw(pdc);
        rpen=pdc->SelectObject(oldpen);
        delete rpen;
    }
}

if(dd.DoModal()==IDOK)
{
    POSITION pos1;
    for(pos1=pos=m_sevent.GetHeadPosition();pos!=NULL;)
    {
        pos1=pos;
        setemp=m_sevent.GetAt(pos1);
        m_sevent.GetNext(pos);
        if(checksevent(setemp)==1)
        {
            decrementevent(setemp);
            m_sevent.RemoveAt(pos1);
        }
    }
    //end of pos1
    for(pos1=pos=m_state.GetHeadPosition();pos!=NULL;)
    {
        pos1=pos;
        temp1=m_state.GetAt(pos1);

```

```

        m_state.GetNext(pos);
        if(checkstate(temp1)==1)
            m_state.RemoveAt(pos1);
    }
    for(pos1=pos=m_event.GetHeadPosition();pos!=NULL;)
    {
        pos1=pos;
        CEventx* et=m_event.GetAt(pos1);
        m_event.GetNext(pos);
        if(checkevent(et)==1)
            m_event.RemoveAt(pos1);
    }
    //end of if
    m_st.RemoveAll();
    m_sevt.RemoveAll();
    m_evt.RemoveAll();
    Invalidate();
    flag1=-1;flag2=-1;
    flag3=FALSE;
}

```

```

void deletestate(CState *state)
{
    POSITION pos;
    CSevent* setemp;
    CPen *rpen,*oldpen;
    CClientDC d(this);
    CDC *pdc=(CDC *)&d;
    OnPrepareDC(pdc);
    insert(state);
    if(!m_st.IsEmpty())
    {
        for(pos=m_st.GetHeadPosition();pos!=NULL;m_st.GetNext(pos))
        {
            temp=m_st.GetAt(pos);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            oldpen=pdc->SelectObject(rpen);
            temp->Draw(pdc);
            rpen=pdc->SelectObject(oldpen);
            delstate(temp);
            delete rpen;
        }
    }
    if(!m_sevt.IsEmpty())
    {
        for(pos=m_sevt.GetHeadPosition();pos!=NULL;)
        {
            setemp=m_sevt.GetNext(pos);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            oldpen=pdc->SelectObject(rpen);
            setemp->Draw(pdc);
        }
    }
}

```

```

        rpen=cdc->SelectObject(oldpen);
        delete rpen;
    }
}
if(!m_evt.IsEmpty())
{
    for(pos=m_evt.GetHeadPosition();pos!=NULL;)
    {
        CEventx* et=m_evt.GetNext(pos);
        rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
        oldpen=cdc->SelectObject(rpen);
        et->Draw(cdc);
        rpen=cdc->SelectObject(oldpen);
        delete rpen;
    }
}
deldlg dd;
if(dd.DoModal()==IDOK)
{ POSITION pos1;
  if(!m_st.IsEmpty())
  {
      for(pos1=pos=m_state.GetHeadPosition();pos!=NULL;)
      {
          pos1=pos;
          temp1=m_state.GetAt(pos1);
          m_state.GetNext(pos);
          if(checkstate(temp1)==1)
              m_state.RemoveAt(pos1);
      }
  }
  //end of m_st
  if(!m_sevt.IsEmpty())
  {
      for(pos1=pos=m_sevent.GetHeadPosition();pos!=NULL;)
      {
          pos1=pos;
          setemp=m_sevent.GetAt(pos1);
          m_sevent.GetNext(pos);
          if(checksevent(setemp)==1)
              { decrementevent(setemp);
                m_sevent.RemoveAt(pos1);
              }
      }
  }
  //end of pos1
  //end of m_sevt
  if(!m_evt.IsEmpty())
  {
      for(pos1=pos=m_event.GetHeadPosition();pos!=NULL;)
      {
          pos1=pos;
          CEventx* et=m_event.GetAt(pos1);
          m_event.GetNext(pos);
          if(checkevent(et)==1)

```

```

        m_event.RemoveAt(pos1);
        } //end of pos1
    } //end of m_evt
} //end of if
Invalidate();
m_st.RemoveAll();
m_sevt.RemoveAll();
m_evt.RemoveAll();
flag1=-1;flag2=-1;
flag3=FALSE;
}

void deleteclass(CClass* cclass)
{
    POSITION pos;
    CSevent* setemp;
    CPen *rpen,*oldpen;
    CClientDC d(this);
    CDC *pdc=(CDC *)&d;
    OnPrepareDC(pdc);
    for(pos=m_state.GetHeadPosition();pos!=NULL;)
    {
        temp=m_state.GetNext(pos);
        if(strcmp(cclass->GetName(),temp->cname)==0)
            insert(temp);
    }
    if(!m_st.IsEmpty())
    { for(pos=m_st.GetHeadPosition();pos!=NULL;m_st.GetNext(pos))
        { temp=m_st.GetAt(pos);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            oldpen=pdc->SelectObject(rpen);
            temp->Draw(pdc);
            rpen=pdc->SelectObject(oldpen);
            delete rpen;
            delstate(temp);
        }
    }
    if(!m_sevt.IsEmpty())
    { for(pos=m_sevt.GetHeadPosition();pos!=NULL;)
        { setemp=m_sevt.GetNext(pos);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));
            oldpen=pdc->SelectObject(rpen);
            setemp->Draw(pdc);
            rpen=pdc->SelectObject(oldpen);
            delete rpen;
        }
    }
    if(!m_evt.IsEmpty())
    { for(pos=m_evt.GetHeadPosition();pos!=NULL;)
        { CEventx* et=m_evt.GetNext(pos);
            rpen=new CPen(PS_SOLID,1,RGB(255,0,0));

```

```

    oldpen=cdc->SelectObject(rpen);
    et->Draw(pdc);
    rpen=cdc->SelectObject(oldpen);
    delete rpen;
}
}

deldlg dd;
if(dd.DoModal()==IDOK)
{ POSITION pos1;
  if(!m_state.IsEmpty())
  {
    for(pos1=pos=m_state.GetHeadPosition();pos!=NULL;)
    {
      pos1=pos;
      temp1=m_state.GetAt(pos1);
      m_state.GetNext(pos);
      if(checkstate(temp1)==1)
        m_state.RemoveAt(pos1);
    }
  }
  if(!m_sevent.IsEmpty())
  {
    for(pos1=pos=m_sevent.GetHeadPosition();pos!=NULL;)
    {
      pos1=pos;
      setemp=m_sevent.GetAt(pos1);
      m_sevent.GetNext(pos);
      if(checksevent(setemp)==1)
      { decrementevent(setemp);
        m_sevent.RemoveAt(pos1);
      }
    }
  } //end of pos1
} //end of if
if(!m_event.IsEmpty())
{
  for(pos1=pos=m_event.GetHeadPosition();pos!=NULL;)
  {
    pos1=pos;
    CEventx* et=m_event.GetAt(pos1);
    m_event.GetNext(pos);
    if(checkevent(et)==1)
      m_event.RemoveAt(pos1);
  } //end of pos1
}
} //end of if
Invalidate();
m_st.RemoveAll();
m_sevt.RemoveAll();
m_cvt.RemoveAll();
flag1=-1;flag2=-1;

```

```

    flag3=FALSE;
}

```

```

void OnGeneratecode()
{
    set=1;
    POSITION pos;
    CClientDC d(this);
    CDC *pdc=(CDC *)&d;
    OnPrepareDC(pdc);
    Invalidate();
    char name1[15],name2[15],name3[15],name4[15];
    CFile fpcls,fpatb,fpInk;
    strcpy(name1,"class");
    strcpy(name2,"{");
    strcpy(name3,"}");
    strcpy(name4,";");
    if(!m_event1.IsEmpty())
        m_event1.RemoveAll();
    if(!sstate.IsEmpty())
        sstate.RemoveAll();
    CSevent* set;
    Event* ett;
    startstate* stt;
    i=0;
    for(pos=pdoc->m_class.GetHeadPosition();pos!=NULL;)
    { CClass* cl=pdoc->m_class.GetNext(pos);
      stt=new startstate;
      stt->cname=cl->GetName();
      stt->starts=cl->st_state;
      stt->tag=++i;
      sstate.AddTail(stt);
    }
    if(!pdoc->m_sevent.IsEmpty())
    { for(pos=pdoc->m_sevent.GetHeadPosition();pos!=NULL;)
      { set=pdoc->m_sevent.GetNext(pos);
        ett=new Event();
        ett->s1=set->s1;
        ett->s2=set->s2;
        ett->s3=set->s3;
        ett->event=set->GetName();
        st1=pdoc->CheckSname(set->s1,set->start);
        ett->c1=st1->cname;
        st1=pdoc->CheckSname(set->s2,set->end);
        ett->c2=st1->cname;
        m_event1.AddTail(ett);
      }
    }
    i=10,j=10;
}

```

```

classsave rtemp;
atbsave atemp;
attrib *amove,*aadd;
classdiagramar *cmove,*clas=NULL;
fpcls.Open("test.cls",CFile::modeRead);
fpInk.Open("test.lnk",CFile::modeRead);
fpatb.Open("test.atb",CFile::modeRead);
if(fpcls.GetLength()==0)
    MessageBox("File is empty","Read Record...");
else
{
    fpcls.SeekToBegin();
    while(fpcls.Read(&rtemp,sizeof(classsave))!=0)
        {
            cmove=new classdiagramar;
            cmove->name=rtemp.name;
            cmove->cno=rtemp.cno;
            cmove->alink=NULL;
            cmove->mlink=NULL;
            if(fpatb.GetLength()==0)
                MessageBox("Methods/attributes file empty"."Read Record...");
            fpatb.SeekToBegin();
            while(fpatb.Read(&atemp,sizeof(atemp))!=0)
                {
                    if(atemp.cno == cmove->cno)
                        {
                            aadd=new attrib;
                            aadd->name=atemp.name;
                            aadd->next=NULL;
                            if(atemp.type==1)
                                {
                                    if(cmove->alink==NULL)
                                        cmove->alink=aadd;
                                    else
                                        {
                                            amove=cmove->alink;
                                            while(amove->next!=NULL)
                                                amove=amove->next;
                                            amove->next=aadd;
                                        }
                                }
                        }
                    else
                        {
                            if(cmove->mlink==NULL)
                                cmove->mlink=aadd;
                            else
                                {
                                    amove=cmove->mlink;
                                    while(amove->next!=NULL)
                                        amove=amove->next;
                                    amove->next=aadd;
                                }
                        }
                }
        }
}

```

```

        }
        // same class object (if condition)
    } // while loop reading attrib
    cmove->next=clas;
    clas=cmove;

    // while reading class info
    fpInk.Close();
    fpcls.Close();
    fpatb.Close();
    cmove=clas;
    CString attrib1,attrib2;
    attrib1="attribute";
    attrib2="methods";
    while(cmove != NULL)
    {
        i=10;
        pdc->TextOut(i,j,name1);
        i=strlen(name1)+20;
        pdc->TextOut(i,j,cmove->name);
        j+=20;
        pdc->TextOut(i,j,name2);
        i+=strlen(name2)+20;
        pdc->TextOut(i,j,attrib1);
        j+=20;
        while(cmove->alink != NULL)
        { pdc->TextOut(i,j,cmove->alink->name);
          j+=20;
          cmove->alink=cmove->alink->next;
        }
        pdc->TextOut(i,j,attrib2);
        j+=20;
        while(cmove->mlink != NULL)
        { pdc->TextOut(i,j,cmove->mlink->name);
          j+=20;
          cmove->mlink=cmove->mlink->next;
        }
        pdc->TextOut(i,j,cmove->mlink->name);
        i+=20;j+=20;
        pdc->TextOut(i,j,name3);
        j+=20;
        pdc->TextOut(i,j,name4);
        j+=20;
        cmove=cmove->next;
    }
}

```

```

CWinThread *t1=AfxBeginThread(thread1.GetSafeHwnd());
CWinThread *t2=AfxBeginThread(thread2.GetSafeHwnd());
CWinThread *t3=AfxBeginThread(thread3.GetSafeHwnd());

```



```

CWinThread *t4=AfxBeginThread(thread4,GetSafeHwnd());
CWinThread *t5=AfxBeginThread(thread5,GetSafeHwnd());

```

```

}
void newview::OnSavecode()
{
    set=0;
    CFile file;
    POSITION pos;
    CString filename;
    CClass *ctemp2;
    CState *stemp2,*stemp1;
    CSevent *setemp2;
    stemp1=NULL;
    codedlg namdlg;
    if(namdlg.DoModal()==IDCANCEL)
    return;
    filename=namdlg.m_filename;
    AfxMessageBox(filename);
    CFileException e;
    if( !file.Open( filename, CFile::modeCreate | CFile::modeWrite. &e ) )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
    for(pos=m_class.GetHeadPosition();pos!=NULL;)
    {
        ctemp2=m_class.GetNext(pos);
        file.Write(ctemp2,sizeof(CClass));
    }
    for(pos=m_state.GetHeadPosition();pos!=NULL;)
    {
        stemp2=m_state.GetNext(pos);
        file.Write(stemp2,sizeof(CState));
    }
    for(pos=m_sevent.GetHeadPosition();pos!=NULL;)
    {
        setemp2=m_sevent.GetNext(pos);
        file.Write(setemp2,sizeof(CSevent));
    }
    file.Close();
}
CTypedPtrList<CObList.Event*>m_event1;
CTypedPtrList<CObList.startstate*>sstate;

CString getstate(int i)
{
    startstate* st;
    POSITION pos;

```

```

for(pos=sstate.GetHeadPosition();pos!=NULL;)
{ st=sstate.GetNext(pos);
  if(st->tag == i)
    return st->starts;
}
return "";
}

CString getclass(int i)
{
  startstate* st;
  POSITION pos;
  for(pos=sstate.GetHeadPosition();pos!=NULL;)
  { st=sstate.GetNext(pos);
    if(st->tag == i)
      return st->cname;
  }
  return "";
}

void enableevent(CString sname,CString cname)
{
  POSITION pos;
  Event* et;
  for(pos=m_eventl.GetHeadPosition();pos!=NULL;)
  {
    et=m_eventl.GetNext(pos);
    if(strcmp(et->s1,sname)==0 && strcmp(et->c1,cname)==0 )
    {
      CString tempdis=et->s1;
      et->e.SetEvent();
    }
  }
}

Event* waitforevent(CString sname,CString cname)
{
  POSITION pos;
  Event* et;
  for(pos=m_eventl.GetHeadPosition();pos!=NULL;)
  { et=m_eventl.GetNext(pos);
    if(strcmp(et->s2,sname)==0
      && strcmp(et->c2,cname)==0)
      return et;
  }
  return NULL;
}

UINT thread1(LPVOID pparam)
{ CString sname;
  sname=getstate(1);

```

```

while(1)
{
CString cname=getclass(1);
enableevent(sname,cname);
Event* et;
et=waitforevent(sname.cname);

if(et==NULL) break;
::WaitForSingleObject(et->e.m_hObject,INFINITE);
sname=et->s3;
}
return 0;
}

```

```

UINT thread2(LPVOID pparam)
{ CString sname;
sname=getstate(2);
while(1)
{
CString cname=getclass(2);
enableevent(sname,cname);
Event* et;
et=waitforevent(sname,cname);
if(et==NULL) break;
::WaitForSingleObject(et->e.m_hObject,INFINITE);
sname=et->s3;
}
return 0;
}

```

```

UINT thread3(LPVOID pparam)
{ CString sname;
sname=getstate(3);
while(1)
{
CString cname=getclass(3);
enableevent(sname,cname);
Event* et;
et=waitforevent(sname,cname);
if(et==NULL) break;
::WaitForSingleObject(et->e.m_hObject,INFINITE);
sname=et->s3;
}
return 0;
}

```

```

UINT thread4(LPVOID pparam)
{ CString sname;
sname=getstate(4);
while(1)
{

```

```

CString cname=getclass(4);
enableevent(sname,cname);
Event* et;
et=waitforevent(sname,cname);
if(et==NULL) break;
::WaitForSingleObject(et->c.m_hObject,INFINITE);
sname=et->s3;
}
return 0;
}

```

```

UINT thread5(LPVOID pparam)
{ CString sname;
sname=getstate(5);
while(1)
{
CString cname=getclass(5);
enableevent(sname,cname);
Event* et;
et=waitforevent(sname,cname);
if(et==NULL) break;
::WaitForSingleObject(et->e.m_hObject,INFINITE);
sname=et->s3;
}
return 0;
}

```

References

1. Booch,Grady.
Object Oriented analysis and design with applications. Grady Booch 2nd edition.
2. Rumbaugh,J.,Blaha,M., Premerlani,W., Eddy,F.,and Lorenson,W.1997.
Object Oriented Modeling and Design, Prentice Hall.
3. Bjarne Stroustrup.
The C++ Programming Language.1997.
Addison-Wesley 3rd edition.
4. David.J.Kruglinski.
Inside VC++. 4th Edition.
Microsoft Press.
5. Yashavant.P.Kanetkar.
VC++ Programming.1998, 1st Edition.
BPB publications.
6. Roger.S.Pressman
Software Engineering, A Practitioners approach.1997,
4th Edition. McGraw Hill International Edition.