A Direct Manipulation Object-Oriented Environment to Support Methodology-Independent CASE Tools

A thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at Massey University

Bei Zhong

1996

Abstract

The aim of the thesis is research into application of direct-manipulable OO graphical environments to the development of methodology-independent CASE tools. In this thesis, a Methodology-Independent Graphical OO CASE Environment (MIGOCE) is proposed.

MIGOCE consists of three parts: OO Notation Workshop, OO Notation Repository and Universal OO Diagramming Tool. OO Notation Workshop is an OO graphical editor which is used to design existing and new notations; OO Notation Repository is a notation database that stores different notations designed by the notation workshop; Universal OO Diagramming Tool is an upper-CASE graphical environment, by which a user can draw arbitrary OO diagrams of different methodologies. The MIGOCE database management system provides OO notation sets management, OOA/OOD diagrams management and OO repository management for data integrity and sharing.

MIGOCE has three outstanding characteristics: Methodology-independence, Directly-manipulable graphical environment and Easily-expanded program structure. MIGOCE is completely methodology-independent. It not only supports existing OO methodologies, but also supports users' own notation designs. It provides support for mixing, updating existing methodologies or defining new ones. It typically allows the user to switch quickly different OO notation sets supported by corresponding methodologies for designing diagrams. Direct manipulation interfaces of MIGOCE enable it more flexible and distinctive. The user can easily add, delete, edit or show notation shapes, and get the system feedback very quick on the screen. The MIGOCE system itself is programmed using object-oriented programming language — C++. Its program structure enable the functions of itself easy to be modified and expanded.

Although MIGOCE is a prototype, it provides a new way to develop the real methodology-independent OO CASE environment. So far, the way and style taken by MIGOCE have not been found in OO CASE literatures. This system gives a complete possibility of implementing a methodology-independent OO CASE tool and shows distinct effectiveness of such a tool in practice.

Acknowledgments

I would like to express my grateful thanks to my first supervisor Dr. Daniela Mehandjiska-Stavreva for her supervision, valuable guidance, and encouragement throughout my study. I deeply appreciate her patience in reading and constructively criticising the manuscript of this thesis.

Special thanks go to my second supervisor Dr. Chris Phillips for his helpful advises on my work, and his patience in modifying my manuscript.

My deep appreciation is also extended to all staff and postgraduate students in the Department of Computer Science for their assistance and friendship.

Lastly, I would especially like to express my gratitude to my family for their continued support and encouragement.

Table of Contents

Chapter 1

Introduction

1.1. Object-Oriented Development

Over the past ten years, object-oriented (OO) technology has indeed moved into the mainstream of industrial-strength software development. It is a software strategy that differs from the functional approach, because it is based on information hiding and views a software system as a set of interacting objects, with their own private state, rather than as a set of functions.

OO technology is the most important software evolution of the 1990s. It is destined to change not only the way we build software, but the way software intercommunicates over worldwide multi-vendor networks. OO modeling will change the way we design business processes and the way we think about enterprises [Martin, 1993].

Object-oriented software development is mainly characterised by four features: information hiding (encapsulation), data abstraction, dynamic binding and inheritance. Data abstraction provides a clear separation between the specification and the implementation of objects. Inheritance allows the application developer to define new application-specific objects based on existing ones. This improves and reinforces reusability. Therefore, object-oriented development of software is one of the best currently available techniques to keep the cost of software development down.

At present, object-oriented analysis and design (OOA & OOD) has aroused exceptional interest both in industrial and in academic areas. Particularly, the object-oriented approach contributes to the production of quality software [Sommerville, 96]. The acceptance of OOA and OOD as effective software strategies has led to the development of OO methodologies. OO methodologies cover the requirements gathering, analysis and design phases of the software development lifecycle.

Three generations of Object-Oriented methodologies have been defined over the past decade. A multiplicity of Object Oriented analysis (OOA), Object Oriented design

(OOD), Object Oriented analysis & design (OOAD), Object Oriented software development (OOSD) methodologies, etc., have been developed over the last 10 years (over 50 different OO methodologies at this time). Some of the well known and widely utilised methodologies are:

- Wirfs-Brock's Responsibility Driven Design [Wirfs-Brook et al. 1990]
- Object Oriented Design [Coad, P. and Yourdon, E. 1990a, 2nd Ed 1991b]
- Object Oriented Analysis [Coad et al. 1991b, Shlaer et al., 1991]
- Object Oriented Analysis and Design [Shlaer et al. 1988, Martin, J. 1993, Booch 1994b, Martin and Odell, 1995]
- Rumbaugh's OMT (Object Modeling Technique) [Rumbaugh et al., 1991a]
- Object Oriented Modeling and Design - Jacobson's Objectory (the Object Factory for Software Development [Jacobson et al. 1992]
- Object Oriented Software Development, [Coleman et al. 1994, Colbert et al. 1994]
- MOSES (Methodology for Object Oriented Software Engineering Systems, [Henderson-Sellers et al 1994a].

First generation methodologies including Wirfs-Brock's responsibility driven design, Booch's OOD [Booch, 91a], Rumbaugh's OMT , Coad and Yourdon OOA/OOD, Shlaer and Mellor OOA , Jacobson's Objectory (Object Factory for Software Development), etc. have been developed in the late 80's and early 90's.

The first generation techniques were applied and evaluated resulting in second generation methodologies such as, Booch's OOA/OOD [Booch 1994b], Graham's SOMA (Semantic Object-Oriented Modeling Approach) [Graham, 94b], Henderson-Sellers's MOSES (Methodology for Object Oriented Software Engineering Systems) [Henderson-Sellers and J. Edwards, 1994a], Martin and Odell's Advanced Object Modeling [Martin and Odell, 1995], Coleman's Fusion methodology and Rumbaugh's second generation OMT [Rumbaugh, 94b, Rumbaugh, 95c].

The plethora of OOMs (OO Methodologies) and their current rapid evolution have made it difficult for a company wishing to adopt the object technology to choose an appropriate methodology. Some of the problems are:

- the large number of existing OOMs and dialects of these methodologies (e.g. Wirfs-Brock's responsibility driven design and its dialects adopted by Budd [Budd, 91] and Booch [Booch 1994b];

- most OOMs are still undergoing modifications such as the Unified method [Booch & Rumbaugh, 95], Unified Modeling Language [Firesmith et al., 96] and Rumbaugh's second generation OMT [Rumbaugh, 94b, Rumbaugh, 95c]. Some of the new models are a major departure from the originals. In some cases the entire approach is different. These ongoing modifications are natural as 'methodologies must evolve' [Brough, 92] and 'any method must grow or die'[Rumbaugh, 94b].

- new methodologies which are revisions of existing ones have also been proposed such as SOMA, a revision of Coad and Yourdon's methodology, FUSION, an extension to OMT, etc.

To address the diversity of first and second generation methodologies two third generation methodologies are under development: the 'Unified method', a convergent methodology comprising Booch, OMT and Objectory; and the OPEN methodology [Henderson-Seller et al, 96b]. The later is proposed by Ian Graham, Brian Henderson-Seller and Donald Firesmith, with input from a Consortium of methodology researchers including: Larry Constantine, Meilir Page-Jones, Betrand Meyer, Adele Goldberg, Rebecca Wirfs-Brock, etc. Furthermore in July 1996 new releases of the two 3rd generation Methodologies were announced 'The Unified Modeling Language' and 'The Open Modeling Language'.

In many concepts of object-orientation, these methodologies are similar to each other; but in representation, they have their own distinct notations. OO methodologies are still in constant evolution and their exhaustive study would be a quite difficult task. Criticisms as regards to methodologies were identified as: they require long learning periods, demand willingness from the developers and users, require users to gain considerable knowledge and experience concerning the methods and require the adaptation of new procedures. Sometimes there is an overlap between OOA and OOD methodologies because object-oriented analysis and design cannot be cleanly separated.

The aim of the thesis is to develop a methodology-independent graphical OO CASE environment to support the existing diversity of OO Methodologies and their future evolution.

1.2. Overview of CASE Tools

CASE (Computer-aided Software Engineering) is an emerging technology with the potential of being one of the most significant advancements in the software field. CASE tools have been created to help make software development faster, cheaper and higher quality. A CASE tool is defined as a computer based product aimed at supporting one or more software engineering activities within a software development process. It is proved that people can get benefits of CASE tools, such as speeding up the development process, improving analyst-user communication and integrating the life cycle activities of software development. A CASE environment is a collection of CASE tools and other components together with an integration approach that supports the interactions that occur among the environment components, and between the users of the environment and the environment itself.

CASE tools can be divided into: upper-CASE tools, lower-CASE tools and cross life cycle CASE tools. Upper-CASE tools support business and Information Systems (IS) planning in the scope required in the preliminary phase, and support system analysis and design. Lower CASE tools help to automate the forward engineering part of a software re-engineering cycle, speeding up the whole process and reducing the cost of software re-engineering. Cross life cycle CASE tools support the activities across the system development life cycle [Whitten et al., 94].

The CASE industry grew up in the late 1980s building non-OO tools. As OO techniques became better understood, some CASE vendors added OO concepts to their toolsets. OO-CASE refers to an object-oriented CASE toolset that is used to support the software development adopting OOA and OOD methods. OO CASE tools are more effective, because these tools have the advantages of OO software strategies.

Several generations of CASE tools have emerged in the past. First generation CASE tools addressed mostly form and representation issues of software development methodologies [Sorenson, 1988a]. Initially the focus of CASE tool development was on program support tools such as translators, compilers, assemblers, linkers and loaders. Later the range of support tools began to expand with the development of program editors, debuggers, code analysers etc., collectively referred to as development tools.

Large-scale software development demanded enhanced support of the entire software development process from CASE tool developers [Sumner, 1992]. Assistance was required for the requirements definition, design and implementation phases of the

software development lifecycle, testing, documentation and version control. A distinction was made between those tools that support activities early in the lifecycle of a software project (such as requirements and design support tools) and those tools that are used later in the lifecycle (such as compilers and testing support tools). The former class of tools are referred to as front-end or upper-CASE tools and the later are referred to as back-end or lower-CASE tools.

In general, the first generation CASE tools aided the user in creating system analysis and design diagrams and detailed textual based specifications. They performed consistency, completeness and correctness checking. Some of them provided a primitive form of code generation. Their main disadvantages were: inadequate methodology support, no customisation facilities; lack of support for reverse engineering; inability to integrate different CASE tools used at various stages of software development.

Second generation CASE tools addressed these problems. Integration was achieved by sharing definitions of objects and relationships described in a common dictionary. CASE tools methodology support was improved by the production of tool sets supporting methodology customisation using a meta-systems approach [Brough, 1992; Rossi *et al.*, 1992; Smolander *et al.*, 1991; Sorenson, 1988b]. However second generation CASE tools were still deficient in a number of important areas. They lacked support for defining new methodologies. They did not provide information interchange of analysis and design results expressed in different methodologies. From a usability perspective they did not facilitate navigation of complex structures of data.

In the past few years, research into CASE technology has been concentrated in two main areas. The first one addressed the development of software environments with an open architecture aiming at the integration of independently developed CASE tools [Lang, 1991; Nilsson, 1990; Sorenson, 1988b; Papahristos *et al.*, 1991]. The second area of research addressed the methodology dependence of CASE tools (see Figure 1).

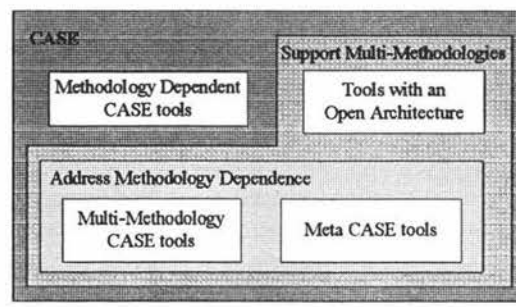


Figure 1. CASE Technology [Mehandjiska-Stavreva et al., 96]

Because of multiplicity of OO methodologies, there are a number of OO CASE tools products which support all kinds of different methodologies. Many of current OO CASE tools suffer from generic problems. The main problem is that these tools lack flexibility. Most of them are method-dependent. They can only support one or two methodologies and cannot meet the needs of different users.

However, flexible, method-independent OO CASE tools are needed because these tools enable the development of software systems more efficient and economical.

1.3. A Methodology-Independent Graphical CASE Environment

In order to overcome the disadvantages of current OO CASE tools, methodology-independent OO CASE tools are needed. These tools are flexible to model different information systems of an enterprise with different methodologies and make use of a repository to store and reuse all the information of the different systems.

In this thesis, an OO upper-CASE tool kit called MIGOCE is proposed. MIGOCE (Methodology-Independent Graphical OO CASE Environment) is an OO graphical design environment that supports all existing OO methodologies. In MIGOCE, users can create different OO notations sets for existing OO methodologies or their own OO notation set according to different requirements. The OO notation sets created in MIGOCE environment can be used by a diagraming tool to support different OO analysis and design methodologies.

MIGOCE implemented under Borland C++ 4.5. The three components of the MIGOCE environment are OO Notation Workshop, OO Notation Repository and Universal OO Diagraming Tool. The main functions of these components are:

- *OO Notation Workshop*: a notation graphical editor which is used to design basic shapes to form different notation sets supported by different OO methodologies, including the notations defined by users.

- *OO Notation Repository*: a notation database that is used to store and manage all notation shapes designed in OO Notation Workshop. MIGOCE provides database management for data integrity and sharing between OO Notation Workshop and Universal OO Diagraming Tool.

- *Universal OO Diagraming Tool*: an OO diagraming environment based on notation sets created by OO Notation Workshop. Under this environment, the user can use different OO notations to draw different OO diagrams supported by arbitrary OO methodologies.

MIGOCE is an upper-CASE tool graphical environment. It is mainly used to analyze and design systems that adopt OO software strategies. The expression of OO concepts relies on displaying OOA or OOD diagrams. Therefore, the graph representation is the key issue of the MIGOCE system. Taking this into account, MIGOCE system is designed to have strong graphical functions. During the design of the MIGOCE interface, domain task analysis is adopted to bridge the gap between the user's conceptual model and the system model.

In the design of the MIGOCE system, OO technique has been taken. The domain model is built that groups together entities and related procedures to form object classes. Because of using OO technique, the functions of the MIGOCE system can be easily modified and enhanced. OO programming language — C++ — provides complete possibility of doing these.

To users, MIGOCE has two outstanding advantages — Methodology-independence and Directly-manipulable graphical environment. These highlights of the system make it more distinguished.

The first, building methodology-independent OO CASE tools aims at overcoming the weakness of existing tools — methodology-dependence. MIGOCE supports all kinds of OO methodologies, including existing methodologies and those defined by users. Particularly, when information interchanges between analysis and design results expressed in different methodologies are required, it is more remarkable. MIGOCE provides support for mixing, updating existing methodologies or defining new ones.

The second, directly-manipulable graphical environment enables MIGOCE more flexible and effective. The interface of MIGOCE emphasizes the interaction with users under the WIMP (Window, Icon, Menus, Pointer) environment. Direct manipulation of it are visualized, quick-response and convenient for users.

1.4. Overview of the Thesis

Chapter 2 reviews of software design approaches — the functional approach and the object-oriented approach, and discusses a few existing OO methods and their advantages and disadvantages.

Chapter 3 states the benefits of CASE tools, and the limitation of current OO CASE tools. In this chapter, an OO upper-CASE environment is proposed. The chapter describes the rationale behind MIGOCE (Methodology-Independent Graphical OO CASE Environment) and the overall architecture of this system.

Chapter 4 describes several task analysis techniques, their applications in the design of Human-Computer Interface (HCI) and the types of interaction in CASE. The chapter presents the user task model of MIGOCE, which is based on the TKS theory.

Chapter 5 presents the static and dynamic object models of OO Notation Workshop and Universal OO Diagraming Tool. All models have been built by using the Booch methodology.

Chapter 6 states the process of building the user interface of OO Notation Workshop, including the characteristics analysis, task analysis and task model based on the TKS theory. This chapter presents the functionality of OO Notation Workshop as well as notation examples created by OO Notation Workshop.

Chapter 7 gives the prototyping of the Universal OO Diagraming Tool, describes its structure, components and functions, as well as interface design.

Chapter 8 presents conclusions — the evaluation of the MIGOCE and states future work and further extensions of MIGOCE.

Chapter 2

Software Development Approaches

Different approaches to software development have evolved in the past. Some of the most widely used are functional decomposition, data modeling and information engineering. This chapter will briefly address some issues of Functional approach and Object-Oriented approach. The well-known OO methodologies are covered including Shlaer and Mellor, Jacobson, Coad and Yourdon, Rumbaugh and Booch.

2.1. Functional Approach

In the functional approach, the system is designed from a functional viewpoint, starting with a high-level view and progressively refining this into a more detailed design. This strategy is exemplified by structured design [Constantine and Yourdon, 79] and step-wise refinement [Wirth, 71; Wirth, 76].

Top-down functional decomposition

The Top-down design method was proposed in conjunction with functional decomposition and it is a valid method where the problem is recursively partitioned into sub-problems until tractable sub-problems are identified. Functional decomposition requires humans to map from the problem domain to functions and sub-functions. Usually, this approach is readily recognized with its step and sub-steps. When a designer is in the process of design, he is often asked with the question like "What do you do to solve the complex problem?" Undoubtedly, the most direct method of solving the problem is to decompose it into a hierarchical series of sub-problems and have an ordered solution of these problems. Here, the process of solving problems is the process of performing tasks.

The general form of the design is hierarchical. This hierarchical structure is a tree structure. Actually, many existing system designs started with Top-down approach.

Data flow diagrams

Data flow diagrams display how input data is transformed to output results through a sequence of functional transformations. They focus on what happens to the data and how the data has been passed between processes. This approach is often described as "structured analysis". With Data flow approach, the analyst maps from the real world into data flows and processes.

Structure charts

Structure charts are graphical means of showing how elements of a data flow diagram can be realized as a hierarchy of program units. These charts can be used as a visual program description with control information defining selection and loops. They are used only to display the static organization of the design.

2.2. Object-Oriented Approach

The history of object-oriented programming starts with the development of the discrete event simulation language Simula, in Norway in 1967, and continues with the development of a language that almost makes a fetish of the notion of an object, Smalltalk, in 1970s. Some other languages which influence the development of OO programming, include Alphard [Wulf et al., 76] and CLU [Liskov et al., 77]. Since then there have been many languages which have been inspired by these developments and have laid claim to the appellation "Object-oriented"[Graham, 94a].

As object-oriented programming began to mature, interest shifted to object-oriented design methodologies and to object-oriented analysis or specification. So far, there are over 50 different methodologies and none of these is entirely complete. These methodologies are still undergoing modifications.

In object-oriented approach, the system is viewed as a collection of objects rather than as functions. During the OO analysis and design, the data flow diagrams in traditional methods are replaced by OO diagrams in which the communication between objects is performed through message passing that causes changes in object state.

2.2.1. Key Concepts

(1) Object: An object is an entity that has a state which is represented as a set of object attributes and a defined set of operations which operate on that state (The internal object representation to the outside world is hidden) [Brumbaugh, 94].

(2) Class: A class is "a set of objects that share a common structure and a common behaviour"[Booch, 94b].

(3) Encapsulation: The data structures and implementation details of an object are hidden from other objects in the system. The only way to access an object's state is to send a message that causes one of the methods to be executed [Brumbaugh, 94].

(4) Message: A message is " an operation that one object performs upon another" [Booch, 94b].

(5) Inheritance: This is " a relationship among classes, wherein one class shares the structure or behaviour defined in one (single inheritance) or more (multiple inheritance) other classes" [Booch, 94b].

(6) Polymorphism: The ability to use the same expression to denote different operations is referred to as polymorphism [Brumbaugh, 94].

An inheritance hierarchy or tree is created when a class inherits attributes and operations from a single super-class. Multiple inheritance means that attributes and operations can be inherited from more than one super-class.

Object-orientation is a technique for system modeling. Here, the word "system" is used with a wide meaning and can be either a dedicated software system or a system in a wider context (for example an integrated software and hardware system or an organization). OO development covers different phases of the software development life cycle, such as analysis, design and programming.

Grady Booch defined OOA(Object-oriented analysis), OOD(Object-oriented design), OOP (Object-oriented programming) as follows [Booch, 94b]:

"Object-oriented analysis (OOA) is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain".

"Object-oriented design (OOD) is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design".

"Object-oriented programming (OOP) is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships".

In fact, OOA, OOD and OOP are three steps which constitute the process of solving a problem. At first, a user, as an analyst, needs to grasp, to understand in depth, the problem domain. In OOA, the analyst may communicate with the potential users and domain experts (people who know the problem area, but not necessarily computer experts), to describe the things they know about the problem. These "things" are candidates for classes and objects. The analyst abstracts objects from the problem domain description.

At the second stage, the designer must figure out *how* to build this functionality defined by the analyst. When performing the design phase, the first step is to get the initial list of candidate classes. Depending on how the analysis is done, this may mean the designer has to create the list from the analysis documents. Once the list is fairly stable, he can get into a little more detail about the design of the individual classes.

Finally, OOP is the last step of solving the problem. If OOA and OOD have been used in the system definition and design, the translation of design into code should be intuitive.

The mechanism of object-orientation is very different from many other techniques used in the past. It attempts to manage the complexity inherent in real-world problems by abstracting out knowledge, and encapsulating it within objects.

Two obvious characteristics of object-oriented programming are modularity and reusability [Booch, 94b]. The modularity makes the program structure compact, not scattered. It is easy to modify a program developed using the object-oriented paradigm. Also, it is much easier and convenient to add, delete or modify the system functionality.

The reusability makes the life of systems longer and more circulative. Object-oriented approach has other important concepts such as encapsulation, polymorphism, and so on. Undoubtedly, the flexibility of many computer systems has been enhanced by adopting the Object-oriented paradigm. The advantages of object-oriented approach has made it widely used.

2.2.2. Existing OO Methodologies

Object-oriented analysis and design (OOAD) is a rapidly evolving discipline, and there have been many related books published in the past few years in addition to numerous research papers. There are many methodologies in the OO field from requirements analysis to design.

Some authorities think that OO methodologies can be divided into OOA and OOD. OOA methodologies have evolved mainly from information modeling: e.g. E-R (Entity-Relationship) modeling helps identify objects based on data components; ERDs (Entity Relationship Diagrams) show entities (objects), relationship, attributes; OOA extends ER with inheritance, and behaviour shown on same diagram. Rumbaugh, Shlaer/Mellor, Coad/Yourdon, Martin/Odell models are extended ERDs. OOD methodologies are naturally based on OOP since designs models are closer to coding than analysis. These methodologies emphasize class hierarchy, instantiation and message passing, and are more useful in design than analysis. OOD methodologies have mostly evolved from programming languages, like C++, Smalltalk, Ada. The typical examples of design methodologies related to programming languages include Booch methodology, Wirfs-Brock methodology [Rosenberg, 95].

In general, OOA and OOD methodologies result from the influence of three domains:

- Semantic data modeling
- Structured analysis methods
- Object-oriented programming languages

In some cases, there is no clear boundary between OOA methodologies and OOD methodologies. Most of existing OO methodologies are still undergoing modifications stimulated by the experience gained from their application in practice. Recently, Coad and Yourdon have extended their methodology, providing guidelines for design, without changing the graphical notation elements involved; Booch, instead, proposed an analysis

phase, for which new graphical elements are added to the existing notation for design. Rumbaugh has replaced his OMT with a second generation methodology [Rumbough, 94b, 95c].

Shlaer and Mellor

Shlaer-Mellor methodology was published in 1988 with a case study prototyping approach [Shlaer & Mellor, 88]. It gave many heuristics for object identification and analysis, which help initial abstraction and object modeling. It was derived from information ER modeling. But the initial methodology did not include any notion of inheritance. In 1991, a later book by Shlaer and Mellor introduced inheritance (Entity Subtyping) [Shlaer & Mellor, 91].

In Shlaer-Mellor methodology, the first step is to break a system into smaller parts, then define the interface for all the parts. These smaller parts are called domains. The subject matter of each domain is distinct from the subject matter of any other domain in the system.

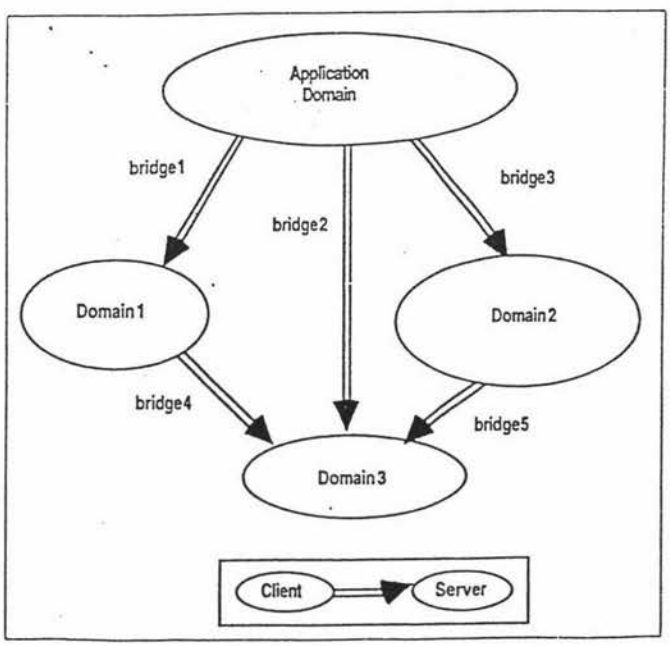


Figure 2.1. Shlaer-Mellor domains and bridges

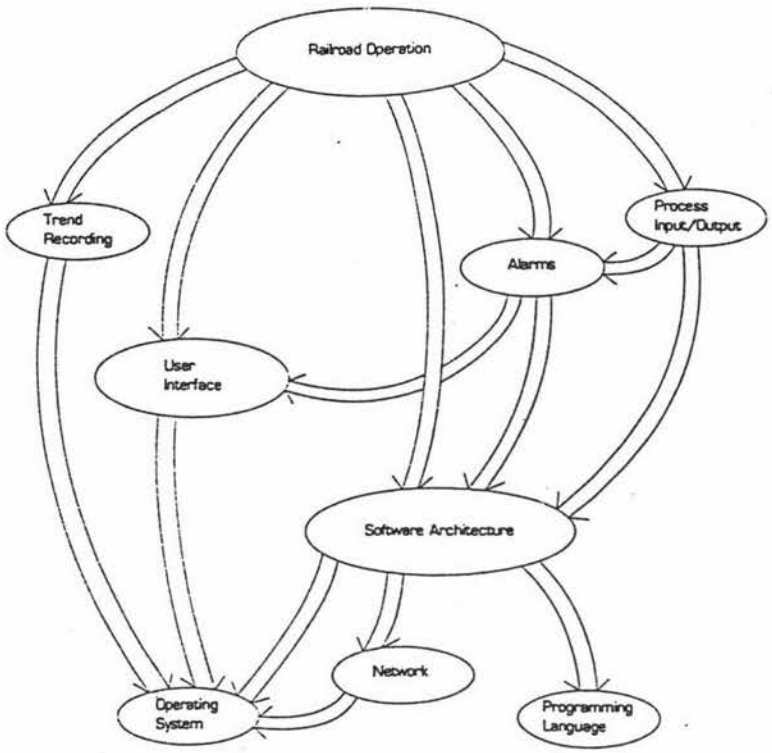


Figure 2.2. Domain chart for the Automated Railroad Management System [Shlaer & Mellor, 91]

Figure 2.1. shows several domains connected by arrows. For any pair of connected domains, one domain acts as a client, the other as a server. The arrows represent what the Shlaer-Mellor methodology calls a bridge: a small amount of code that holds domains together by conveying control and data between the client and the server.

Domains are always depicted on a domain chart. Figure 2.2. illustrates a domain chart for an Automated Railroad Management System.

According to Shlaer-Mellor methodology, the steps during the process of OO analysis are the following:

1. Domain analysis
   (a) Identification of domains
   (b) Identification of bridges
2. Definition of subsystems
3. Construction of information models
   (a) Objects
   (b) Attributes
   (c) Relationships
   (d) Subtypes and supertypes
4. Construction of state models
   (a) Life-cycles for objects
   (b) Life-cycles for relationships
   (c) Object communication model
5. Construction of process model
   (a) Action data flow diagram
   (b) Object access model

Jacobson

OOSE (Object-oriented Software Engineering) [Jacobson et al., 92] is a methodology for OOAD derived from Jacobson's Objectory. It is derived from experience of a single, although large, project: telephony for Ericsson, in Sweden. It is unique among OO methodologies in attempting to address the entire software development life cycle and is proved as a successful OO methodology in many industrial environments. Many of the ideas of OOSE are similar to those of other OO methodologies, but one original idea stand out: the Use Case. A use case is a dialogue, between the system and the user, which is carried out to achieve some goal.

In Jacobson's methodology, analysis starts with a description of the set of typical interaction sequence (event sequence), called a "Use Case". During establishing a Use Case model, the concept of "Actors" is introduced. Actors are external entities with which the system interacts. An actor may be a human, also may be an arbitrary system. The introduction of "Actors" has two advantages:

- the actors can be used to find the use cases and be instantiated in actual interactions with users.

- the same event can initiate different use cases depending on the state of the system (mode) and the purpose and subsequent behaviour of the user.

The process of Objectory includes three phases: requirements, analysis, and construction, which is shown in Figure 2.3.

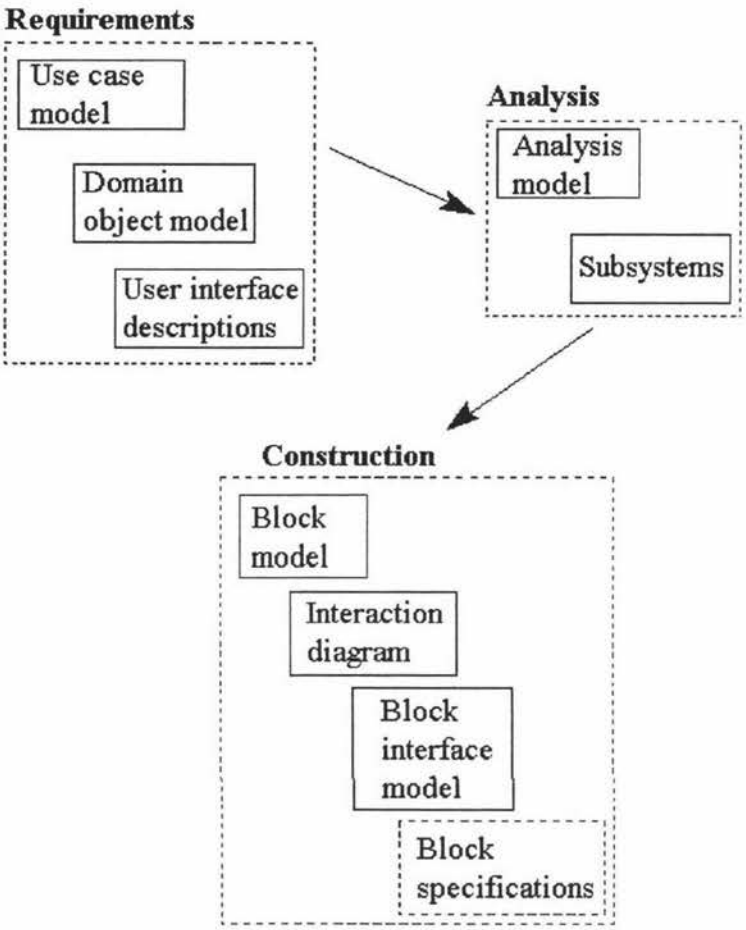


Figure 2.3. Objectory: Process Overview [Coleman, 94]

The description of users' requirements results usually in building three models: Use-case model; User interface descriptions; Domain object model. Use-case model (Figure 2.4.) is used to generate a domain object model with objects drawn from the entities of the business. User interface descriptions bridge the gap between the system and users. Domain object model supports the use cases by defining the concepts with which the system works.

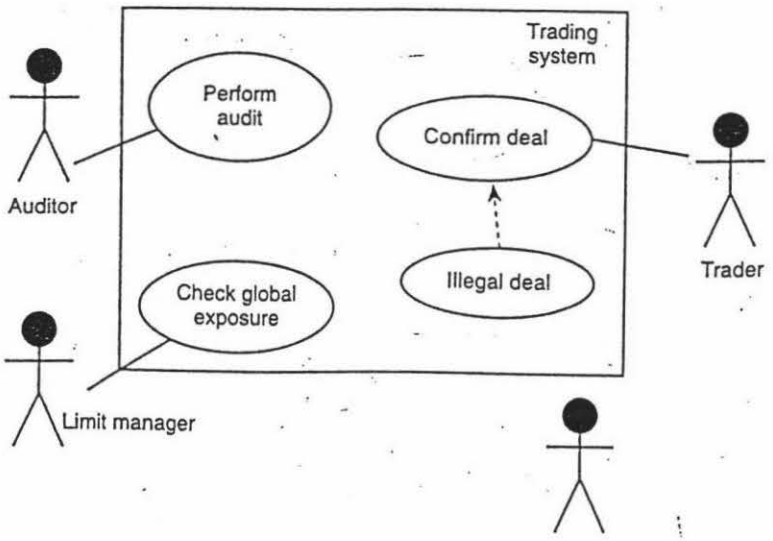


Figure 2.4. The Use-case Model for a financial trading system [Graham, 94a]

During the analysis phase, the requirements models are refined to produce an ideal description of the system. Actually, the analysis model is a form of entity-relationship model. Sybsystems are groups of objects with similar behaviour. They are used to reduce complexity and structure the system for further development.

During the construction, the analysis models are refined. Four models are produced: Block model; Interaction diagrams; Block interfaces; Block specification. A block is an abstraction of a code module. Each block is formed from one or more analysis objects. Interaction diagrams are timeliness diagrams for each of the use cases developed previously. Construction phase is the last phase in which object communication is more precisely defined, and the characteristics of the implementation environment are considered.

By Objectory, the steps during the process of OO analysis and design are the following:

1. Requirements analysis — requirements model
   (a) Use-case model
      — actors
      — use cases
   (b) User interface descriptions

(c) Problem domain object model
— object name
— logical attributes
— static instance connections (inheritance, dynamic instance connections, operations)

2. Analysis — analysis model
(a) Identification of analysis objects
— interface objects
— entity objects
— control objects
(b) Working with the analysis objects
(c) Definition of subsystems

3. Construction
(a) Building Block model
(b) drawing Interaction diagrams for use cases
(c) forming each block interface
(d) implementing the block specifications for the target system

Based on Objectory, OOSE distinguishes five different models: a requirements model, an analysis model, a design model, an implementation model and a test model. In terms of the life cycle OOSE recommends stages of analysis, design, implementation and testing. The advantage of this approach is that requirements can be traced right through the whole life cycle, and this is what OOSE emphasizes.

The main contribution of Jacobson's methodology is the Use Case concept. It adds user-centred analysis and design to the OO approach for specification of the user interfaces and tasks provided by object services. The application of Use Case concept makes Jecobson's methodology itself practical. The users need not know which use case they are going to perform at the outset, only that eventually closure will be achieved on completion of some use case. This is useful approach to define high-level system scope. The weakness of Jacobson's methodology is "simple notation" and that it is not clear that it can "support prototyping as an approach except for the UI, although incremental development is encouraged as is sound risk analysis and the collection of metrics"[Graham, 94a].

Coad and Yourdon

Object-oriented analysis [Coad & Yourdon, 90a] is primarily based on an entity-relationship model mixed with some typical features of object-oriented programming languages or databases such as message passing and system-generated object identifier[Eckert &Golder, 94]. It is used during the analysis phase of the software life cycle. Later, Coad/Yourdon methodology has been extended, using the same notation, to the design phase [Coad & Yourdon, 91b]. Coad and Yourdon suggested that OOA consists of five major activities:

Finding Class-&-Objects
Identifying Structures
Identifying Subjects
Defining Attributes
Defining Services

- Subject: "A subject is a mechanism for guiding a reader (analyst, problem domain expert, manager, client) through a large, complex model. Subjects are also helpful for organizing work packages on larger projects, based upon initial OOA investigations " [Coad and Yourdon, 91b]. The problem area can be decomposed into "subjects" which correspond to the notion of "levels" in dataflow diagrams.

- Structure: Two different structures are identified, Generalisation-Specialisation structure and Whole-Part structure (structural links). The Generalisation-Specialisation structure points out the inheritance relations. The Whole-Part structure evidences aspects union of parts and part, container and content, set and member.

- Service: After having established the attributes of each class, the operations manipulating these attributes and characterizing the class behaviour have to be defined. Service (or method) is a procedure that alters the state of an object (the values of its attributes) or causes the object to send messages.

The OOA model is presented and reviewed in five layers:

Subject layer
Class & Object layer
Structure layer
Attribute layer
Service layer

For the behaviour dimension, Coad/Yourdon's methodology gives instance connections and message connections. From a description of the semantics of a message connection arrow: (a) The sender "sends" a message; (b) The receiver "receives" the massage; and (c) The receiver returns a response to the sender.

Coad/Yourdon's object-oriented design adds to these five activities (or layers) four components. Principally, OOD consists in refining the results of OOA into what is defined as the Problem Domain Component of the design (by adding new solution space classes) and adding three new components called the Human Interaction Component (HIC), Task Management Component (TMC) and Data Management Component (DMC).

The OOD model [Coad and Yourdon, 91b] consists of four components:

Problem Domain Component
Human Interaction Component
Task Management Component
Data Management Component

In an overall approach, the four components correspond to the four major activities in OOD:

Designing Problem Domain Component
Designing Human Interaction Component
Designing Task Management Component
Designing Data Management Component

The Strengths of Coad/Yourdon methodology are its conceptual simplicity and streamlined process, specially for OOA. One of the most notable features of its notations is the explicitness of attributes. Coad/Yourdon introduce a less clumsy notation than that found in Booch, Shlaer/Mellor or most of the design-oriented approaches. The emphasis on analysis as opposed to design. This is the reason why some authorities classify this methodology under OOA methodologies.

This methodology is weakest when it comes to specifying the dynamics of systems, but a state transition approach is suggested for object dynamics though no notation for this is defined [Graham, 94a]. The simple notation of this methodology is not enough for dealing with complicated systems. Therefore, it is difficult to design a real-time, dynamic

and complex system. Figure 2.5 gives the example of an Airlift system using Coad/Yourdon methodology.

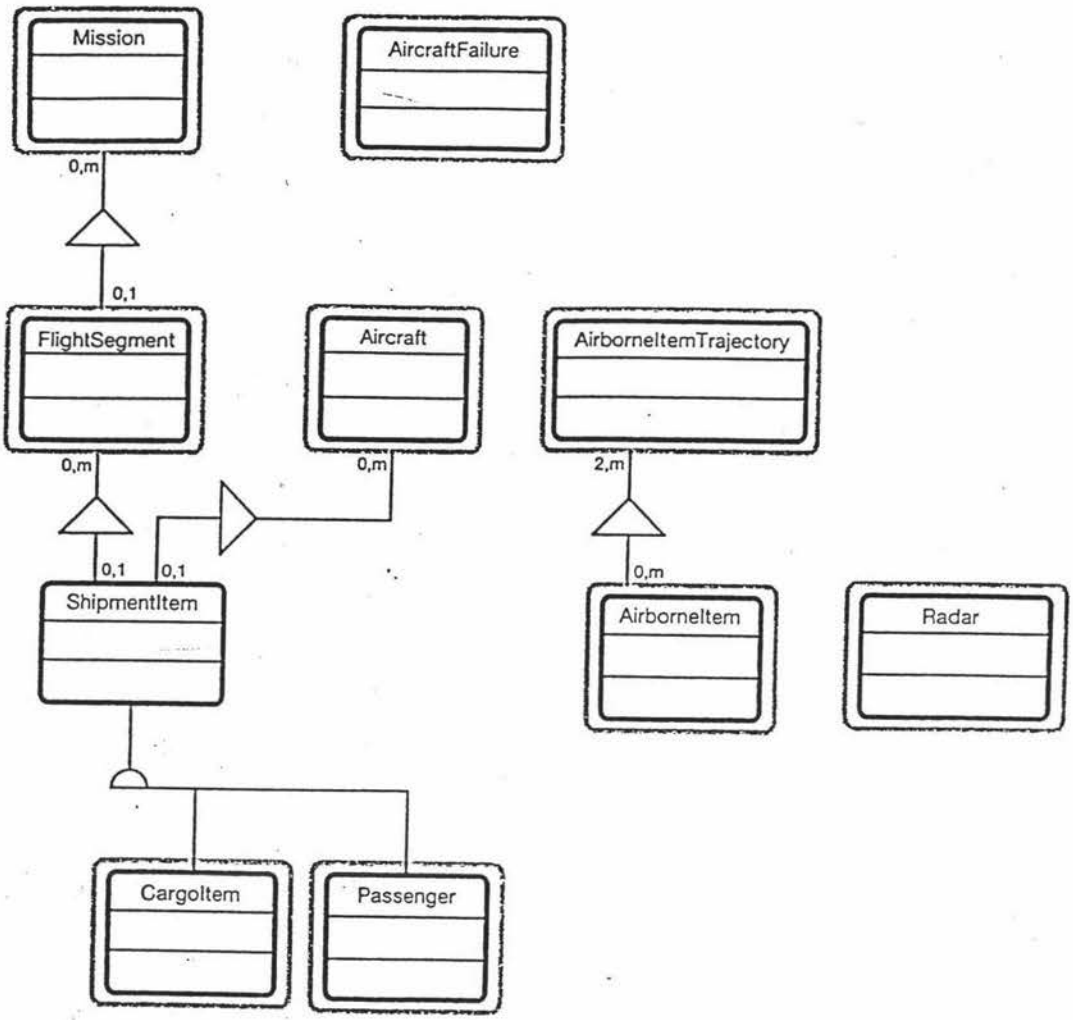


Figure 2.5. Airlift System — Class-&-Object and Structure layers [Coad/Yourdon, 91b]

Rumbaugh

The Object Modeling Technique (OMT) [Rumbaugh et al., 91a] originates from the work of James Rumbaugh and his colleagues at General Electric. OMT of Rumbaugh et al. is widely regarded as one of the most complete object-oriented analysis systems so far

published [Graham, 94a]. OMT process consists of analysis, design and implementation (Figure 2.6).

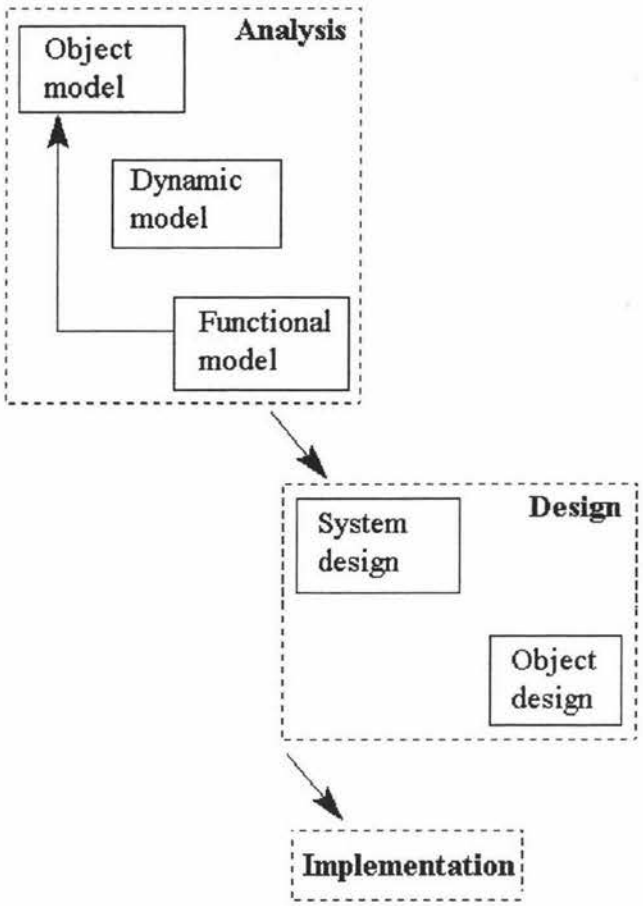


Figure 2.6. Object Modeling Technique: Process Overview [Coleman, 94]

There are three models in the analysis phase: the object model (OM), the dynamic Model (DM) and the functional model (FM). Object model consists of diagrams similar to those of Coad/Yourdon, and a data dictionary. The notation is based on that of ER modeling, but operations and other annotations are added to the entity icons. The OMT object model is meant to capture the static structure of a system by showing the classes in the system, relationships between the classes, and the attributes and operations that characterize each class. It is the most important of the three models.

A dynamic model is built for every object. The dynamic model consists of STDs (State Transition Diagrams) drawn in an extended Harel notation and global event flow diagrams. Dynamic Model captures the dynamic behaviour of each class by using state machine diagrams. Each state machine shows how the class objects respond to events. Events are objectified in that they are seen as constituting classes that can be even in

inheritance relationships. Events may occur with attributes, cause actions that may be guarded by conditional statements (preconditions).The third step is to build Functional Model which uses data flow diagrams (DFD) to show how outputs are computed for inputs. The purpose of the functional model is to specify the computations without regard for the order in which the values are computed. FM is not always done and most practitioners seem to use it at a high level.

The design phase does not introduce any models of its own. It builds on the analysis models. This phase is divided into two stages: the system design and the object design. The system design proceeds by organizing the objects into subsystems, identifying concurrency from the DM, allocating subsystems to processors or tasks. This stage includes resource management and organization of data stores.

The object design stage includes the steps:

(1) design algorithms;
(2) optimize access paths;
(3) implement control;
(4) adjust structures;
(5) design attribute details;
(6) package structures into modules;
(7) write the design report, including a detailed OM, DM and FM.

The implementation phase concentrates on converting the object design into code. Rumbaugh gives some sound heuristics for implementing object-oriented designs in relational database and in conventional languages [Rumbaugh et al., 91a].

The primary strength of OMT is its analysis phase. It has a well-defined process, and all the models use concise and understandable notation which are much richer than Coad/Yourdon. OMT addresses more development issues than most other methodologies.

The main weakness is that the design stage lacks the step-by-step approach of the analysis[Coleman & Arnold et al., 94]. It still remains incomplete in some areas and the notations is very complex to learn and use [Graham, 94a].

Booch

Booch methodology is considered one of rather complete design methodologies. This methodology is described in his first edition of "Object-Oriented Design with Applications"[Booch, 91a]. In 1994, the second edition was published [Booch, 94b]. In this book, more specific detail about the unified notation is added to enhance the reader's understanding; more programming examples using C++ have been introduced; and certain idioms and architectural frameworks have emerged in various application domain. Booch suggests using domain analysis to discover candidate classes, and then model the world by identifying the classes and objects that form the vocabulary of the problem domain.

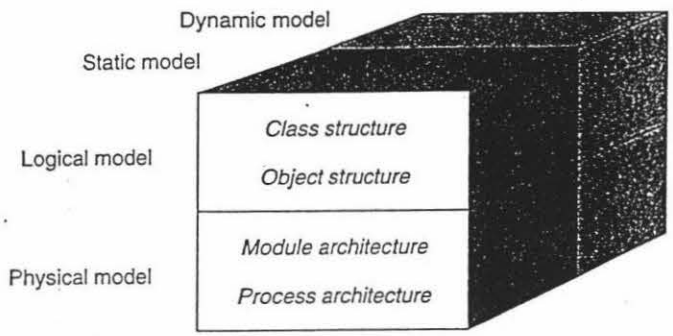


Figure 2.7. The models of object-oriented analysis and design [Booch, 94b]

The design methodology and notation consists of four major activities and six notations. Different models can be built: dynamic model, static model, logical model and physical model (Figure 2.7). The six notations are: class diagrams, object diagrams, timing diagrams, state diagrams, module diagrams and process diagrams.

Class Diagrams and Object Diagrams

Class diagrams show classes and their relationships. A class diagram is a notational variation of entity-relationship diagrams. These relationships include *inheritance, instantiation, and use*, which are described by using a set of rich notation. Class relationships are mostly static. To deal with complex diagrams, Booch recommends a form of layering, so that classes can be grouped into " class categories" containing

several related classes. These layers are shown as rectangles. A class category diagram is a logical collection of classes or other class categories.

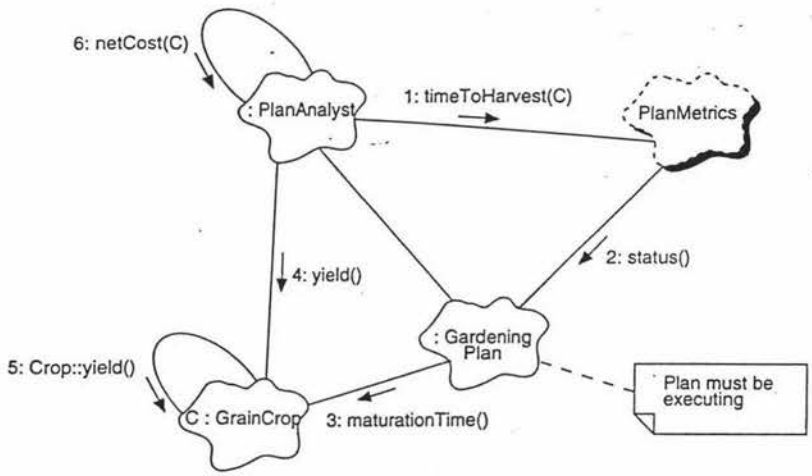


Figure 2.8. Object Diagram for the Hydroponics Gardening System [Booch, 94b]

An object diagram shows object and their relationships. In this instance "relationship" means that the objects can send messages to each other. In general, object relationships are dynamic. Figure 2.8. shows the object diagram for the Hydroponics Gardening System. Like a class diagram, an object diagram provides the logical view of a system, allowing the description of these abstractions or entities, justifying their presence and meaning in the whole system architecture.

Drafts of class diagrams and object diagrams are produced during early stages of development and completed during the identification of the relationship among classes and objects.

Module Diagrams and Process Diagrams

Class and object diagrams and their associated templates describe the logical static design of a system. The physical design may differ from the logical. Booch distinguishes between logical and physical view of a system. Module and process diagrams are simple

graphs giving a physical view of the system under development. A module diagram shows the allocation of classes and objects to modules. The notation for modules is based on the earlier Gradygram notation. A process diagram which is a simple block diagram, shows processors (annotated with the process which they run), devices and the communication connections between them.

Timing Diagrams and State Diagrams

Booch suggests that the dynamic semantic of a system should be described in two ways. One of them is taking state transition diagrams to show how the instances of classes move from one state to another under the influence of events and what actions result from such state changes. Like OMT, Booch commends that Harel notation can be used to express state transition diagrams. The other way is using timing diagrams to display ordering information. A timing diagram has time as the x-axis and the different objects as the y-axis, a line represent the flow of control between objects. The notation is borrowed from the field of hardware design, as shown in Figure 2.9.

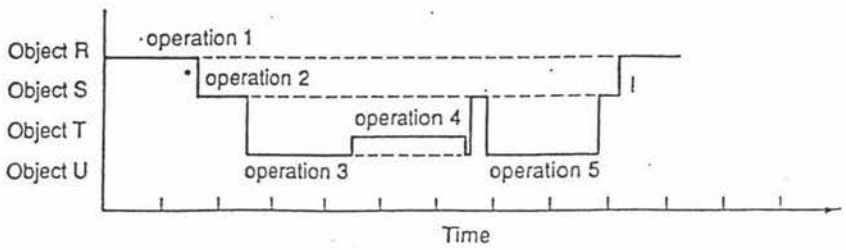


Figure 2.9. A timing diagram [Graham, 94a]. The asterisk (*) indicates the creation of an instance and the shriek (!) its destruction.

Booch places particular emphasis on discovering the "key mechanisms" of a design. A mechanism is any structure whereby objects work together to provide some behaviour that satisfies a requirement of the problem. Booch acknowledges that analysis and design cannot proceed in isolation of each other, and advocates a piecemeal approach; keep improving the design until you are satisfied. Figure 2.10. gives an overview of the Booch process.

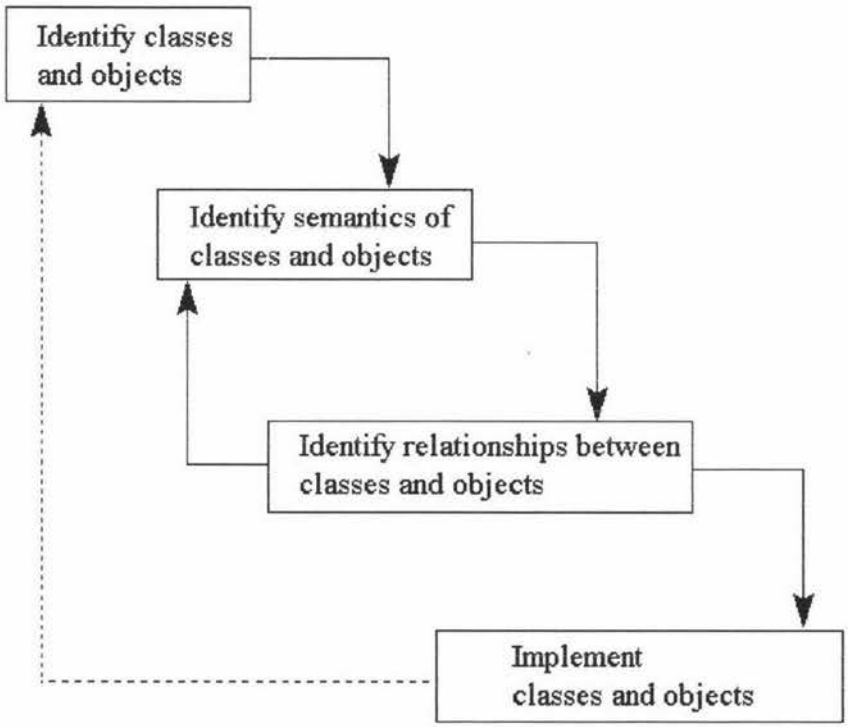


Figure 2.10. Booch: Process Overview [Coleman, 94]

During the whole process, the classes, instances, semantics and structures should be refined on the basis of what has been learnt. The process stops when it is believed that all the key abstractions and functions have been defined, and is highly non-linear.

Booch methodology is a well-known analysis and design methodology which is powerful in dealing with a complex system. Its notation covers all aspects of an object-oriented system (Static, dynamic, logical and physical). A major weakness in Booch is the absence of a defined process for developing system [Coleman, 94].

Chapter 3

CASE Tools

"Any software that saves time, increases quality, or enhances communication during software development is a CASE tool"[Brumbangh, 94].

The acronym CASE refers to Computer-aided Software Engineering. CASE technologies span a wide range of topics that encompass software engineering methods and project management procedures. In fact, the crux of CASE is the use of tools that provide leverage at any point in the software development cycle. CASE tools are developed for the purpose of automating the software process.

The true history of CASE started in mid-1970s. The ISDOS project, under the direction of Dr. Daniel Teichrowe at the University of Michigan, developed a language called Problem Statement Language (PSL) for describing user problems and solution requirements for an information system into a computerized dictionary. A companion product called Problem Statement Analyzer (PSA) was created to analyze the problem and requirements statement for completeness and consistency. PSL/PSA ran on large mainframe computers that consumed precious and expensive machine resources. Therefore, few companies could afford to dedicate computer resources to PSL/PSA.

The real breakthrough came with the advent of the IBM personal computer. In 1984, an upstart company called Index Technology (now known as INTERSOLV) created a PC software tool called Excelerator. Its success established the CASE acronym in industry. Since then, more and more CASE products have been running on personal computers or intelligent workstations.

In the past several years, much has happened in the area of CASE. Research efforts have been concentrated on improving CASE tools functionality and efficiency. New products are continually appearing. Particularly, the studies which compare traditional software development with software development using CASE have been conducted. They showed that the productivity and quality can be greatly improved by using CASE tools. Although modern CASE technology is still very young, the technology is improving at a staggering rate. New products (called tools) are emerging monthly, and best existing

products are improving annually. Of course, some of products fail in the marketplace due to different reasons.

3.1. Life Cycle of Software System Development

The identification of the "software crisis" in the late 1960s and the notion that software development is an engineering discipline led to the view that the process of software development is like other engineering processes [Sommerville, 96]. The model of the software development process is called "software life cycle".

The software development life cycle is a phased approach to analysis and design which holds that system are best developed through the utilization of a specific cycle of analyst and user activities.

3.1.1. "Waterfall" Life Cycle

There are numerous variations of software life cycle. The typical one is known as the "waterfall" model. This model includes five stages [Sommerville, 96], as shown in Figure 3.1. They are:

(1) *Requirements analysis and definition*: The system's services, constrains and goals are established by consultation with system's users. They are then defined in a manner which is understandable by both users and developers.

(2) *System and software design*: The systems design process partitions the requirement to either hardware or software system and establishes an overall system architecture. Software design involves representing the software system functions so that they may be transformed into one or more executable programs.

(3) *Implementation and unit testing*: During this stage, the software design is represented as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

(4) *Integration and system testing*: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met.

(5) *Operation and maintenance*: During this phase, the system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

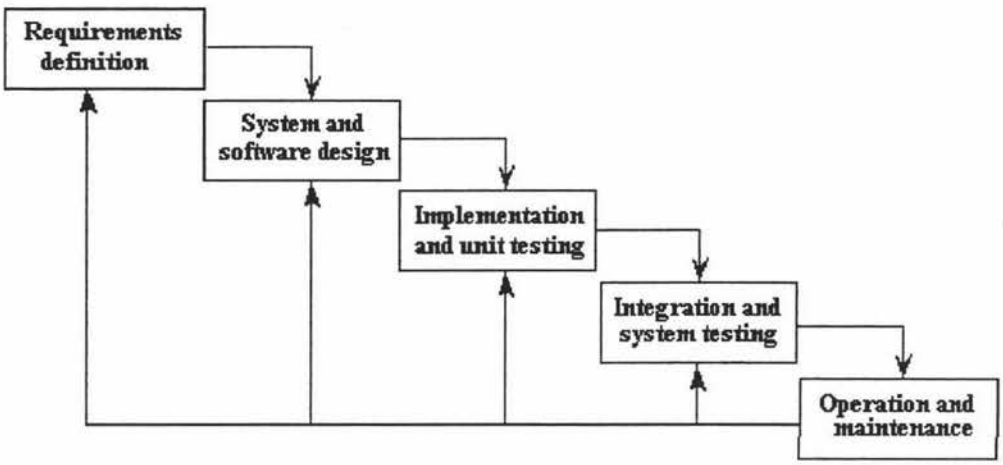


Figure 3.1. The waterfall model of the lifecycle [Sommerville, 96].

It is very important that during the final phase of the life cycle (operation and maintenance), information is fed back to previous development phases. Errors and omissions in the original software requirements are discovered, program and design errors come to light and the need for new functionality is identified. Maintenance may involve changes in requirements design and implementation, and it may highlight the need for further system testing. All the modifications can be made in this phase and the software system is refined again and again.

3.1.2. OO Life Cycle

A high-level reference model for object-oriented software development life cycle has been defined by the Object Management Group (OMG), as shown in Figure 3.2. This model covers the activities that must be carried out during an OO project.

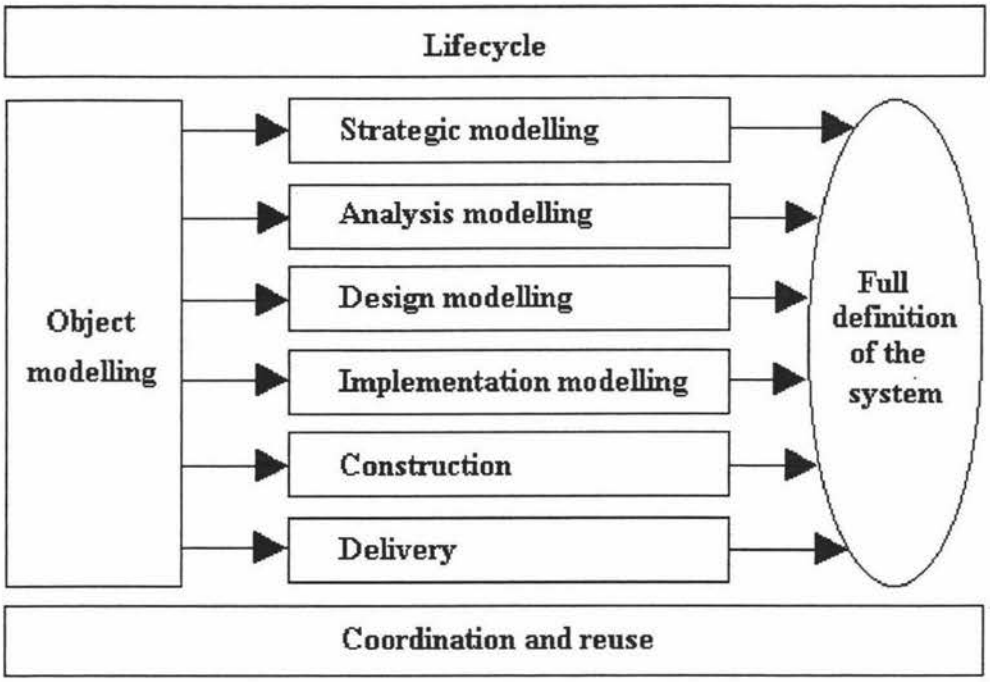


Figure 3.2. The OO software development lifecycle proposed by OMG [Yourdon, 94].

Object Modeling: One of the important characteristics of an OO project is that all the life-cycle activities share common vocabulary, notation, and strategies — that is, everything revolves around objects [Yourdon, 94]. Therefore, Object Modeling provides a set of terms and concepts for representing everything within the scope of analysis and design as an object.

Strategic modeling: Enterprise and business modeling. It covers requirements capture and development planning.

Analysis Modeling and Design Modeling: The concepts of Analysis Modeling and Design Modeling are familiar to most software engineers. During the process of obtaining a description of the problem domain, the OO analysis model is built. The Design Modeling covers specifications of object types, operations and interfaces. The design must meet quality requirements.

Implementation Modeling: This is physical design and involves designing modules and the distribution strategy [Graham, 94a]. During this phase, the software and hardware to be used should be considered.

3.2. CASE Tools Categories

In general, the CASE tools which are used in different areas can be classified in two, orthogonal ways:

(1) Function- oriented --- In this way, the classification of tools is based on the functions they provide.

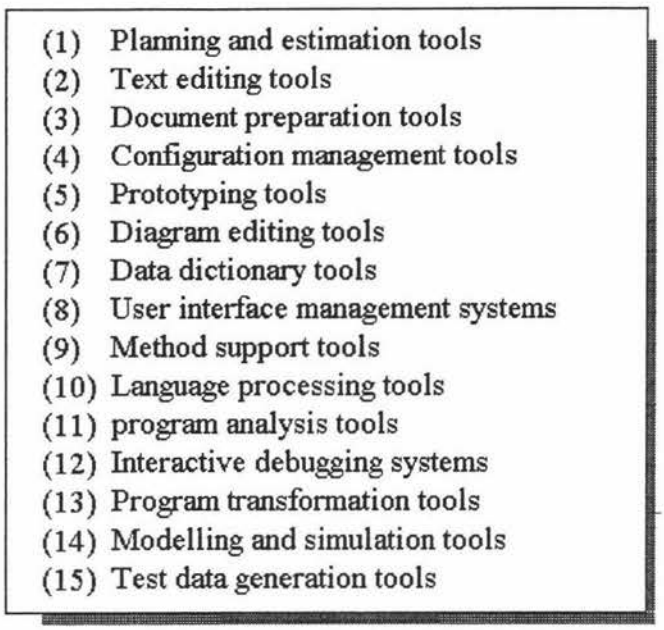


Figure 3.3. The classification of tools based on functions [Sommerville, 96]

CASE tools can be classified by the functions they perform. Figure 3.3 shows the function-oriented classification of tools. These CASE tools are grouped according to their uses in various areas. In other words, these tools can separately automate the solving of different types of problems.

(2) Activity-oriented — This classification of tools is based on the process activities they support, such as requirements specification, design, implementation, etc.

Nowadays, the term "CASE" refers to tools supporting the entire software lifecycle, including specification, design, development, and maintenance phases. CASE "tools" are programs (software) that automate or support one or more phases of a systems development lifecycle. Therefore CASE tools might often be classified into three broad categories according to the CASE problems on which they focus [Whitten et al., 94]:

a. Upper-CASE tools (front-end CASE tools):

The term *upper-CASE* describes tools that automate or support the "upper" or front-end phases of the systems development life cycle; namely, systems planning, systems analysis, and general systems design [Whitten et al., 94].

Upper-CASE tools for system planning help analysts and consultants to capture, store, organize, and analyze models of the business and define business strategies. Most planning tools can perform analysis on different matrices to identify logical "groups" of data, functions, locations and other planning information , and help develop the planned databases, networks, and information systems.

Upper-CASE tools for system analysis and design deal with the high-level design, specification, and analysis of software requirements. These tools allow users to create and modify the design of systems, help primarily analysts and designers to support the modeling of an organization's functional requirements, and assist them in drawing the boundaries for a given project.

b. Lower-CASE tools (back-end tools):

The term *lower-CASE* describes tools that automate or support the "lower" or back-end phases of the life cycle; namely, detailed systems design, systems implementation, and systems support [Whitten et al., 94].

Lower-CASE tools deal with the detailed design, coding, assembly, and testing of software. These tools are used more often by programmers and workers who must implement the systems designed via upper CASE tools. They have mainly functions as the following:

- Helping programmers to more quickly test and debug their program code.
- Helping programmers or analysts to automatically generate program code from analysis and design specifications.
- Helping designers and programmers to design and automatically generate special or detailed system design components like screens and databases.
- Automatically generating complete application code from analysis and design specifications.

c. Cross life cycle CASE tools:

The term *cross life cycle CASE* refers to tools that support activities across the entire life cycle. This includes activities such as project management and estimation [Whitten et al., 94].

The main activities across the entire systems development life cycle are: project management, estimation and documentation. Project management tools are used to help managers plan, schedule, report on, and manage their projects and resources. Estimation tools are used to assist users in the estimation of time and cost to complete the project. Documentation tools allow users to assemble documentation that may have been created in different phases of the life cycle.

3.3. OO CASE Tools

3.3.1. Background

Object-oriented CASE tools are these tools which are used to support the software development adopting object-oriented analysis (OOA) and object-oriented design (OOD) methods.

The purpose of designing OO CASE tools is to help a user to perform analysis and design tasks adopting Object-Oriented approaches.

Since the CASE tool focuses on OO techniques, it should support the following concepts:

- Classes (should include Attributes and Operations)
- Objects (Instances of a Class)
- Inheritance
- Communication and relationships between Objects and Classes

OO graphical CASE tools can help the analyst or designer to draw a graphical representation of the developed model The diagram is composed of a set of graphical primitives. In general, OO CASE Graphical tools have these distinct features:

- The tool uses simple graphical primitives: Shapes and Connectors.
- The tool can explicitly supports classes, objects, functions, inheritance and message passing.
- The tool should have flexible modification abilities (shapes can be moved, resize, create and delete, and so on).

3.3.2. Types of OO CASE Tools

So far, the main kinds of tools that are applicable to Object- Oriented Development can be classified as follows:

(1) A graphics- based system supporting the Object-Oriented notations — a tool that can be used during analysis to capture the semantics of scenarios, as well as early in the development process to capture strategic and tactical design decisions, maintain control over the design products. This tool is also useful during systems maintenance.

(2) A class browser — a tool that knows about the class structure and module architecture of a system. Using this tool, a developer can find all of the abstractions that are part of the design or are candidates for reuse. For instance, the standard Smalltalk environment allows users to browse all the classes of a system.

(3) An incremental compiler — a compiler that can compile single declarations and statements. This kind of evolutionary development corresponds to the principles of object-oriented approach and provides the support that is necessary for OOP. But, unfortunately, most existing compile systems are still traditional, batch-oriented.

(4) A debugger with knowledge of classes — a debugging tool that knows about class and object semantics. Traditional debuggers for non- object-oriented programming languages do not include knowledge about classes and objects, and can not permit the developer to find the really important information needed to debug an object- oriented program (eg. using a stand C debugger for C++ programs, while possible).

(5) A configuration management and version- control tool.

(6) A class librarian — a librarian tool that allows developers to locate classes and modules in the library according to different criteria and add useful classes and modules to the library as they are developed.

(7) A GUI (Graphical User Interface) builder — a tool to interactively create dialogues.

3.4. Evaluation of the Current CASE Tools

3.4.1. Advantages

Now, more and more people realize that it is important to use CASE tools. A lot of research in evaluation of the use of CASE tools has been done in the past. Many surveys about the use of CASE tools have been conducted. For example, a survey of 834 organizations was carried out by G.M.Wijers [1990]; an investigation of CASE tools utilization which covers hundreds of companies was undertaken in different countries [Stobart et al., 93].

These reports stated the advantages of using CASE tools and major reasons for purchasing CASE tools. The software development areas that were identified as having the largest increase in quality were those of: Code Generation, Prototyping and Project Control. The software development phases that were identified as having the largest increase in efficiency were those of: Program Design, Code Generation, Prototyping and Maintenance.

It is reported that many financial institutions have dynamic systems delivery environments and have used CASE products to assist in their systems delivery to their client base.

Many companies identified the following commonly found advantages that could be gained from the use of CASE products:

(1) increased productivity,
(2) reduced development and maintenance costs,
(3) faster delivery,
(4) improved documentation.

In summary, it was identified that the theorised and promised benefits and advantages from using CASE are the same world-wide.

3.4.2. CASE Tools Problems

Some research reports pointed out that many of the current CASE tools suffer from generic problems. These tools have too many inconvenient characteristics which prevent companies realizing the benefits that could be obtain by their use. The major problems are:

(1) Tools do not offer the possibility to use several different methods in an integrated manner.
(2) The integration between tools for different software development phases is poor.
(3) Tools often required a change to methods other than those used within the organisations.
(4) Tools for project management are not integrated with tools for the other software development activities.
(5) The coupling of design data to the tool repository is weak.
(6) There are no multi-user application development possibilities, for communication between the CASE tools users.

Currently CASE tools are costly and immature products. Because of rigidity and weak support of the user's native methods and methodologies in existing CASE tools, a well planned training investment must be considered before CASE can be successfully used. Therefore, high costs of tools seem to be a major reasons for their rejection by many organisations.

Figure 3.4 shows the major problems with current CASE tools. The lack of multi-user development support was the most common. Some of the tools can only support a limited set of methods, and therefore cannot satisfy users' needs.

CASE tools without multi-user support, version control and some notion of work packages are virtually useless except for smallest projects. Multi-user support is essential for any large project and especially when development teams are geographically distributed [Graham, 94a]. It is, however, obvious that all organisations' methods are not similar to those supported in CASE tools. Therefore, one way to alleviate this problem is

to develop CASE environments that can cover a variety of different methods and can thereby satisfy most organisations' different needs.

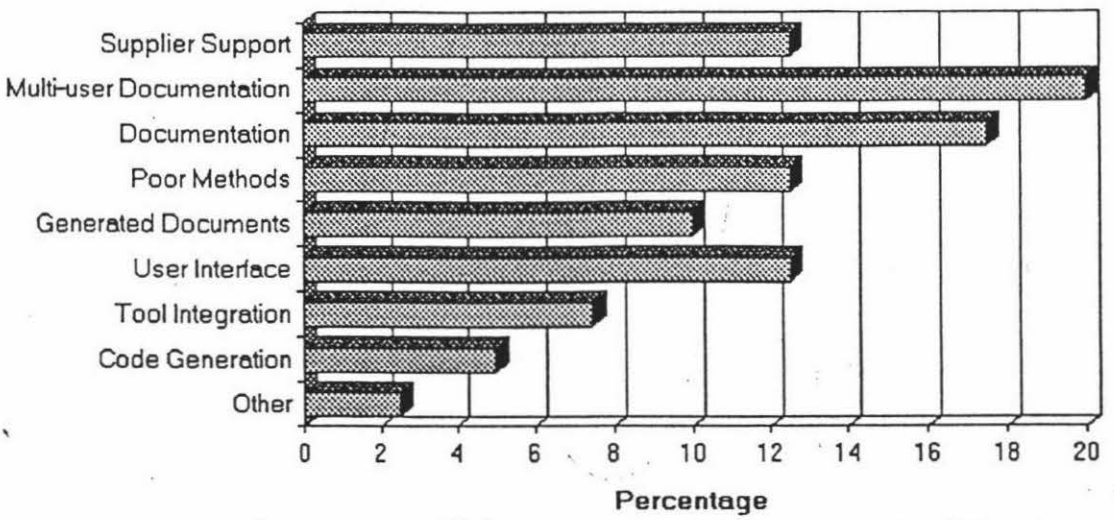


Figure 3.4. Major Problems with Current CASE Tools
[Stobart et al., 93]

In summary, CASE tools in all countries seem to be considered immature. They suffer from a number of distinct problems. Poor tool integration, lack of multi-user development facilities and poor method support were common problems identified in many countries.

Investigations and researches have indicated that [Stobart et al., 93]:

(1) A major reason given for using CASE is to improve the overall software production process, especially in the area of control. Undoubtedly, users can get great benefits from the use of CASE.

(2) The implementation of CASE is a costly business and should only be undertaken in a planned manner. Management support is essential if it is to succeed.

(3) CASE tools need to be flexible and capable of customisation and integration.

(4) The exact relationship between the use of CASE and the use of methods is far from clear. However, it would appear that since there are so many differing methods in use that CASE products need to remain very flexible.

3.5. CASE Tools Development Research Strategies

Although CASE tools were designed to address productivity and quality issues of systems development, it is now widely accepted that they have not totally succeeded in achieving their stated goals. They have been criticised for their lack of flexibility and potential inefficiencies. These current CASE tools cannot meet all kinds of customers' needs. In order to overcome these weaknesses, two major approaches have been investigated.

3.5.1. Federated CASE Environment

In spite of the recent advances made in systems development methodologies there has been some uneasiness about methodology choice and utilization, and in some cases their practical validity. Sometimes, because of a number of limitations that characterise most of the current methodologies, it is hard to decide which methodology should be chosen. In methodologies, the most serious of these limitations are:

- explicitly prescriptive,
- problem specific,
- inadequate support for all phases of the life-cycle.

One Approach to overcome the weaknesses of existing methodologies is to create an open methodology environment in which different methodologies co-exist in a federated architecture, to contribute to a system's development [Papahristos & Grey, 91]. This implies that the supporting CASE tools of these methodologies must also be allowed to co-exist in a way that enable reusability of information collected through one CASE tool by another. In the open methodology environment, users can select any one of the existing methodologies. Such a development environment can:

(1) Result in a highly flexible methodology being used which is capable of blending with other organisational procedures.
(2) Facilitate the development of procedural pattern in response to a system's contextual changes.
(3) Allow choice for the system developer in selecting the most appropriate problem representation or the preferred development methods.
(4) Result in a complete and consistent methodology, supporting all phases of the life-cycle.

One of the methods implementing such an open methodology environment is to make use of a Data Dictionary (DD) during the information resource management (IRM). The Data Dictionary is used in the planning, administration and operation of an organisation's information processing activities. A more suitable approach for achieving an open methodology environment could be to interface a common Data Dictionary to a variety of CASE tools while letting them retain their own local data dictionaries.

In general, there are three data levels in an Information Resource Dictionary System (IRDS): Fundamental level, IRD definition level, and IRD level. The Fundamental level consists of the types of data, instances of which are to be recorded on the IRD definition level; The IRD definition level provides an extensible definition of the types of information which may be recorded at the IRD level; the IRD level is the level on which the content of an Information Resource Dictionary is recorded.

CASE tools are supported by a Data Dictionary in which a user's various projects are stored. Each CASE tool available in the market today has its own, usually non-standard, Data Dictionary, thus making it difficult to integrate them. This problem can be effectively overcome if the tools are made to operate from a common shared Data Dictionary. Under the open methodology environment, the common Data Dictionary would be made to act as the co-existence mechanism for the different methodologies, while the local data dictionaries would maintain the autonomy of an individual CASE tool, allowing their techniques and information to be shared between methodologies. This environment is called Federated CASE Environment (FCE).

Figure 3.5 illustrates the major components of the FCE. The corn of the FCE architecture is a meta-translator toolkit which consists of Query Meta-translation Module (QMM) and Diagram Meta-translation Module (DMM). The meta-translator can complete translation of queries and diagrams between CASE tools.

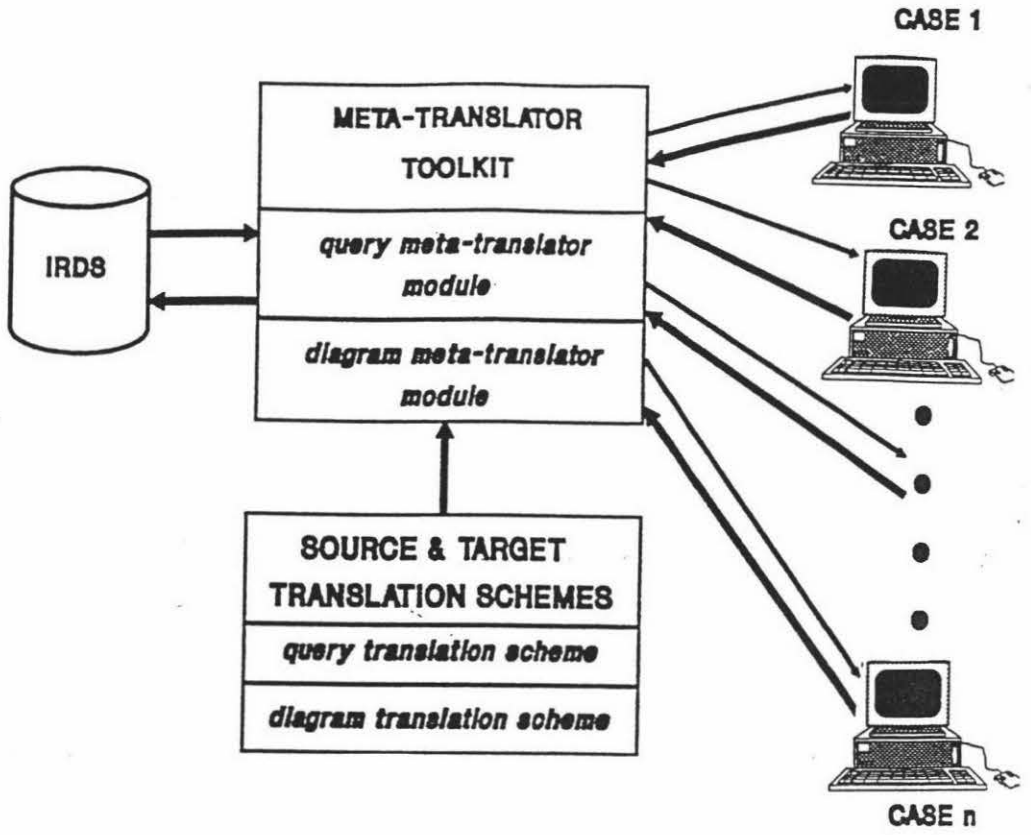


Figure 3.5. FCE Architecture [Papahristos & Grey, 91]

3.5.2. CASE Shell

Today, CASE has been taken into extensive use and more and more CASE tools have emerged in the world. But most companies do not choose, however, a strategy too incorporate existing methods into tools. More likely the methods are determined by the chosen tool. The problem for users in adopting CASE tools is that new methods and models must often be learned while introducing the tool. One way to alleviate this problem is to develop CASE environments that can cover a variety of different methods and can thereby satisfy most organizations' different needs.

Because many existing CASE tools support only a few methodologies and a limited set of methods, there is a growing need for customizable CASE tools — CASE shells (or meta-CASE environment). CASE shells "Offer mechanisms to specify a CASE tool for an arbitrary method or a chain of methods" [Rossi et al., 92]. They are different from ordinary CASE tools supporting a fixed set of methods. A CASE shell allows the user to

specify his own method or metamodel. With a CASE shell, the user can more easily build computer supported methods for a given task or project due to its flexibility. CASE shell may be classified as follows [Marttiin et al., 93]:

database oriented: to define the description languages which deals with methods by using a metalanguage.

interface oriented: to establish a CASE environment around generic graphical notations.

extension kit: to build an extension for existing CASE tools will allow new methodologies to be specified and added into the corresponding CASE tool.

knowledge oriented: in knowledge oriented tools, the meta-metamodel is often based on a set of logical rules.

In the field of Software Engineering and Information System Development (SE/ISD), the software process is called "the total set of software engineering activities needed to transform user's requirements into software" [Oquendo et al., 92]. The purposes of process models include facilitating human understanding and communication, improving project and process management, facilitating automatic guidance in performing processes, and supporting automatic execution. Further, these are fitted to varying organizational and project based practices. A successful meta-CASE environment needs to be powerful enough to manage these diversified needs. The flexibility in process models can be implemented under such a meta-CASE environment:

(1) Support for different life cycle strategies: ISD process should not be forced to follow only one life cycle strategy.

(2) Support for varied methodology processes: Finding a single best methodology suitable for all development projects seems to be impossible. To satisfy different users' needs, the meta-CASE environment should support varying methodologies (including the methodologies defined by the users).

(3) Management of products evolution: Due to the complexity of ISD, user requirements, solution candidates and chosen models may change all the time. Therefore, the tools for handling versions and revisions of products, and for navigating between them, are needed in the meta-CASE environment. Also, these tools should support any managerial activities.

(4) Modification: The process model structure and activities should be modifiable during the actual process.

Since methods are undergoing very rapid evolution at present it may be wise either to use very low-cost tools that can be replaced or adopt a meta-CASE approach. The latter is more expensive but more adaptable and the tools should be less likely to need replacement as methods settle down. This gives advantages in terms of training and the continuity of large projects.

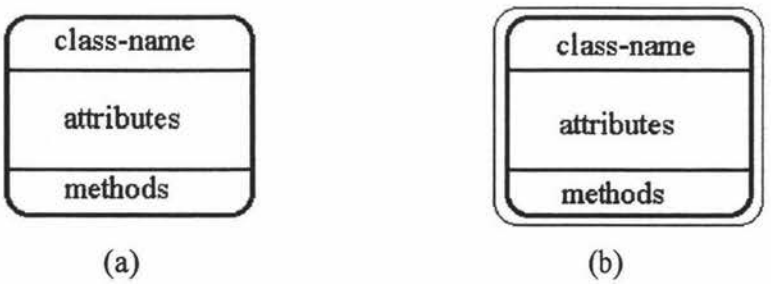
3.6. Rationale Behind MIGOCE Development

3.6.1.Motivation

The limitation of current tools

(1) Multiplicity of OO methodologies

There are over 50 different Object-Oriented analysis and design methodologies. Each of these methodologies has itself advantages and disadvantages. For a certain methodology, the Object Model is expressed in terms of a graphical and/or textual notation. Notations show the differences between one methodology and another. Figure 3.6 — Figure 3.8 illustrate separately the notations for classes and objects of a few OO methodologies described in Chapter 2.



(a) (b)

Figure 3.6. The Coad/Yourdon notation for Classes and objects
(a). Graphical representation for a class
(b). Graphical representation for class and objects

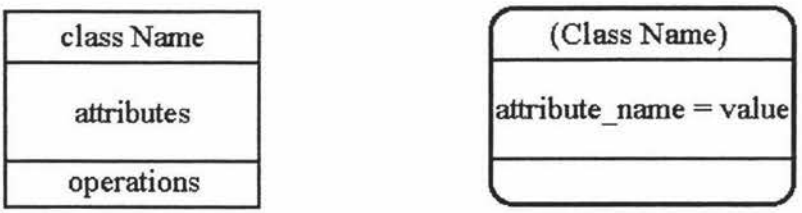


Figure 3.7. The Rumbaugh notation for classes and objects

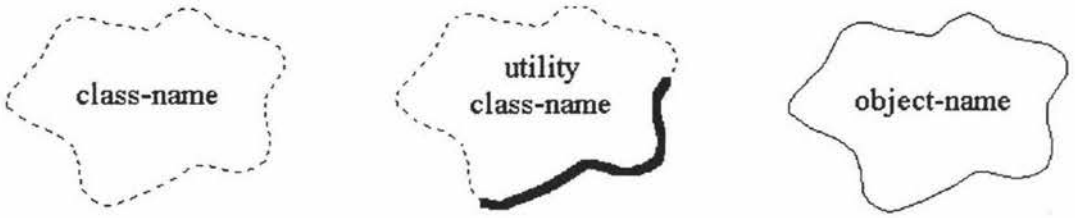


Figure 3.8. The Booch notation for classes and objects

From these figures it is obvious that the notations for the same concept are very different from each other among different OO methodologies. Users have to choose the methodologies they need and the corresponding notations and semantics of these methodologies before they start working on their projects.

(2)Methodology-dependence

So far, most of OOAD CASE tools, particularly upper tools, are methodology-dependent. They support only a limited set of methodologies. Many of these tools support one or two methodologies. Users cannot choose arbitrarily CASE tools according to real requirements, but rely on existing CASE products which support particular methodologies.

None of the existing methodologies and corresponding CASE tools can support the whole process of system development nor meet all kinds of users' requirements. Analysts who follow formal rules blindly do not always develop the skill of being able to deal successfully with the unexpected. Thus there will be a need on some projects to adapt and alter even the best of the existing methodologies. This indicates a strong requirement for tools which are useful independently of any particular methodology: tools where any appropriate diagram type or documentation style can be used [Graham, 94a].

Sometimes, it is more difficult to decide which tool is better when a complicated system is to be built from analysis, through design to implementation, because some CASE tools stress system analysis phase, but some stress system design. For example, it is suggested that the analysis of Coad/Yourdon and Booch design can be applied together to deal with some problem domains [F. Losavio et al., 94]. It is clear that there is a growing need for methodology-independent CASE tools.

The evolution of OO methodologies

OO methodologies themselves are actually in constant evolution. The change is necessary as OO methodologies should meet the real requirements to describe and model real problem domains. Therefore, new concepts and notations might be introduced. The improvement of OO methodologies themself results in the generation of new version of CASE tools which support corresponding OO methodologies. This limits the usage of these CASE tools which are being used by some organizations and their new versions which just emerge at market. These organizations have to buy the new versions to help them to develop more complex systems.

However, there is one interesting aspect of OO methodologies that should be considered: while the OO CASE tools are just now appearing in the early to mid-1990s, the underlying OO methodologies are still undergoing a significant evolutionary development [Yourdon, 94].

In order to meet users' different requirements, flexible, customizable CASE tools are needed. These tools should cover a range of methodologies and can be used to define the user's own methodologies, and the corresponding notations and underlying semantics.

3.6.2. MIGOCE Overall Architecture

OO upper-CASE tools can help users to analyze and design systems using OO methodologies. In these CASE tools, the expression of OO concepts are performed by building OO models and displaying OO diagrams on a screen. The representation of graphs is the key issue of many OOA and OOD CASE tools. Therefore, these tools often have strong graphical functions and ran under Direct Manipulation (DM) environments. But, as mentioned before, most of current upper CASE tools are methodology-

dependent. The OO diagrams can only support one or two special notations. CASE tools have so far not been meeting the expectations of users and management.

An OO upper-CASE tool kit is proposed in the thesis. This is a Methodology-Independent Graphical OO CASE Environment (MIGOCE). To build such an environment, the key points are:

(1) To design notations that support a range of different methodologies, including the user's own defined notations.

(2) To form corresponding OOA and OOD diagraming environments.

Figure 3.9 represents MIGOCE major components.

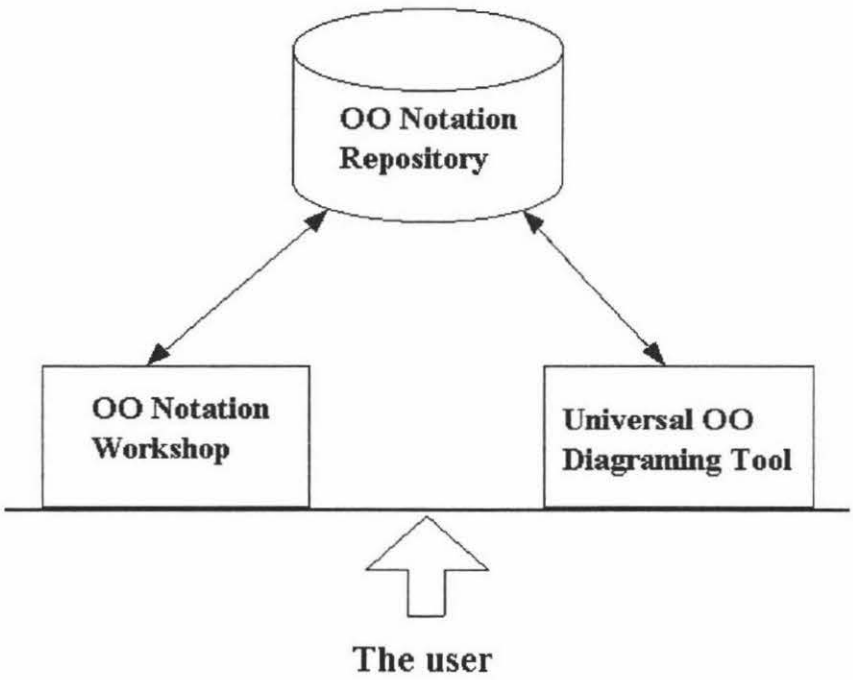


Figure 3.9. The major components of the MIGOCE system

The MIGOCE system consists of three parts — OO Notation Workshop, OO Notation Repository and Universal OO Diagraming Tool. A user can communicate with the MIGOCE system through the interface.

OO Notation Workshop

OO Notation Workshop is actually an OO graphical editor which has the following characteristics:

- It allows designing all kinds of notations supported by existing OO methodologies
- It allows the design of users' own defined notations
- Easy browsing of the existing notations
- Strong file management functions
- Flexible graphical functions

The user can use OO Notation Workshop to design arbitrary notations which are stored into OO Notation Repository.

OO Notation Repository

This is a notation database which contains many sets of notations. It allows the storage of information about all notations that are basis of forming the Object-Oriented analysis and design environment.

The two-way arrow between OO Notation Workshop and OO Notation Repository indicates the interactive communications between them. Once a notation is defined by the user, it can be stored into the repository; On the other hand, the user can extract some notations from the repository when he (or she) wants to modify them.

In addition, OO Notation Repository communicates with Universal OO Diagraming Tool. It can provide the diagraming tool with what the tool needs at all times. The two-way arrow between OO Notation Repository and Universal OO Diagraming Tool shows the data-flow between them. The new diagrams created by the diagraming tool also can be stored in the repository.

Universal OO Diagraming Tool

This is a graphical OOA and OOD environment that consists of some individual design windows. The user may complete his (or her) Object-Oriented analysis and design under

the corresponding environment that supports a certain methodology which the user needs.

Universal OO Diagraming Tool accesses the notations in OO Notation Repository to generate a certain OOA or OOD window which the user requires. For example, when the user wants to perform object-oriented analysis by using Coad/Yourdon methodology, he (or she) can choose the Coad/Yourdon's notation from the repository and draw the OO diagrams, adopting Coad/Yourdon's methodology.

Figure 3.10 illustrates the architecture of MIGOCE. The components included in OO Notation Workshop (OONW) are: *Notation File Manager, Notation Set Manager, Basic Shape Designer and Window Manager*. The functions of these components are:

- *Notation File Manager*: Managing notation files and communicating with the repository
- *Notation Set Manager*: Managing notation sets
- *Basic Shape Designer*: Designing basic shapes
- *Window Manager*: Window management

The components included in Universal OO Diagraming Tool (UOODT) are: *OO Methodology Selector, OO Diagram Creator, OOA/OOD File Manager and Window Manager*. The following are the functions of these components:

- *OO Methodology Selector*: Selecting the OO methodologies stored in the repository
- *OO Diagram Creator*: Creating OO Diagrams
- *OOA/OOD File Manager*: Managing OOA/OOD diagram files and communicating with the repository
- *Window Manager*: Window management

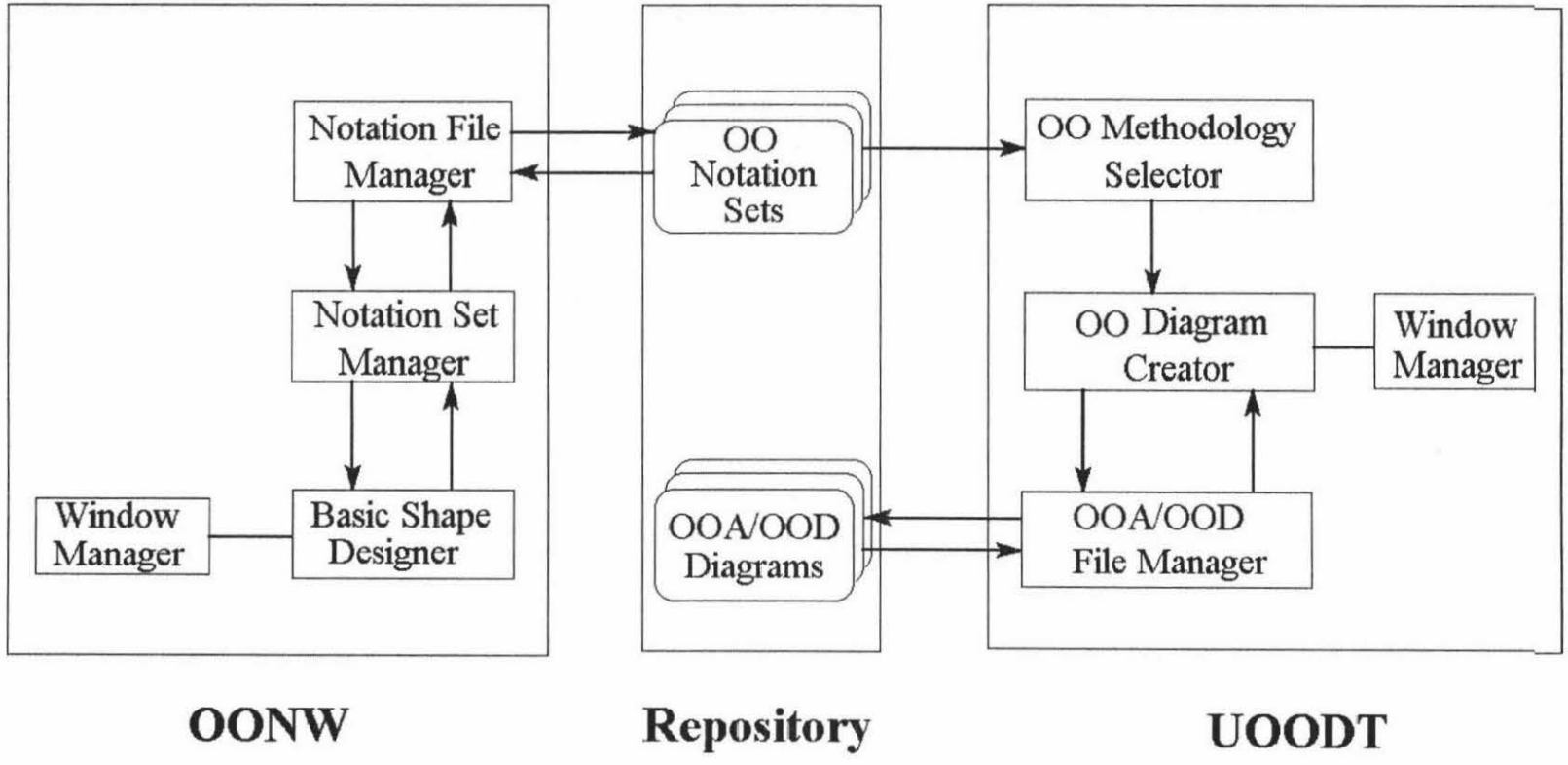


Figure 3.10. The Architecture of MIGOCE

Chapter 4

Task Analysis and User Interface Design

"Task analysis is the process of analyzing the way people perform their jobs: the things they do, the things they act on and the things they need to know" [Dix et al., 93].

Task analysis covers a range of techniques used by ergonomists, designers, operators and assessors to describe, and in some cases evaluate, the human-machine and human-human interactions in systems. Therefore, task analysis is a methodology which is supported by a number of specific techniques to help the analyst collect information, organize it, and then use it to make various judgements or design decisions. In recently years, more and more people have paid attention to a problematic area in the design of new system and new work places — Task Analysis (TA) and Task Modeling (TM).

As an important domain, Task Analysis (TA) is researched in order to enable computer systems to effectively serve users. The application of task analysis methods provides the user with a "blueprint" of human involvements in a system, building a detailed picture of that system from the human perspective. Such structured information can be used to ensure that there is compatibility between system goals and human capabilities and organization, so that the system goals will be achieved. Usage of explicit task analysis approaches should therefore lead to more efficient and effective integration of the human element into system design and operations.

Task analysis is a similar activity to requirements analysis as practised in systems analysis and design. Actually, task analysis involves the study of what an operator is required to do to achieve a system goal. Task analysis can be used when designing a system, evaluating a system design, or if a particular human-machine system performance problem has been "targeted" to be analysed and resolved. Task analysis can be undertaken at any stage in the life cycle of a system and exits during the whole process of the system design.

Task analysis can help in decisions about where to automate processes, how to determine user's requirements, and how to train staff and ensure efficiency. Task analysis can form the basis for the analysis of human errors in systems, and can be used in incident or

accident investigations, to define what went wrong and to help identify remedial measures, thus to enhance the safety of systems. Task analysis can be also used to identify maintenance support tools and systems of work. Specially, Task analysis has played a very important role in the design of HCI. It help designers to analyse user's job and achieve successfully the user's goal.

4.1. User Interface Design of OO CASE Tools

Human-computer interfaces are the physical surface and facilities, between the human user and the computer, providing the medium through which they can connect and interact; the physical (visual, audio, tactile) means, methods, and patterns that support human-computer interaction [Treu, 94a].

Interaction involves at least two participants: the user and the system. Both are complex, and are very different from each other in the way that they communicate and view the domain and task. The interface must therefore effectively translate between them to allow the interaction to be successful. If the actions allowed by the system correspond to those intended by the user, the interaction will be effective. Psychologists are insistent on this point: interactive system design should be *user- centered* and "early and continual focus on the user" [Shneiderman, 93].

4.1.1. The Role of Task Analysis

Task analysis is potentially the most powerful method in the field of Human-Computer Interaction (HCI) either for evaluating systems or for producing requirement specifications.

The task analysis in the design of OO CASE tool user interface is "user centered". Users bring tasks to the system which have been defined within their occupational role. Task cognition impinges upon the design of a computer system in the degree to which it defines the required functionality of the system, in the extent to which working practices in the external task domain must be accommodated within the internal system domain and the degree to which task-specific assistance can be given.

In general, most users treat the system as a "black box". They know what want to achieve and they know which interface operations achieve that goal, but they have little

conception of the precise effects of commands upon the internal states of the computing system. The system designer should link the user's conceptual model with conceptual objects within the computer system. In another words, it is hoped that the user's conceptual understanding is complete and totally consistent with the design model employed by the system designer.

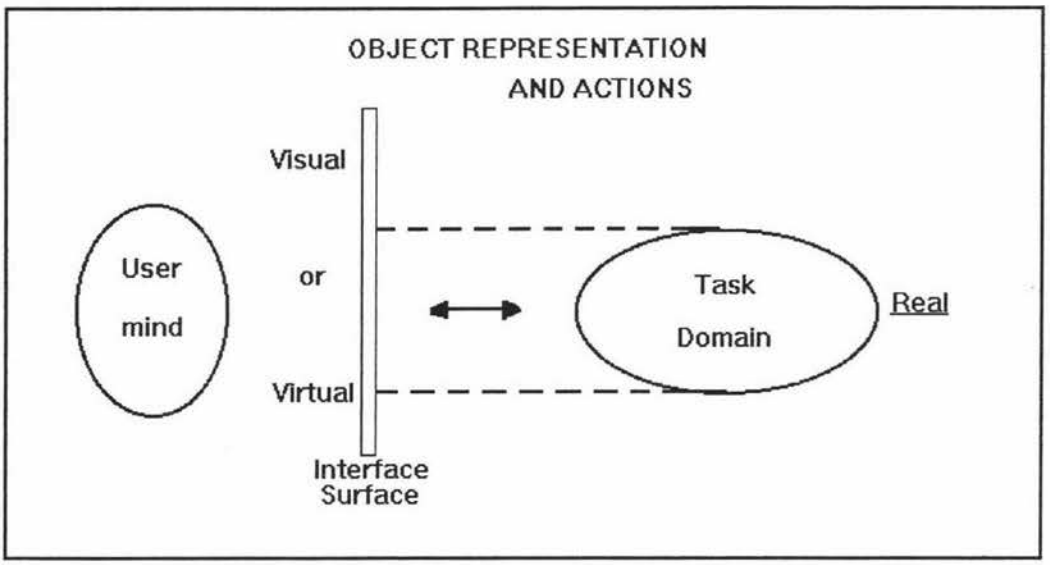


Figure 4.1. Behind the Visible Surface [Treu, 94a]

As Figure 4.1 shows, in the processes of designing OO CASE tool interfaces, designers take the user's real requirements into account through task analysis. Task cognition, therefore, is related to objects and states external to the system. Many Objects can be represented by easily recognizable, visual means that the interface surfaces. They can take on pictorial, iconic, symbolic, graphical, diagrammatic, and other forms. The main contributions of task analysis methods in the design of HCI are:

(1) Applications at different stages of system design

- User requirements — During the initial stages of design, task analysis methods are needed which capture the overall basic dimensions of current and future user tasks.

- Conceptual design — At this stage, the designer should map the task description onto the detailed design of the computer system interface.

- Evaluation of conceptual design — Once the interface design is complete, it is possible to evaluate the demands it makes upon users. The availability of the interface is evaluated by the designer and the user.

- Documentation and "reverse engineering"

After the design of the system has stabilized, the task analysis information may be utilized in the development of documentation. The documentation can be tutorial materials and is also likely to be used during the processes of system maintenance. Finally, fully detailed task analysis documentation would be valuable for the process of "reverse engineering" in order to evaluate the interface again and improve it.

(2) Task analysis in task-system mapping

There are several things should be done by the designer for task-system mapping:

- Define task functions
- Analyze information input and output requirements
- Identify task objects
- Adopt task methods

4.1.2. User Interface Styles

Interface can be considered as a form of dialogue between a computer and a user. The common styles of dialogue are:

***Menus*:** The structure of a menu-driven interface is based on the hierarchical tree, which allows selection from a large number of choices with a relatively small number of options in each menu and few levels of the hierarchy.

***Dialogue boxes*:** Dialogue boxes are information windows. They are used by system to bring the user's attention to some important information, possibly an error or a warning used to prevent a possible error.

***Direct manipulation (DM)*:** It is well known that WIMP (Window, Icon, Menus, Pointer) is called DM environment. The essential characteristic of such dialogues is that some kind of direct representation of the task is presented to the user by the system, with

the result that the desired operation or command is achieved by directly manipulating the virtual reality embodied in the display. The interface supports the user's task by portraying a realistic "virtual world" on the screen. The interface presents a metaphor of the application to provide the context for interaction. As this should be based on the users' domain model, it should be compatible with their expectations.

WIMP environments often have many advantages as the following:

- Explicit action: the user points at and manipulates objects directly on the screen.
- Immediate feedback: the results of the user's actions are immediately visible.
- Task analogy: interaction matches the user's conceptual model of how the system should operate and the display shows pictures of familiar objects. It is very suitable for task analysis and implementation.
- Graphic tool available: many graphic design tools are available under WIMP environments. This save users' time to complete the design of systems.

The major disadvantages of direct-manipulation dialogues are that they require complex and expensive hardware to support a suitable graphics display, and substantial software development to implement the chosen metaphor realistically.

4.1.3. Overview of Techniques for Task Analysis

Three important kinds of techniques of Task Analysis should be mentioned:

(1) Task decomposition

Task decomposition is an analysis method by which a task often is split into sub-task and is completed in order. Hierarchical Task Analysis (HTA) is a typical approach of task decomposition, which prompts the analyst to establish the conditions when various subtasks should be carried out in order to meet a system's goals. HTA was proposed originally by Annett and Duncan [Annett & Duncan, 67] as a method of analysis decomposition. HTA is concerned with establishing an accurate description of the steps that are required in order to complete a task. It was initially developed for training applications, but has since been developed and applied in a number of other contexts. It can be used to deal with specific issues, such as interface design, work organization, the development of operator manuals and job aids, training and human error analysis.

(2) Knowledge based techniques

Knowledge based techniques are analysis methods based on the knowledge about a task which the user want to perform. So, the objects and actions are defined and classified according to the knowledge in a given domain. Well-known appoaches of knowledge-based task analysis include Task Analysis for Knowledge Description (TAKD) [Johnson et al., 84], Task Knowledge Structures (TKS) [Johnson et al., 88], Knowledge Analysis of Task (KAT) [Johnson and Johnson, 90] developed from TKS, etc.

(3) Entity- relation based analysis

Entity-relation based analysis is a technology which associated with Entity-Relationship (E-R) modeling [Chen, 76], and more recently object-oriented programming. It emphases the relationships between actions and objects. The entities chosen for task analysis will be those which are expected to be represented in the resulting computer system.

4.2. Task Knowledge Structures and Knowledge Analysis of Tasks

Knowledge Based Analysis has emerged as an important aid to early design in human computer interaction. It provides an information source from which design decisions can be made, and a basic for evaluating designed system. Task Knowledge Structures (TKS) is a technique of knowledge-based task analysis. Based on the TKS theory, a method called Knowledge Analysis of Tasks (KAT) has been developed by Johnson and Johnson (1990).

4.2.1. Task Knowledge Structures

TKS has been developed by Johnson et al.[Johnson et al., 88] to allow user tasks to be modelled and designed. A TKS represents the different types of knowledge that users are expected to recruit and use in task performance.

TKS is a method of cognitive task analysis. It is concerned with informing the design process through the application of cognitive theories. The Task knowledge that people possess is an important subset of their total knowledge[Johnson, 92]. This knowledge should be taken into account in the analysis, design and development of software systems.

The TKS theory holds that task knowledge is represented in a person's memory and can be described by a TKS which is assumed to be activated during task execution [Johnson, 92]. Contained within TKSs are goal-oriented and taxonomic substructures: goal-oriented substructures represent a person's knowledge about goals and enabling states, subgoals, plans and procedures; taxonomic substructures represent knowledge about the properties of task objects and their associated actions.

Components of a Task Knowledge Structure

A task is an activity that is undertaken by one or more agents to bring about some change of state in a given domain. Agents can include people, animals, or machines. Tasks can be grouped together in many ways. One grouping of tasks is in terms of a *role*. An agent assuming a particular role is expected to carry out the set of tasks associated with that role [Johnson, 92].

Each task is modelled by a TKS and collections of tasks are modelled by related TKSs. A TKS consists of the following components (See Figure 4.2.):

Goals: the state of affairs that the successful performance of a task can produce, i.e. desired state changes in the domain of application.

Subgoals: goals which, when achieved, contribute to the achievement of a further or higher order goal.

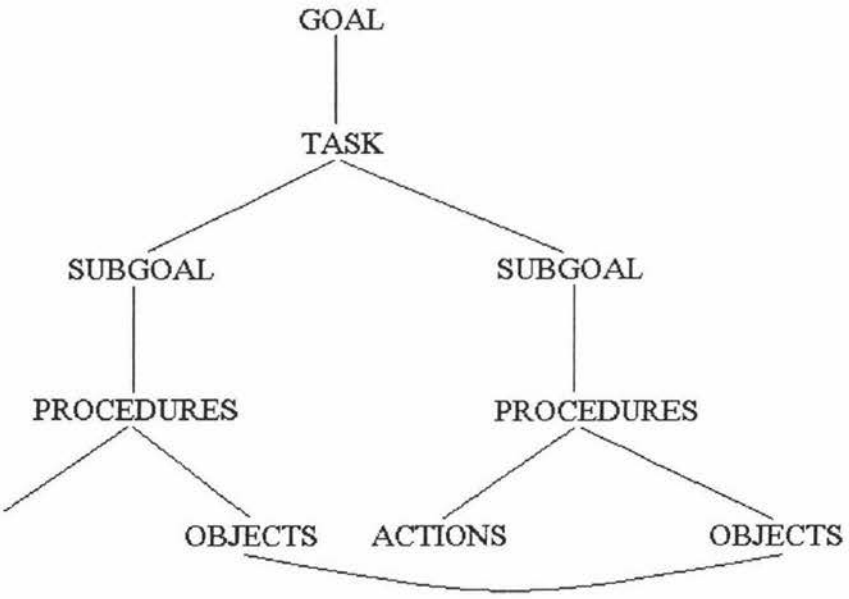


Figure 4.2. A schematic view of the components of a Task Knowledge Structure [Johnson, 91]

Procedures: executable activities that produce changes of state and result in a goal or subgoal being achieved.

Actions: units of a activity that make up the executable procedures, each procedure has one or more actions.

Objects: entities upon which or by which the actions are performed and upon which the procedures, subgoals and goals are predicated.

A TKS model includes a *goal-oriented substructure*, *task procedures* and a *taxonomic substructure*. The *goal-oriented substructure* describes the goals and subgoals identified within the TKS. There must be an appropriate procedure to allow a goal or a subgoal to be executed. *Task procedures* are executable forms of the task. They rely on knowledge of objects and actions, which, when combined, constitute a given procedural unit. Properties of objects in a given task context are represented in a *taxonomic substructure*. A taxonomic substructure includes the analysis of actions, objects, the relations between them, and the procedures in that which actions and objects are included and how they are used.

Relations

A TKS is related to other task knowledge structures by a number of different relations. The two forms of relations need to be mentioned:

Within-role relations: One form of relation between TKSs is in terms of their association with a given role that a person may assume. Within a role each task to be performed will have a corresponding TKS.

Between-role relations: A second form of relation between TKSs is in terms of the similarity of tasks across different roles. A task may be part of the collection of tasks associated with more than one role.

Representativeness and Centrality

Representativeness is concerned with the instances and their relations to the class, and is a matter of degree rather than an all-or-none property. *Centrality* is concerned with critical points in the task at which success or failure is determined [Johnson, 92].

Using TKS, it is important to note that each task is to satisfy a particular goal and that there may be many layers of subgoals. Having constructed a TKS model the analyst has identified a number of important properties of users' task knowledge of benefit to the system designer. Showing the relations between tasks and roles provides the designer with a view as to the different kinds of tasks that the system should support by virtue of common task-role properties, and which different roles might expect to have access to the same task functions and to those task functions which are specific to particular roles [Johnson, 89].

4.2.2. Knowledge Analysis of Tasks

Knowledge Analysis of Tasks (KAT) is a method based on the TKS theory. KAT identifies the elements of knowledge represented in a task knowledge structure. KAT includes the following key points:

Collecting task data: A variety of knowledge-gathering techniques of can be used to obtain different types of information about a task. These techniques of collecting data include protocol analysis, observations, questionnaires and experimental methods, etc.

Analyzing collected data: After obtaining adequate data based on knowledge elements of a task, an analysis of those data is required in terms of TKS components.

Building task models: a task knowledge structure is produced to model people's knowledge in terms of goal structure, procedural substructure and taxonomic substructure.

4.3. A User Task Model

User task models identify the user's concept of how a task is constructed in terms of its functions and operational sequence [Preece & Rogers et al., 94]. These models attempt to discover how much users know about the system in terms of its operation and what their expectations are about how it will work. The establishment of user task models enable a developer to analyze, design and implement a system according to the user's real requirements.

The TKS theory and the KAT method have been adopted for building the task model of a methodology-independent graphical OO CASE environment (called MIGOCE, see Section 3.6), because:

- The TKS task model has both an analytical and descriptive component that enables existing users and computers to be analyzed in terms of the tasks they perform and also allows models of proposed users and computers to be explicitly described and hence reasoned about [Johnson, 91]. TKS can bridge the gap between the knowledge in users' mind and the design of computer systems.

- The TKS model includes both static and dynamic analysis: the *goal-oriented substructure* gives a static goal-subgoal hierarchy; *task procedures* provide dynamic executable processes of the tasks, in which actions are called upon objects. For a whole system, both high-level analysis and low-level design are needed. The designer can build the model starting with the top goal that derives subgoals. Whereas a *taxonomic susstructure* gives the lower level description of objects.

• KAT is concerned with identifying knowledge relevant to the task. It provides a way how to gain the knowledge about a task and steps for building the TKS model. The KAT method can help system designers to identify how much information should be provided in designing systems, and help users to identify what users expected to have available to them at any time.

As an application of the TKS theory, a user task model of the MIGOCE system has been established following the rules of the KAT method. The steps of building the task model are:

(1)Collecting task data

The knowledge about designing the MIGOCE system comes from current Object-Oriented Development, OO CASE literatures, and analysis of the results of using existing tools. As mentioned in chapter 3, the MIGOCE system is a methodology-independent graphical OO CASE environment. MIGOCE has two tools: OO Notation Workshop and Universal OO Diagraming Tool. The connection between them is a database: OO Repository.

(2)Analyzing task data and building task model

Figure 4.3 illustrates the high-level user task model based on TKS (See Figure 4.2.). *Using MIGOCE* is the goal the user want to achieve. It derives three subgoals *OO notations management, OOA/OOD diagrams management* and *OO repository management. OOA/OOD diagrams management* derives four subgoals *Create new OO diagrams, Modify existing OO diagrams, View OO diagrams* and *Print OO diagrams. Create new OO diagrams* is decomposed into two procedures *Draw OO notation icons* and *Edit OO notation icons.*

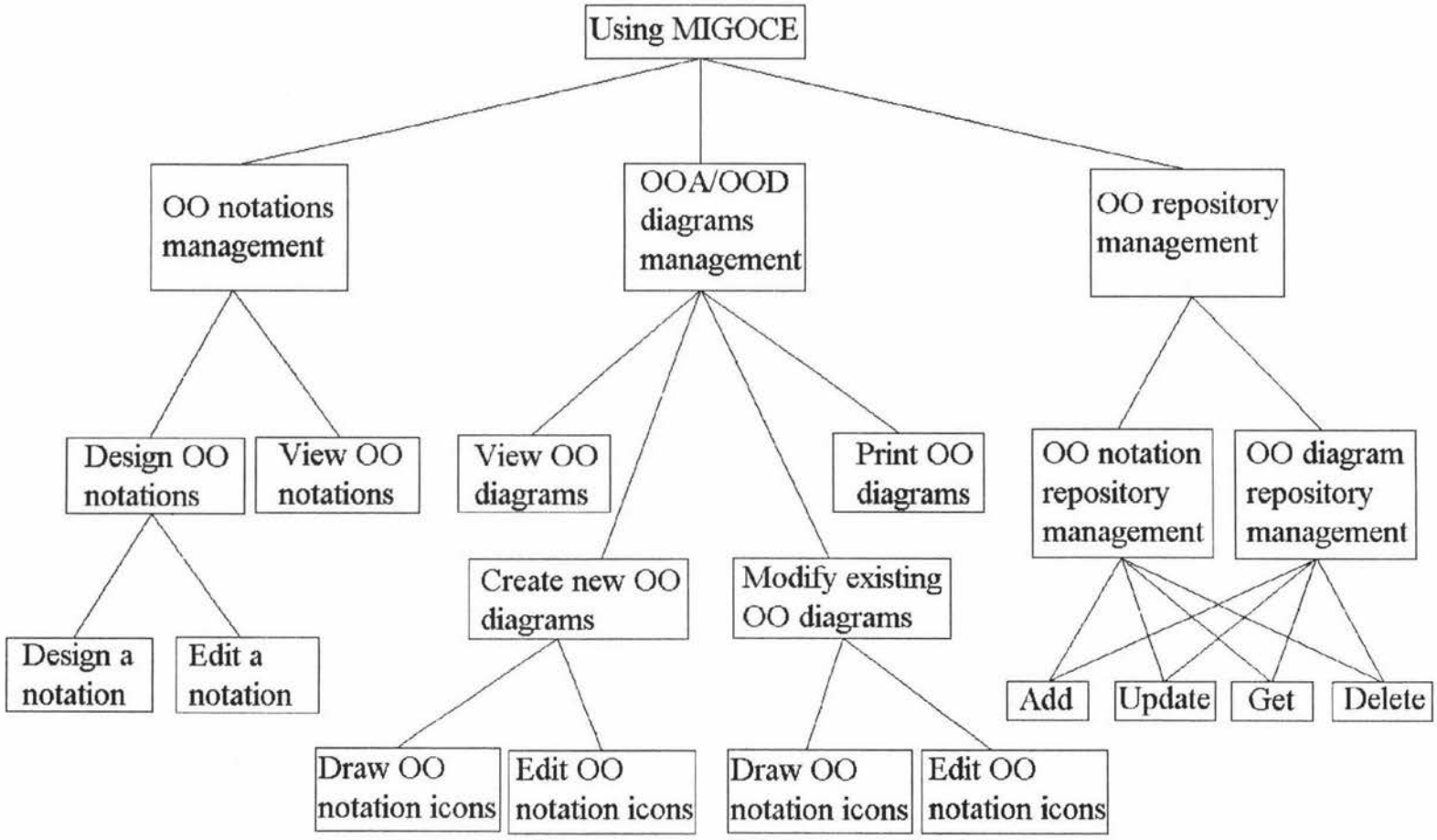


Figure 4.3. A user task model for the MIGOCE system

Goal-oriented substructure*

Goal: *Using MIGOCE*

Subgoal: *OO notations management*
- Subgoal: *Design OO notations*
  - Subgoal: *Design a notation*
  - Subgoal: *Edit a notation*
- Subgoal: *View OO notations*

Subgoal: *OOA/OOD diagrams management*
- Subgoal: *View OO diagrams*
- Subgoal: *Print OO diagrams*
- Subgoal: *Create new OO diagrams*
  - Procedure: *Draw OO notation icons*
  - Procedure: *Edit OO notation icons*
- Subgoal: *Modify existing OO diagrams*
  - Procedure: *Draw OO notation icons*
  - Procedure: *Edit OO notation icons*

Subgoal: *OO repository management*
- Subgoal: *OO notation repository management*
  - Procedure: *Add a notation*
  - Procedure: *Update a notation*
  - Procedure: *Get a notation*
  - Procedure: *Delete a notation*
- Subgoal: *OO diagram repository management*
  - Procedure: *Add a diagram*
  - Procedure: *Update a diagram*
  - Procedure: *Get a diagram*
  - Procedure: *Delete a diagram*

---

* The goal-oriented substructure is based on the user task model for MIGOCE (See Figure 4.3) and is used for designing the user interface.

Task procedures

Task procedure is a procedure associated with an executable process. For example, during the *Edit OO notation icons* procedure, the process of copying a notation icon is: action *Select* is invoked upon the *OO notation icon* object, then the *Copy* operation is called on this object. The following are two examples of Task Procedures:

- Procedure *Edit OO notation icons* (steps):

  a. Select *Select Tool* if it is not active
  b. *select* a notation shape
  c. *Edit* the selected notation shape
  Cut
  or Copy
  or Paste
  d. Repeat step b and c if it is needed

- Procedure *Draw OO notation icons* (steps):

  a. Select *Draw Tool* if it is not active
  b. *Choose* the node icon from the OO notation set
  c. *Position* the copy of the selected node icon in the design window by clicking the left mouse button
  d. *Repeat* step c if many copies of the selected node icon are needed in an OO diagram

Taxonomic substructure

The taxonomic substructure of a TKS model is associated with low-level design. For the interface design of the MIGOCE system, it plays an important role of matching the user's tasks to visual components of the user interface, such as menu, icon, etc., by which the user can send messages to the system. The taxonomic substructure of MIGOCE will be discussed in Section 6.1. (GUI of OO Notation Workshop). The identified task objects are addressed and further modelled in Chapter 5.

Chapter 5

The Static and Dynamic Object Models of MIGOCE

In an object-oriented methodology, the static models of an application specify the classes within the application and the relationships that exist among those classes; the dynamic models express the behaviours specified by the requirements of the application and describe the messages that are passed between objects. The objects described by the dynamic models are instances of the classes from the static models which must support the behaviours demanded by the requirements and expressed in the dynamic models.

Applying the TKS theory and the KAT method has resulted in identification of the task objects. The Booch methodology has been adopted to represent the static and dynamic object models of MIGOCE.

5.1. Modeling OO Notation Workshop

Graphical OO notations are symbols for representing the OO concepts, e.g. classes, objects, relationships, and their modifiers. Different OO methodologies have their unique notations. OO Notation Workshop (OONW) is a window based graphical tool which can be used to build existing as well as define new OO notations. It supports methodology-independent OO Diagraming Tool by providing different notations through OO Notation Repository. Based on the notations stored in OO Notation Repository, the designer can select different OO notation sets to support different OO methodologies.

5.1.1. OO Notation Workshop Specifications

A graph editor is an interactive tool that presents a graph to the user pictorially and allows the user to edit the graph. Notation Workshop is an interactive graphic editor. With it, the user can create, modify, name, and edit (cut, copy, paste) notations. It has two modes, the design mode and the view mode. The user can easily change its mode. In

the design mode, two kind of tools exist: drawing tools and the selection tool. In the view mode, only the selection tool exists.

In the design mode, when a drawing tool is active, the user can create basic shapes using the current drawing tool; when the selection tool is active, the user can select a basic shape by clicking the left mouse button on it. A selected basic shape is indicated visually by displaying the control points (small squares) that appear along the edges of the bounding rectangle of it. A selected basic shape can be edited (move, copy, cut, resize) and its color, pen size and pen style can be changed. If an action produces an unwanted result when the user creates or edits a basic shape, s/he can reverse or *undo* the action before performing any other actions. The user can change the current pen color, pen size and pen style when a drawing tool is active or when the selection tool is active but the current selection is empty.

In the view mode, the user can select a notation shape, a selected notation shape is indicated visually by displaying the control points (small squares) that appear along the edges of it. A selected notation shape can be modified, edited (move, copy, cut, resize) or named.

The user can delete all the basic shapes from the design screen or delete all the notation shapes from the view screen. The user can zoom in or zoom out basic shapes and notation shapes.

All the notation shapes created using OO Notation Workshop can be stored for later retrieval, inspection, or modification.

5.1.2. The Class Hierarchy of the Basic Shapes

The basic shapes which are used to form graphical notations are drawing elements of OO Notation Workshop. These basic shapes are arc, curve, ellipse, line, polygon, rectangle, and rounded rectangle. Each basic shape has its bounding rectangle, position, and is displayed using its pen size, its pen type and its colour. The class hierarchy for the basic shapes is shown in Figure 5.1. On the top of the inheritance hierarchy is the abstract class *Basic Shape*. It defines the attributes & operations common for all graphical icons. At this level the attributes *Color*, *PenSize*, *PenType*, *Bound* & operations *GetSize()*, *GetBound()*, *GetPos()*, *QueryPenSize()*, *QueryPenType()*, *UpdateBound()*, *UpdateSize()*, *UpdatePosition()* are defined.

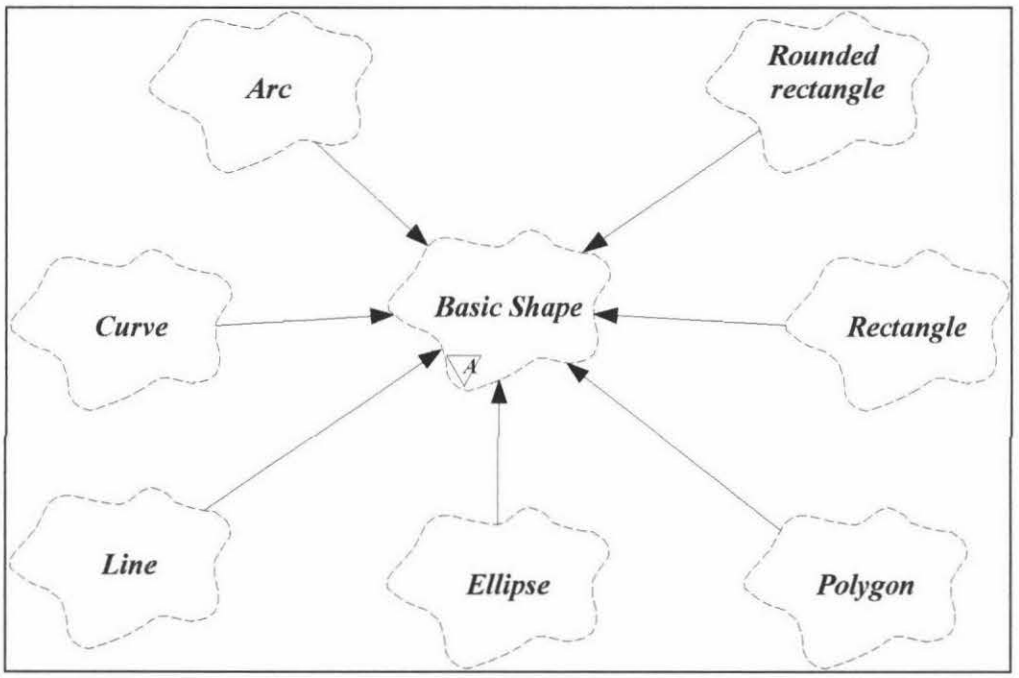


Figure 5.1. The class hierarchy for the basic shapes

Some basic shapes can be filled with a color, these basic shapes are ellipse, polygon, rectangle, and rounded rectangle. An abstract class called *Filled Basic Shape* is created to represent all basic shapes with an inside color. The modified class hierarchy for the basic shapes is shown in Figure 5.2.

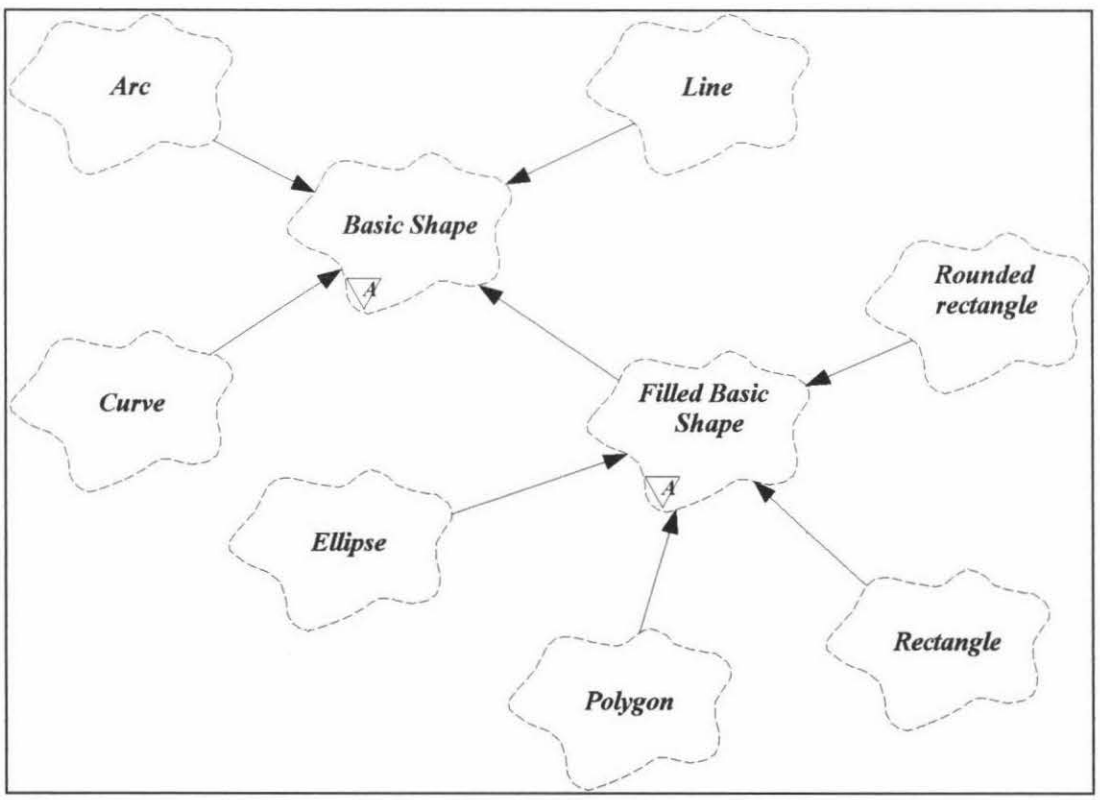


Figure 5.2. The modified class hierarchy for the basic shapes

In Notation Workshop, a notation shape is a collection of basic shapes. A new filled basic shape may partially or wholly overlap other basic shapes. When an inside color of a filled basic shape is white, sometime it is very useful when it is transparent. The abstract class *Filled Basic Shape* has defined three display methods: Filled, Overlap and Transparent. Figure 5.3 shows an example, where three filled ellipses share a common structure but each ellipse uses its own display method.

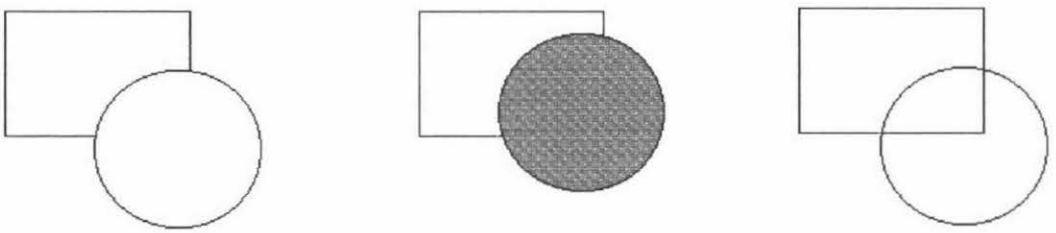


Figure 5.3. An example of three different filled ellipses

Each filled basic shape has three different display methods. The class hierarchy is further refined by the creation of an abstract class for each filled basic shape. Three subclasses which are derived from this abstract class. Each subclass shares a common structure of its superclass and has its own display method. The final class hierarchy for the basic shapes is shown in Figure 5.4.

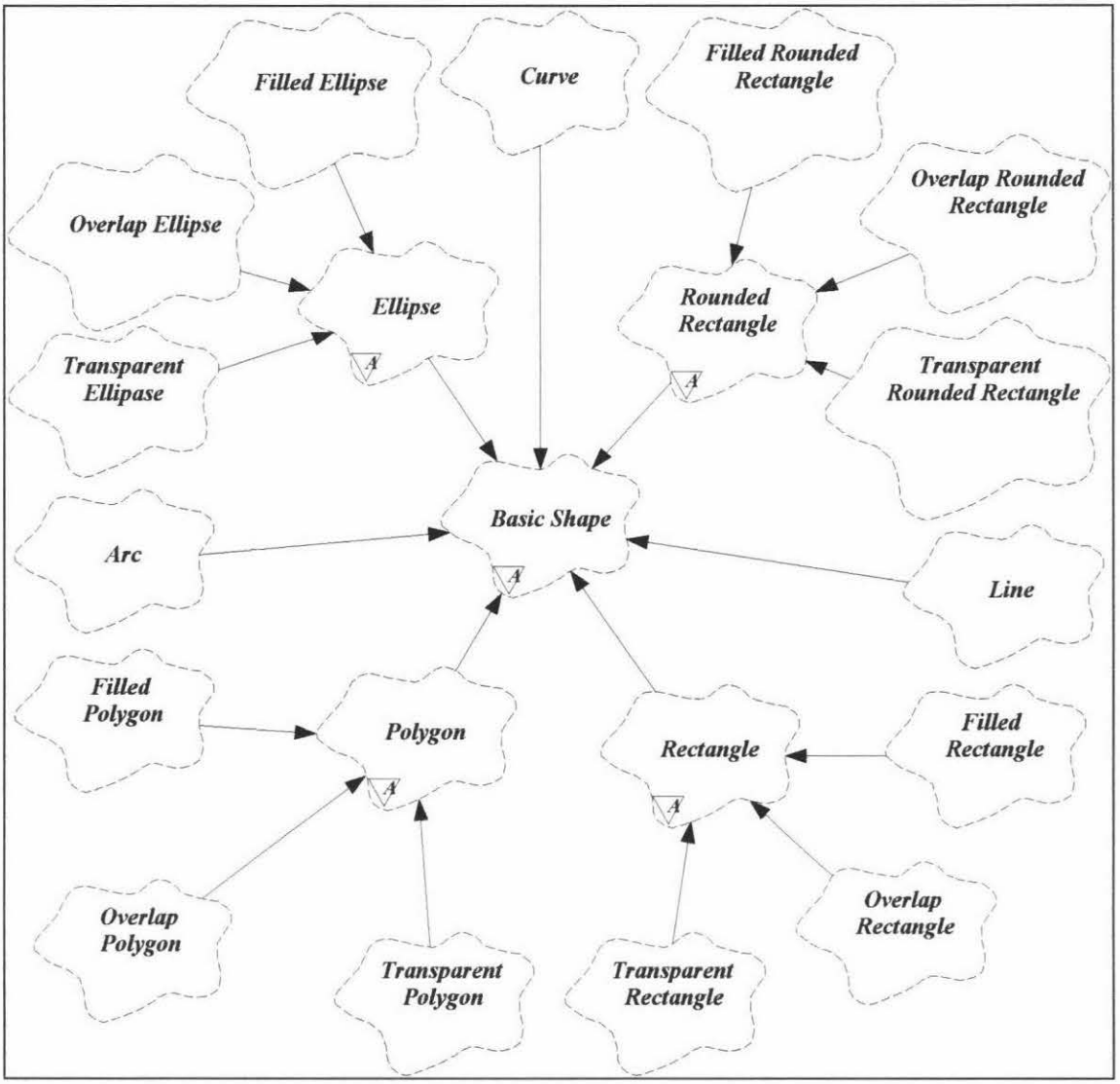


Figure 5.4. The final class hierarchy for the basic shapes

5.1.3. The Static Object Model of OO Notation Workshop

In Notation Workshop, OO Repository contains zero or more OO methodologies; an OO methodology contains a set of notation shapes and a notation shape is a collection of basic shapes. Figure 5.5 shows the basic static model of OO Notation Workshop.

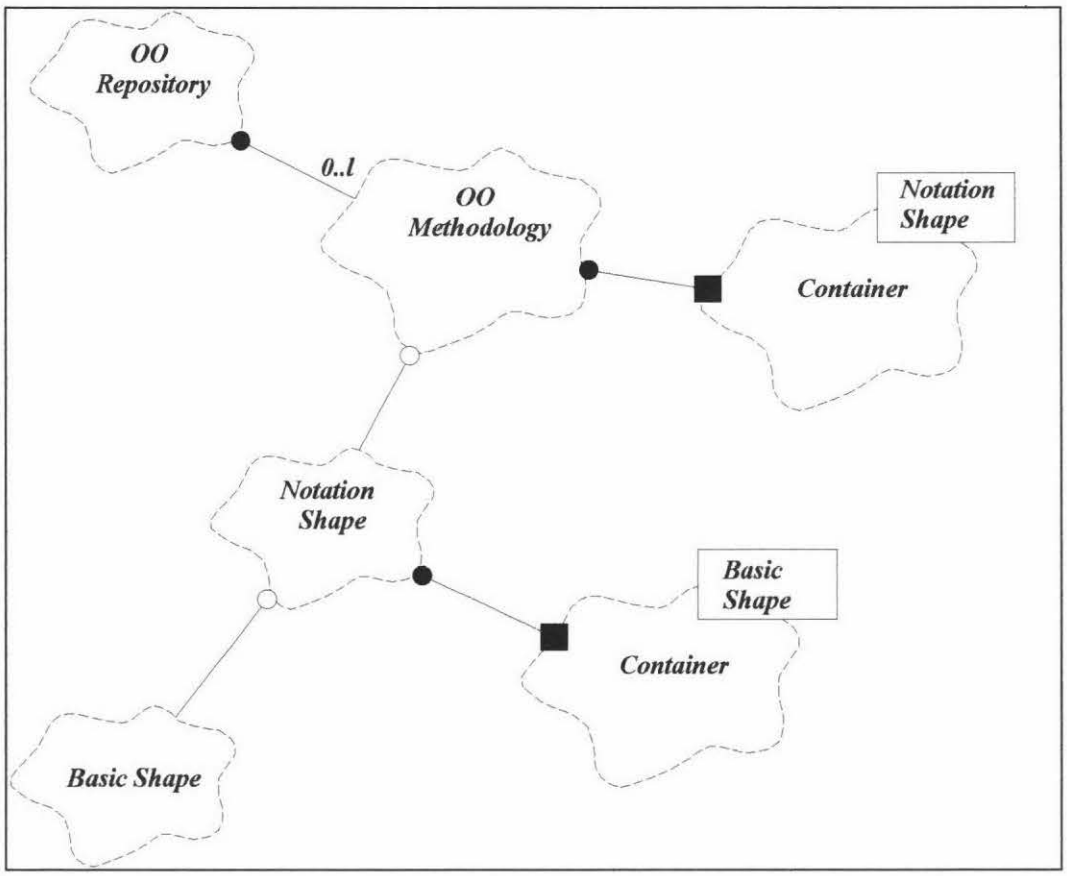


Figure 5.5. The basic static model of OO Notation Workshop

The refined static model of OO Notation Workshop is given in Figure 5.6. In order to separate data from the interface for that data, objects of the *TDrawDocument* class are used to contain many different types of data and provide methods to access that data, objects of the *TDrawView* class are used to form interfaces between objects of the *TDrawDocument* class and the user interface and to control how the data is displayed on the screen and how the user can interact with the data. Instances of the *TDrawView* class contain instances of the *TDrawDocument* class, the *TNotation* class, the *TBasicShape* class, the *TPoint* class and the *TPen* class; instances of the *TDrawDocument* class contain instances of the *TNotations* class, the *TNotation* class and the *TBasicShape* class; instances of the *TNotations* class contain a *container* that can hold an unspecified number of *TNotation* objects; instances of the *TNotation* class contain a *container* that can hold an unspecified number of *TBasicShape* objects; and instances of the *TBasicShape* class contain a *container* that can hold an unspecified number of *TPoint* objects. In C++, a *container* class is the template class and has its corresponding argument [Martin, 95].

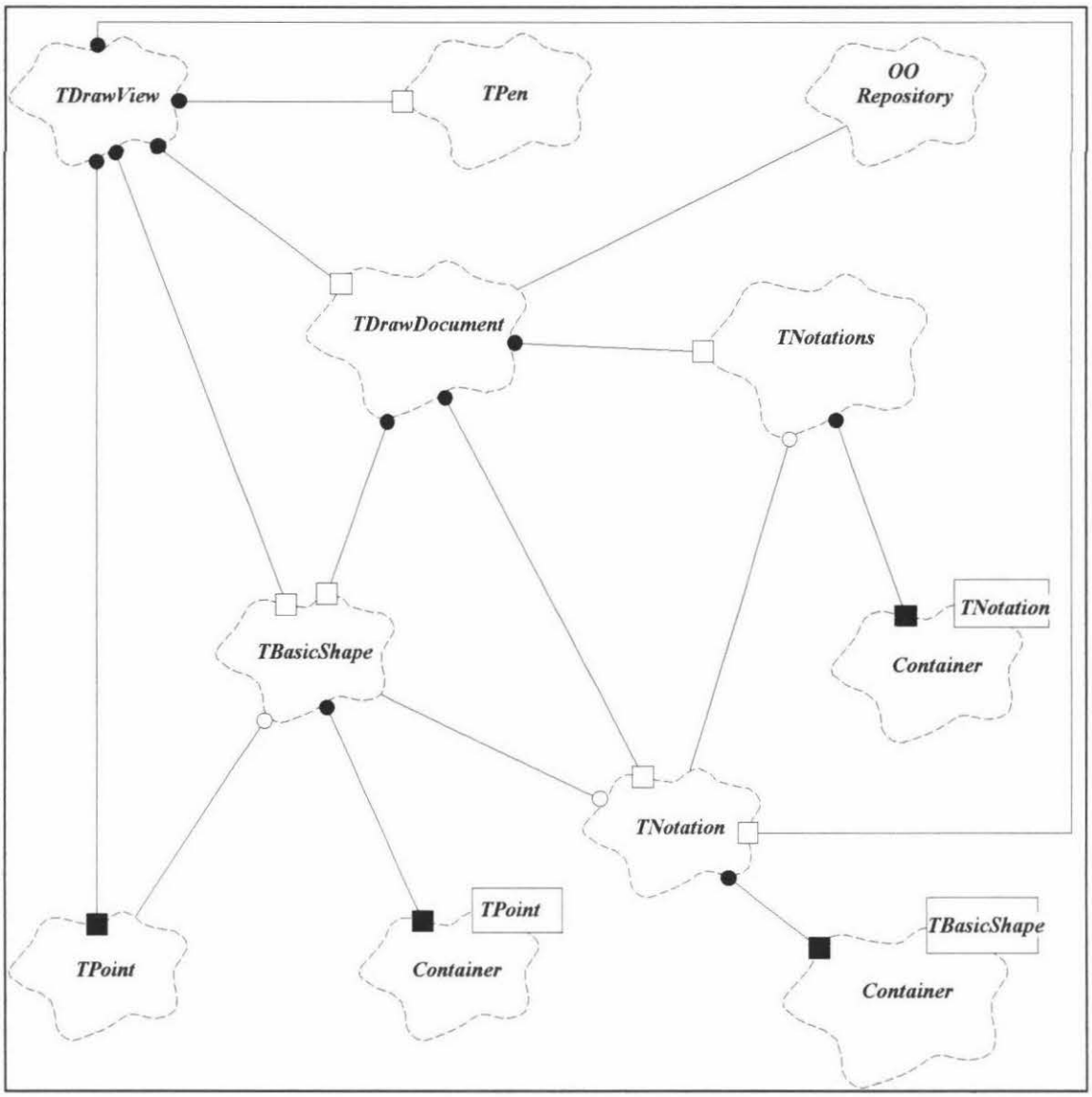


Figure 5.6. The refined static model of OO Notation Workshop

5.1.4. The Main Behaviours of OONW

Figure 5.7 shows the object diagram modeling the creation of a basic shape. Creating a basic shape requires the collaboration of several different objects. The *:TDrawView* object which is the instance of the *TDrawView* class invokes the operation *leftButtonDown()* upon the *BasicShape* object that is the instance of the *TBasicShape* class. Subsequently, the *BasicShape* object calls *Add(type)* upon the *:Tpoints* object.

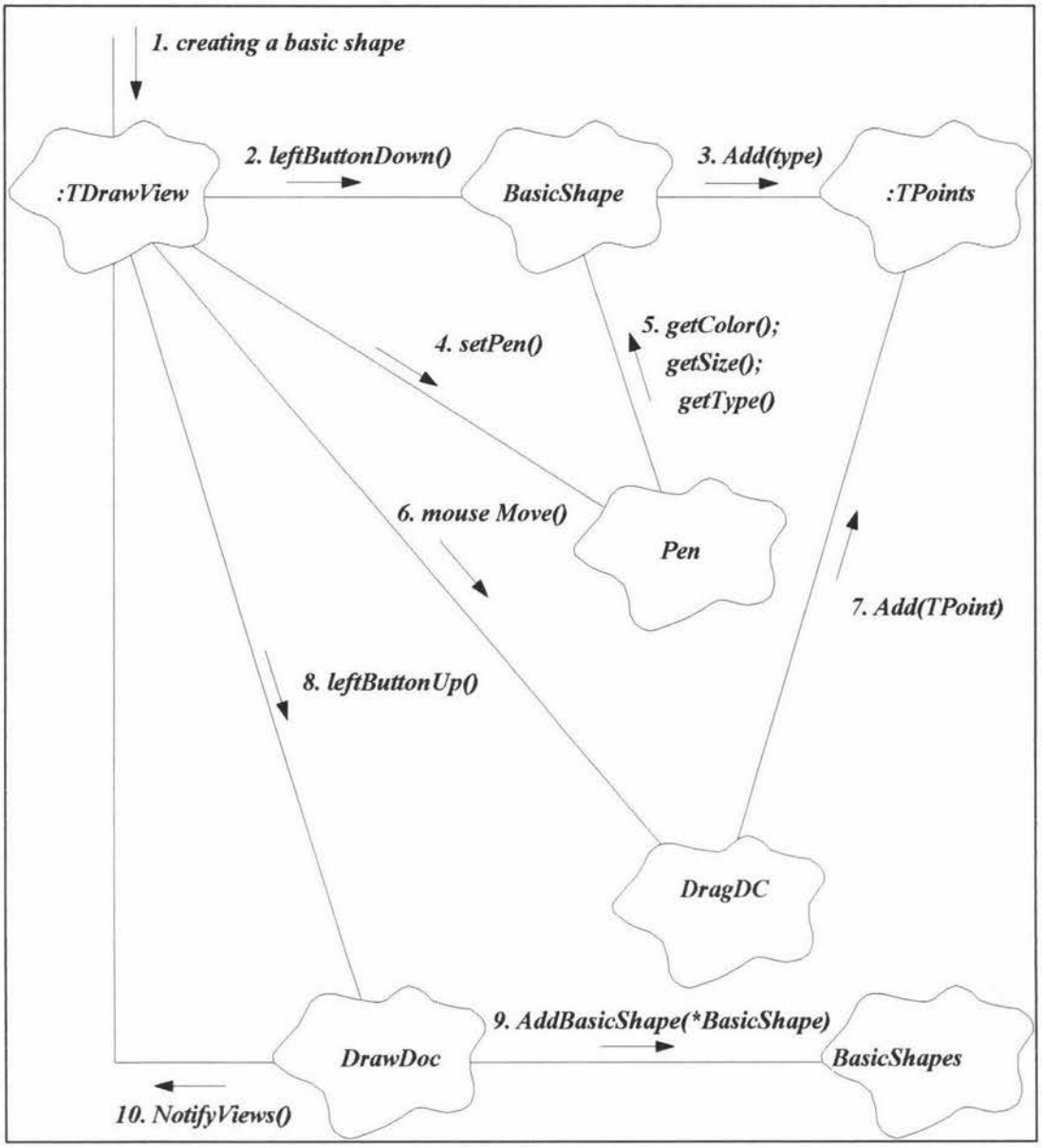


Figure 5.7. The object diagram modeling the creation of a basic shape

Before drawing a basic shape, the *:TDrawView* object invokes the operation *setPen()* on the object *Pen*. In turn, the object *Pen* calls *getColor()*, *getSize()* and *getType()* upon the *BasicShape* object. When the mouse is moving, the *:TDrawView* object sends the *mouseMove()* massage to the *DragDC* object, then the *DragDC* object adds the new point (current mouse position) to the *:Tpoints* object. When the button is up, the *:TDrawView* object invokes the operation *leftButtonUp()* upon the *DrawDoc* object which is the instance of the *TDrawDocument* class. In turn the *DrawDoc* object sends the *AddBasicShape(*BasicShape)* massage to the *BasicShapes* object that is the instance

of the *TNotation* class. Finally, the *DrawDoc* object calls *NotifyViews()* to update the *:TDrawView* object.

Figure 5.8 shows the object diagram modeling changing the color of a selected basic shape. The user sends message *ChangeColor* by selecting the menu item *Pen Color* or Bitmap button , the *:TDrawView* object receives *ChangeColor* message from the user interface, and sends message *GetColor* to the instance of *TChooseColorDialog,* then sends *SetColor* message to object *Selected* and *ModifyBasicShape* messages to the *DrawDoc* object. After sending *Find* and *Modify* messages to object *Basicshapes*. The *DrawDoc* object invokes the operation *SetDirty(ture)* upon itself, and sends message *NotifyViews()* to the instance of class *TDrawView* to update the color of the selected basic shape.

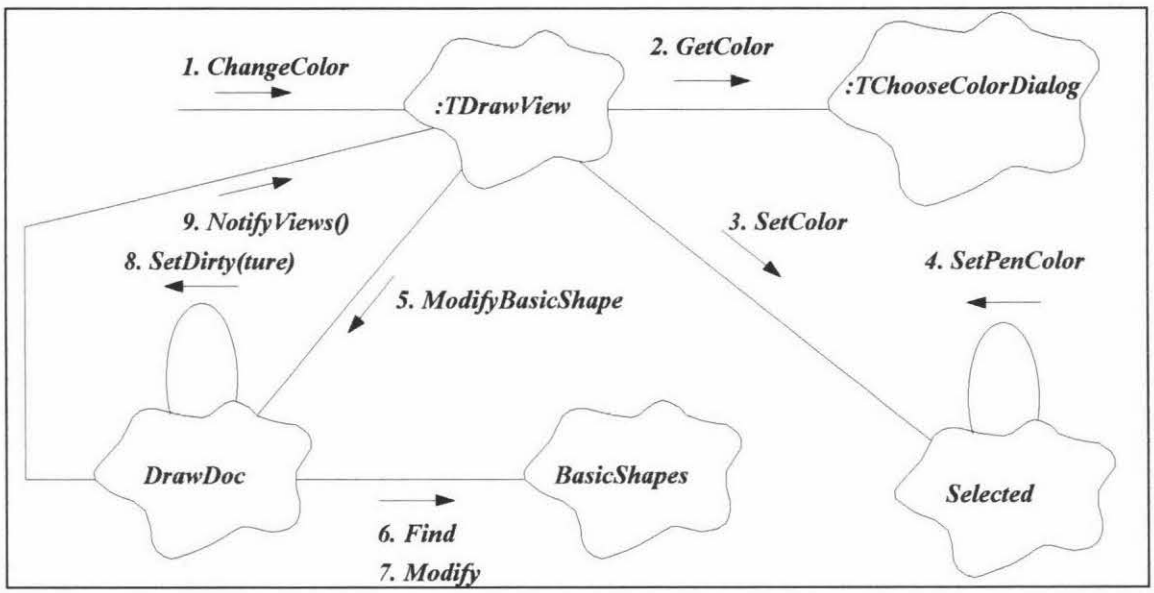


Figure 5.8. The object diagram modeling changing the color of a selected basic shape

The object diagram in Figure 5.9 models the operations followed when adding an OO notation to the *DrawDoc* object and the object diagram in Figure 5.10 shows the message passing used to save an OO notation set. The numbers indicate the general sequence in which these operations will be invoked.

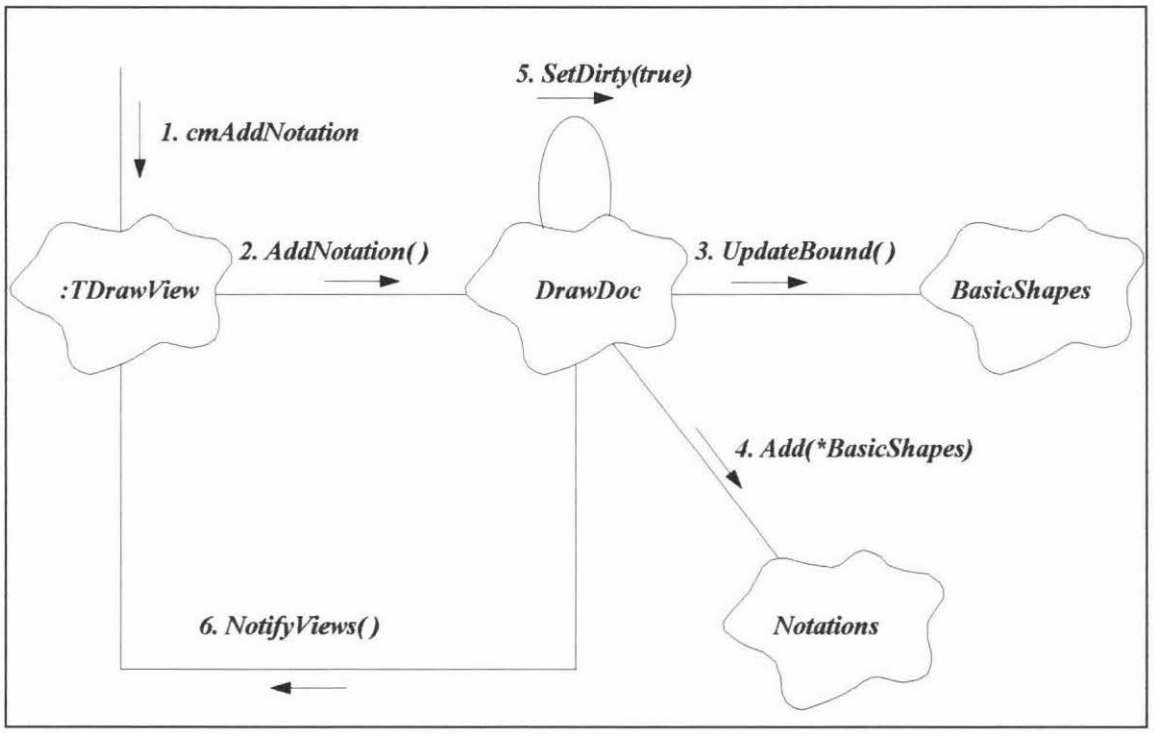


Figure 5.9. The object diagram modeling the creation of an OO notation

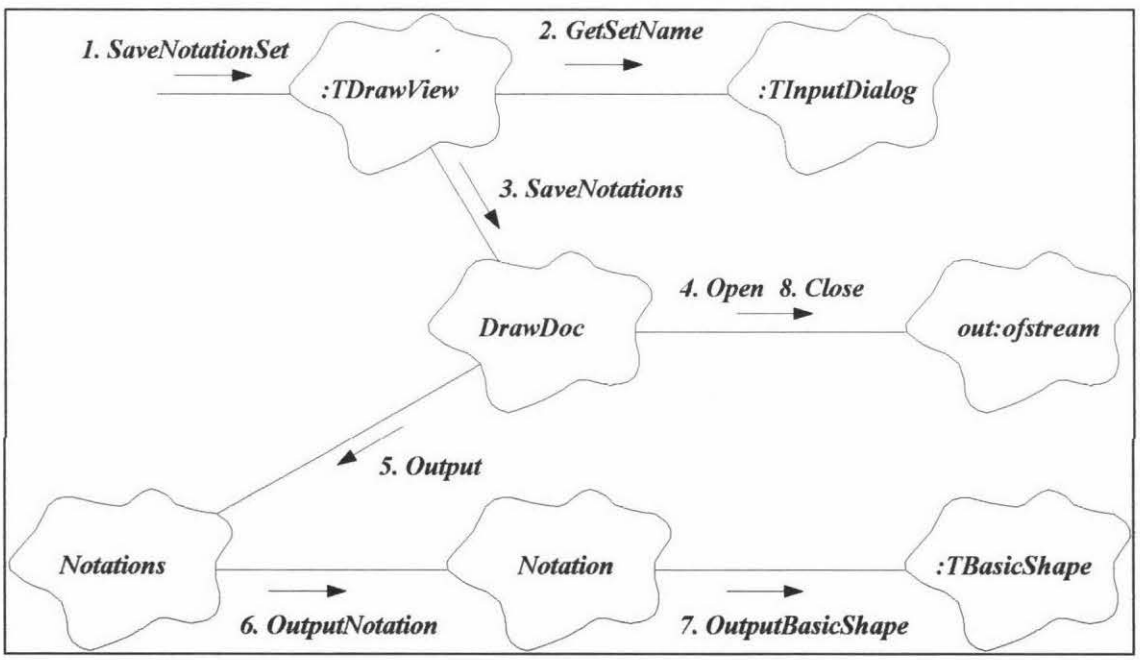


Figure 5.10. The object diagram modeling saving an OO notation set

5.2. Modeling Universal OO Diagraming Tool

In the MIGOCE OO development environment, Universal OO Diagraming Tool (UOODT) is a window based graphical design tool which can be customized to support different OO methodologies.

5.2.1. UOODT Specifications

Based on the OO notation sets stored in OO Notation Repository, UOODT can be used to draw graphical OO models of any OO methodology provided that the OO notation set of the methodology has been loaded to UOODT.

Each OO design window has its OO notation set selected by the designer. The OO notation set of the current OO design window can be changed at any time by loading another OO notation set.

Each OO design window has two modes: the design mode and the view mode. Each mode of an OO design window has its own tools.

In the design mode, the designer can draw node icons in the current design window based on the selected notation shape or create relationships between node icons based on the selected relationship. A node icon can be selected, a selected node icon can be moved, cut or copied. A cut or copied node icon can be pasted to the current design window or other design windows. The designer can also scale down or scale up a selected node icon, zoom in or zoom out all OO diagrams and delete all OO diagrams from the current design window.

In the view notation mode, the designer can select a notation shape from the current OO notation set. The selected OO notation shape can be cut or copied. A cut or copied notation shape can be pasted to the OO notation set of the current design window or OO notation sets of other design windows. A selected OO notation shape can be scaled down or scaled up.

The designer can save the OOD document in the active design window to disk or select a OOD document and then load it into the client window of UOODT. The client window can have several OO design windows at the same time.

5.2.2. The Static Object Model of UOODT

The static model of UOODT is shown in Figure 5.11, instances of the *TDrawView* class contain instances of the *TDrawDocument* class, the *TNotation* class, the *TPoint* class and the *TPen* class; instances of the *TDrawDocument* class contain instances of the

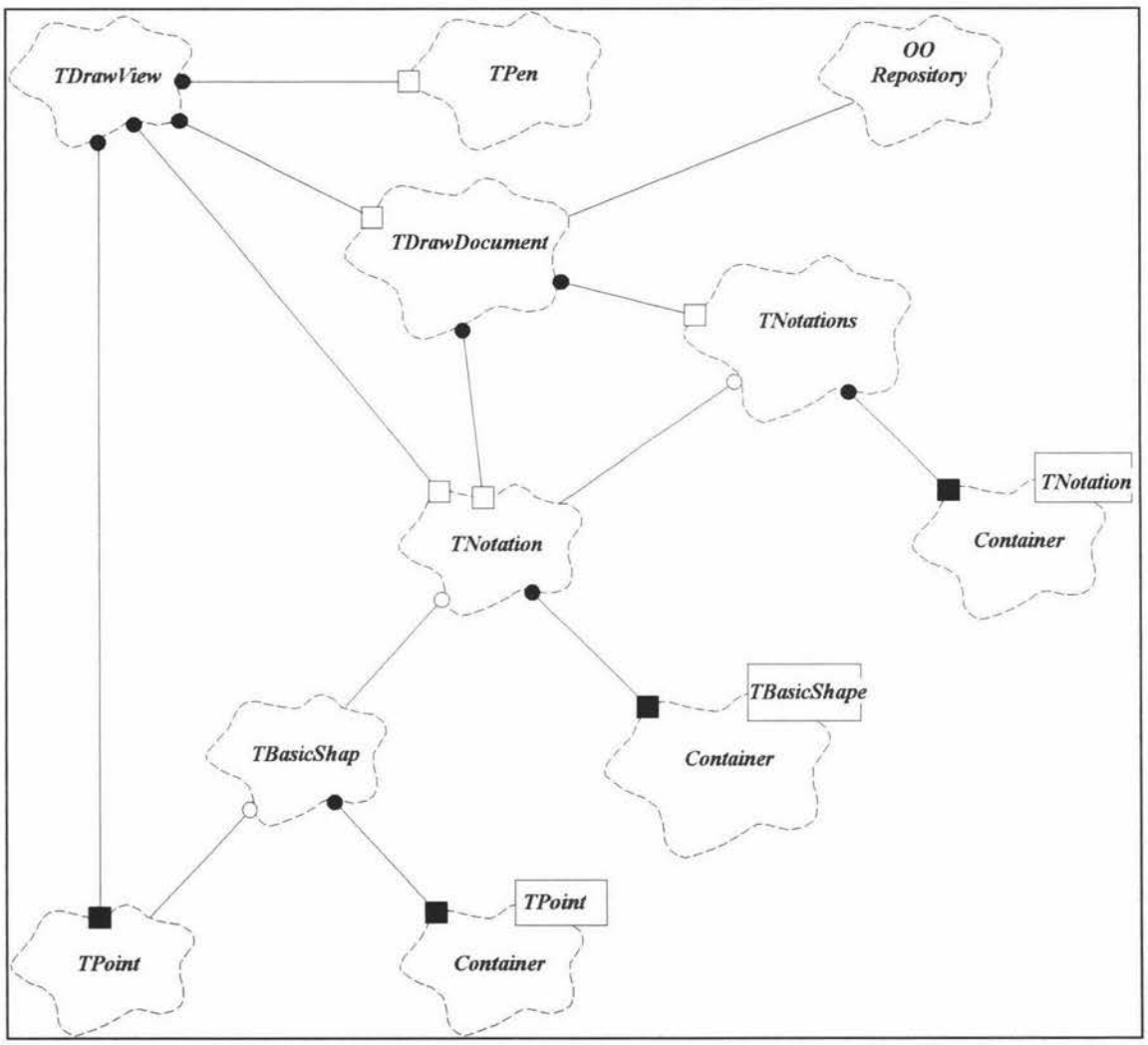


Figure 5.11. The static object model of UOODT

TNotations class and the *TNotation* class; instances of the *TNotations* class contain a *container* that can contain an unspecified number of *TNotation* objects; instances of the *TNotation* class contain a *container* that can contain an unspecified number of

TBasicShape objects; and instances of the *TBasicShape* class contain a *container* that can contain an unspecified number of *TPoint* objects. The *TBasicShape* class, the *TNotation* class and the *TNotations* class used in the static model of OONW are reused in the static model of UOODT. But the *TDrawView* class and the *TDrawDocument* class of UOODT have different specifications because they do not need to contain instances of the *TBasicShape* class and operations which are called upon these instances.

5.2.3. Object Diagrams in UOODT

The following figures illustrate main object diagrams in UOODT. Figure 5.12 shows the object diagram for loading a notation set for current design window.

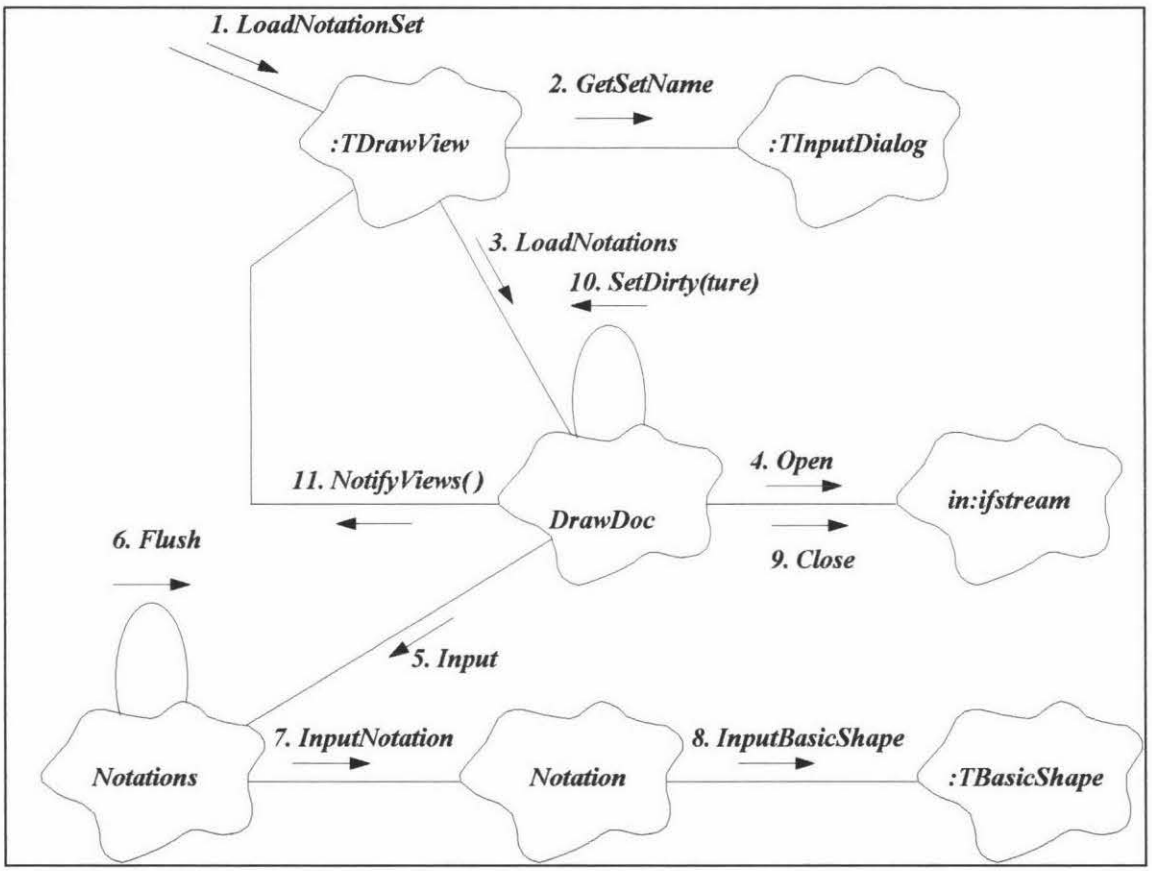


Figure 5.12. The object diagram for loading a notation set

In Figure 5.12, The user sends message *LoadNotationSet* by selecting the Bitmap button  , the *:TDrawView* object receives *LoadNotationSet* message from the user interface, and sends message *GetSetName* to the instance of *TInputDialog*, then sends *LoadNotations* message to the *DrawDoc* object. The *DrawDoc* object sends *Open*

message to the *in:ifstream* object and sends *Input* message to the *Notations* object, after invoking the operation *Flush()* upon itself, the *Notations* object sends *InputNotation* message to object *Notation* which in turn sends *InputBasicShape* message to the *:TBasicShape* object. Finally, the *DrawDoc* object sends *Close* message to the *in:ifstream* object and then invokes the operation *SetDirty(ture)* upon itself, and sends message *NotifyViews()* to the instance of class *TDrawView* to notify change.

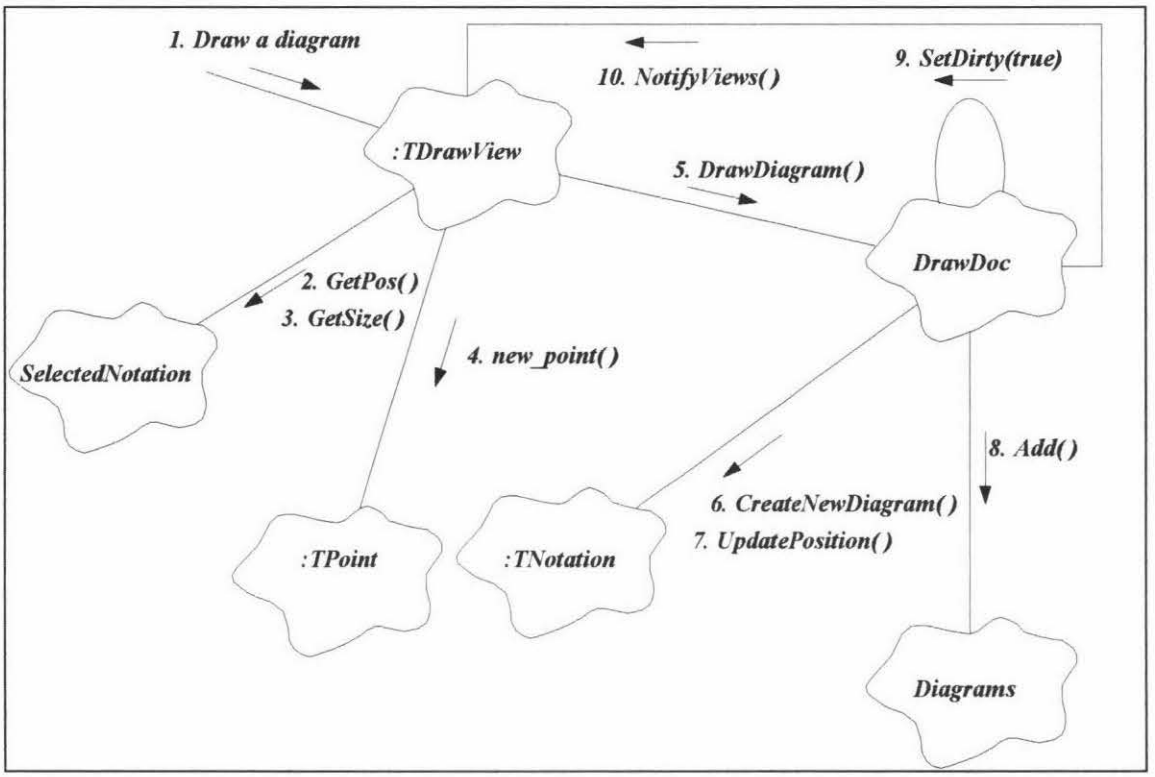


Figure 5.13. The object diagram for drawing a diagram

As mentioned before, UOODT has two modes: the view mode and the design mode. In the view mode, the user can select a notation shape from the notation set; in the design mode, the user can draw OO diagrams using the selected OO notation. Figure 5.13 shows the object diagram for drawing a diagram. The *Draw a diagram* massage is sent to the instance of class *TDrawView* by clicking the left button of the mouse on the design window. The instance of class *TDrawView* invokes *GetPos()* and *GetSize()* operations on the *SelectedNotation* object, the *GetPos()* operation returns the position of the selected notation shape, and the *GetSize()* operation returns the size of the selected notation shape. Based on the position and size of the selected notation shape and the clicking point of the mouse, the operation *new_point()* determines the position of the new notation shape. The *DrawDiagram()* massage is generated by the instance of class *TDrawView* and is sent to the *DrawDoc* object which is the instance of the

TDrawDocument class, the *DrawDoc* object invokes the *CreateNewDiagram()* and *UpdatePosition()* operations upon the instance of the *TNotation* class. The *DrawDoc* object invokes the *Add()* operation on object *Diagrams*. Finally, the *DrawDoc* object calls *SetDirty(ture)* upon itself, that means the file has been changed, and sends message *NotifyViews()* to the instance of class *TDrawView* to update the current drawing.

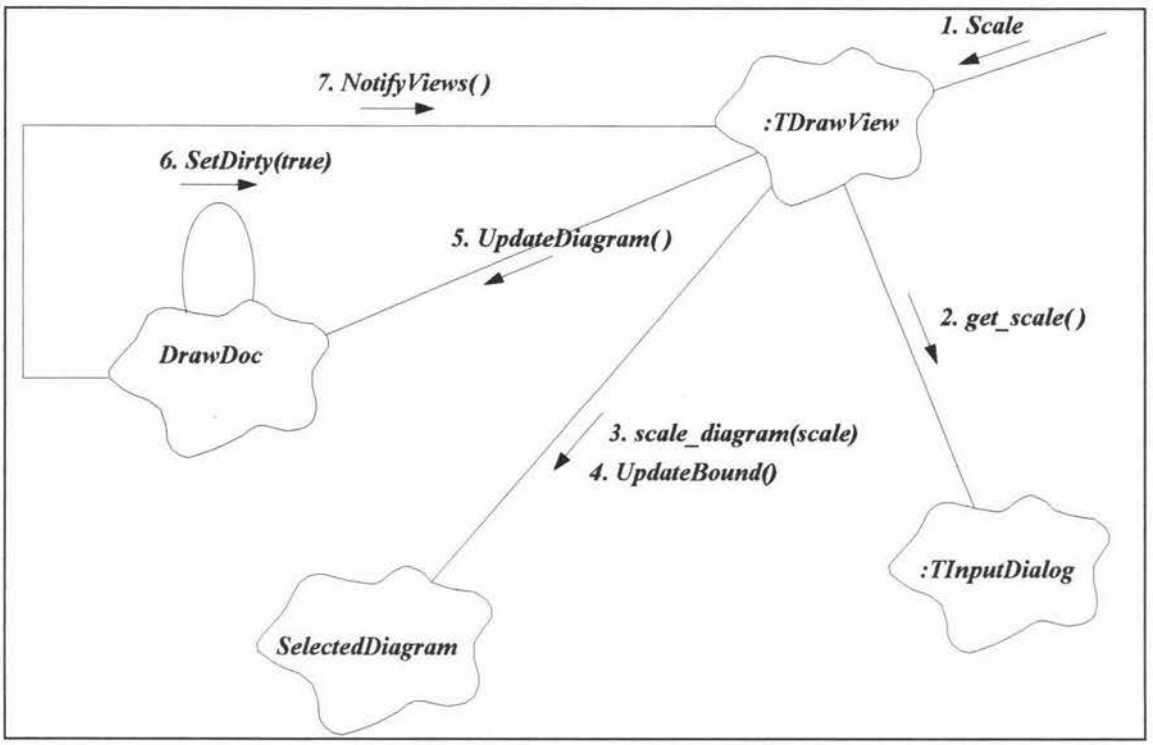


Figure 5.14. The object diagram for scaling a selected diagram

The object diagram in Figure 5.14 shows the operations in use when scaling a selected diagram. The diagram which needs to be scaled should be selected before the *Scale* message comes from the user interface. The instance of *TDrawView* invokes the operation *get_scale()* on the instance of *TInputDialog*, gets a scale from the user through a input dialog box, and then sends *scale_diagram(scale)* and *UpdateBound* messages to the *SelectedDiagram* object. The instance of *TDrawView* sends *UpdateDiagram()* message to the *DrawDoc* object. After updating its data, the *DrawDoc* object invokes the operation *SetDirty(ture)* upon itself, and sends message *NotifyViews()* to the instance of class *TDrawView* to update the current drawing.

Chapter 6

OO Notation Workshop

OO Notation Workshop (OONW) is a window-based design tool used by the user to create different OO notations. Different OO notations can be used by Universal OO Diagraming Tool to support different OO methodologies.

6.1. GUI of OO Notation Workshop

6.1.1. Users of OO Notation Workshop

Human-computer interfaces should be built to suit the needs of people; consequently, it is important to discover what types of people will be using the interface.

Ideally a system should be sufficient for users' needs and compatible with their experience. It is the analyst's task to capture that knowledge and build the new system to be as compatible as possible with the users' expectations [Sutcliffe, 1995].

Users of OO Notation Workshop will be developers who have the knowledge about OO analysis and design. Some of these people are object-oriented experts. They know the problem-solving domain very well, know what they want and how they proceed to achieve their goal. The graphical user interface of OO Notation Workshop should meet the requirements of these users.

6.1.2. Task Analysis and Task Model

As mentioned in section 4.1.1., the key issues in task-system mapping are: Define task functions; Analyze information input and output requirements; Identify task objects and adopt task methods.

- Defining task functions:

A UI of an application can informally be defined as the part of the application that is in charge of the communication with a user. The rest of the application is a non interactive part, which provides the application functionality [Kovacevic, 92], as shown in Figure 6.1.

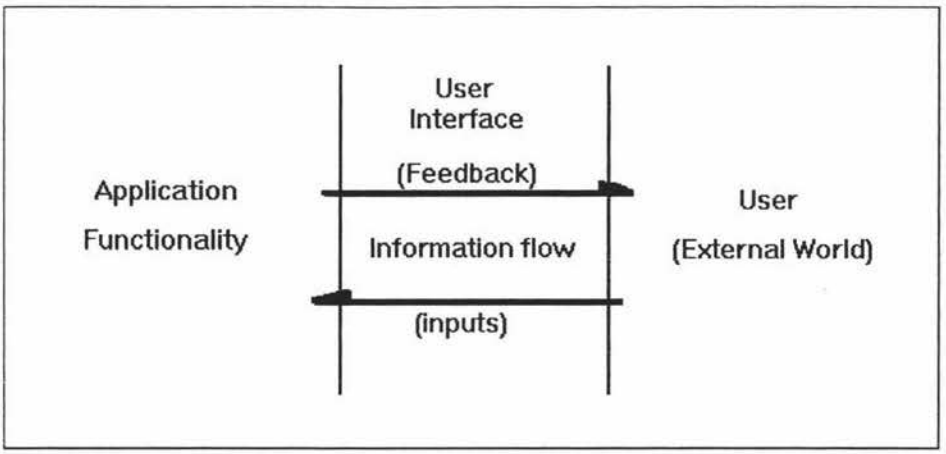


Figure 6.1. A user interface as an intermediary between a user and the application's functional part [Kovacevic, 92]

For OO Notation Workshop, its functions include creating, modifying, naming and editing notations. The purpose of building the interface is to perform the communication between the user and the functional parts of these applications.

- Analyzing information input and output:

Because WIMP environments have many advantages (see Section 4.1.2), this style of UI has been adopted for OO Notation Workshop. Usually when the information is input, the user selects an item on the menu or an icon on the tool bar by clicking the mouse; whereas the output is a display of corresponding graphs on the screen.

With the interface of OO Notation Workshop, the user may specify each unit of information in different ways, such as to type in all parameters, point at a position, select colour and fill pattern from a menu. Feedback from the system may also vary. OO Notation Workshop can provide feedback simply by displaying the object at the new position, or with new colour, as well as showing another pattern, etc.

- Identifying task objects and adopting task methods:

Before designing the interface, task objects should be identified. Because OO Notation Workshop is a graphical editor, processing graphs is the key issue. A graph consists of different shapes. Every shape also has its own colour, pen size, pattern etc. Therefore, shape, colour, pen size, and so on, can be considered task objects.

One of task analysis approaches — TKS (Task Knowledge Structures, see section 4.2.1.) provides us with a way to build interface task models. TKS has been adopted as a method to establish the task model.

The main task of OO Notation Workshop is to design OO notations. Based on the TKS theory discussed in section 4.2.1., the task model of the graphical user interface of OO Notation Workshop has been built and shown in Figure 6.2. This task model gives the process in which to design a notation is the user's goal.

Actually, the goal *Design a notation* in Figure 6.2 is a subgoal in figure 4.3. It derives two procedures *Create a new notation* and *Modify an old notation*. The following show the *Task procedures* and the *Taxonomic substructure* of the model:

Task procedures

Procedure *Create a new notation* (steps):

a. Decide (or select) a new basic shape
b. Draw a new basic shape
    Select color
    Select pen type
    Select pen size
c. IF more basic shapes are needed
    THEN go back to step a.
  IF design of the new notation is completed
    THEN add the new notation to the OO notation repository

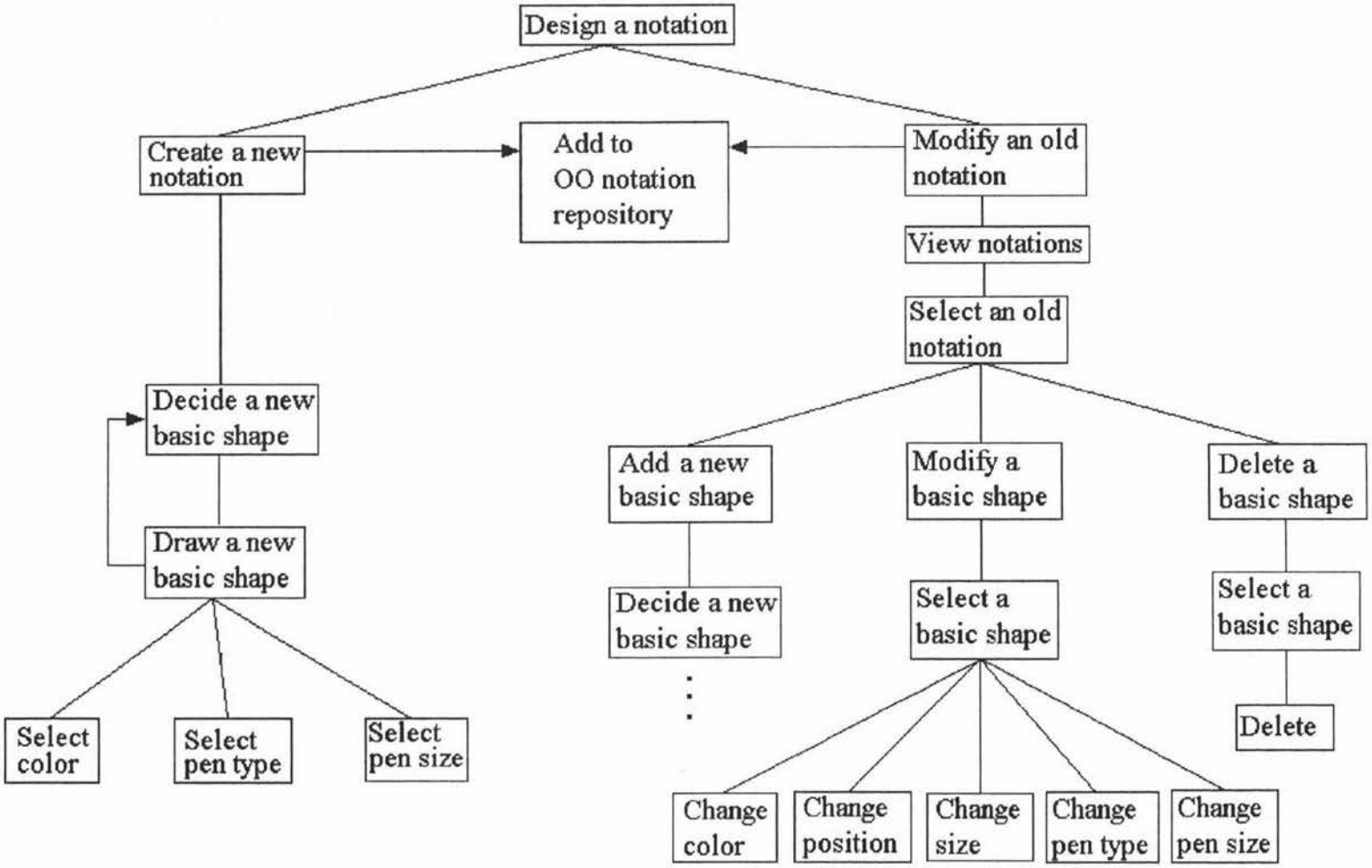


Figure 6.2. A task model for designing an OO notation

Procedure *Modify an old notation* (steps):

a. View notations
b. Select an old notation
   - Add a new basic shape, then go to the step a of the *Create a new notation* Procedure (Decide a new basic shape)

   or • Modify a basic shape → Select a basic shape (Change color, or position, or size, or pen type, or pen size)

   or • Delete a basic shape → Select a basic shape → Delete the basic shape
c. IF the modification of the old notation is finished
   THEN add the modified notation to the OO notation repository

Taxonomic substructure

The actions and objects relevant with procedure *Create a new notation are:*

Actions: *Deciding (or Selecting) which basic shape needs to be drawn*
Selecting color
Selecting pen type
Selecting pen size
Drawing a basic shape
Adding a notation to the OO repository

Objects: *Basic shape, Color, Pen type, Pen size*

The actions and objects relevant with procedure *Modify an old notation* are:

Actions: *Viewing notations*
Selecting an old notation
Adding a basic shape
Modifying a basic shape
Selecting a basic shape
Changing color, or position, or size, or pen type, or pen size

Objects: *Notation, Basic shape, Color, Size, Pen type, Pen size*

6.1.3. Visual Components of OO Notation Workshop

Menu design

To match the user interface to the task model, the visual components of UI enable the user easily to perform operations for achieving his (or her) goal. Summing up the actions and objects in the taxonomic substructure, the menu system of OO Notation Workshop are divided into 8 units: *File*, *Edit*, *Pen*, *BasicTools*, *Notation*, *Zoom*, *Window* and *Help*. The functions of these menu units (or groups) are as follows:

File: File management
Edit: editing graphs
Pen: Setting pen size, pen color and different pen types
BasicTools: Selecting a basic shape, drawing different basic shapes
Notation: Notation management
Zoom: Zooming basic shapes and notations
Window: Window management
Help: Helping the user to use OO Notation Workshop

Each unit consists of some menu items. For example, the *BasicTools* menu unit consists of *Select, Curve, Line, Polyline, Circle, Rect, RoundRect, Polygon, Diamond and Parallelogram* items which are used to determine the current tool. The menu commands of all menu groups of OO Notation Workshop are shown Figure 6.3.

The main window

All functions of OO Notation Workshop should be completely included in a main window. Taking account of the graphical features of OO Notation Workshop, the WIMP environment has been adopted to make it more attractive, easier and more intuitive to use. This main window is shown in Figure 6.4.

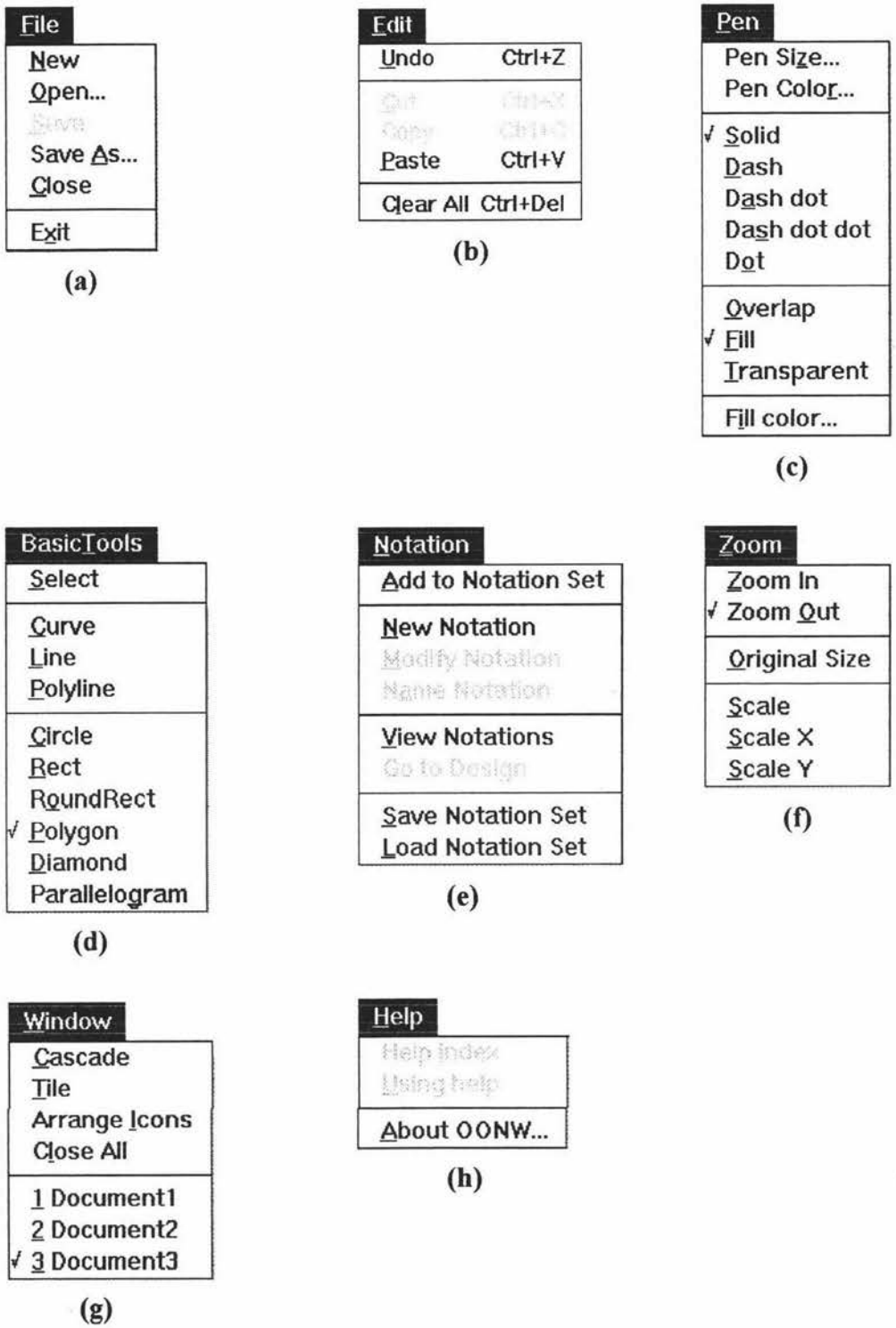


Figure 6.3. The group menu commands of OO Notation Workshop

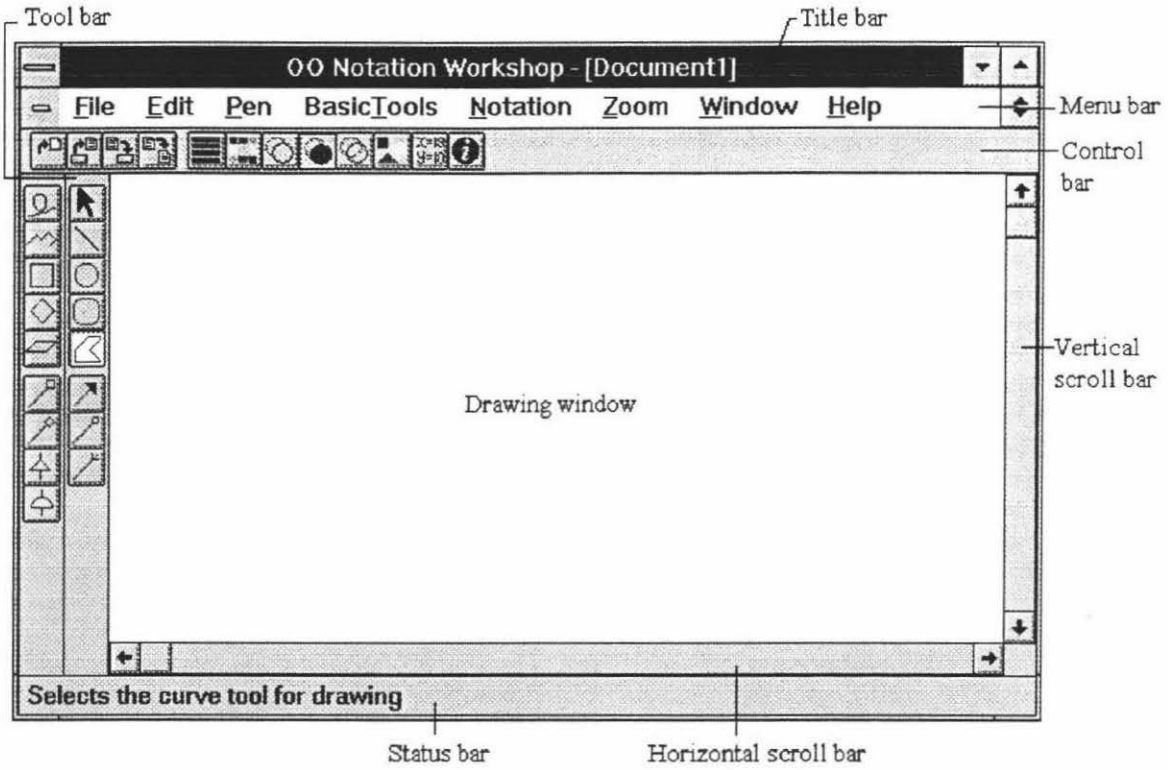


Figure 6.4. The main window of OO Notation Workshop

In Figure 6.4, the window border is the outside edge of a window, the user can change the window size by lengthening or shortening the border on each side of a window. The Maximize button can be used to enlarge the application window so that it fills the entire desktop; the Minimize button can be used to reduces the application window to an icon which is on the desktop.

The menu bar contains the group menus from which the user can choose menu commands. Not all group menus are available at all times. Some group menus , such as *Edit*, *Pen*, *BasicTools*, *Notation* and *Zoom*, appear on the menu bar when a notation document is opened. The details of some menu commands will be discussed in following sections.

The control bar and the tool bar use bitmap buttons to represent some of the menu commands. Clicking one of the bitmap buttons from the control bar or from the tool bar is a quicker alternative to choosing a command from the menu. Figure 6.5 shows the bitmap buttons used by OO Notation Workshop and their correspondent menu commands.

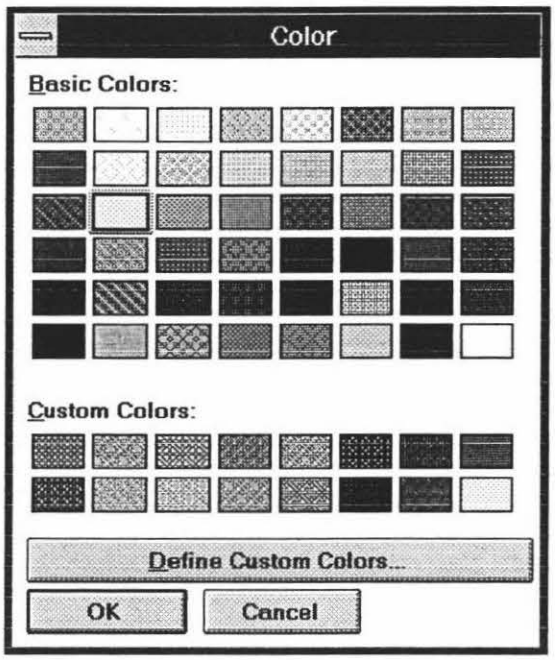
The status bar is used to display help messages when the user move the mouse pointer over a bitmap button or a menu command.

Bitmap button	Menu command
	New
	Open...
	Save
	Save As...
	Pen Size...
	Pen Color...
	Overlap
	Fill
	Transparent
	Fill color
	
	Select
	Curve
	Line
	Polyline
	Ellipse
	Rect
	RoundRect
	Polygon
	Diamond
	Parallelogram
	
	
	
	
	
	
	
	About OONW...

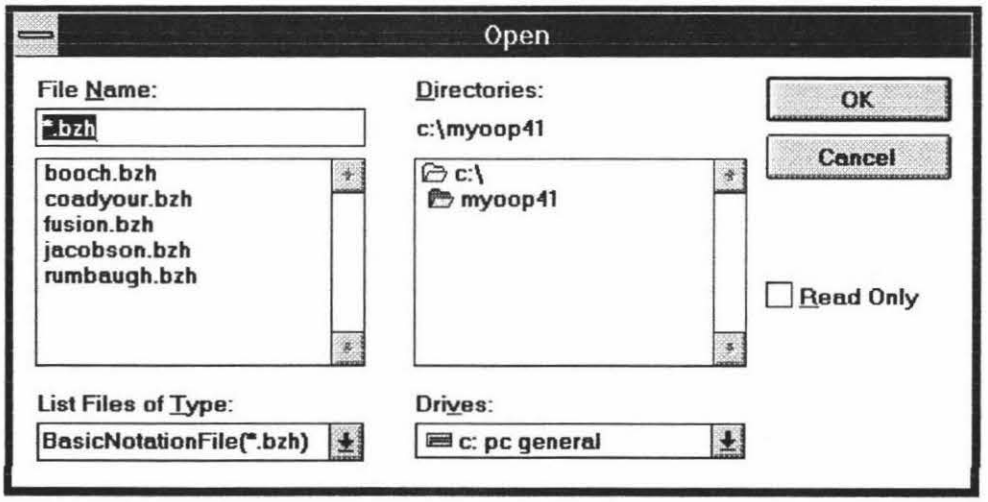
Figure 6.5. The bitmap buttons and their correspondent menu commands.

The scroll bars enable the user to control scrolling which is the movement of data in and out of the client area. It is a way for the user to see a notation document in parts if the design window cannot fit the entire notation document inside the client area.

The graphical user interface of OO Notation Workshop also includes some dialog boxes. A dialog box can be displayed in response to a menu selection, a bitmap button click, prompting, warning or error condition. Figure 6.6 shows some examples of the dialog boxes used by OO Notation Workshop.

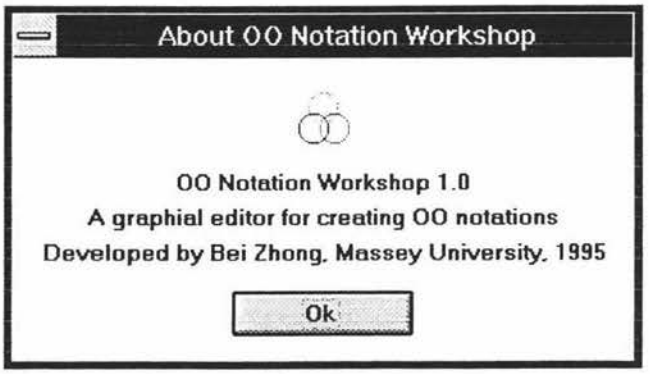


(i)

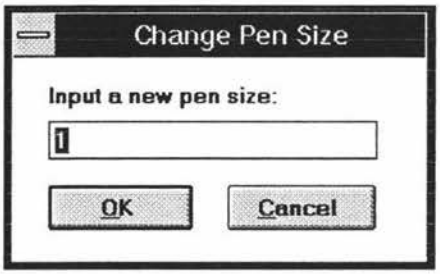


(ii)

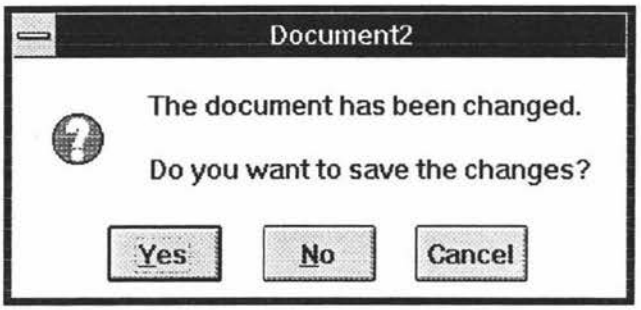
Figure 6.6a. Examples of the common dialog boxes.



(i)



(ii)



(iii)

Figure 6.6b. Examples of the custom dialog boxes

Examples of the common dialog boxes shown in Figure 6.6a are Windows' common dialog boxes: the color dialog box and the file open dialog box. The color dialog box (Figure 6.6a (i)) is used when the user want to select or create a new color (a new pen color or a new fill color) for drawing or when the user want to change the pen color or the fill color of a selected basic shape; The file open dialog box (Figure 6.6a (ii)) is used to open OO notation definition files. The file save dialog box (not presented in Figure 6.6a) is used to let users choose file names.

Examples of the custom dialog boxes shown in Figure 6.6b are the information dialog box (Figure 6.6b (i)), a input dialog box (Figure 6.6b (ii)) and a prompting dialog box (Figure 6.6b (iii)). The information dialog box shows information about OO Notation Workshop; the input dialog box is used to input a new pen size for the drawing tools; and the prompting dialog box is used to prompt the user to save OO notation definition when the user wants to close a design window but the definition of OO notation has been changed since it was loaded.

All dialog boxes used in OO Notation Workshop are modal dialog boxes. The menu commands used by OO Notation Workshop to invoke dialog boxes are *Open...*, *Save**, *Save As...*, *Pen Size...*, *Pen Color...*, *Fill Color...* and *About OONW...*; the bitmap buttons which can invoke dialog boxes are  ,  ,  ,  ,  , and  .

---

* Only when the menu command *Save* or bitmap button  is used to save a new, unnamed notation document, it invokes a dialog box to prompt the user for a file name.

6.2. Designing the OO Notation

As discussed in Section 5.1.2, an OO notation shape is a collection of the basic shapes, so the task of designing an OO notation shape is the task of creating and editing those basic shapes which are used to form this OO notation shape. The process of designing an OO notation shape includes four main steps: selecting the basic drawing tools; creating basic shapes; editing or modifying basic shapes and adding the new OO notation shape to the OO notation set.

6.2.1. Selecting the Basic Drawing Tools

OO Notation Workshop provides two types of basic drawing tools: the drawing tools for unclosed shapes and the drawing tools for closed shapes. The basic drawing tools for unclosed shapes are the curve tool, the line tool ,and the polyline tool; the basic drawing tools for closed shapes are the ellipse tool, the rectangle tool, the rounded rectangle tool, the polygon tool, the diamond tool, and the parallelogram tool. Figure 6.7 shows the possible basic shapes which can be drawn using the basic drawing tools.

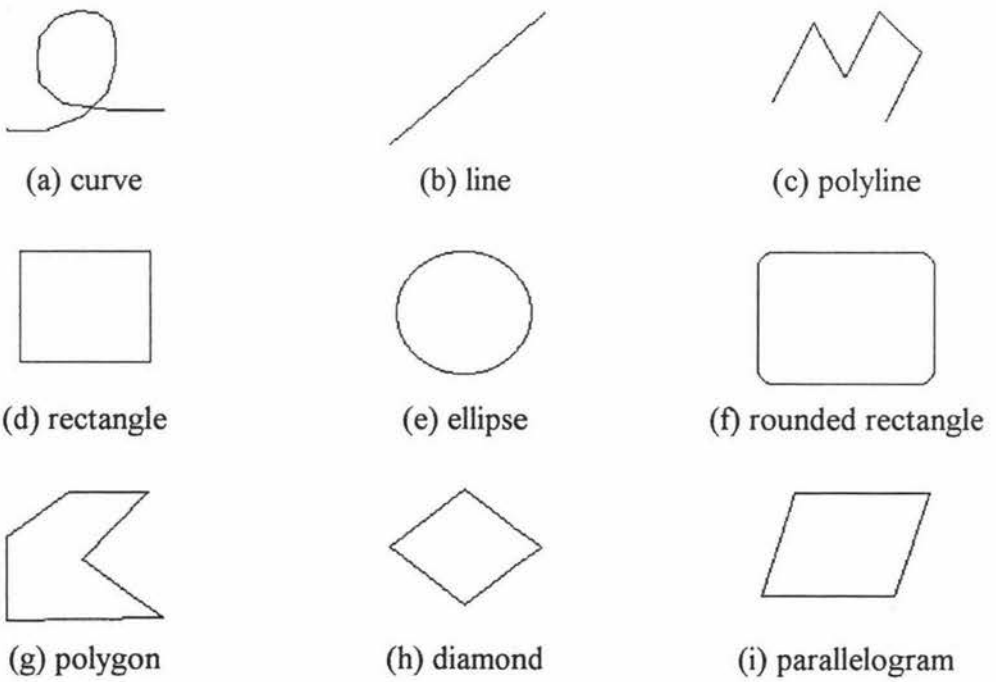


Figure 6.7. Basic shapes

Because the basic drawing tools for closed shapes have three modes: overlap, filled and transparent (see Figure 5.3), there are 18 selection for the drawing tools used to draw closed shapes.

The designers can select different pen color, pen size and pen type for the drawing tools used to draw the unclosed shapes (curve, line or polyline); also in addition they can select different pen color, pen size and pen type for the drawing tools used to draw closed shapes (ellipse, rectangle, rounded rectangle, polygon, diamond and parallelogram). A fill color can be selected for the drawing tools used to draw closed shapes when they are in fill mode. The pen color or fill color can be selected or created from the color common dialog box (Figure 6.6a (i)); the pen size can be input from the input dialog box (Figure 6.6b (ii)); and the pen type (solid, dash, dash dot, dash dot dot and dot) can be selected from the *Pen* group menu.

Besides two type basic drawing tools discussed above, OO Notation Workshop also provides a group of special drawing tools used to draw some special basic shapes which can be used to construct OO notation shapes or can directly be added to some OO notation sets. Figure 6.8 shows the special tool buttons and the examples of these special basic shapes.

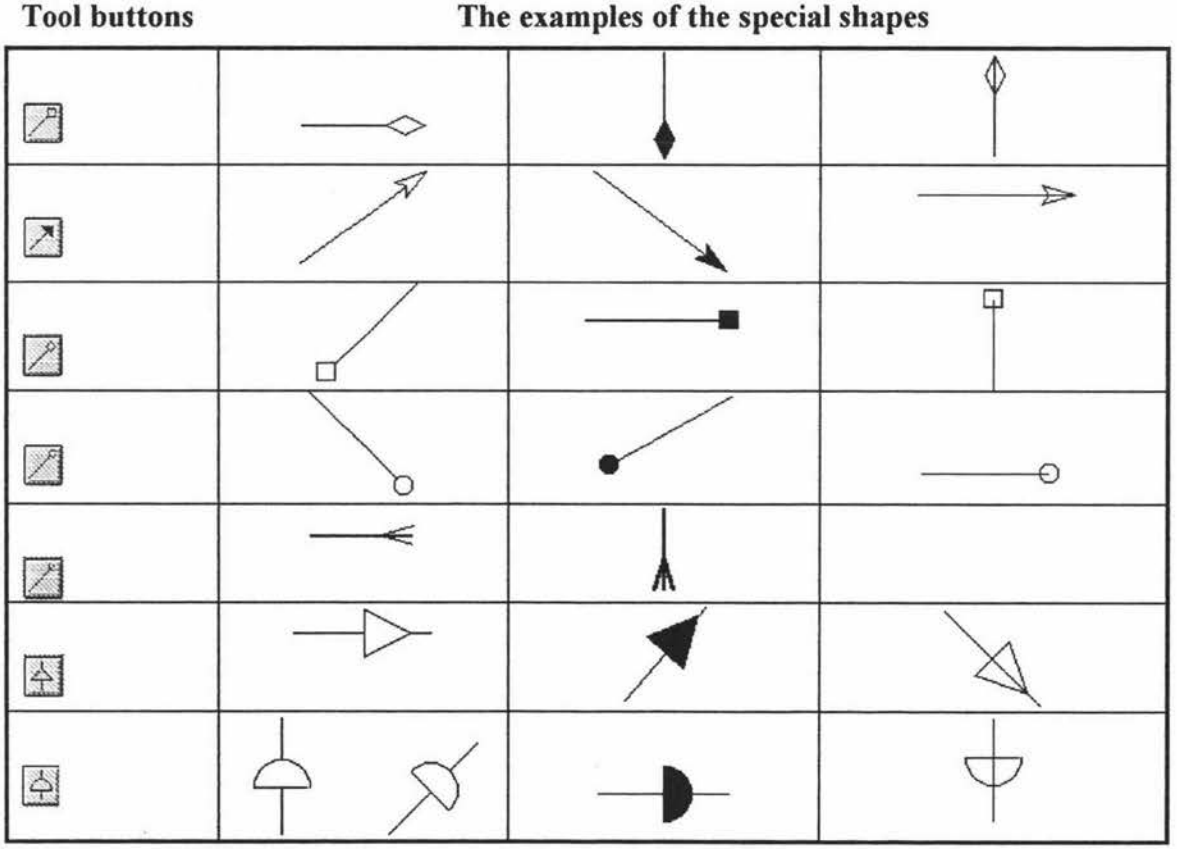


Figure 6.8. The special tool buttons and the examples of special basic shapes.

When using the special tools, the designers can also select different pen color, pen size and pen type for these special drawing tools, and select a drawing mode (overlap, filled or transparent) for them if a special basic shape includes closed shapes. The designer could also select the fill color if the drawing mode is the filled mode.

It should be noted that the special basic shapes drawn by the special drawing tools can be constructed from the basic shapes drawn by the basic drawing tools. Using the special drawing tools to draw these special basic shapes is more convenient than using the basic drawing tools.

6.2.2. Creating Basic Shapes and Special Basic Shapes

OO Notation Workshop is an interactive graphic system. It uses the selected drawing tool to draw basic shapes or special basic shapes on the design window by handling a number of mouse events.

6.2.3. Editing or Modifying Basic Shapes*

The basic shapes displayed in the current design window can be edited or modified. The *Edit* menu (Figure 6.3 (b)) provides commands, *Cut*, *Copy*, and *Paste*. They allow to cut, copy, and paste the selected basic shape in the current design window or to cut, copy the selected basic shape from one design window, then paste it to another design window. The *Edit* menu also provides *Undo* command to undo last action, and *Clear All* command to clear all basic shapes from the current design window.

The attributes, size and location (the reference point) of the selected basic shape can be modified. If the selected basic shape is an unclosed shape, its attributes are pen color, pen size and pen type; if the selected basic shape is a closed shape, besides its pen color, pen size and pen type, its attributes also include its drawing mode and its fill color if its drawing mode is the *filled* mode.

---

* Here basic shapes include the basic shapes and the special basic shapes.

To change the pen color of the selected basic shape, use the color common dialog box (Figure 6.6a (i)) invoked by selecting the menu command *Pen Color* from the *Pen* menu or by selecting the bitmap buttons  from the control bar.

To change the pen size of the selected basic shape, use the input dialog box (Figure 6.6b (ii)) invoked by selecting the menu command *Pen Size* from the *Pen* menu or selecting the bitmap buttons  from the control bar.

To change the pen type of the selected basic shape, select one of the menu items *Solid*, *Dash*, *Dash dot*, *Dash dot dot* and *Dot* from the *Pen* group menu.

When a closed basic shape is selected, to change its drawing mode, select one of the menu items *Overlap*, *Fill* and *Transparent* from the *Pen* group menu or select one of the bitmap buttons  (*Overlap*)  (*Fill*)  (*Transparent*) from the control bar. The menu commands *Overlap*, *Fill* and *Transparent* and the bitmap buttons    are only available when the selected basic shape is a closed shape.

When a closed basic shape is selected and its drawing mode is the *filled* mode, its fill color can be modified. To accomplish this task, use the color common dialog box (Figure 6.6a (i)) invoked by selecting the menu item *Fill Color* from the *Pen* menu or selecting the bitmap button  from the control bar. The menu item *Fill Color* and the bitmap button  are only available when a closed basic shape is selected and its drawing mode is the *filled* mode.

To move a basic shape, select the basic shape you want to relocate, use the left mouse button to drag the basic shape to a new position. A dotted rectangle indicates the position of the basic shape as you drag it. The real basic shape moves to its new location when you release the left mouse button.

6.2.4 Viewing and Editing OO Notation Shapes

The basic shapes created in the design window form an OO notation shape, an OO notation shape can be added to the container of the current OO notation set by selecting the menu command *Add to notation Set* from the *Notation* menu (see Figure 6.3 (e)).

After having been added to the container of the current OO notation set, the basic shapes in the design window remain unchanged. The designer can carry on a new OO notation

shape design based on these existing basic shapes, or ignore the existing basic shapes and begin a new design session by selecting the menu command *New Notation* from the *Notation* menu.

During a design session, the designer can view and edit the OO notation shapes contained in the container of the current OO notation set. After selecting the menu command *View Notations* from the *Notation* menu, the mode of the current design window is changed from its design mode to its view mode and the all OO notation shapes contained in the container of the current OO notation set are displayed in the design window.

The OO notation shapes displayed in the current design window can be edited. The *Edit* menu* (see Figure 6.3 (b)) provides commands, *Cut*, *Copy*, and *Paste*. They allow to cut, copy, and paste the selected OO notation shape in the current design window or to cut, copy the selected OO notation shape from one design window, then paste it to another design window which is also in the view mode. The *Edit* menu also provides *Undo* command to undo last action, and *Clear All* command to clear all OO notation shapes from the current design window. Besides above common editing functions, a selected OO notation shape can also be moved, named and modified.

The view mode of the current design window can be changed into its design mode by selecting one of the three menu commands, *New Notation*, *Modify Notation* and *Go to Design*, from the *Notation* menu. The designer can always go back to the design mode from the view mode and begin a new design session by selecting the menu command *New Notation*; or go back to the design mode from the view mode, restore the existing basic shapes and carry on the previous design by selecting the menu command *Go to Design*. The menu command *Modify Notation* is only available when there is a selected OO notation shape. By selecting the menu command *Modify Notation*, the designer can go back to the design mode from the view mode, extract all basic shapes from the selected OO notation shape and display them in the design window, then carry on the new design which is based on these basic shapes.

---

* In the design mode and view mode, OO Notation Workshop uses the same menu commands from the menu *Edit* but different operation objects. In the design mode, the operation object is a basic shape; in the view mode, the operation object is an OO notation shape.

6.3. OO Notation Examples

To demonstrate the designing ability of OO Notation Workshop, several graphical OO notation examples are created. The following paragraphs show the OO notation sets of the Booch methodology [Booch, 94b], the Coad/Yourdon methodology [Coad & Yourdon, 90a], the Fusion methodology [Coleman, 94], Object Modelling Technique [Rumbaugh et al., 91a], Objectory [Jacobson et al., 92] and Shlaer-Mellor methodology [Shlaer & Mellor, 91].

6.3.1. The Notation of the Booch Methodology

Some of the graphical notation shapes created for the Booch methodology are shown in Figure 6.9, Figure 6.10 and Figure 6.11. Figure 6.9 shows *Object icon* and *Class icons*; Figure 6.10 shows *Class relationships* and *Containment adornments*, and Figure 6.11 shows *Module icons*.

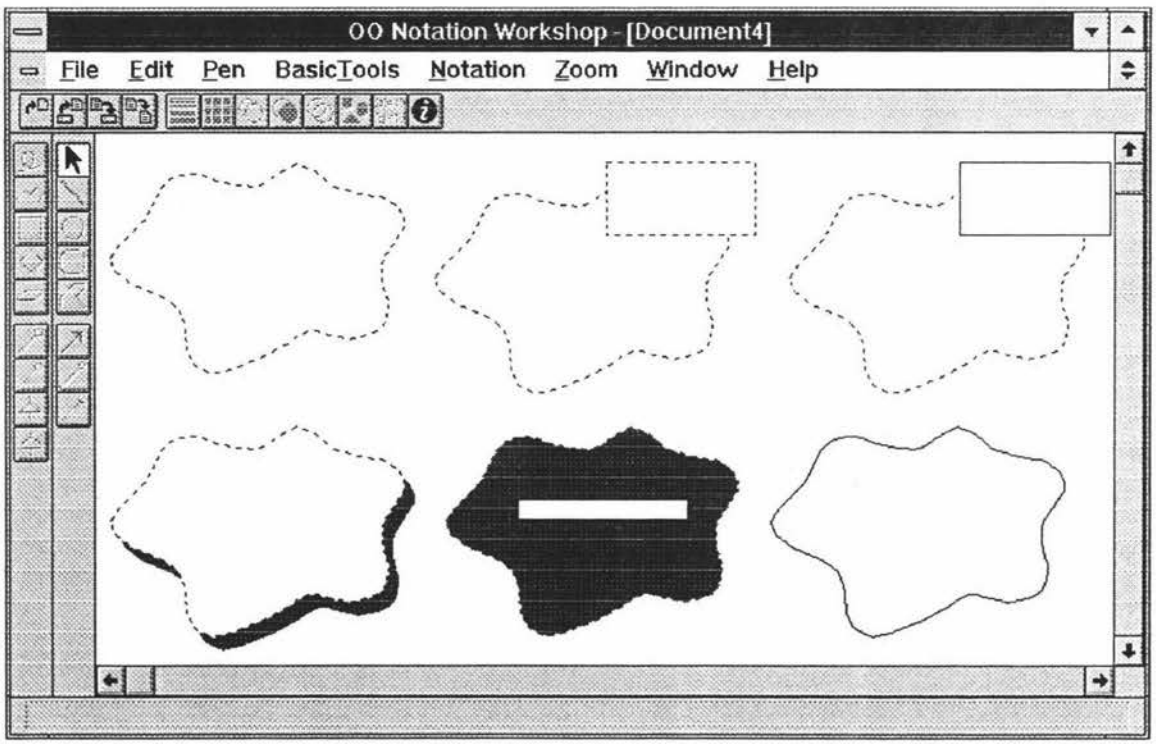


Figure 6.9. *Object icon* and *Class icons* of the Booch methodology

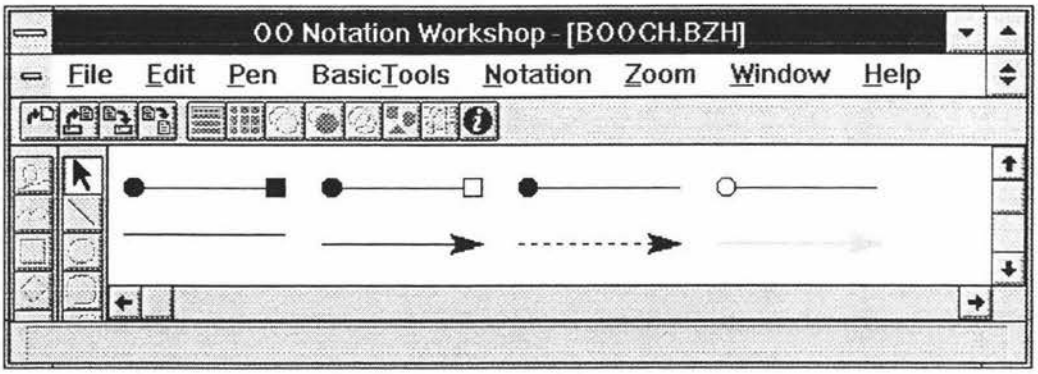


Figure 6.10. *Class relationships* and *Containment adornments* of the Booch methodology

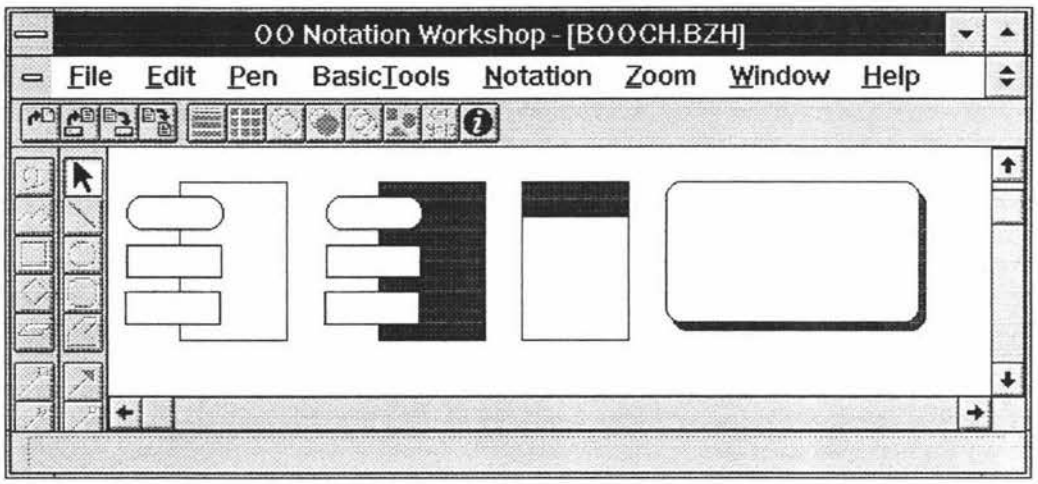


Figure 6.11. *Module icons* of the Booch methodology

The basic shape of a class icon of the Booch methodology is a cloud. In Figure 6.9, the class icon is created by using the *Polygon* tool which uses the *Dash* pen. To ensure that this class icon is the same shape as it should be, the data used to draw the class icon are entered from the input dialog box. The object icon which is similar to a class icon except that it has a solid line as a boundary is obtained by simply modifying the pen-type attribute of the polygon of the class icon from *Dash* to *Solid*. Based on the class icon, other class icons (the parameterized class icon, the instantiated class icon and the metaclass icon) as will as utility icons (the class utility icon, the parameterized class utility icon and the instantiated class utility icon) can be drawn easyly. For example, the

parameterized class icon is a cloud with a dashed-line box in the upper right corner, Figure 6.12 shows the steps used to create the parameterized class icon.

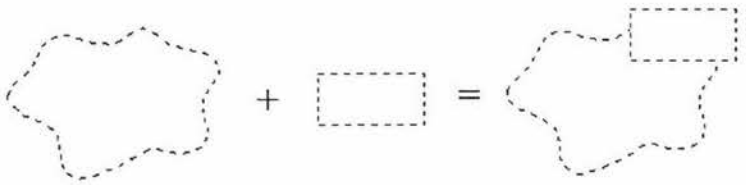


Figure 6.12. The steps used to create the parameterized class icon.

The instantiated class icon is a cloud with a solid-line box in the upper right corner, so it is obtained by simply modifying the pen-type attribute of the dashed-line box of the parameterized class icon from *Dash* to *Solid*. The metaclass icon is a gray-filled cloud with a unfilled solid-line box inside it. Figure 6.13 shows the steps used to create the metaclass class icon.

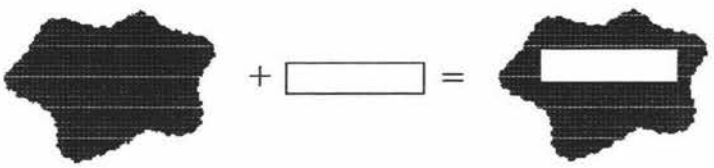


Figure 6.13. The steps used to create the metaclass class icon.

The class utility is a cloud with a gray shadow at the lower edge of the cloud, Figure 6.14 shows the steps used to create the class utility icon.

Figure 6.14. The steps used to create the class utility icon.

A parameterized class utility icon is a cloud with a dashed-line box in the upper right corner and a gray shadow at the lower edge of the cloud. Figure 6.15 shows the steps used to create the parameterized class utility icon from the class utility icon.

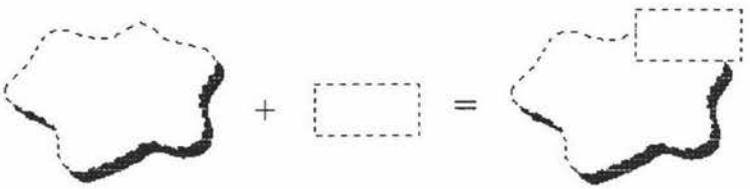


Figure 6.15. The steps used to create the parameterized class utility icon.

The instantiated class utility icon is a cloud with a solid-line box in the upper right corner and a gray shadow at the lower edge of the cloud, it is obtained by simply modifying the pen-type attribute of the dashed-line box of the parameterized class icon from *Dash* to *Solid*.

Similar steps are used to create *Module icons*. Figure 6.16 shows the steps used to create the specification icon of *Module Diagram* of the Booch methodology.

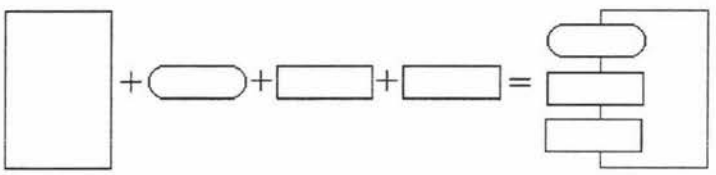


Figure 6.16. The steps used to create the specification icon.

The design steps used to create the icons shown in Figure 6.17 are different from the steps discussed above, because each of the icons of *Class relationships* and *Containment adornments* must have its reference direction and length. To create an icon, the first step is to create a horizontal line which represents the reference direction and length of the icon. Figure 6.17 shows the steps used to create the icon of the has-by-reference relationship.

Figure 6.17. The steps used to create the icon of the has-by-reference relationship.

The use of the reference direction and length will be discussed in Chapter 7.

6.3.2. The Notation Examples of Other OO Methodologies

The following figures show some notation examples which are created in OO Notation Workshop. These notation sets come from existing widely used OO methodologies.

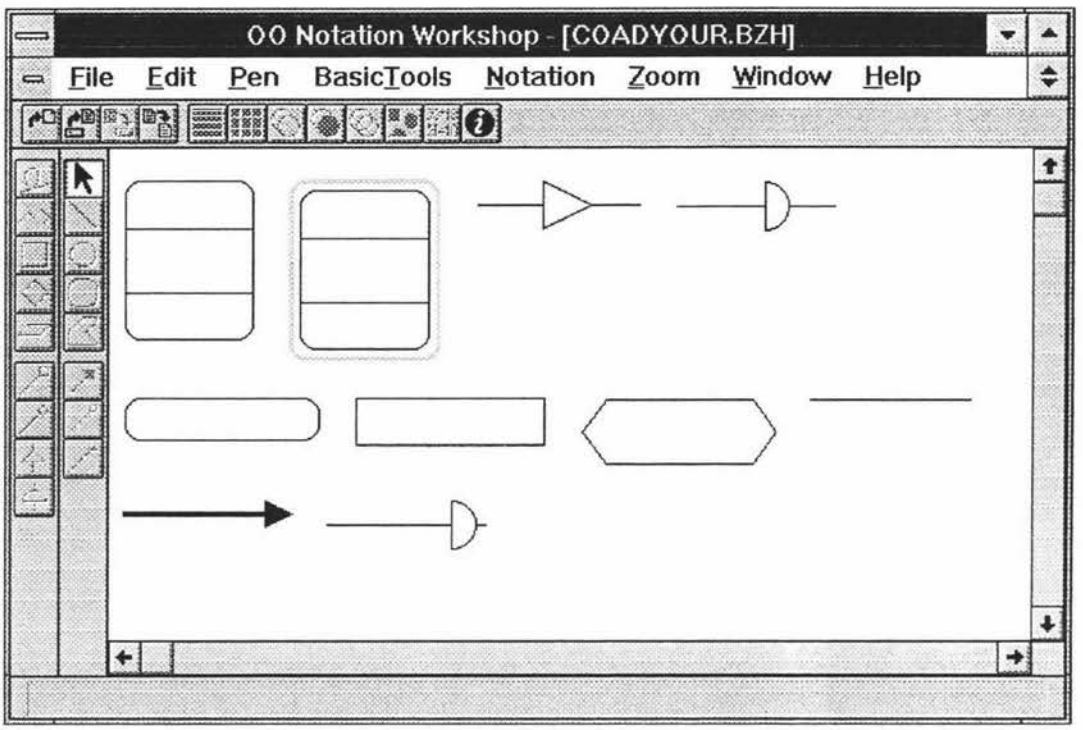


Figure 6.18. The notation of the Coad/Yourdon methodology

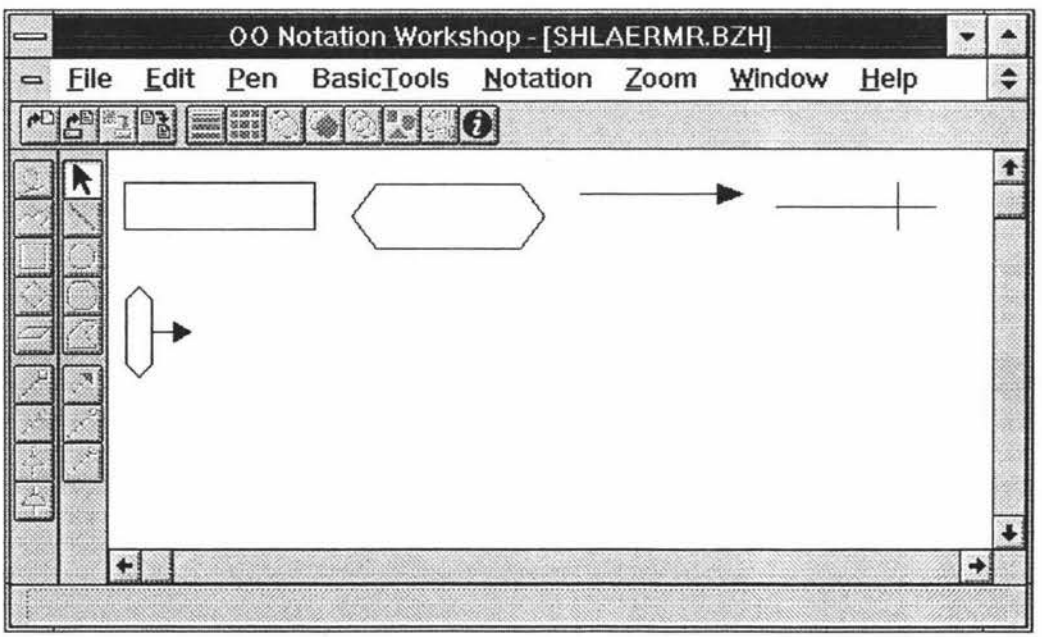


Figure 6.19. The notation of the Shlaer-Mellor Methodology

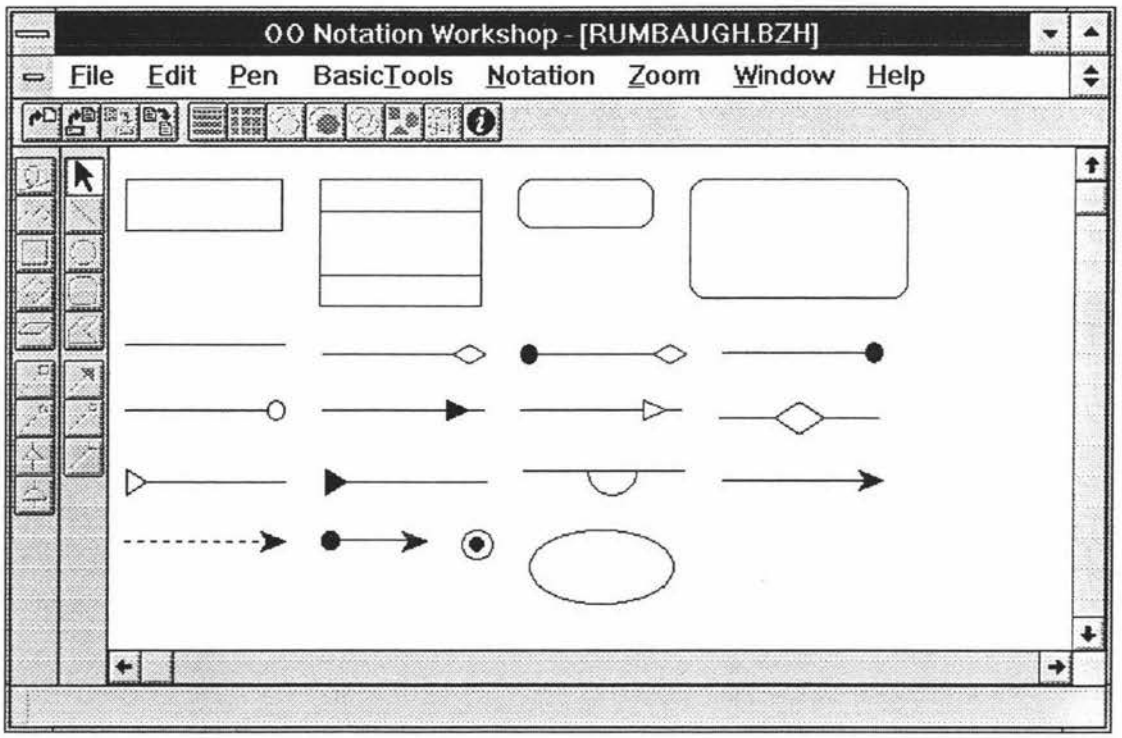


Figure 6.20. The notation of the Object Modelling Technique

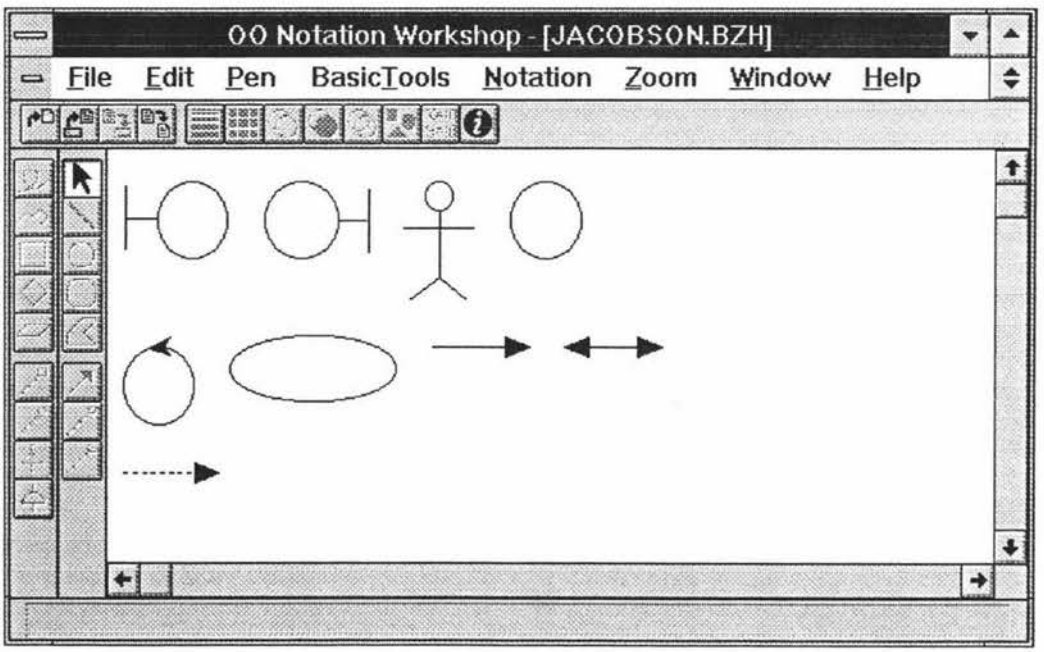


Figure 6.21. The notation of the Objectory

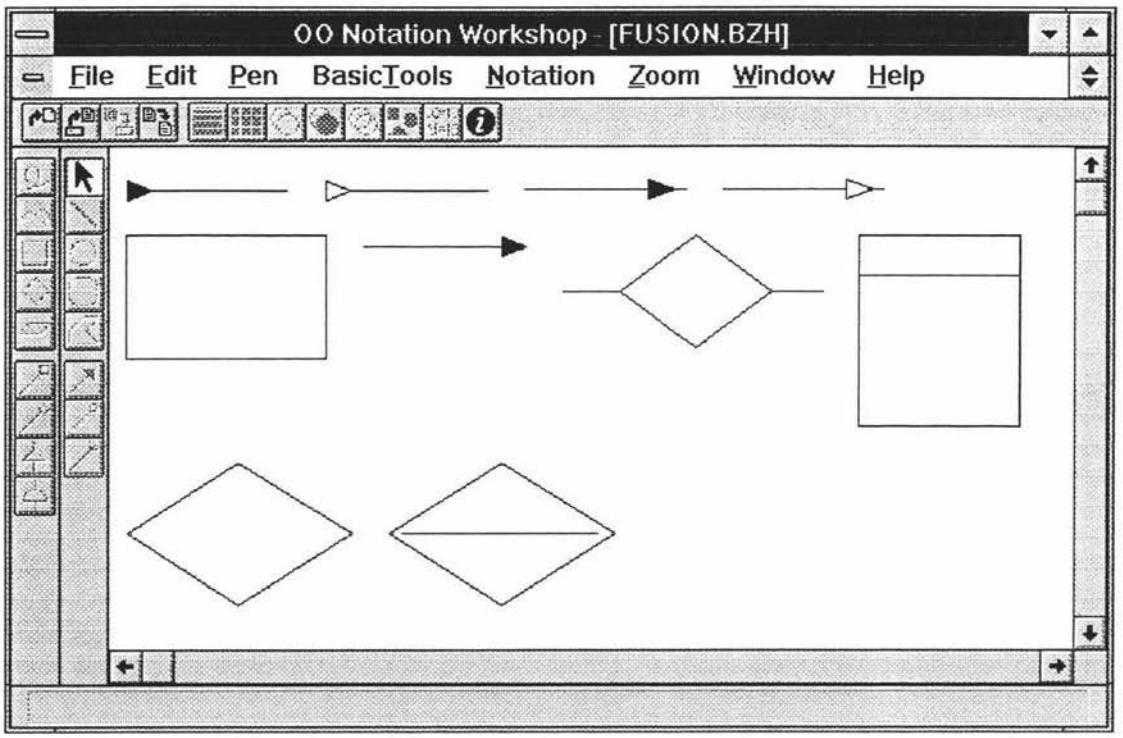


Figure 6.22. The notation of the Fusion methodology

Chapter 7

Universal OO Diagraming Tool

7.1. Overview of Universal OO Diagraming Tool

In the MIGOCE OO development environment, UOODT (Universal OO Diagraming Tool) is a window based graphical design tool which can be customized to support different OO methodologies.

7.1.1. GUI of UOODT

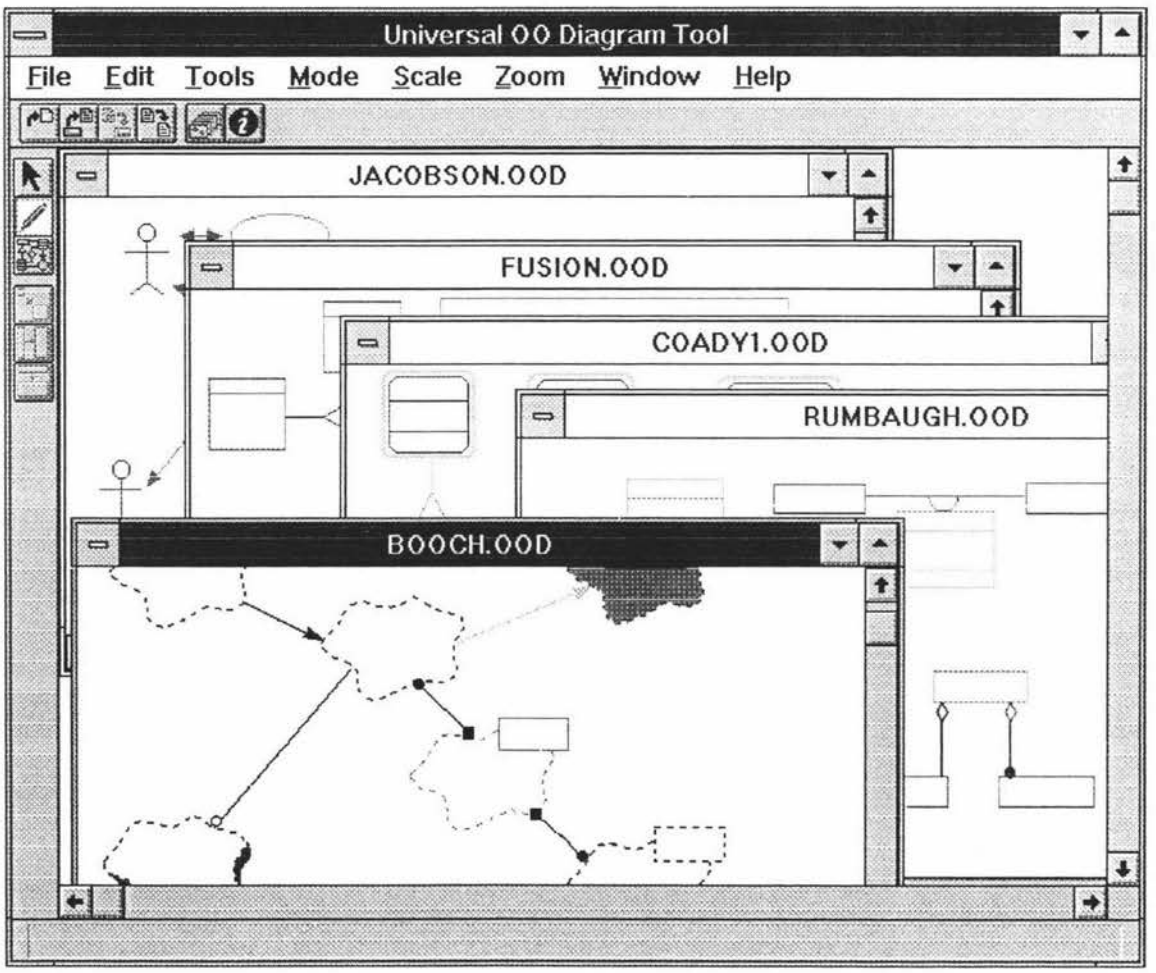


Figure 7.1. The application window of UOODT

The application window of UOODT is shown in Figure 7.1, it contains a menu bar, a control bar, a tool bar, a status bar and a client window which can have several OO design windows. OO design windows are the windows that the designer actually works with and that display the OO diagrams contained in each OOD document.

The menu commands of all menu groups of UOODT are shown in Figure 7.2. Note not all group menus are available at all times. Some group menus, such as *Edit*, *Tools*, *Mode*, *Scale* and *Zoom*, appear on the menu bar when at least an OOD document is opened. The details of some menu commands will be discussed in following sections.

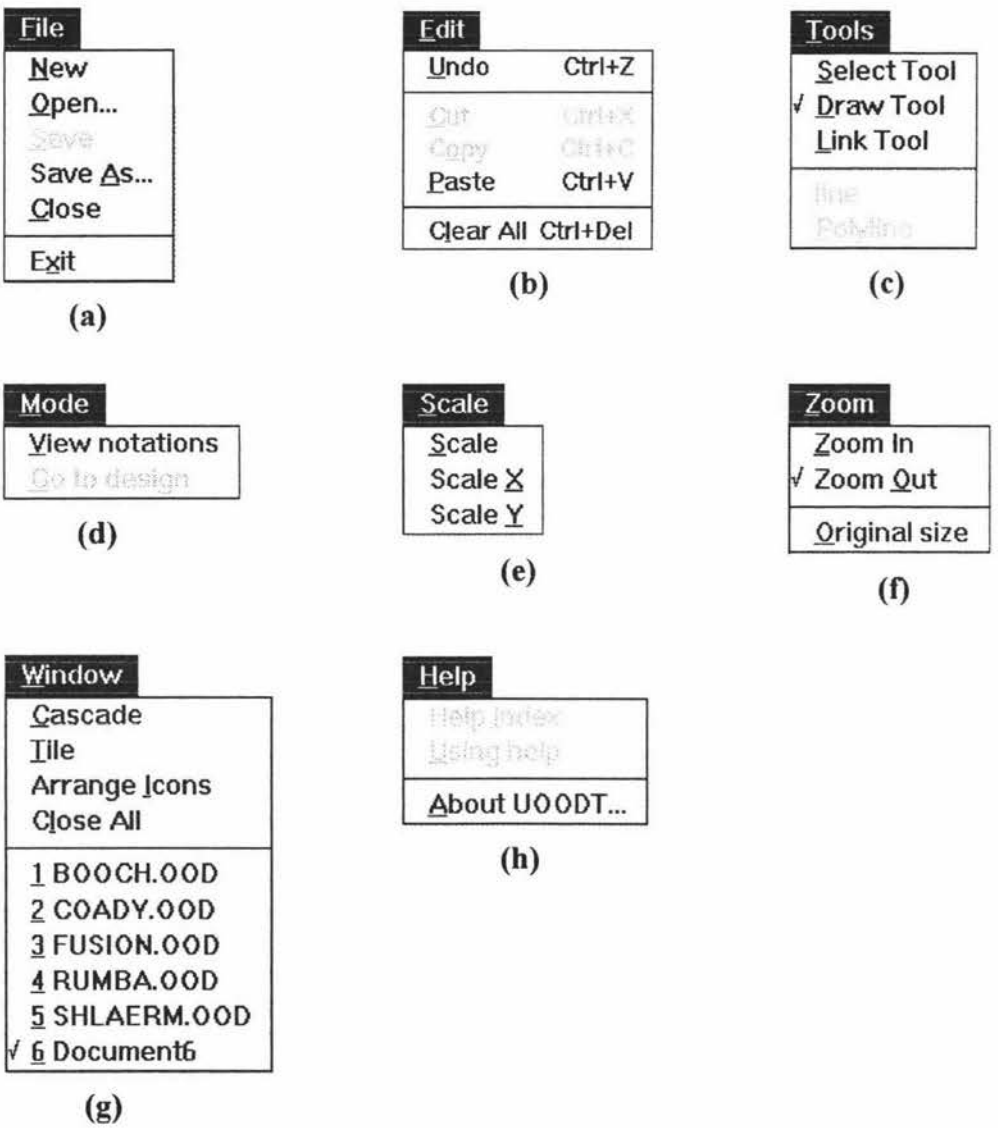


Figure 7.2. The menu commands of all menu groups of UOODT

Figure 7.3 shows the bitmap buttons used by UOODT and their correspondent menu commands, the control bar and the tool bar use these bitmap buttons to represent some of the menu commands. Clicking one of the bitmap buttons from the control bar or from the tool bar is a quicker alternative to choosing a command from the menu.

Bitmap button	**Menu command**
	New
	Open...
	Save
	Save As...
	
	About UOODT...
	Select Tool
	Draw Tool
	Link Tool
	Scale
	Scale X
	Scale Y

Figure 7.3. The bitmap buttons used by UOODT

The graphical user interface of UOODT also has a status bar used to display help hints when the user move the mouse pointer over a bitmap button or a menu command. Each of OO design windows has scroll bars used to control scrolling if the design window cannot fit the entire OOD document inside its client area.

Some dialog boxes are used in the graphical user interface of UOODT, these dialog boxes are the common dialog box *File open* and *File save* and the custom dialog box *Input* and *About*. All the dialog boxes used by UOODT are model dialog boxes.

All OO design windows which are in the client window of UOODT can be re-arranged and iconified. The *Cascade* menu command arranges the design windows to overlap so that each title bar is visible. The *Tile* menu command causes the design windows in smaller sizes to fit next to each other on the client window of UOODT.

7.1.2. The Main Functions of UOODT

Based on the OO notation sets stored in OO Notation Repository, UOODT can be used to draw graphical OO diagrams for any OO methodology provided the OO notation set of the methodology has been loaded to UOODT.

Each OO design window has its OO notation set selected by the designer, the OO notation set of the current OO design window can be changed at any time by loading another OO notation set.

Each OO design window has two modes: the view notation mode and the design mode. The mode of an OO design window is easy to change.

Each mode of an OO design window has its tools. In the view notation mode, the *Select Tool* is used to select a notation shape from the current OO notation set. In the design mode, the *Draw Tool* is used to draw node icons in the current design window based on the selected notation shape; the *Link Tool* is used to draw relationships between node icons in the current design window based on the selected relationship; and the *Select Tool* is used to select or move a node icon or a relationship.

UOODT uses the same *Edit, Scale* menu commands but different operation object or objects in both the design mode and the view notation mode. In the design mode, the operation object is a node icon (or a relation) or node icons (or relationships); but in the view notation mode, the operation object is an OO notation shape or OO notation shapes.

In the design mode, a selected node icon can be cut or copied. a cut or copied node icon can be pasted to the current design window or other design windows.

In the view notation mode, a selected OO notation shape can be cut or copied. a cut or copied notation shape can be pasted to the OO notation set of the current design window or OO notation sets of other design windows.

A selected node icon can be scaled down or scaled up in the design mode and a selected OO notation shape can be scaled down or scaled up in the view notation mode. When scaling down or scaling up a selected object, the user can select three different scaling mode: *Scale*, *Scale X*, or *Scale Y*. The examples in Figure 7.4 show different scaling up and scaling down effect.

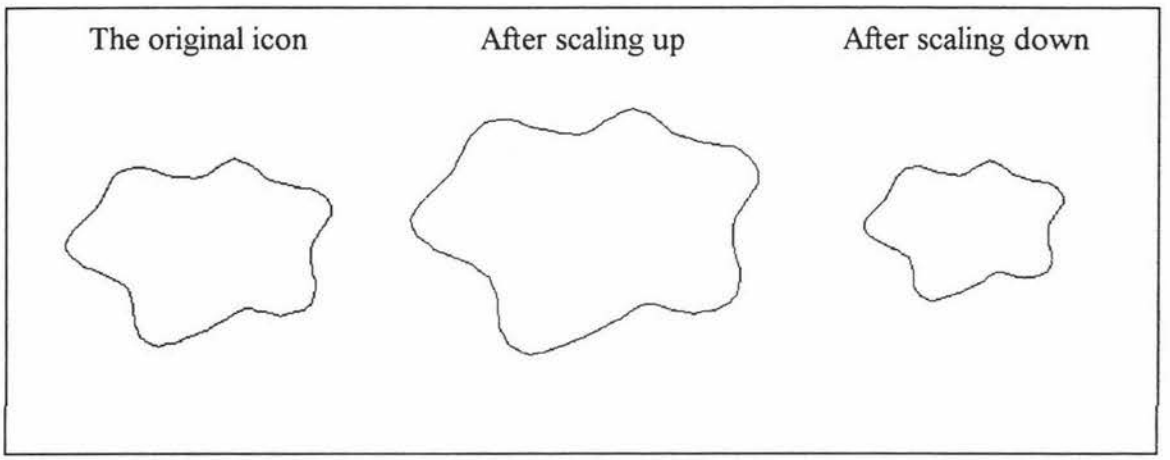


Figure 7.4a. The example of using the *Scale* command.

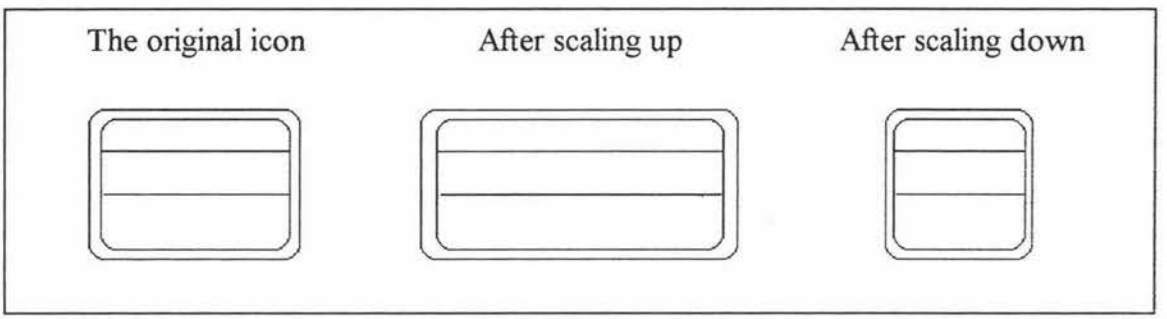


Figure 7.4b. The example of using the *Scale X* command.

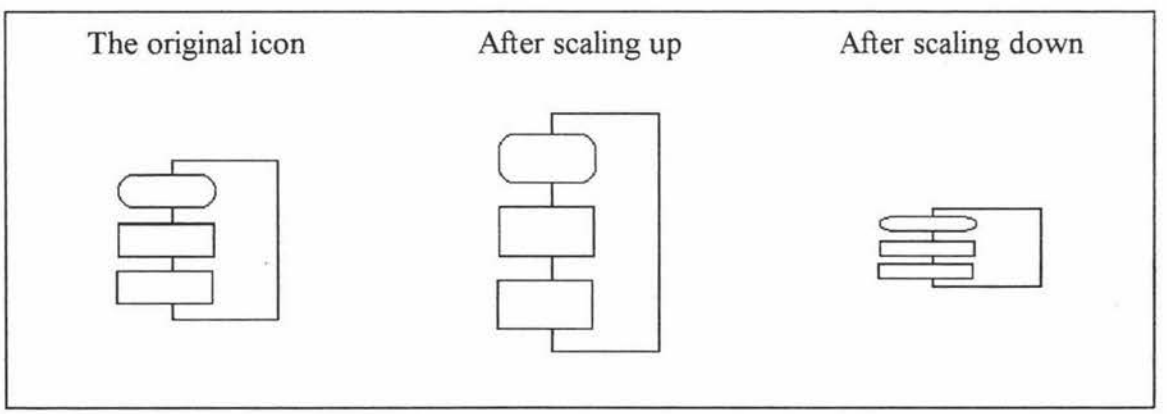


Figure 7.4c. The example of using the *Scale Y* command.

In the design mode, a selected node icon can be moved, to move a node icon, select the node icon you want to relocate, use the left mouse button to drag the node icon to a new

position, a dotted rectangle indicates the position of the node icon as you drag it, the real node icon moves to its new location when you release the left mouse button.

In the design mode, the *Clear All* command can be used to clear all OO diagrams from the current design window.

In the view notation mode, the *Clear All* command can be used to clear all OO notation shapes from the OO notation set of the current design window.

UOODT offers strong file management functions to manage OOD documents, the *File* menu provides commands for creating new OOD documents, opening existing OOD documents, saving OOD documents, and exiting the application window of UOODT.

The *New* command opens a new design window with the default name Documentx (the x stands for a number) in the client window of UOODT and automatically makes the new design window active.

The *Open* command displays the common dialog box *File open* so the designer can select a OOD document which has the *.ood* extension and then load it into the client window of UOODT, the client window can have several OO design windows at the same time.

The *Save* command saves the OOD document in the active design window to disk. If the document has a default name (such as Document8), UOODT opens the common dialog box *Save As* so the designer can rename the OOD document as well as save it in a different directory or on a different drive. This command is unavailable if no design window is active.

The *Save As* command displays the common dialog box *Save As*, the document in the active design window can be saved under a different name, in a different directory, or on a different drive. This command is unavailable if no edit window is active.

The *Exit* command exits the application of UOODT and prompts the designer to save the modified OOD document(s) before exiting.

7.2. Using UOODT in Object-Oriented Development

7.2.1. Selecting an OO Methodology

As discussed above, UOODT can be customized to support different OO methodologies; the client window of UOODT can have several OO design windows at the same time, and each OO design window has its OO notation set used by the designer to carry out an OOA or OOD assignment.

A new design window opened by the *New* command must be given an OO notation set before it can be used. The given OO notation set of the design window determines the OO methodology used in the design window. When the content of the design window are saved as a OOD document to disk, the given OO notation set of the design window is also be saved.

When an existing OOD document which has the *.ood* extension is opened by the *Open* command, its OO notation set is loaded to the design window and determines the OO methodology used in the design window.

At any time, the OO methodology used in a design window can be changed by loading a new OO notation set from OO Notation Repository. It is very easy to update the OO notation set of an existing OOD document and it is possible to mix different OO methodologies in the same design window.

To load an OO notation set, select the bitmap button  from the control bar and give the name of the OO notation set through the input dialog box.

7.2.2. Drawing and Editing OO Diagrams

An OO diagram simply captures an OO design and consists of several node icons and relationships. To draw OO diagrams in the design window is to place node icons in the design window and then link them with relationships.

When working in an OO diagram, the behavior of the mouse is determined by the tool that is active in the tool bar along the left side of the client window of UOODT. To

activate a tool, point to the bitmap button and click the left mouse button. The tool bar contains three kinds of tools:  (*Select Tool*),  (*Draw Tool*),  (*Link Tool*). When *Select Tool* is active, the mouse can be used to select the icon(s) on which UOODT is to perform an operation (Cut, Copy, Scale up, move ...). When *Draw Tool* is active, based on the selected notation shape, the mouse can be used to place node icons (such as class icons) in the current design window. When *Link Tool* is active, the mouse can be used to draw relationships between node icons in the current design window based on the selected relationship.

7.3. Examples of Object-Oriented Development

Figure 7.5 – Figure 7.9 show Object-oriented diagrams which are designed by using Universal OO Diagraming Tool. These diagrams support existing OO methodologies.

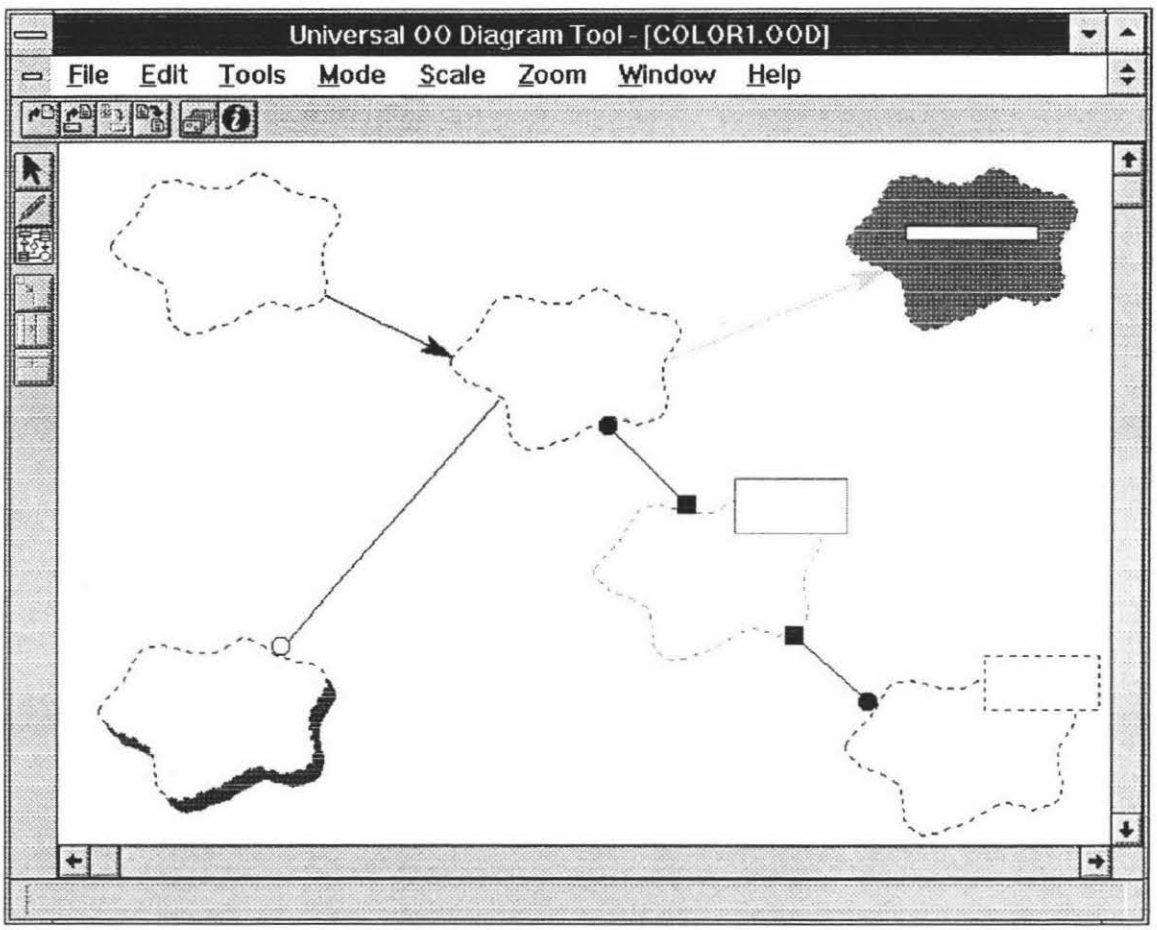


Figure 7.5. An OO diagram for the Booch methodology

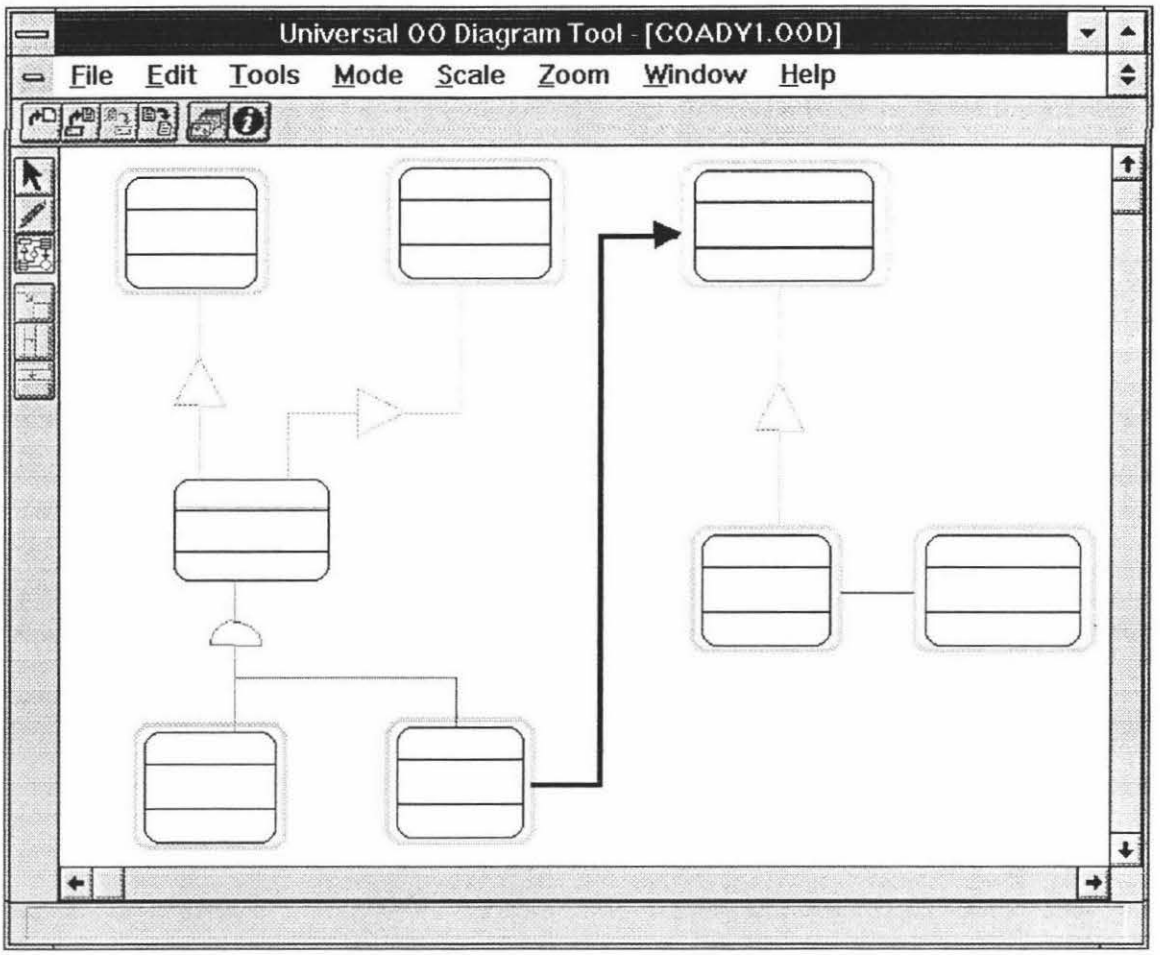


Figure 7.6. An OO diagram for the Coad/Yourdon methodology

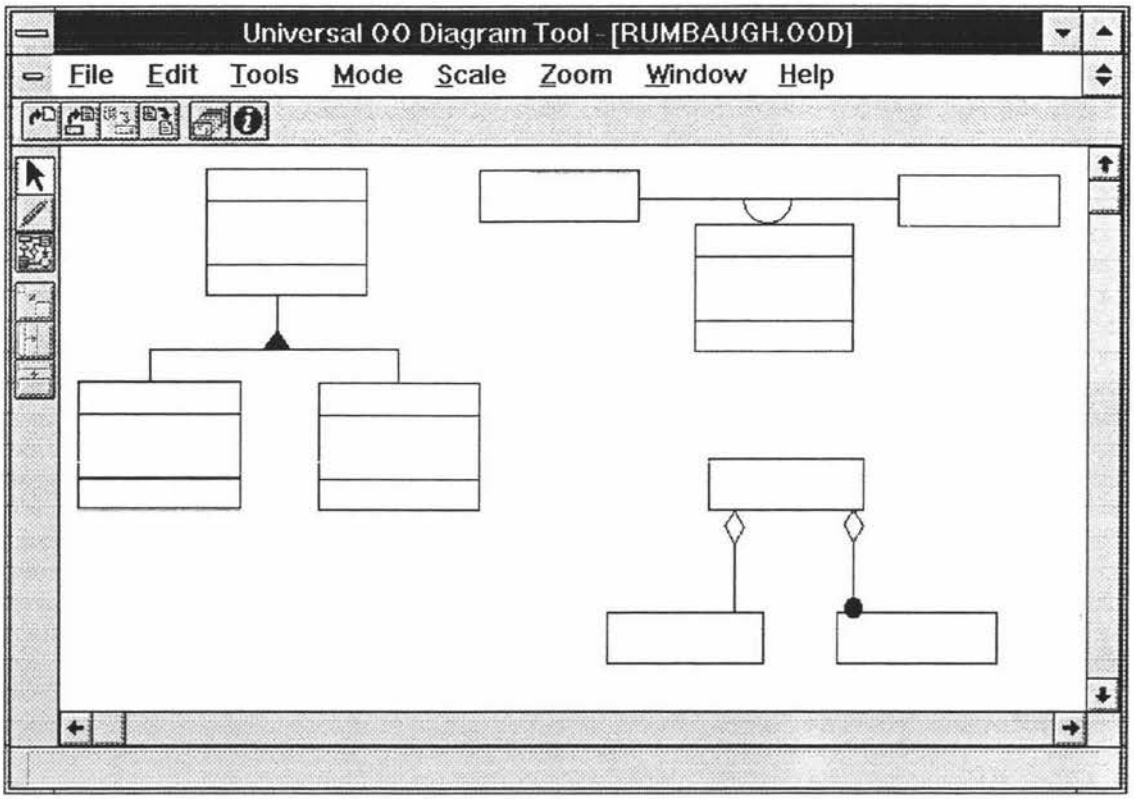


Figure 7.7. An OO diagram for the Rumbaugh methodology

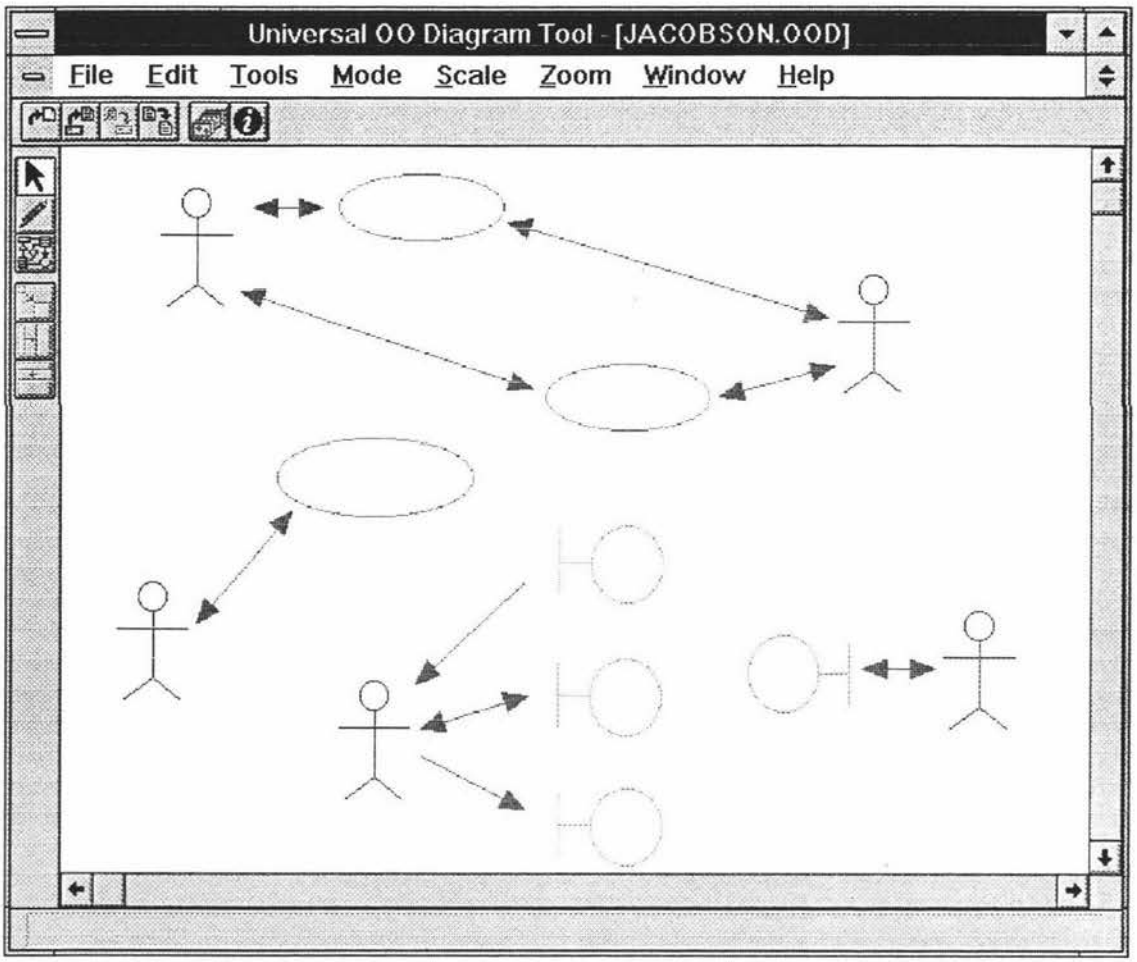


Figure 7.8. An OO diagram for Jacobson's Objectory

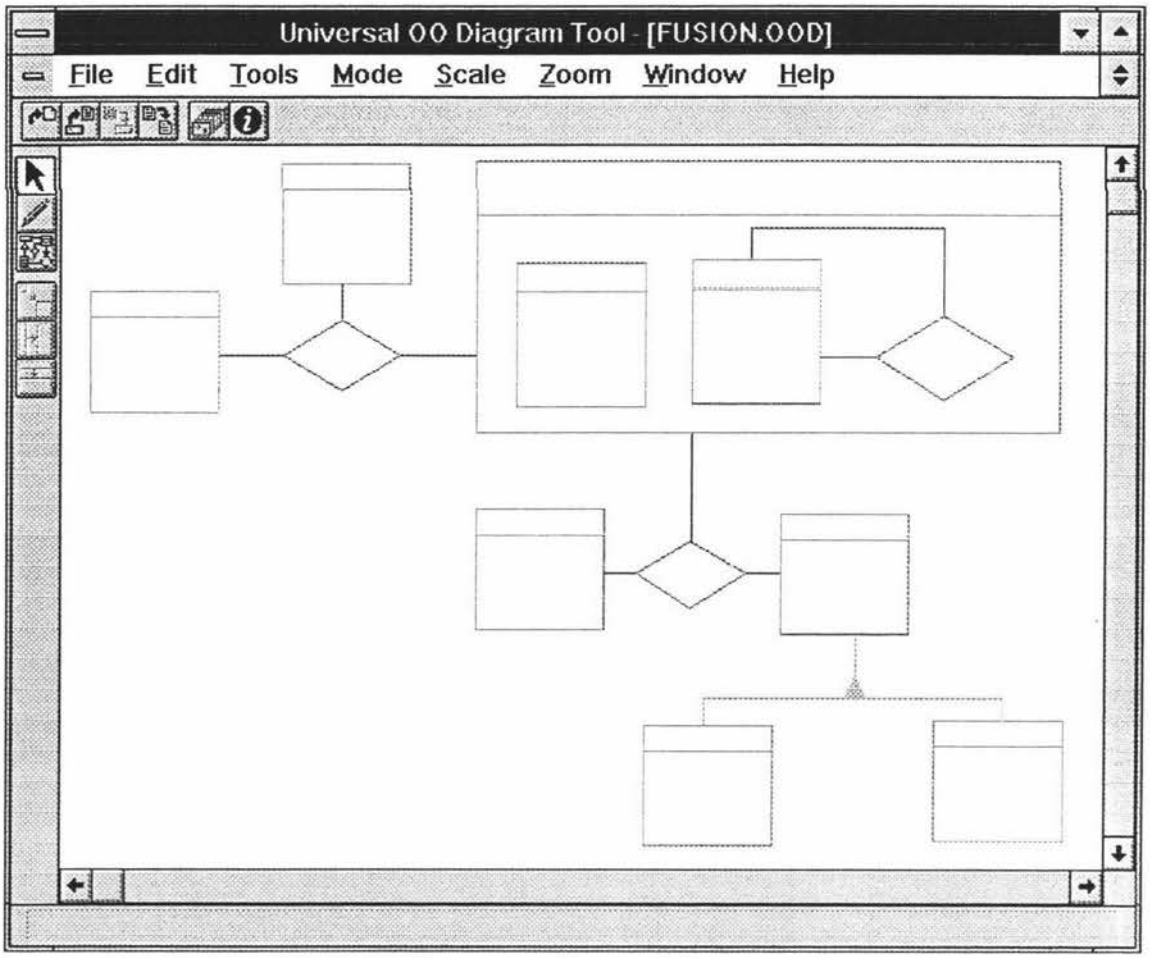


Figure 7.9. An OO diagram for the Fusion methodology

With Universal OO Diagraming Tool, the user can choose notations from different existing OO methodologies and draw a mixed-methodology diagram with these notations. The example of a mixed-methodology OO diagram is shown in Figure 7.10.

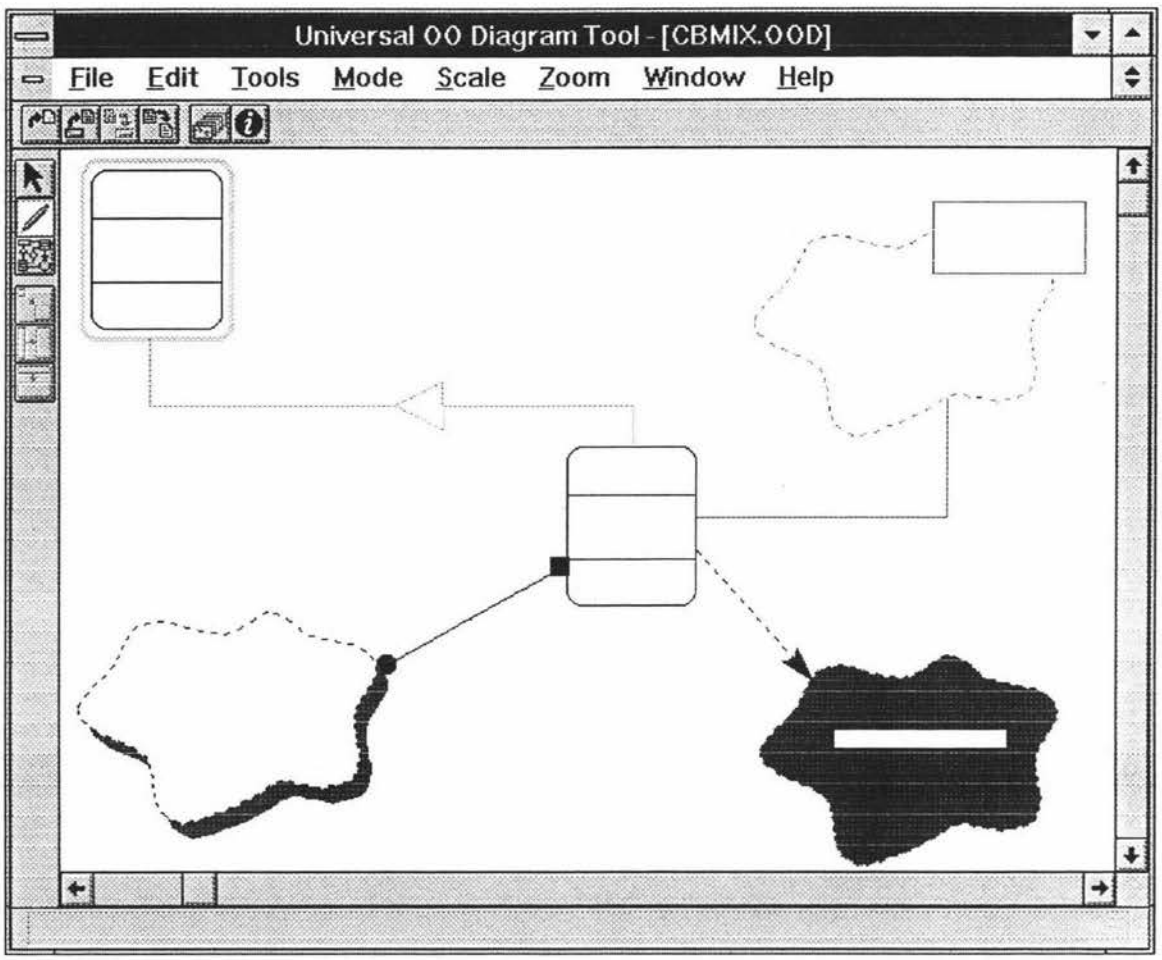


Figure 7.10. The example of a mixed-methodology OO diagram

One outstanding advantage of the MIGOCE system is that it supports users' own methodologies. The user can draw OO diagrams using Universal OO Diagraming Tool with their own notations which are created by OO Notation Workshop. An example of supporting the user-defined notation is shown in Figure 7.11.

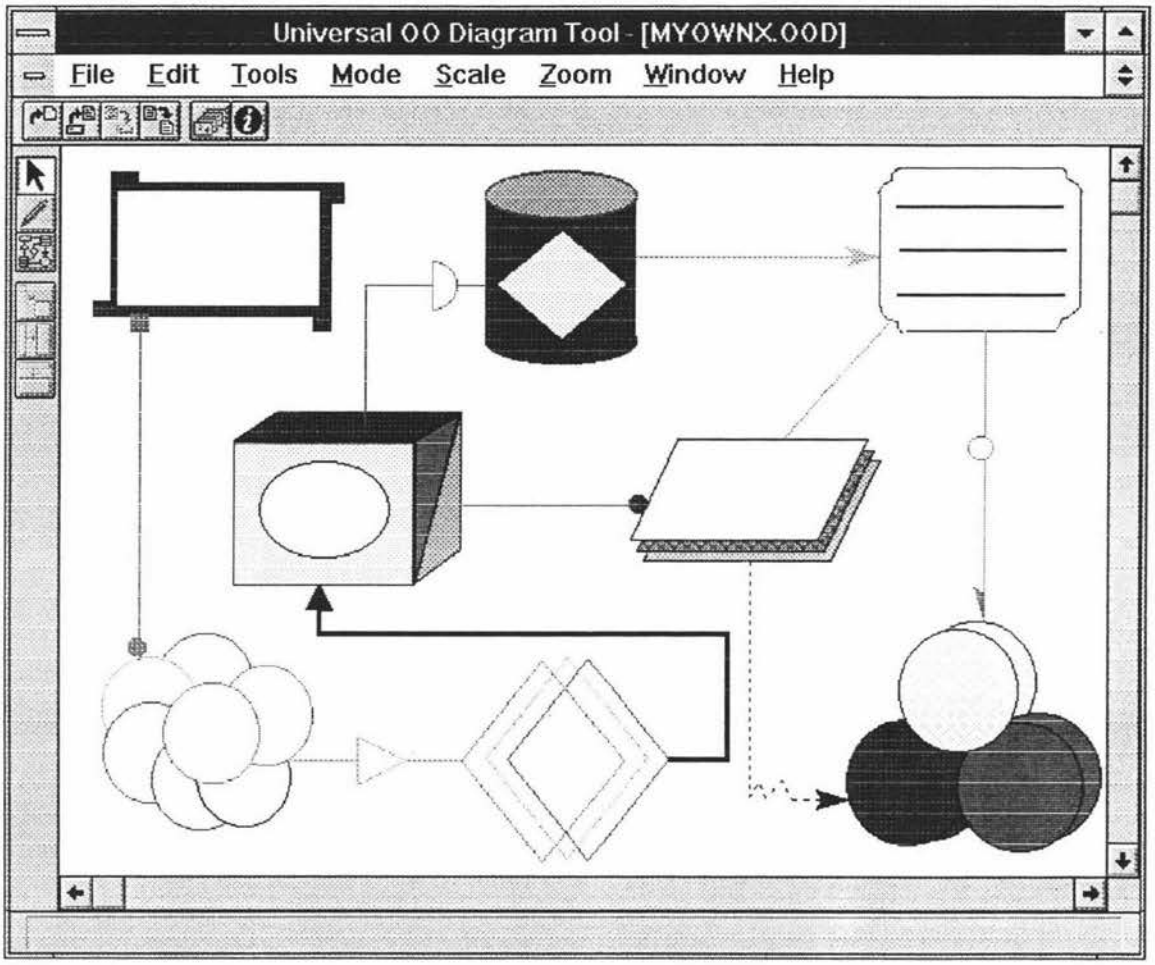


Figure 7.11. An example of supporting the user-defined notation

UOODT is a MDI (multiple-document interface) application which can support multiple OO design documents open at the same time (see Figure 7.12 and Figure 7.13).

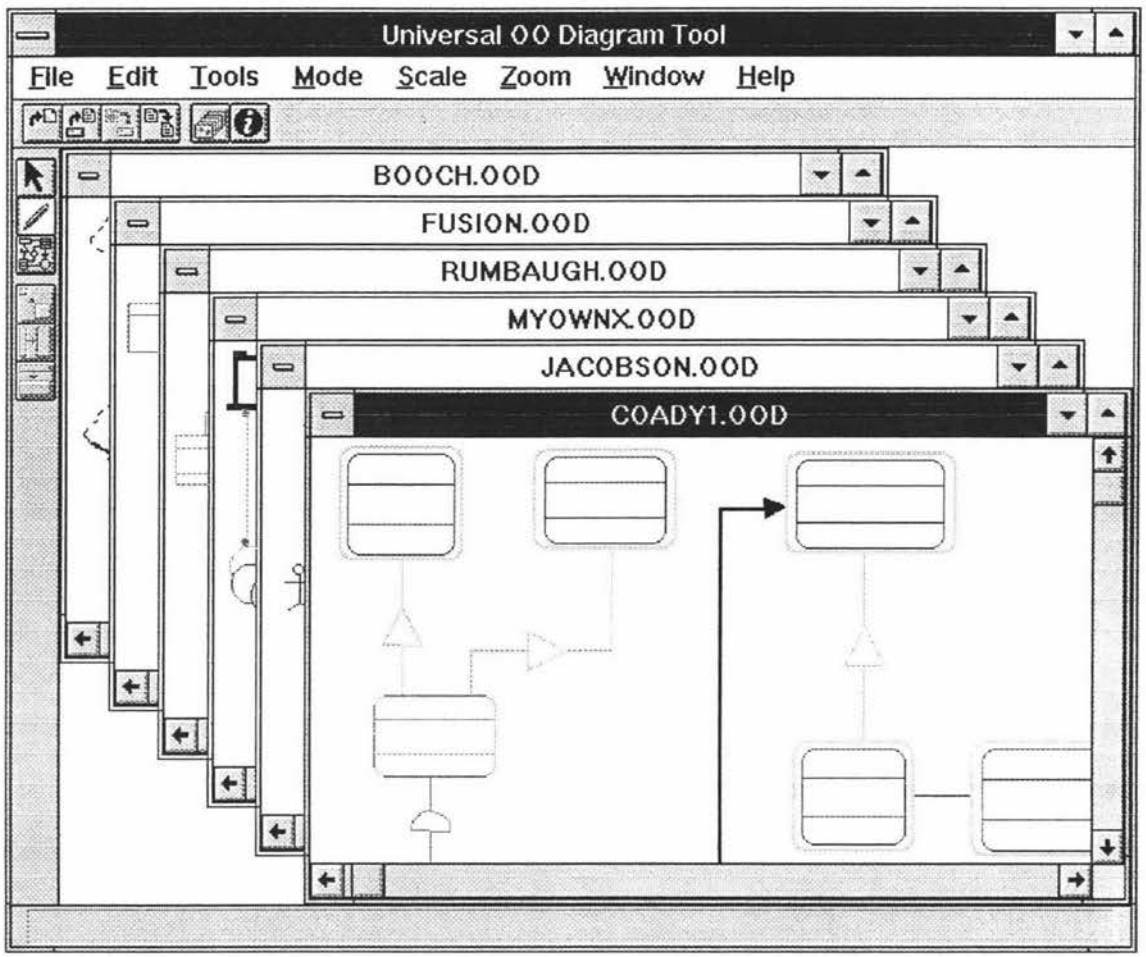


Figure 7.12. The multiple OO design windows(Cascade) of UOODT

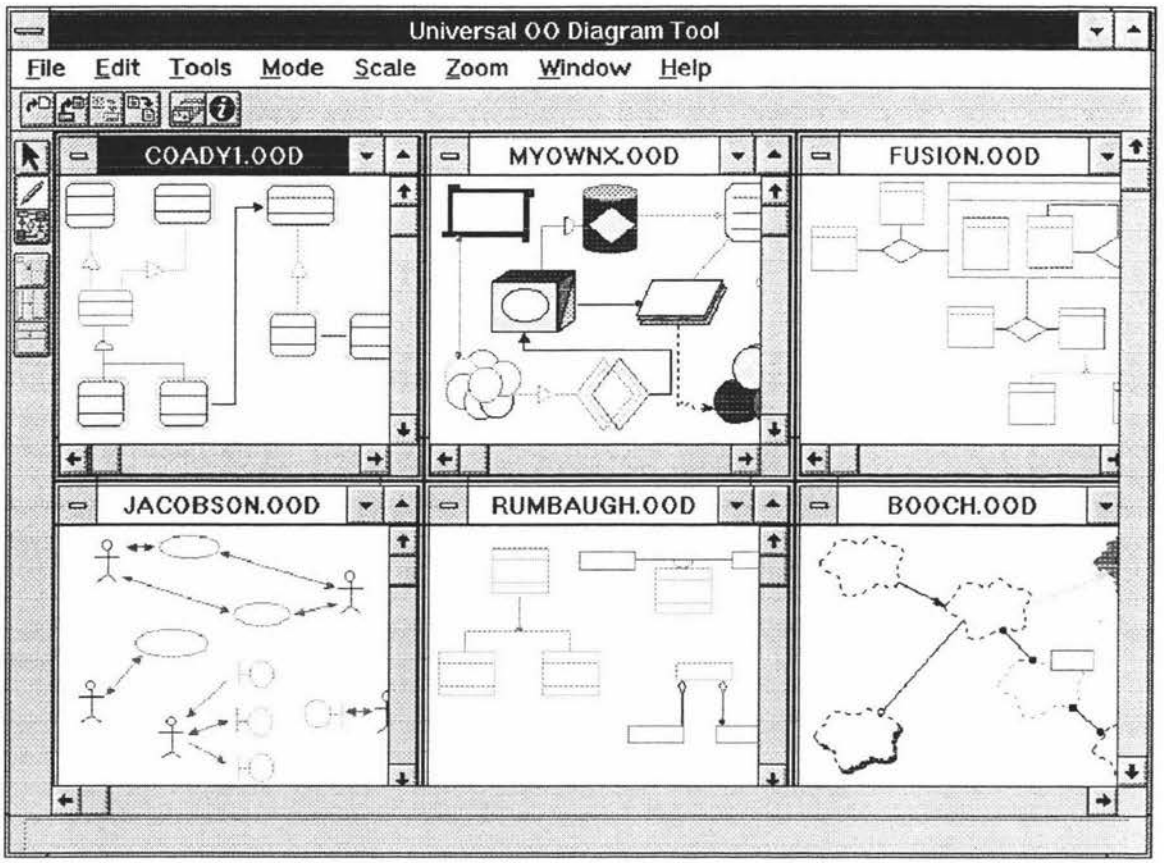


Figure 7.13. The multiple OO design windows(Tile) of UOODT

7.4. Summary

UOODT (Universal OO Diagraming Tool) is a window based graphical design tool which can be customized to support different OO methodologies. It is a methodology-independent OO Diagraming Tool which is provided with different notations by OO Notation Repository.

Based on the OO notation sets stored in OO Notation Repository, UOODT can be used to draw graphical OO diagrams for any OO methodologies. Each OO design window has its OO notation set selected by the designer, the OO notation set of the current OO design window can be changed at any time by loading another OO notation set.

Chapter 8

Conclusion

The mechanism of object-orientation is very different from many other technologies used in the past to develop software systems. It attempts to manage the complexity inherent in real-world problems by abstracting out knowledge, and encapsulating it within objects.

Two obvious characteristics of object-oriented programming are modularity and reusability. The modularity makes the program structure compact, not scattered. The reusability makes the life of systems longer and more circulative. It is easy to modify a program developed using the object-oriented paradigm. Also, it is much easier and convenient to add, delete or modify the system functionality.

Computer-aided software engineering (CASE) became fashionable during the 1980s along with object-oriented programming. More and more OO CASE tools have emerged in the past several years. People have got a lot of benefit from them. These tools can speed the development process, increase communicate more effectively with users, and integrate the work that software developers do on the system from the beginning to the end of the software development life cycle.

The benefits in productivity, efficiency and quality have resulted from the adoption and utilization of CASE tools in many companies. But, most of current OO CASE tools still have problems. They are methodology-dependent and lack flexibility. Therefore, to build methodology-independent OO CASE tools is one of effective ways of solving these problems. The implementation of MIGOCE indicates the possibility and efficiency of this way, and shows that it is worth doing this work and it is a great success.

8.1. Overview of MIGOCE

OO Notation Workshop

OO Notation Workshop is an OO graphical editor which can be used to design existing and new OO notations. It supports methodology-independent OO Diagraming Tool by

providing different notations through OO Notation Repository. It has characteristics as the following:

- It allows designing all kinds of notations supported by existing OO methodologies
- It allows the design of users' own defined notations
- Easy browsing of the existing notations
- Strong file management functions
- Flexible graphical functions

OO Notation Repository

OO Notation Repository is a notation database which contains many sets of notations. It allows the storage of information about all the notations that are basis of forming the Object-Oriented analysis and design environment. OO Notation Repository connects with Universal OO Diagraming Tool. It can provide the diagraming tool with what it needs at all times.

Universal OO Diagraming Tool

Universal OO Diagraming Tool is an upper-CASE tool. In fact, it is a graphical OOA and OOD environment that consists of some individual design windows. The user may complete his Object-Oriented analysis and design under the corresponding environment that supports a certain methodology which the user needs.

Universal OO Diagraming Tool accesses the notations in OO Notation Repository to generate a certain OOA or OOD window which the user requires. It is completely methodology-independent and can be used to draw any OO diagrams according to the user's requirements.

8.2. The Characteristics of MIGOCE

Methodology-Independence

Methodology-Independence is the most important feature of MIGOCE. MIGOCE supports not only the notation graphs of existing methodologies, but also those which

are defined by users. Therefore, MIGOCE can meet all kinds of users' requirements and can be used for many different applications.

The implementation of the MIGOCE system provides a way in which people can overcome one of the main shortcomings of existing OO CASE tools — methodology-dependence.

Directly-manipulable graphical environment

MIGOCE is a directly-manipulable graphical environment. With WIMP interface of MIGOCE, the user can point at and manipulate objects on the screen, and can arbitrarily draw, modify, drag a graph on the workspace. Meanwhile, MIGOCE has strong functions of graph edit and file management. The user can easily add, delete, edit or show notation shapes, and get the system feedback very quick on the screen.

Various input styles make MIGOCE more flexible. The user can drag the mouse to draw a graph like by hand; he can also input data (coordinate points) to get an accurate notation shape. The pull-down menu can be used to specify attributes for these graphs, such as colour, pensize, etc.

Easily-expanded program structure

The MIGOCE system is programmed using object-oriented programming language — C++. The system itself has advantages of OO programming. It is easy to modify and expand the MIGOCE system. This flexibility can meet the requirements of different users and enable the functions of MIGOCE to be enhanced.

MIGOCE has many outstanding advantages. It can do the job which many OO CASE tools cannot do. But, because of methodology-independence, it is more complicated to implement such a system than those systems that are methodology-dependent.

8.3. Further Work

MIGOCE is not a product at the moment. The functions of MIGOCE need to be enhanced. Specially Universal OO Diagraming Tool needs to be completed.

In fact, the variety of relations between objects enhances the complexity of graphical expression. All kinds of possibilities should be taken into account when the diagraming tool is designed. Further work also includes adding the text input function (such as name classes, identify the attributes and operations, etc.).

References

[Aaen, 93] I.Aaen, "CASE User Satisfaction—Impact Evaluations in User Organizations", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*,Singapore, July 19-23, 1993.

[Ammeraal, 91] L.Ammeraal, *C++ for Programmers*, John Wiley & Sons Ltd.1991.

[Anderson, 91] B.Anderson, "Object-oriented programming and interface construction", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Andonoff et al., 95] E.Andonoff, G.Hubert, A.L.Parc and G.Zurfluh, "Modelling Inheritance, Composition and Relationship Links Between Objects, Object Versions and Class Versions", *Advanced Information Systems Engineering*, 7th International Conference, CAiSE'95.

[Annett and Duncan, 67] J.Annett and K.D.Duncan, "Task Analysis and Training Design", *Occupational Psychology*, 41: 211-221, 1967.

[Avison & Fitzgerald, 93] D.E.Avison & G.Fitzgerald, *Information Systems Development -- Methodologies, Techniques and Tools*, Alfred Waller Ltd., 1993.

[Bailin, 89] S.C.Bailin, "An Object-Oriented Requirements Specification Method", *Communications of the ACM*, May 1989 Volume 32 Number 5.

[Bar-David, 93] T.Bar-David, *Object-Oriented Design for C++*, P T R Prentice-Hall, Inc., 1993.

[Barker, 90] R.Barker, *CASE*Method-- Entity Relationship Modelling*, Addison-Wesley Publishing Company, 1990.

[Barros, 93] O.Barros, "Requirements Elicitation and Formalization Through CASE-Supported External Design and Object-Oriented Specification", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*,Singapore, July 19-23, 1993.

[Beck et al., 94] A.Beck, C.Janssen, A.Weisbecker, J.Ziegler, "Integrating Object-Oriented Analysis and Graphical User Interface Design", *Software Engineering and Human-Computer Interaction*, ICSE'94 Workshop on SE-HCI:Joint Research Issues, 1994.

[Beynon-Davies 93] P.Beynon-Davies, *Information Systems Development*, THE MACMILLAN PRESS LTD, 1993.

[Blanc & Korn, 94] L.A.L.Blanc & W.M.Korn, "A Phased Approach to the Evaluation and Selection of CASE Tools", *Information and Software Technology*, 1994 36(5) 267-273.

[Booch, 91a] G.Booch, *Object-Oriented Design with Applications*, The Benjamin Cummings Publishing Company, Inc., 1991.

[Booch, 94b] G.Booch, *Object-Oriented Analysis and Design with Applications*, The Benjamin Cummings Publishing Company, Inc., 1994.

[Booch & Rumbaugh, 95] G.Booch & J.Rumbaugh, *Unified Method*, Rational Software Corporation, 1995.

[Bonfatti & Monari, 94] F.Bonfatti & P.D.Monari, "Towards a General Purpose Approach to Object-Oriented Analysis", *Object-Oriented Methodologies and Systems*, International Symposium, ISOOMS'94.

[Brinkkemper & Hofstede, 90] S.Brinkkemper & A.H.M.ter Hofstede, "The Conceptual Task Model: a Specification Technique between Requirements Engineering and Program Development", *Advanced Information Systems Engineering*, Second Nordic Conference CAiSE'90.

[Brockers & Gruhn, 93] A.Brockers & V.Gruhn, "Computer-Aided Verfication of Software Process Model Properties", *Advanced Information Systems Engineering*, 5th International Conference, CAiSE'93.

[Brough, 92] M.Brough, "Methods for CASE: A Generic Framework", *Advanced Information Systems Engineering*, 4th International Conference, CAiSE'92.

[Brown et al., 93] A.W.Brown, E.J.Morris and P.F.Zarrella, "Experiences with a Federated Environment Testbed", *Software Engineering*--ESEC'93.

[Brumbaugh, 94] D.E.Brumbaugh, *Object-Oriented Development--Building CASE Tools with C++*, John Wiley & Sons, Inc., 1994.

[Budd, 91] T.Budd, *An Introduction to Object Oriented Programming*, Addison Wesley, Reading, MA, 1991.

[Budiardjo, 93] E.K.Budiardjo, "A Custodial Application Generator for Use in a CASE Tool Environment", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*, Singapore, July 19-23, 1993.

[Castano & Antonellis] S.Castano & V.D.Antonellis, "Reuse in Object-Oriented Information Systems Development", *Object-Oriented Methodologies and Systems*, International Symposium, ISOOMS'94.

[Champeaux & Faure, 92] Dennis de Champeaux & Penelope Faure, " A Comparative Study of Object-Oriented Analysis Methods", *Journal of Object-Oriented Programming*, March/April 1992.

[Chen, 76] P.Chen, "The Entity relationship Model: Toward a Unified View of Data",. *ACM Trans. on Database Systems,* 9-36, 1976.

[Chen et al., 92] D.J.Chen, P.J.Lee and S.K.Huang, " Requirements Organization Approach for Object-Oriented Construction of Software Systems", *Information and Software Technology,* Vol 34, No.7, July 1992.

[Christian, 92] K. Christian, *The Microsoft guide to C++ programming*, Microsoft Press, 1992.

[Civello, 93] F.Civello, " Roles for Composite Objects in Object-Oriented Analysis and Design", *OOPSLA'93, pp.376-393.*

[Coad & Yourdon, 90a] P.Coad & E.Yourdon, *Object-Oriented Analysis*, Prentice-Hall International, Inc., 1990.

[Coad & Yourdon, 91b] P.Coad & E.Yourdon, *Object-Oriented Design*, Prentice-Hall International, Inc., 1990.

[Codd, 70], E.F.Codd, "A Relational Model for Large Shared Data Banks", *CACM*, 13(6), 377-87, 1970.

[Coleman et al., 94] D. Coleman, P. Arnold et al, *Object-Oriented Development —The Fusion Method*, Prentice Hall International, Inc. 1994.

[Connell, 89] John L. Connell & Linda B. Shafer, *Structured Rapid Prototyping*,.Yourdon Press Computing Series,1989.

[Constantine & Lockwood, 94] "Essential Use Cases: Essential Modeling for User Interface Design", *Tutorial presented at OZCHI'94, CHISIG of Ergonomics Soc. of Australia, Downer, ACT,* 1994.

[Constantine and Yourdon, 79] L.L. Constantine & E. Yourdon, *Structure Design,* Englewood Cliffs NJ: Prentice-Hall, 1979.

[Cox & Walker, 90] K.Cox & D.Walker, *User-Interface Design*, Woden Printers & Publishers, 1990.

[Dampney, 94] C.N.G.Dampney, "Semantic Correspondence in integrating CASE Tool Repository Schemas", *Information and Software Technology*, 1994 36(2) 87-96.

[Desmarais et al., 90] M.C.Desmarais, L.Giroux and S.Larochelle, "User Modelling for a Text-Editor Coach", *Mental Models and Human-Computer Interaction 1*, (eds.) D.Ackermann and M.J.Tauber, Elsevier Science Publishers B.V.(North-Holland), 1990.

[Diaper,89] D.Diaper, *Task Analysis for Human-Computer Interaction,* D.Diaper/Ellis Horwood Limited,1989.

[Dix, 93] A.Dix, J.Finlay, G.Abowd, and R.Beale, *Human-Computer Interaction*, Prentice Hall International(UK) Limited, 1993.

[Downton, 91] A.Downton, "Engineering the human-computer interface: an overview", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Downton & Leedham, 91] A.Downton & G.Leedham, "Human aspects of human-computer interaction", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Downton, 91] A.Downton, "Dialogue style: basic techniques and guidelines", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Ebert & Engels, 94] J.Ebert & G.Engels, "Structural and Behavioral Views on OMT-Classes", *Object-Oriented Methodologies and Systems,* International Symposium, ISOOMS'94.

[Eckert & Golder,94] G Eckert & P Golder, "Improving Object-Oriented Analysis", *Information and Software Technology,* 1994 36(2) 67-86.

[Ege, 94] R.K.Ege, *Object-Oriented Programming with C++*, Academic Press, Inc., 1994.

[Finkelstein, 90] C.Finkelstein, *An Introduction to Information Engineering From Strategic Planning to Information Systems,* Addison-Wesley Publishing Company, 1990.

[Finkelstein, 93] C.Finkelstein, *Strategic Systems Development*, Addison-Wesley Publishing Company, 1993.

[Firesmith, 93] D.G.Firesmith, *Object-Oriented Requirements Analysis and Logical Design*, John Wiley & Sons, Inc., 1993.

[Firesmith et al, 96] D.Firesmith, B.Henderson-Sellers and I Graham, Open Modeling Language (OML),by the OPEN Consortium, 1996.

[Flamig, 93] B.Flamig, *Practical Data Structures in C++*, John Wiley & Sons, Inc.,1993.

[Fowler, 92] M.Fowler, "A Comparison of Object-Oriented Analysis and Design Methods", Vancouver, Canada: *OOPSLA'92.*

[Freeman & Layzell, 94] M.J.Freeman & P.J.Layzell, "A Meta-Model of Information Systems to Support Reverse Engineering", *Information and Software Technology*, 1994 36(5) 283-294.

[Gane, 90] C.Gane, *Computer-Aided Software Engineering*, Prentice-Hall, Inc., 1990.

[Gibson & Conheeney, 95] M.D.Gibson & K.Conheeney,"Domain Knowledge Reuse During Requirements Engineering", *Advanced Information Systems Engineering*, 7th International Conference, CAiSE'95.

[Glinert, 90] E.P.Glinert, *Visual Programming Environments*, IEEE Computer Society Press, 1990.

[Gobson, 88] M.Gobson, "A Guide to Selecting CASE Tols". *Datamation*, July 1, pp 65-66, 1988.

[Graham, 94a] I Graham, *Object Oriented Methods*, Addison-Wesley Publishing Company Inc., 1994.

[Graham, 94b] I Graham, *Migrating to Object Technology*, Addison-Wesley Publishing Company Inc., 1994.

[Grosberg, 95] J.A.Grosberg, " Design and Construction of Shlaer-Mellor Bridges", *Object Magazine*, June 1995.

[Harrison & Ossher, 93] W.Harrison & H.Ossher, "Subject-Oriented Programming (A Critique of Pure Objects)", *IBM T.J.Watson Research Center,* in OOPSLA'93.

[Hartman et al.,94] M.Hartman, F.W.Jewell, C.Scott And D.Thornton, "Taking an Object-Oriented Methodology into the Real World" OOPSA'94, Addendum to the Proceedings October 23-27,1994.

[Henderson-Sellers & Edwards, 94a] B.Henderson-Sellers & J.Edwards, *The Working Object*, Prentice Hall, Book Two of Object Oriented Knowledge, Prentice Hall, 1994.

[Henderson-Sellers & Graham, 96b] B.Henderson-Sellers & I.Graham, *OPEN:Toward Method Convergance?*, IEEE Computer, April, 1996.

[Hoppe, 90] H.U.Hoppe, "A Grammar-Based Approach to Unifying Task-Oriented and System-Oriented Interface descriptions", *Mental Models and Human-Computer Interaction 1*, (eds.) D.Ackermann and M.J.Tauber, Elsevier Science Publishers B.V.(North-Holland), 1990.

[Hurley, 94] W.D.Hurley, " Integrating User Interface Development and Modern Software Development ", in *The Impact of CASE Technology on Software Processes* (Editor: D.E.Cooke), World Scientific Publishing Co. Pte. Ltd., 1994.

[Høydalsvik & Sindre, 93] G.M. Høydalsvik & G.Sindre, " On the purpose of Object-Oriented Analysis", *OOPSLA'93* pp.240-255.

[Iivari, 91] J.Iivari, "Object-Oriented design of information systems: the Design Process", in *Object Oriented Approach in Information Systems* (Editor: F.V.Assche, B.Moulin, C.Rolland), IFIP, 1991.

[Iivari, 95] J.Iivari, "Object-Orientation as Structural, Fuctional and Behavioural Modelling: A Comparison of Six Methods for Object-Oriented Analysis", *Information and Software Technology*, 1995 37 (3) 155-163.

[Ip & Holden, 92] S.Ip & T.Holden, "A Knowledge Based Technique for the Process Modelling of Information Systems: the Object Life Cycle Diagram", *Advanced Information Systems Engineering*, 4th International Conference, CAiSE'92.

[Jarzabek & Tan, 94] S.Jarzabek & C.L.Tan, "Modeling multiple view of common features in software reengineering for reuse", *Advanced Information Systems Engineering*, 6th International Conference, CAiSE'94.

[Jacobson et al.,92] I. Jacobson, M. Christerson et al.,*Object-Oriented Software Engineering: A Use Case Driven Approach.* Wokingham: Addison-Wesley, 1992

[Jacobson, 93] I. Jacobson, *Object-Oriented Software Engineering,* Addison-Wesley Publishing Company,1993.

[Johnson et al., 84] P.Johnson, D.Diaper and J.B.Long, "Task, skills a.d knowledge", in *Interact'84,* Shackel, B.(ed.), Elsevier, Amsterdam, 1984.

[Johnson and Johnson, 88] P.Johnson, H.Johnson, R.Waddington and A.Shouls, "Task related knowledge structure: analysis, modelling and application", in People and computers IV: From Research to implementation,D.M.Jones and R.Winder (eds), Cambridge University Press, Cambridge, 1988.

[Johnson and Johnson, 90] H.Johnson & P.Johnson,"Integrating task analysis and system design: Surveying designer's needs", *Ergonomics*, 32, 11, 1451-67, 1990.

[Johnson and Johnson, 91] P.Johnson & H.Johnson, "Knowledge analysis of tasks: analysis and specification for human-computer systems", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Johnson, 91] P.Johnson, "User Interface – a framawork to relate tasks, users and designs", *Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals,* edited by H.-J. Bullinger, Elsevier Science Publishers B.V., 1991.

[Johnson et al., 92] P.Johnson, P.Makopoulos, S.Wilson and J.Pycock, "Task Based Design: Mapping Between User Task Models and User Interface Designs", in Proceeding of the Second Interdisciplinary Workshop on Models, 1992.

[Johnson, 92] P.Johnson, *Human Computer Interaction, -- Psychology, Task analysis and Software Engineering,* McGRAW-HILL Book Company Europe, 1992.

[Jones, 91] P.Jones, "Dialogue Design", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Jones & Downton, 91] S.Jones & A.Downton, "Windowing Systems: High-and Low-Level Design Issues", in *Engineering the Human-Computer Interface* (Editor: A.Downton), McGRAW-HILL Book Company(UK) Limited, 1991.

[Karam & Casselman, 93] G M.Karam & R S.Casselman, " A Cataloging Framework for Software Development Methods", *IEEE Computer,* Feb.1993.

[Kendall, 94] K.E. Kendall & J.e.Kendall, *Systems analysis and Design,* Prentice-Hall, Inc., 1994.

[King Galliers, 94] S.King & R.Galliers, "Modelling the CASE Process: Empirical Issues and Future Directions", *Information and Software Technology*, 1994 36(10) 587-596.

[Kovacevic, 92] S. Kovacevic, "A Compositional Model of Human-Computer Dialogues", *Multimedia Interface Design,* (eds.) M. Blattner & R. Dannenberg, ACM Press, Addison-Wesley Publishing Company, 1992.

[Krickhahn et al., 91] R.Krickhahn, M.-J.Schachter-Radig and K.Streng, "The Use of Co-operation Models for Specification and design of User Interfaces", *in Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals (editor: H--J.Bullinger)*, Elsevier Science Piblishers B.V., 1991.

[Kristensen, 94] B.B.Kristensen, " Complex Associations: Abstractions in Object-Oriented Modelling", *OOPSLA 94-* 10/94 Portland, Oregon USA.

[Kruglinski, 94] David j.Kruglinski, *Inside Visual C++*, Microsoft Press,1994.

[Kusters & Wijers, 93] R.J.Kusters & G.M.Wijers, "On the Practical Use of CASE-tools Results of a survey", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*,Singapore, July 19-23, 1993.

[Lacroix & Vanhoedenaghe, 89] M.Lacroix & M.Vanhoedenaghe, "Tool Integration in an Open Environment", 2nd European Software Engineering Conference, ESEC'89.

[Lafore, 93] Robert Lafore, "Lafore's Windows Programming Made Easy", Waite Group Press, 1993.

[Lang, 91] B.Lang, "CASE Support for the Software Process: Advances and Problems", 3rd European Software Engineering Conference, ESEC'91.

[Lee, 93] G.Lee, Object Oriented GUI Application DEvelopment, Prentice Hall, Eaglewood Cliffs, New Jersey, 1993.

[Lee et al., 94] P.J.Lee, D.J.Chen and C.G.Chung, " An Object-Oriented Modelling Approach to System Software Design", *Information and Software Technology*, 1994 36 (11) 683-694.

[Lee, 94] W.Lee, " How to Adapt OO Development Methods in a Software Development Organization - A Case Study", *OOPSLA'94 Addendum to the Proceedings*, October 23-27,1994.

[Leung & Apperley, 93] Y.Leung & M.Apperley, "A Taxonomy of Distortion -Oriented Display Techniques for Graphical Data Presentation," Proceeding of HCI International'93.

[Leung & Apperley, 94] Y.Leung & M.Apperley, "A Review and Taxonomy of Distortion-Oriented Presentation Techniques", ACM Transactions on CHI, 1(2), 126-160, 1994.

[Lewis & Rosenstein et al., 95] T.Lewis & L.Rosenstein et al., *Object Oriented Application Frameworks*, Manning Publications Co., 1995.

[Liddle et al. 94] S.W.Liddle, D.W.Embley and S.N.Woodfield, "A Seamless Model for Object-Oriented System Development", *Object-Oriented Methodologies and Systems*, International Symposium, ISOOMS'94.

[Liskov et al., 77] B.Liskov, A.Snyder, R.Atkinson and C.Schaffert, "Abstraction mechanisms in CLU". *Comm. ACM*, 20(8), 1977.

[Losavio et al., 94] F.Losavio, A.Matteo and F.Schlienger, " Object-Oriented Methodologies of Coad and Yourdon and Booch: Comparison of Graphical Notations", *Information and Software Technology*, 1994 36 (8) 503-514.

[Markopoulos et al., 92] P.Markopoulos, J.Pycock, S.Wilson and P.Johnson, "Adept - A Task Based Design Environment", Proceeding of the Hawaii International Conference on System Sciences, Vol.2, *IEEE Comp.* Soc.Press, 1992.

[Martin, 91] J.Martin, *Rapid Application Development*, MACMILLAN Publishing Company, 1991.

[Martin, 93] J.Martin, *Principles of Object-Oriented Analysis and Design*, James Martin and James J.Odell, 1993.

[Martin & Odell, 95] J.Martin & J.J.Odell, *Object-Oriented Methods: A Foundation* Englewood Cliffs, NJ: Prentice Hall, 1995.

[Martin & Odell, 92] J.Martin & J.J.Odell,

[Martin, 95] R.C.Martin, *Designing Object-Oriented C++ Applications Using The Booch Method*, Prentice-Hall, Inc., 1995.

[Marttiin et al., 93] P.Marttiin, M.Rossi, V.Tahvanainen and K.Lyytinen, "A Comparative Review of CASE Shells: A Preliminary Framekork and Research Outcomes", *Information & Management,* 25 (1993) 11-31.

[Marttiin, 94] P.Marttiin, "Towards Flexible Process Support with a CASE Shell", *Advanced Information Systems Engineering*, 6th International Conference, CAiSE'94.

[Mehandjiska et al., 94] D Mehandjiska, D.Page and P.Clark, "Development of an Intelligent Object Oriented CASE Tool", *Proceeding OOIS'94*, Springer-Verlag, London, 1994.

[Mehandjiska et al., 96] D Mehandjiska, D.Page and M.Choi, "Meta-Modeling and Methodology Support in Object-Oriented CASE tools", *OOIS'96* (submitted paper), 1996.

[Mehandjiska et al., 96] D Mehandjiska, D.Page and S.Dasari, "Generic Knowledge Base for a Methodology Independent Object-Oriented CASE Tool", *IASTED International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, Honolulu, Hawaii, USA, August 1996.

[Mercer, 93] A.Mercer, "Providing Generic CASE Integration", in *Integration Technology for CASE* (Editor: R.Daley), UNICOM Seminars Ltd 1993.

[Mercer, 95] R.Mercer, *Computing Fundamentals with C++*, Franklin, Beedle and Associates Incorporated, 1995.

[Meyer, 87] B.Meyer, "Reusability: The Case for Object-Oriented Design", *IEEE Software vol. 4(2)*, March 1987.

[Meyer, 88] B.Meyer, *Object-oriented Software Construction,* Prentice Hall International (UK) Ltd, 1988.

[Michon, 92] B.Michon, "Highly Iconic Interfaces", *Multimedia Interface Design,* (eds.) M. Blattner & R. Dannenberg, ACM Press, Addison-Wesley Publishing Company, 1992.

[Mosley, 92] V.Mosley, "How to Assess Tools Efficiently and Quantitatively", *IEEE Software*, Vol.9, No.3, May 1992, 99,29-32, 1992.

[Nelson, 94] M.L.Nelson, " Considerations in Choosing a Concurrent/Distributed Object-Oriented Programming Language", *ACM SIGPLAN Notices, Volume 29, No.12*, December 1994.

[Nguyen et al., 92] G.T.Nguyen, D.Rieu and J.Escamilla, "An Object Model for Engineering Design", ECOOP'92(European Conference on Object-Oriented Programming, 1992.)

[Nilsson, 90] E.G.Nilsson, "CASE Tools and Software Factories", *Advanced Information Systems Engineering,* Second Nordic Conference CAiSE'90.

[Normand & Coutaz, 92] V.Normand & J.Coutaz, "Unifying the Design and Implementation of User Interfaces through the Object Paradigm", *ECOOP'92(European Conference on Object-Oriented Programming, 1992.)*

[Oquendo et al., 92] F.Oquendo, J.Zucker and P.Griffiths, "A Meta-CASE Environment for Software Process-centred CASE Environments", *Advanced Information Systems Engineering*, 4th International Conference CAiSE'92.

[OMG, 92] *Object Management Architecture Guide*, OMG C Document 92.11.1, Revision 2.0, Framingham, USA, Sept 1.

[Page et al, 94] D.Page, P.Clark, D.Mehandjiska, An Abstract Definition of Graphical Notations for Object-Oriented Information Systems, in Proceedings OOIS'94, Springer-Verlag, London, 1994.

[Papa et al., 94] M.P.Papa, G.Ragucci, G.Corrente, M.Ferrise, S.Giurleo and D.Vitale, "The Development of an Object-Oriented Multimedia Information System", *Object-Oriented Methodologies and Systems,* International Symposium, ISOOMS'94.

[Papahristos & Gray, 91] S.Papahristos &W.A.Gray, "Federated CASE Environment", *Advanced Information Systems Engineering*, Third International Conference CAiSE'91.

[Papazoglou et al., 94] M.P.Papazoglou, L.Marinos and N.G.Bourbakis, " The Organizational Impact of Integrating Multiple Tools ", in *The Impact of CASE Technology on Software Processes* (Editor: D.E.Cooke), World Scientific Publishing Co. Pte. Ltd., 1994.

[Paulisch, 93] F.N.Paulisch, *The Design of an Extendible Graph Editor,* Lecture Notes in Computer Science 704, Springer-Verlag, 1993.

[Peterson, 92] M.Peterson, *Borland C++--Developer's Bible,* The Waite Group, Inc., 1992.

[Pettersson, 95] M.Pettersson, "Designing the User Interface on Top of a Conceptual Model",*Advanced Information Systems Engineering*, 7th International Conference, CAiSE'95.

[Preece & Rogers et al., 94] J.Preece & Y.Rogers et al., *Human-Computer Interaction*, Addison-Wesley Publishing Company, 1994.

[Pressman, 92] R.S.Pressman, *Software Engineering-- A Practitioner's Approach*, McGraw-Hill, Inc., 1992.

[Rosenberg, 95] D.Rosenberg, " Applying Object-Oriented Methods to Interactive Multiwedia Projects", *Object Magazine*, June 1995.

[Rosenberg, 95] D.Rosenberg, *An Object Methodology Overview*, SIGS BOOKS, ICONIX Software Engineering, Inc., 1995

[Rossi et al., 92] M.Rossi, M.Gustafsson, K.Smolander, L.Johansson and K.Lyytinen, "Metamodeling Editor as a Front End Tool for a CASE Shell", *Advanced Information Systems Engineering*, 4th International Conference CAiSE'92.

[Rouxel et al., 94] C.Rouxel, J.-P.Velu, M.Texier and D.Leroy, "Object-Oriented Methodologies for Large Scale Projects: Does It Work?", *Object-Oriented Methodologies and Systems,* International Symposium, ISOOMS'94.

[Rumbaugh et al., 91a].J.Rumbaugh, M.Blaha, W.Premerlani et al., *Object-Oriented Modelling and Design.* Englewood cliffs. NJ: Prentice-Hall, 1991.

[Rumbaugh, 94b].J.Rumbaugh, "OMT: The Object Model", *Journal of Object Oriented Programming,* Vol.7.No.8., 1994.

[Rumbaugh, 95c].J.Rumbaugh, "OMT: The Functional Model", *Journal of Object Oriented Programming,* Vol.8.No.1., 1995.

[Schildt, 91] H.Schildt, *C++: The Complete Reference*, McGraw-Hill, 1991.

[Schildt, 92] H.Schildt, *Turbo C/C++: The Complete Reference*, McGraw-Hill, 1992.

[Selamat et al., 93] M.H.Selamat, A.T.Othman, M.M.Rahim and I.Khalil, "A Model for CASE Implementation: A Malaysian Experience", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*,Singapore, July 19-23, 1993.

[Senn, 90] J.A.Senn, *Information Systems in Management*, Wadsworth, Inc., 1990.

[Shlaer & Mellor, 88] S.Shlaer & S.J.Mellor, *Object-oriented System Analysis — Modelling the World in Data.* Englewood cliffs. NJ: Yourdin Press, 1988.

[Shlaer & Mellor, 91] S.Shlaer & S.J.Mellor, *Object-Lifecycles: Modelling the World in States.* Englewood cliffs. NJ: Yourdin Press, 1991.

[Shneiderman,93] B.Shneiderman, *Designing the User Interface*, Addison-Wesley Publishing Company, 1993.

[Smolander et al., 91] K.Smolander, K.Lyytinen, Veli-Pekka Tahvanainen and P.Marttiin, "MetaEdit-A Flexible Graphical Environment for Methodology Modelling", *Advanced Information Systems Engineering*, Third International Conference CAiSE'91.

[Sommerville, 96] I.Sommerville, *Software Engineering*, Addison-wesley Publishers Ltd., 1996.

[Song et al., 93] X.Song, H.Fischer and J.Skeer, "Applying Meta-Models to Developing the Semantics Schema for ObjectMaker", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*, Singapore, July 19-23, 1993.

[Sorenson, 88a] P.Sorenson, "First Generation CASE Tools: All Form but Little substance", Research Report, Dept Computational Science, University of Saskatchewan, Saskatchewan, Canada, 1988.

[Sorenson, 88b] P.Sorenson, "On-The Metaview System for Many Specification Environments", *IEEE Software*, Vol. 5, No 2, March, 1988.

[Stobart et al., 93] S.C.Stobart, A.J.van Reeken, G.C.Low, J.J.M.Trienekens, J.O.Jenkins, J.B.Thompson &.D.R.Jeffery, " An Empirical Evaluation of the Use of CASE Tools", *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering, CASE'93*,Singapore, July 19-23, 1993.

[Sumner, 92] M.sumner, the /impact of computer-assisted software Engineering on Systems Development, IFIP Transactions-The Impact of Computer Supported Technologies on Information systems Development edited by Kendall K.E, Lyytinen K, DeGross J I, Elsevier Science Publishers, Amsterdam, 1992.

[Sutcliffe, 91] A.G.Sutcliffe, " Object-Oriented System Development: Survey of Structured Methods", *Information and Software Technology*, Vol 33 No.6 July/August 1991.

[Sutcliffe, 95] A.G.Sutcliffe, *Human-Computer Interface Design*, THE MACMILLAN PRESS LTD,1995.

[Szekely, 94] P.Szekely, "User Interface Prototyping: Tools and Techniques", *Software Engineering and Human-Computer Interaction*, ICSE'94 Workshop on SE-HCI:Joint Research Issues, 1994.

[Tan & Ling, 95] H.B.K.Tan & T.W.Ling, "Recovery of Object-Oriented Design from Existing Data-Intensive Business Programs", *Information and Software Technology*, 1995 37 (2) 67-77.

[Taylor, 94] R.N.Taylor, "User Interface Technology and Software Engineering Environments", *Software Engineering and Human-Computer Interaction*, ICSE'94 Workshop on SE-HCI:Joint Research Issues, 1994.

[Tello, 91] E.R.Tello, *Object-Oriented Programming for Windows*, John Wiley & Sons, Inc., 1991.

[Tichy & Newbery, 87] W.F.Tichy & F.J.Newbery, "Knowledge-based Editors for Directed Graphs", *ESEC'87*. (1st European Software Conference, Strasbourg, France,Sep. 1987.)

[Treu, 94a] S.Treu, *User Interface Design*, Plenum Press, 1994.

[Treu, 94b] S.Treu, *User Interface Evaluation*, Plenum Press, 1994.

[Van den Goor et al, 92] G.van den Goor, S.Hong and S.Brinkkemper, A Comparison of Six Object Oriented Analysis and Design Methodologies, Method Engineering Institute, University of Twente, Netherlands, 1992.

[Verhoef & Hofstede, 95] T.F.Verhoef &A.H.ter Hofstede, "Feasibility of Flexible Information Modelling Support", *Advanced Information Systems Engineering*, 7th International Conference, CAiSE'95.

[Viehstaedt & Minas, 95] G.Viehstaedt & M.Minas, "Graphical Representation and Manipulation of Complex Structures Based on a Formal Model", *Advanced Information Systems Engineering*, 7th International Conference, CAiSE'95.

[Vliet, 93] H.V.Vliet, Software Engineering Principles and Practice, John Wiley & Sons, Ltd., 1993.

[Wasserman & Pircher, 91] A.Wasserman & P.Pircher, "Object-Oriented Structured Design and C++", *Computer Language* Vol. 8(1), 1991.

[Weiss, 92] M.A.Weiss, *Data Structures and Algorithm Analysis*, The Benjamin/Cummings Publishing Company, 1992.

[Whitten et al., 94] J.L.Whitten, L.D.Bentley, and V.M.Barlow, *Systems Analysis & Design Methods*, Richard D. Irwin, Inc., 1994.

[Wieringa, 91] R.J.Wieringa, "Object-Oriented Analysis, Structured Analysis, and Jackson System Development", in *Object Oriented Approach in Information Systems* (Editor: F.V.Assche, B.Moulin, C.Rolland), IFIP, 1991.

[Wijers, 90] G.M.Wijers, "Experiences with the use of CASE-tools in the Netherlands", *Advanced Information Systems Engineering*, Second Nordic Conference CAiSE'90.

[Windsor, 90] P.Windsor, "An Object-Oriented Framework for Prototyping User Interfaces", *in Human-Computer Interaction - INTERACT'90 (Editors:D.Diaper et al.)*, Elsevier Science Publishers B.V.(North-Holland), IFIP, 1990.

[Wirfs-Brock et al., 90] R.Wirfs-Brock, B.Wilkerson and L.Wiener, *Designing Object-Oriented Software,* Prentice-Hall, Inc., 1990.

[Wirth, 71] N.Wirth, "Program devlopment by stepwise refinement", *Comm. ACM*, 14(4), 221-7, 1971.

[Wirth, 76] N.Wirth, *Systematic Programmimg, An Introduction*. Englewood Cliffs NJ: Prentice-Hall, 1976.

[Wulf et al., 76] W.A.Wulf, R.L.London and M.Shaw, "An introduction to the construction and verification of Alphard programs". *IEEE transactions on Software Engineering*, SE-2, 1976.

[Wuwongse & Ma, 91] V.Wuwongse & J.Ma, "An Object-Oriented Approach to Model Management", *Advanced Information Systems Engineering*, Third International Conference CAiSE'91.

[Yourdon, 94] E.Yourdon, *Object-Oriented Systems Design--An Integrated Approach*, P T R Prentice Hall, Inc., 1994.

Appendix Program Listing

```
//-------------------------------------------------------------------------
// winterfa.h -- the head file for the user interface of
//                  OO Notation Workshop
//                  Developed by Bei Zhong (1995)
//-------------------------------------------------------------------------
#if !defined(BASIC_H)
#define BASIC_H

class _USERCLASS TDrawDocument;

class TDrawApp : public TApplication, public TOcModule {
 public:
  TDrawApp();

 protected:
  TMDIClient* Client;
  int       DocMode;
  TView*      View;
  // Override methods of TApplication
  void InitInstance();
  void InitMainWindow();
  // Event handlers
  void EvNewView(TView& view);
  void EvCloseView(TView& view);
  void EvDropFiles(TDropInfo dropInfo);
  void CmAbout();
 private:
  void SetShow(bool visible);
  bool GetShow();
  TDrawDocument* OpenDoc(const char far* name = 0);
  const char far* GetPath();
  // method of TModule ==> const char far* GetName()
 DECLARE_RESPONSE_TABLE(TDrawApp);
 DECLARE_AUTOAGGREGATE(TDrawApp)
  AUTOPROP (Visible, GetShow,SetShow,     TAutoBool, )
  AUTOFUNC0 (NewDoc,  OpenDoc, TAutoObject<TDrawDocument>, )
  AUTOFUNC1 (OpenDoc, OpenDoc, TAutoObject<TDrawDocument>,TAutoString,)
  AUTOPROPRO(AppName, GetName, TAutoString, )
  AUTOPROPRO(FullName,GetPath, TAutoString, )
};
#endif
```

```
//-------------------------------------------------------------------------
// wclasses.h -- the head file for OO Notation Workshop
//           Developed by Bei Zhong (1995)
//-------------------------------------------------------------------------
#if !defined(BASIC1_H)
#define BASIC1_H

#if !defined(OCF_OCLINK_H)
# include <ocf/oclink.h>
#endif

const int Margin  = 15;  // Margins used for copying selection

class _ICLASS TOcPart;
class _USERCLASS TDrawView;

typedef TArray<TPoint> TPoints;
typedef TArrayIterator<TPoint> TPointsIterator;

class TBasicShape : public TPoints {
 public:
            // Constructor to allow construction from a color and a pen size.
            // Also serves as default constructor.
            TBasicShape(const TColor& color = TColor(0),int penSize = 1 ) :
                        TPoints(5, 0, 5), PenSize(penSize),PenType(PS_SOLID),Color(color),
                        Bound(0, 0, 0, 0),Selected(false),FillType(0),FillColor(255,255,255){}
            // Functions to modify and query pen attributes.
            const TColor& QueryColor () const {return Color;}
            const TColor& QueryFillColor() const {return FillColor;} //FillColor
            int   QueryFillType() const {return FillType;}//FillType
            int   QueryPenSize() const {return PenSize;}
            int   QueryPenType() const {return PenType;}
            void  SetFillType(int fillType){ FillType = fillType;}
            void  SetFillColor(TColor fillColor){ FillColor = fillColor;}
            void  SetPen  (TColor& newColor, int penSize = 0,int penType = 0);
            void  SetPen  (int penSize);
            void  SetPenType  (int penType);
  friend bool GetFillColor(TWindow* parent, TBasicShape& BasicShape);
            friend bool GetPenSize(TWindow* parent, TBasicShape& BasicShape);
            friend bool GetPenColor(TWindow* parent, TBasicShape& BasicShape);
            // TBasicShape draws itself.  Returns true if everything went OK.
            virtual bool Draw(TDC&) const;
            void DrawSelection(TDC& dc);
```

```
        // The == operator must be defined for the container class, even if unused
        bool operator ==(const TBasicShape& other) const {return &other == this;}
        friend ostream& operator <<(ostream& os, const TBasicShape& BasicShape);
        friend istream& operator >>(istream& is, TBasicShape& BasicShape);
        // bound related functions
        void Invalidate(TDrawView& view);
        TPoint& GetPos() { return Bound.TopLeft(); }
        TSize& GetSize() {return Bound.Size();}
        TRect& GetBound(){return Bound;}
        void UpdateBound();
        void UpdatePosition(TPoint&);
        void UpdateSize(TSize& newSize);
        bool IsSelected(){return Selected;}
        void Select(bool select = true){Selected = select;}
  void Scale(double scale);
        void ScaleX(double scale);
        void ScaleY(double scale);
        TUIHandle::TWhere Where;
        TRect  Bound; // the enclosing rectangle for BasicShape
 protected:
        bool   Selected;
        int    FillType;
        int    PenSize;
        int    PenType;
        TColor  Color;
        TColor  FillColor;
};

typedef TArray<TBasicShape> TBasicShapes;
typedef TArrayIterator<TBasicShape> TBasicShapesIterator;

class TNotation : public TBasicShapes {
 public:
        TNotation() : TBasicShapes(5, 0, 5),Bound(0, 0, 0, 0),Selected(false) {}
        virtual bool Draw(TDC&) const;
        void DrawSelection(TDC& dc);
        bool operator ==(const TNotation& other) const { return &other == this; }
        friend ostream& operator <<(ostream& os, const TNotation& Notation);
        friend istream& operator >>(istream& is, TNotation& Notation);
        void Invalidate(TDrawView& view);
        TPoint& GetPos(){return Bound.TopLeft();}
        TSize& GetSize() { return Bound.Size();}
        TRect& GetBound() {return Bound;}
        void UpdateBound();
        void UpdatePosition(TPoint&);
        void UpdateSize(TSize& newSize);
        bool IsSelected() {return Selected;}
        void Select(bool select = true){Selected = select;}
        void Scale(double scale);
        void ScaleX(double scale);
        void ScaleY(double scale);
        TUIHandle::TWhere Where;
 protected:
        TRect Bound;   // the enclosing rectangle for a notation
        bool Selected;
};

typedef TArray<TNotation> TNotations;
typedef TArrayIterator<TNotation> TNotationsIterator;

class _USERCLASS TDrawDocument : public TOleDocument {
 public:
        enum {
                PrevProperty = TOleDocument::NextProperty-1,
                BasicShapeCount,
                Description,
                NextProperty,
        };
        enum {UndoNone, UndoDelete, UndoAppend, UndoModify};
        // OCServer changes begin
        TDrawDocument(TDocument* parent = 0);
        // OCServer changes end
        ~TDrawDocument();
        // implement virtual methods of TDocument
        bool   Open(int mode, const char far* path=0);
        bool   OpenSelection(int mode, const char far* path, TPoint far* where);
        bool   Close();
        bool   Commit(bool force = false);
        bool   CommitSelection(TOleWindow& oleWin, void* userData);
        int      FindProperty(const char far* name);  // return index
        int      PropertyFlags(int index);
        const char* PropertyName(int index);
        int      PropertyCount(){ return NextProperty - 1;}
        int      GetProperty(int index, void far* dest, int textlen=0);
        // data access functions
        TBasicShape*  GetBasicShape(uint index);
```

```
            TNotation*    GetNotation(uint index); ////
//          TBasicShapes* GetBasicShapes(){ return BasicShapes;}
            TNotation*    GetBasicShapes(){ return BasicShapes;} ////
            TNotations*   GetNotations() { return Notations;} ////
            TBasicShape*  GetBasicShape(TString& moniker);
            int    AddBasicShape(TBasicShape& BasicShape);
  void  NewNotation();
            int    AddNotation();
            int    AddNotation(TNotation& Notation);
            void   ModifyNotation(TNotation& Notation, uint index);
  void  GoToDesign();
            void   DeleteBasicShape(uint index);
            void   DeleteNotation(uint index);
            void   CopyBasicShape(uint index);
            void   CopyNotation(uint index);
            void   PasteBasicShape();
            void   PasteNotation();
            void   ModifyBasicShape(TBasicShape& BasicShape, uint index);
            void   ClearBasicShapes();
            void   ClearNotations();
            void   LoadNotationSet(const char *name);
            void   SaveNotationSet(const char *name);
  void  UpdateNotation(TNotation& Notation, uint index);
            void   Undo();
            TSize& GetDocSize(){return DocSize;}
 protected:
//          TBasicShapes* BasicShapes;
            TNotation*    BasicShapes; ////
            TBasicShape*  UndoBasicShape;
  TNotations*   Notations;
            int         UndoState;
            int         UndoIndex;
            string      FileInfo;
            TSize       DocSize;     // Document size in log unit
 private:
            // To not interfere with normal mouse operations, the automation
            // entry points have their own BasicShapes with their own attributes.
            long GetPenColor(){ return AutoPenColor;}
            void SetPenColor(long color){
                        AutoPenColor = color;
                        AutoBasicShape->SetPen(TColor(color));
                        }
            short GetPenSize(){ return AutoPenSize;}
            void SetPenSize(short penSize){
                        AutoPenSize = penSize;
                        AutoBasicShape->SetPen(penSize);
                        }
            void AddPoint(short x, short y){AutoBasicShape->Add(TPoint(x,y));}
            void AddBasicShape(){
                        AddBasicShape(*AutoBasicShape);
                        ClearBasicShape();
                        }
            void ClearBasicShape(){
                        delete AutoBasicShape;
                        AutoBasicShape = new TBasicShape(AutoPenColor, AutoPenSize);
                        }
            TBasicShape*  AutoBasicShape;
            long        AutoPenColor;
            short       AutoPenSize;

 DECLARE_AUTOCLASS(TDrawDocument)
            AUTOPROP(PenSize,   GetPenSize,  SetPenSize,  short, )
            AUTOPROP(PenColor,  GetPenColor, SetPenColor, long, )
            AUTOFUNC2V(AddPoint, AddPoint, short, short, )
            AUTOFUNC0V(AddBasicShape,  AddBasicShape, )
            AUTOFUNC0V(ClearBasicShape,ClearBasicShape, )
};

class _USERCLASS TDrawView : public TOleView {
 public:
            TDrawView(TDrawDocument& doc, TWindow* parent = 0);
            ~TDrawView(){ delete BasicShape; }
            static const char far* StaticName(){return "Draw View";}
            const char far* GetViewName() {return StaticName();}
            static void CleanUp(void* userData);
            void SetBasicShapeSelection(TBasicShape* BasicShape);
            void SetBasicShapeSelection(int index);
            TBasicShape* BasicShapeHitTest(TPoint& pt);
  void SetNotationSelection(TNotation* Notation);
            void SetNotationSelection(int index);
  TNotation*  NotationHitTest(TPoint& pt);
            bool Select(uint modKeys, TPoint& point);
            void SetupWindow();
            TBasicShape* GetSelected() {return Selected;}
//          bool EvOcViewSetLink(TOcLinkView& view);
            enum DRAWTOOL {
```

```
                DrawSelect = 0,
                DrawPen,
                DrawLine,
                DrawCircle,
                DrawRect,
                DrawRoundedRect,
                DrawPolygon,
                DrawPolyline,
                DrawArrow,
                DrawDiamond,
                DrawParagram,
                DrawDiamondend,
                DrawSquarend,
                DrawCirclend,
                DrawLinesend,
                DrawHas,
                DrawIs,
        };

protected:
        // Message response functions
        void EvSize(uint sizeType, TSize& size);
        void EvLButtonDown(uint, TPoint&);
 void EvRButtonDown(uint, TPoint&);
        void EvMouseMove(uint, TPoint&);
        void EvLButtonUp(uint, TPoint&);
        bool ShowCursor(HWND, uint, uint);
        void Paint(TDC&, bool, TRect&);
        void CmPenSize();
        void CmPenColor();
        void CmDataInput();
        void CeDataInput(TCommandEnabler& ce);
        void CmClear();
        void CmUndo();
        void CmEditCut();
        void CmEditCopy();
        void CmEditPaste();
        void CmPsSolid();
        void CmPsDash();
        void CmPsDashDot();
        void CmPsDashDotDot();
        void CmPsDot();
        void CePsSolid(TCommandEnabler& ce);
        void CePsDash(TCommandEnabler& ce);
        void CePsDashDot(TCommandEnabler& ce);
        void CePsDashDotDot(TCommandEnabler& ce);
        void CePsDot(TCommandEnabler& ce);
        void CmOverlap();
        void CmFill();
        void CmTransparent();
        void CmFillColor();
        void CeOverlap(TCommandEnabler& ce);
        void CeFill(TCommandEnabler& ce);
        void CeTransparent(TCommandEnabler& ce);
        void CeFillColor(TCommandEnabler& ce);
        void CePenSize(TCommandEnabler& ce);
        void CePenColor(TCommandEnabler& ce);
        void CeEditCut(TCommandEnabler& ce);
        void CeEditCopy(TCommandEnabler& ce);
        void CeEditPaste(TCommandEnabler& ce);
        void CePen(TCommandEnabler&);
        void CeLine(TCommandEnabler&); // CM_LINE 301
        void CeCircle(TCommandEnabler&); // CM_CIRCLE 303
        void CeRect(TCommandEnabler&); // CM_RECT 304
        void CeRoundedRect(TCommandEnabler&); // CM_RECT 305
        void CePolygon(TCommandEnabler&); // CM_ 317
        void CePolyline(TCommandEnabler&); // CM_ 318
        void CeArrow(TCommandEnabler&); // CM_ 319
        void CeDiamond(TCommandEnabler&); // CM_ 329
        void CeParagram(TCommandEnabler&); // CM_ 330
        void CeDiamondend(TCommandEnabler&); // CM_ 331
        void CeSquarend(TCommandEnabler&); // CM_ 332
        void CeCirclend(TCommandEnabler&); // CM_ 333
        void CeLinesend(TCommandEnabler&); // CM_ 334
        void CeHas(TCommandEnabler&); // CM_ 335
        void CeIs(TCommandEnabler&); // CM_ 336
        void CeSelect(TCommandEnabler&);
        void CeOrgSize(TCommandEnabler&);
        void CeDoubleSize(TCommandEnabler&);
        void CeHalfSize(TCommandEnabler&);
        void CmPen();
        void CmLine();
        void CmCircle();
        void CmRect();
        void CmRoundedRect();
        void CmPolygon();
```

```
void CmPolyline();
void CmArrow();
void CmDiamond();
void CmDiamondend();
void CmSquarend();
void CmCirclend();
void CmLinesend();
void CmHas();
void CmIs();
void CmParagram();
void CmSelect();
void CmOrgSize();
void CmDoubleSize();
void CmHalfSize();
void CmNewNotation();
void CmAddNotation();
void CmModifyNotation();
void CmNameNotation();
void CmViewNotations();
void CmGoToDesign();
void CmSaveNotationSet();
void CmLoadNotationSet();
void CeNewNotation(TCommandEnabler& ce);
void CeAddNotation(TCommandEnabler& ce);
void CeModifyNotation(TCommandEnabler& ce);
void CeNameNotation(TCommandEnabler& ce);
void CeViewNotations(TCommandEnabler& ce);
void CeGoToDesign(TCommandEnabler& ce);
void CeLoadNotationSet(TCommandEnabler& ce);
void CeSaveNotationSet(TCommandEnabler& ce);
 void CmScale();
void CeScale(TCommandEnabler& ce);
void CmScalex();
void CeScalex(TCommandEnabler& ce);
void CmScaley();
void CeScaley(TCommandEnabler& ce);
bool VnCommit(bool force);
bool VnRevert(bool clear);
bool VnAppend(uint index);
bool VnDelete(uint index);
bool VnModify(uint index);
TDrawDocument*  DrawDoc;//same as Doc member,but cast to derived class
TPen*           Pen;
TBrush*         Brush;
TBasicShape*    BasicShape;//a single BasicShape sent or received from document
TBasicShape*    Selected;  // To hold a single selected BasicShape
TNotation*      SelectedNotation;
DRAWTOOL        Tool,OldTool;    // current tool selected
TControlBar*    ToolBar;
TPoint          FirstPoint, LastPoint;
 int            CurrentFillMode;
int             DesignMode;
int                     hasCutCopy;
int             ZoomScale;
private:
void InsertObject(TOcInitInfo &init);
void AdjustScroller();
 DECLARE_RESPONSE_TABLE(TDrawView);
};

const int vnDrawAppend    = vnCustomBase+0;
const int vnDrawDelete    = vnCustomBase+1;
const int vnDrawModify    = vnCustomBase+2;

NOTIFY_SIG(vnDrawAppend, uint)
NOTIFY_SIG(vnDrawDelete, uint)
NOTIFY_SIG(vnDrawModify, uint)

#define EV_VN_DRAWAPPEND  VN_DEFINE(vnDrawAppend,  VnAppend,  int)
#define EV_VN_DRAWDELETE  VN_DEFINE(vnDrawDelete,  VnDelete,  int)
#define EV_VN_DRAWMODIFY  VN_DEFINE(vnDrawModify,  VnModify,  int)

#endif // __BASIC1_H
```

```
//----------------------------------------------------------------------------
// winterfa.cpp -- the implementation of the interface classes
//                      for OO Notation Workshop
// Developed by Bei Zhong (1995)
//----------------------------------------------------------------------------
#include <owl/owlpch.h>
#include <owl/applicat.h>
#include <owl/dialog.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/listbox.h>
#include <owl/statusba.h>
#include <owl/docmanag.h>
#include <owl/olemdifr.h>
#include <owl/oledoc.h>
#include <owl/oleview.h>
#include <classlib/arrays.h>
#include <ocf/automacr.h>
#include <stdlib.h>
#include <string.h>
#include "interfac.rc"
#include "classes.h"
#include "interfac.h"

DEFINE_APP_DICTIONARY(AppDictionary);
static TPointer<TOcRegistrar> Registrar;

// Registration

REGISTRATION_FORMAT_BUFFER(100)


BEGIN_REGISTRATION(AppReg)
  REGDATA(clsid,     "{5E4BD340-8ABC-101B-A23B-CE4E85D07ED2}")
  REGDATA(progid,    "Basic.DesignTool.10")
  REGDATA(description,"Tool Used to Design Basic OO Notation")
//  REGDATA(appname,   "Basic OO Notation Design Tool")
  REGDATA(appname,   "OO Notation Workshop")
  REGDATA(cmdline,   "/automation")
  REGDATA(usage,     ocrMultipleUse)
END_REGISTRATION

// TDrawApp

DEFINE_RESPONSE_TABLE1(TDrawApp, TApplication)
  EV_OWLVIEW(dnCreate, EvNewView),
  EV_OWLVIEW(dnClose,  EvCloseView),
  EV_WM_DROPFILES,
  EV_COMMAND(CM_ABOUT, CmAbout),
END_RESPONSE_TABLE;

DEFINE_AUTOAGGREGATE(TDrawApp, OcApp->Aggregate)
  EXPOSE_PROPRW(Visible, TAutoBool,   "Visible",    "Main window shown", 0)
  EXPOSE_METHOD(NewDoc,  TDrawDocument,"NewDocument", "Create new document", 0)
  EXPOSE_METHOD(OpenDoc, TDrawDocument,"OpenDocument","Open existing document", 0)
   REQUIRED_ARG(       TAutoString,  "Name")
  EXPOSE_PROPRO(AppName, TAutoString,  "Name",      "Application name", 0)
  EXPOSE_PROPRO(FullName,TAutoString,  "FullName",   "Complete path to application", 0)
  EXPOSE_APPLICATION(TDrawApp,       "Application", "Application object", 0)
  EXPOSE_QUIT(                "Quit",      "Shutdown application", 0)
END_AUTOAGGREGATE(TDrawApp,tfAppObject|tfCanCreate,"TDrawApp","Application class", 0)

const char far*
TDrawApp::GetPath()
{
  static char buf[_MAX_PATH];
  GetModuleFileName(buf, sizeof(buf)-1);
  return buf;
}

bool
TDrawApp::GetShow()
{
  TFrameWindow* frame = GetMainWindow();
  return (frame && frame->IsWindow() && frame->IsWindowVisible());
}

void
TDrawApp::SetShow(bool visible)
{
  TFrameWindow* frame = GetMainWindow();
  if (frame && frame->IsWindow()) {
    unsigned flags = visible ?
SWP_NOACTIVATE|SWP_NOSIZE|SWP_NOMOVE|SWP_SHOWWINDOW
          :
SWP_NOACTIVATE|SWP_NOSIZE|SWP_NOMOVE|SWP_NOZORDER|SWP_HIDEWINDOW;
    frame->SetWindowPos(HWND_TOP, 0,0,0,0, flags);
  }
}
```

```
extern TDocTemplate drawTpl;

TDrawDocument*
TDrawApp::OpenDoc(const char far* name)
{
 long flags = name ? 0 : dtNewDoc;
 TDocManager* docManager = GetDocManager();
 if (!docManager)
  return 0;
 HWND hWnd = ::GetFocus();
 TDocument* doc = GetDocManager()->CreateDoc(&drawTpl, name, 0, flags);
 ::SetFocus(hWnd);
 return dynamic_cast<TDrawDocument*>(doc);
}

TDrawApp::TDrawApp() :
 TApplication(::AppReg["appname"], ::Module, &::AppDictionary)
{
}

void
TDrawApp::InitMainWindow()
{
 View = 0;

 // Determine whether it's MDI or SDI app
 // If single use or DLL server, run multiple instances as SDI apps
 DocMode = (IsOptionSet(amSingleUse) ||
        !IsOptionSet(amExeMode)) ? dmSDI : dmMDI;

 TOleFrame* frame;
 if (DocMode == dmMDI)
  // Construct the TOleMDIFrame frame window
  frame = new TOleMDIFrame(GetName(), 0, *(Client = new TMDIClient), true);
 else
  // Construct the TOleFrame frame window
  frame = new TOleFrame(GetName(), 0, true);

 // Construct a status bar
 TStatusBar* sb = new TStatusBar(frame, TGadget::Recessed);

 // Construct a control bar
 TControlBar* cb = new TControlBar(frame);
 cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_FILESAVEAS, CM_FILESAVEAS, TButtonGadget::Command));
 cb->Insert(*new TSeparatorGadget);
 cb->Insert(*new TButtonGadget(CM_PENSIZE, CM_PENSIZE, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_PENCOLOR, CM_PENCOLOR, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_OVERLAP, CM_OVERLAP, TButtonGadget::Exclusive));
 cb->Insert(*new TButtonGadget(CM_FILL, CM_FILL, TButtonGadget::Exclusive));
 cb->Insert(*new TButtonGadget(CM_TRANSPARENT, CM_TRANSPARENT,
TButtonGadget::Exclusive));
 cb->Insert(*new TButtonGadget(CM_FILLCOLOR, CM_FILLCOLOR, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_DATAINPUT, CM_DATAINPUT, TButtonGadget::Command));
 cb->Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT, TButtonGadget::Command));
 cb->SetHintMode(TGadgetWindow::EnterHints);
 cb->Attr.Id = IDW_TOOLBAR;

 TControlBar* cb1 = new TControlBar(frame,TGadgetWindow::Vertical);//my...Horizontal
 cb1->Insert(*new TButtonGadget(CM_PEN, CM_PEN, TButtonGadget::Exclusive));
 cb1->Insert(*new TButtonGadget(CM_POLYLINE, CM_POLYLINE, TButtonGadget::Exclusive));
 cb1->Insert(*new TButtonGadget(CM_RECT, CM_RECT, TButtonGadget::Exclusive));
 cb1->Insert(*new TButtonGadget(CM_DIAMOND, CM_DIAMOND, TButtonGadget::Command));
 cb1->Insert(*new TButtonGadget(CM_PARAGRAM, CM_PARAGRAM, TButtonGadget::Command));
 cb1->Insert(*new TSeparatorGadget);
 cb1->Insert(*new TButtonGadget(CM_DIAMONDEND, CM_DIAMONDEND,
TButtonGadget::Exclusive));
 cb1->Insert(*new TButtonGadget(CM_SQUAREND, CM_SQUAREND, TButtonGadget::Exclusive));
 cb1->Insert(*new TButtonGadget(CM_HAS, CM_HAS, TButtonGadget::Exclusive));
 cb1->SetHintMode(TGadgetWindow::EnterHints);
 cb1->Attr.Id = IDW_TOOLBAR;

 TControlBar* cb2 = new TControlBar(frame,TGadgetWindow::Vertical);//my...Horizontal
 cb2->Insert(*new TButtonGadget(CM_SELECT, CM_SELECT, TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_LINE, CM_LINE, TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_CIRCLE, CM_CIRCLE, TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_ROUNDEDRECT, CM_ROUNDEDRECT,
TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_POLYGON, CM_POLYGON, TButtonGadget::Exclusive));
 cb2->Insert(*new TSeparatorGadget);
 cb2->Insert(*new TButtonGadget(CM_ARROW, CM_ARROW, TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_CIRCLEND, CM_CIRCLEND, TButtonGadget::Exclusive));
 cb2->Insert(*new TButtonGadget(CM_LINESEND, CM_LINESEND, TButtonGadget::Exclusive));
```

```
  cb2->SetHintMode(TGadgetWindow::EnterHints);
  cb2->Attr.Id = IDW_TOOLBAR;

  // Insert the status bar and control bar into the frame
  frame->Insert(*sb, TDecoratedFrame::Bottom);
  frame->Insert(*cb, TDecoratedFrame::Top);
  frame->Insert(*cb1, TDecoratedFrame::Left);
  frame->Insert(*cb2, TDecoratedFrame::Left);

  // Set the main window and its menu
  SetMainWindow(frame);
  if (DocMode == dmMDI)
    GetMainWindow()->SetMenuDescr(TMenuDescr(IDM_MDICMNDS));
  else
    GetMainWindow()->SetMenuDescr(TMenuDescr(IDM_SDICMNDS));

  // Install the document manager
  SetDocManager(new TDocManager(DocMode));
}

void
TDrawApp::InitInstance()
{
  TApplication::InitInstance();
  GetMainWindow()->DragAcceptFiles(true);
}

void
TDrawApp::EvDropFiles(TDropInfo dropInfo)
{
  int fileCount = dropInfo.DragQueryFileCount();
  for (int index = 0; index < fileCount; index++) {
    int fileLength = dropInfo.DragQueryFileNameLen(index)+1;
    char* filePath = new char [fileLength];
    dropInfo.DragQueryFile(index, filePath, fileLength);
    TDocTemplate* tpl = GetDocManager()->MatchTemplate(filePath);
    if (tpl)
      GetDocManager()->CreateDoc(tpl, filePath);
    else { // Embedding from file
      TOleWindow* oleWin = TYPESAFE_DOWNCAST(View, TOleWindow);

      // Let the TOleWindow handle the dropped file
      //
      if (oleWin)
        oleWin->ForwardMessage();
    }

    delete filePath;
  }
  dropInfo.DragFinish();
}

void
TDrawApp::EvNewView(TView& view)
{
  View = &view;
  TOleView* ov = TYPESAFE_DOWNCAST(&view, TOleView);
  if (DocMode == dmMDI) {
    if (view.GetDocument().IsEmbedded() && ov && !ov->IsOpenEditing()) {
      TWindow* vw = view.GetWindow();
      TOleFrame* mw = TYPESAFE_DOWNCAST(GetMainWindow(), TOleFrame);
      TWindow* rvb = mw->GetRemViewBucket();
      vw->SetParent(rvb);
      vw->Create();
    }
    else {
      TMDIChild* child = new TMDIChild(*Client, 0);
      if (view.GetViewMenu())
        child->SetMenuDescr(*view.GetViewMenu());
      child->Create();
      child->SetClientWindow(view.GetWindow());
    }
  }
  else {
    TFrameWindow* frame = GetMainWindow();
    frame->SetClientWindow(view.GetWindow());
    if (!view.IsOK())
      frame->SetClientWindow(0);
    else if (view.GetViewMenu())
      frame->MergeMenu(*view.GetViewMenu());
  }

  if (!ov || !ov->GetOcRemView())
    OcApp->SetOption(amEmbedding, false);
}
```

```
void
TDrawApp::EvCloseView(TView& /*view*/)
{
 // nothing needs to be done here for MDI
 if (DocMode == dmSDI) {
  GetMainWindow()->SetClientWindow(0);
 }
}

void
TDrawApp::CmAbout()
{
 TDialog(&TWindow(::GetFocus(), this), IDD_ABOUT).Execute();
}

int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
 try {
  Registrar = new TOcRegistrar(AppReg, TOleDocViewAutoFactory<TDrawApp>(),
                               TApplication::GetCmdLine(), ::DocTemplateStaticHead);
  if (Registrar->IsOptionSet(amAnyRegOption))
   return 0;

  return Registrar->Run();
 }
 catch (xmsg& x) {
  ::MessageBox(0, x.why().c_str(), "Exception", MB_OK);
 }
 return -1;
}
```

```
//----------------------------------------------------------------------------
// wshape.cpp -- the implementation of the TBasicShape class for
//                    OO Notation Workshop
//                    -- Developed by Bei Zhong (1995)
//----------------------------------------------------------------------------
#include <owl/owlpch.h>
#include <owl/dc.h>
#include <owl/gdiobjec.h>
#include <ocf/automacr.h>
#include <ocf/ocdata.h>
#include <ocf/ocremvie.h>
#include "classes.h"
//The implementation of TBasicShape class

void
TBasicShape::SetPenType(int penType)
{
 if (penType < 0)
            PenType = 0; //PS_SOLID
 else
            PenType = penType;
}


void
TBasicShape::SetPen(int penSize)
{
 if (penSize < 1)
            PenSize = 1;
 else
            PenSize = penSize;
}

void
TBasicShape::SetPen(TColor& newColor, int penSize, int penType)
{
// If penSize isn't the default (0), set PenSize to the new size.
 if (penSize)
             PenSize = penSize;
 if (penType < 0)
             PenType = 0;
 else
             PenType = penType;
 Color = newColor;
}

void
TBasicShape::Invalidate(TDrawView& view)
{
 TOleClientDC dc(view);
 TRect rUpdate(GetBound());
 rUpdate.Inflate(1, 1);
 dc.LPtoDP((TPoint *)&rUpdate, 2);
 TUIHandle handle(rUpdate, TUIHandle::Framed);
 rUpdate = handle.GetBoundingRect();
 view.GetDocument().NotifyViews(vnInvalidate, (long)&rUpdate, 0);
}

void
TBasicShape::UpdateBound()
{
 // Iterates through the points in the BasicShape i.
 TPointsIterator j(*this);
 if (!j) return;
 TPoint p = j++;
 if (p.x < 1000) {
                        p = j++;
                        Bound.Set(p.x, p.y, p.x, p.y);

                        while (j) {
                         p = j++;
                         if ((p.x - PenSize) < Bound.left)
                         Bound.left = (p.x - PenSize);
                         if ((p.x + PenSize) > Bound.right)
                         Bound.right = (p.x + PenSize);
                         if ((p.y - PenSize) < Bound.top)
                         Bound.top = (p.y - PenSize);
                         if ((p.y + PenSize) > Bound.bottom)
                         Bound.bottom = (p.y + PenSize);
                         }
                         Bound.right  += 1;
                         Bound.bottom += 1;
 }
 else {;}// type > 1000
}
```

```
void
TBasicShape::UpdatePosition(TPoint& newPos)
{
        TPointsIterator i(*this);
        i++;
        for (i; i; i++) {
  TPoint* pt = (TPoint *)&i.Current();
  pt->x += newPos.x;
  pt->y += newPos.y;
 }
 Bound.Offset(newPos.x, newPos.y);
}

void
TBasicShape::UpdateSize(TSize& newSize)
{
 TSize delta = newSize - GetSize();
 TPointsIterator i(*this);
 i++;
 for (i; i; i++) {
  TPoint* pt = (TPoint *)&i.Current();
  pt->x += (((pt->x - Bound.left) * delta.cx + (GetSize().cx >> 1))/GetSize().cx);
  pt->y += (((pt->y - Bound.top) * delta.cy + (GetSize().cy >> 1))/GetSize().cy);
 }
 Bound.right = Bound.left + newSize.cx;
 Bound.bottom = Bound.top  + newSize.cy;
}

void
TBasicShape::DrawSelection(TDC &dc)
{
// TUIHandle(Bound, TUIHandle::Grapples|TUIHandle::DashFramed).Paint(dc);
        TUIHandle(Bound, TUIHandle::Grapples).Paint(dc);
}

bool
TBasicShape::Draw(TDC& dc) const
{
// Set pen for the dc to the values for this BasicShape
//  TPen pen(Color, PenSize, PS_INSIDEFRAME);
 TPen pen(Color, PenSize, PenType);
 dc.SelectObject(pen);
 TBrush* Brush;

 // Iterates through the points in the BasicShape i.
 TPointsIterator j(*this);
 TPoint FirstPoint,LastPoint, p = j++;

 if (p.x > 800 && p.x < 900 ) { // lines and curves
                bool first = true;
                while (j) {
//   TPoint p = j++;
                p = j++;
                if (!first)
                        dc.LineTo(p);
                else {
                        dc.MoveTo(p);
                        first = false;
                        }
                }
        }
        else if (p.x == 998) { // circle
         FirstPoint = j++;
         LastPoint  = j++;
         if(FillType == 1){ //overlap
                        Brush = new TBrush(TColor(255,255,255));
                        dc.SelectObject(*Brush);
//                      dc.Rectangle(FirstPoint, LastPoint);
                        dc.Ellipse(FirstPoint, LastPoint);
                        delete Brush; // delete overlap brush
                        }
         else if (FillType == 2){ // fill
                        Brush = new TBrush(TColor(FillColor));
                        dc.SelectObject(*Brush);
//                      dc.Rectangle(FirstPoint, LastPoint);
                        dc.Ellipse(FirstPoint, LastPoint);
                        delete Brush; // delete fill brush
                        }
         else {  // transparent
                        dc.SelectStockObject(NULL_BRUSH);
//                      dc.Rectangle(FirstPoint, LastPoint);
    dc.Ellipse(FirstPoint, LastPoint);
                        }

        } // circle

        else if (p.x == 999) {  // recrangle
```

```
 FirstPoint = j++;
 LastPoint  = j++;
 if(FillType == 1){ //overlap
            Brush = new TBrush(TColor(255,255,255));
            dc.SelectObject(*Brush);
            dc.Rectangle(FirstPoint, LastPoint);
            delete Brush; // delete overlap brush
            }
 else if (FillType == 2){ // fill
            Brush = new TBrush(TColor(FillColor));
            dc.SelectObject(*Brush);
            dc.Rectangle(FirstPoint, LastPoint);
            delete Brush; // delete fill brush
            }
 else {  // transparent
            dc.SelectStockObject(NULL_BRUSH);
            dc.Rectangle(FirstPoint, LastPoint);
            }

} // recrangle

          else if (p.x == 997) {  // rounded recrangle
 FirstPoint = j++;
 LastPoint  = j++;
 if(FillType == 1){ //overlap
            Brush = new TBrush(TColor(255,255,255));
            dc.SelectObject(*Brush);
            dc.RoundRect(FirstPoint, LastPoint,TPoint(20,20));
            delete Brush; // delete overlap brush
            }
 else if (FillType == 2){ // fill
            Brush = new TBrush(TColor(FillColor));
            dc.SelectObject(*Brush);
            dc.RoundRect(FirstPoint, LastPoint,TPoint(20,20));
            delete Brush; // delete fill brush
            }
 else {  // transparent
            dc.SelectStockObject(NULL_BRUSH);
            dc.RoundRect(FirstPoint, LastPoint,TPoint(20,20));
            }

} // rounded recrangle
else if ( p.x == 995 ) { // polyline
            TPoint points[400];
            int count = 0;
            while (j && count < 400) {
                        points[count]=j++;
                        count+=1;
                        }
            dc.Polyline(points, count);
} // polyline

else if ( p.x == 996 || p.x == 993 || p.x == 994) { // polygon//diamond//parallolgram
            TPoint points[400];
            int count = 0;
            while (j && count < 400) {
                        points[count]=j++;
                        count+=1;
                        }
            if(FillType == 1){ //overlap
             Brush = new TBrush(TColor(255,255,255));
             dc.SelectObject(*Brush);
             dc.Polygon(points, count);
             delete Brush; // delete overlap brush
             }
 else if (FillType == 2){ // fill
             Brush = new TBrush(TColor(FillColor));
             dc.SelectObject(*Brush);
             dc.Polygon(points, count);
             delete Brush; // delete fill brush
             }
 else {  // transparent
             dc.SelectStockObject(NULL_BRUSH);
             dc.Polygon(points, count);
             }
} // polygon

 else if (p.x == 992 ) { // arrow
             FirstPoint = j++;
             LastPoint  = j++;
             dc.MoveTo(FirstPoint);
             dc.LineTo(LastPoint);

             TPoint points[40];
             int count = 0;
             points[count]=LastPoint;
```

```
                    count+=1;

            while (j) {
                    points[count]=j++;
                    count+=1;
                    }
            if(FillType == 1){ //overlap
             Brush = new TBrush(TColor(255,255,255));
             dc.SelectObject(*Brush);
             dc.Polygon(points, count);
             delete Brush; // delete overlap brush
             }
 else if (FillType == 2){ // fill
             Brush = new TBrush(TColor(FillColor));
             dc.SelectObject(*Brush);
             dc.Polygon(points, count);
             delete Brush; // delete fill brush
             }
 else {  // transparent
             dc.SelectStockObject(NULL_BRUSH);
             dc.Polygon(points, count);
             }
} // arrow + line

             else if (p.x == 990) { //Diamondend
            FirstPoint = j++;
            LastPoint  = j++;
            dc.MoveTo(FirstPoint);
            dc.LineTo(LastPoint);

            TPoint points[40];
            int count = 0;

            points[count]=LastPoint;
                    count+=1;

            while (j) {
                    points[count]=j++;
                    count+=1;
                    }
            if(FillType == 1){ //overlap
             Brush = new TBrush(TColor(255,255,255));
             dc.SelectObject(*Brush);
                    dc.Polygon(points, count);
                    delete Brush; // delete overlap brush
                    }
             else if (FillType == 2){ // fill
                    Brush = new TBrush(TColor(FillColor));
                    dc.SelectObject(*Brush);
                    dc.Polygon(points, count);
                    delete Brush; // delete fill brush
                    }
             else {  // transparent
                    dc.SelectStockObject(NULL_BRUSH);
                    dc.Polygon(points, count);
                    }
            } //  line with diamonedend

                    else if (p.x == 989 ) { // square end
                    FirstPoint = j++;
                    LastPoint  = j++;
                    dc.MoveTo(FirstPoint);
                    dc.LineTo(LastPoint);

FirstPoint = j++;
                    LastPoint  = j++;

                    if(FillType == 1){ //overlap
                     Brush = new TBrush(TColor(255,255,255));
                     dc.SelectObject(*Brush);
                     dc.Rectangle(FirstPoint, LastPoint);
                     delete Brush; // delete overlap brush
                     }
             else if (FillType == 2){ // fill
                     Brush = new TBrush(TColor(FillColor));
                     dc.SelectObject(*Brush);
                     dc.Rectangle(FirstPoint, LastPoint);
                     delete Brush; // delete fill brush
                     }
             else {  // transparent
                     dc.SelectStockObject(NULL_BRUSH);
                     dc.Rectangle(FirstPoint, LastPoint);
                     }
            } // line with diamond end

                    else if (p.x == 991 ) { // circle end
```

```
                    FirstPoint = j++;
                    LastPoint  = j++;
                    dc.MoveTo(FirstPoint);
                    dc.LineTo(LastPoint);

                    FirstPoint = j++;
                    LastPoint  = j++;

                    if(FillType == 1){ //overlap
                     Brush = new TBrush(TColor(255,255,255));
                     dc.SelectObject(*Brush);
                     dc.Ellipse(FirstPoint, LastPoint);
                     delete Brush; // delete overlap brush
                     }
               else if (FillType == 2){ // fill
                     Brush = new TBrush(TColor(FillColor));
                     dc.SelectObject(*Brush);
                     dc.Ellipse(FirstPoint, LastPoint);
                     delete Brush; // delete fill brush
                     }
               else {  // transparent
                     dc.SelectStockObject(NULL_BRUSH);
                     dc.Ellipse(FirstPoint, LastPoint);
                     }
             } // line with circle end

             else if (p.x == 988 ) { // linesend
                     FirstPoint = j++;
                     LastPoint  = j++;
                     dc.MoveTo(FirstPoint);
                     dc.LineTo(LastPoint);

FirstPoint = j++;
                     LastPoint  = j++;
                     dc.MoveTo(FirstPoint);
                     dc.LineTo(LastPoint);

                     FirstPoint = LastPoint;
                     LastPoint  = j++;
                     dc.MoveTo(FirstPoint);
                     dc.LineTo(LastPoint);
            } // line with short lines end
                     else if (p.x == 987||p.x == 986  ) { // has||is
                     FirstPoint = j++;
                     LastPoint  = j++;
                     dc.MoveTo(FirstPoint);
                     dc.LineTo(LastPoint);

                     TPoint points[40];
                     int count = 0;

                     while (j) {
                                points[count]=j++;
                                count+=1;
                                }
                     if(FillType == 1){ //overlap
                      Brush = new TBrush(TColor(255,255,255));
                      dc.SelectObject(*Brush);
                      dc.Polygon(points, count);
                      delete Brush; // delete overlap brush
                      }
               else if (FillType == 2){ // fill
                      Brush = new TBrush(TColor(FillColor));
                      dc.SelectObject(*Brush);
                      dc.Polygon(points, count);
                      delete Brush; // delete fill brush
                      }
               else {  // transparent
                      dc.SelectStockObject(NULL_BRUSH);
                      dc.Polygon(points, count);
                      }
             } // Has

             else if (p.x > 1000){;} // others

 dc.RestorePen();
 return true;
}

ostream&
operator <<(ostream& os, const TBasicShape& BasicShape)
{
 // Write the number of points in the BasicShape
 os << BasicShape.GetItemsInContainer();

 // Get and write pen attributes.
```

```
 os << '' << BasicShape.Color  << '' << BasicShape.PenSize
            << '' << BasicShape.PenType << '' << BasicShape.FillColor
            << '' << BasicShape.FillType;

 // Get an iterator for the array of points
 TPointsIterator j(BasicShape);

 // While the iterator is valid (i.e. we haven't run out of points)
 while(j)
  // Write the point from the iterator and increment the array.
  os << j++;
 os << '\n';

 // return the stream object
 return os;
}

istream&
operator >>(istream& is, TBasicShape& BasicShape)
{
 unsigned numPoints;
 is >> numPoints;

 COLORREF color, fillColor;
 int penSize,penType, fillType;
 is >> color >> penSize >> penType  >> fillColor >> fillType;
 BasicShape.SetPen(TColor(color), penSize, penType);
 BasicShape.SetFillType(fillType);
 BasicShape.SetFillColor(fillColor);

 while (numPoints--) {
  TPoint point;
            is >> point;
  BasicShape.Add(point);
 }

 // return the stream object
 return is;
}

bool
GetPenSize(TWindow* parent, TBasicShape& BasicShape)
{
 char inputText[6];

 wsprintf(inputText, "%d", BasicShape.QueryPenSize());
 if (TInputDialog(parent, "BasicShape Thickness",
                                        "Input a new thickness:",
                                        inputText,
          sizeof(inputText)).Execute() != IDOK)
           return false;
 BasicShape.SetPen(atoi(inputText));

 if (BasicShape.QueryPenSize() < 1)
           BasicShape.SetPen(1);

 return true;
}

bool
GetPenColor(TWindow* parent, TBasicShape& BasicShape)
{
 TChooseColorDialog::TData colors;
 static TColor custColors[16] =
 {
  0x010101L, 0x101010L, 0x202020L, 0x303030L,
           0x404040L, 0x505050L, 0x606060L, 0x707070L,
           0x808080L, 0x909090L, 0xA0A0A0L, 0xB0B0B0L,
           0xC0C0C0L, 0xD0D0D0L, 0xE0E0E0L, 0xF0F0F0L
 };

 colors.Flags = CC_RGBINIT;
 colors.Color = TColor(BasicShape.QueryColor());
 colors.CustColors = custColors;
 if (TChooseColorDialog(parent, colors).Execute() != IDOK)
           return false;
 BasicShape.SetPen(colors.Color,BasicShape.QueryPenSize(),BasicShape.QueryPenType());
 return true;
}

bool
GetFillColor(TWindow* parent, TBasicShape& BasicShape)
{
 TChooseColorDialog::TData colors;
 static TColor custColors[16] =
 {
```

```
  0x010101L, 0x101010L, 0x202020L, 0x303030L,
            0x404040L, 0x505050L, 0x606060L, 0x707070L,
            0x808080L, 0x909090L, 0xA0A0A0L, 0xB0B0B0L,
            0xC0C0C0L, 0xD0D0D0L, 0xE0E0E0L, 0xF0F0F0L
 };

 colors.Flags = CC_RGBINIT;
 colors.Color = TColor(BasicShape.QueryFillColor());
 colors.CustColors = custColors;
 if (TChooseColorDialog(parent, colors).Execute() != IDOK)
            return false;
 BasicShape.SetFillColor(colors.Color);
 return true;
}

void
TBasicShape::Scale(double scale)
{
 TPointsIterator i(*this);
 i++;
 for (i; i; i++) {
            TPoint* pt = (TPoint *)&i.Current();
            pt->x = Bound.left + (pt->x - Bound.left)*scale;
            pt->y = Bound.top  + (pt->y - Bound.top)*scale ;
 }
 Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
 Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
 // UpdateBound()
 }

void
TBasicShape::ScaleX(double scale)
{
 TPointsIterator i(*this);
 i++;
 for (i; i; i++) {
            TPoint* pt = (TPoint *)&i.Current();
            pt->x = Bound.left + (pt->x - Bound.left)*scale;
//          pt->y = Bound.top  + (pt->y - Bound.top)*scale ;
 }
 Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
// Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
// UpdateBound()
}

void
TBasicShape::ScaleY(double scale)
{
 TPointsIterator i(*this);
 i++;
 for (i; i; i++) {
            TPoint* pt = (TPoint *)&i.Current();
//          pt->x = Bound.left + (pt->x - Bound.left)*scale;
            pt->y = Bound.top  + (pt->y - Bound.top)*scale ;
 }
//Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
 Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
// UpdateBound()
}
```

```
//----------------------------------------------------------------------------
// wnotatio.cpp -- the implementation of the TNotation class
//                      for OO Notation Workshop
//                  -- Developed by Bei Zhong (1995)
//----------------------------------------------------------------------------
#include <owl/owlpch.h>
#include <owl/dc.h>
#include <owl/gdiobjec.h>
#include <ocf/automacr.h>
#include <ocf/ocdata.h>
#include <ocf/ocremvie.h>
#include "classes.h"

//============The implementation of TNotation class==================

ostream&
operator <<(ostream& os, const TNotation& Notation)
{
//  os << Notation.GetItemsInContainer();

  TBasicShapesIterator j(Notation);

  while(j)
            os << j++;
//  os << '\n';
  // return the stream object
  return os;
}


istream&
operator >>(istream& is, TNotation& Notation)
{
  unsigned numNotation;
  is >> numNotation;

  while (numNotation--) {
            TBasicShape BasicShape;
            is >> BasicShape;
            Notation.Add(BasicShape);
  }
  return is;
}

bool
TNotation::Draw(TDC& dc) const
{
  // Iterates through the basic shapes in a notation.
  TBasicShapesIterator j(*this);

  while (j) {
  TBasicShape basicshape = j++;
  basicshape.Draw(dc);
  }
  return true;
}

void
TNotation::DrawSelection(TDC &dc)
{
//  TUIHandle(Bound, TUIHandle::Grapples|TUIHandle::DashFramed).Paint(dc);
    TUIHandle(Bound, TUIHandle::Grapples).Paint(dc);

}

void
TNotation::Invalidate(TDrawView& view)
{
  TOleClientDC dc(view);
  TRect rUpdate(GetBound());
  rUpdate.Inflate(1, 1);
  dc.LPtoDP((TPoint *)&rUpdate, 2);
  TUIHandle handle(rUpdate, TUIHandle::Framed);
  rUpdate = handle.GetBoundingRect();
  view.GetDocument().NotifyViews(vnInvalidate, (long)&rUpdate, 0);
}


void
TNotation::UpdatePosition(TPoint& newPos)
{
            TBasicShapesIterator i(*this);
            for (i; i; i++)
            {
                        TBasicShape* bs = (TBasicShape*)&i.Current();
                        bs->UpdatePosition(newPos);
            }
```

```
            Bound.Offset(newPos.x, newPos.y);
}

void
TNotation::UpdateSize(TSize& newSize)
{
 TBasicShapesIterator i(*this);
 for (i; i; i++)
            {
             TBasicShape* bs = (TBasicShape*)&i.Current();
             bs->UpdateSize(newSize);
            }
 Bound.right = Bound.left + newSize.cx;
 Bound.bottom = Bound.top  + newSize.cy;
}

void
TNotation::UpdateBound()
{
 // Iterates through the basic shapes in the notation i.
 TBasicShapesIterator j(*this);
 if (!j)  return;
            TBasicShape bs = j++;
//  TBasicShape* bs = (TBasicShape*)&j.Current();
//  TBasicShape bs = TBasicShapes(j);
 Bound.Set(bs.Bound.left,bs.Bound.top,bs.Bound.right,bs.Bound.bottom);

 while (j) {
            bs = j++;
            if (bs.Bound.left < Bound.left)
                        Bound.left = bs.Bound.left;
                        if (bs.Bound.right > Bound.right)
                        Bound.right = bs.Bound.right;
                        if (bs.Bound.top < Bound.top)
                        Bound.top = bs.Bound.top;
                        if (bs.Bound.bottom > Bound.bottom)
                        Bound.bottom = bs.Bound.bottom;
            }
}

void
TNotation::Scale(double scale)
{
TPoint nPos = GetPos();
TPoint bPos,newPos;
TBasicShapesIterator i(*this);
 for (i; i; i++)
            {
             TBasicShape* bs = (TBasicShape*)&i.Current();
             bs->Scale(scale);
             bPos = bs->GetPos();
             newPos.x = nPos.x + (bPos.x - nPos.x)*scale - bPos.x;
             newPos.y = nPos.y + (bPos.y - nPos.y)*scale - bPos.y;
             bs->UpdatePosition(newPos);
             bs->UpdateBound();
            }
 Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
 Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
 // UpdateBound()
}

void
TNotation::ScaleX(double scale)
{
 TPoint nPos = GetPos();
 TPoint bPos,newPos;
 TBasicShapesIterator i(*this);
 for (i; i; i++)
            {
             TBasicShape* bs = (TBasicShape*)&i.Current();
             bs->ScaleX(scale);
   bPos = bs->GetPos();
             newPos.x = nPos.x + (bPos.x - nPos.x)*scale - bPos.x;
             newPos.y = 0;
             bs->UpdatePosition(newPos);
             bs->UpdateBound();
            }
 Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
//  Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
 // UpdateBound()
}

void
TNotation::ScaleY(double scale)
{
 TPoint nPos = GetPos();
```

```
TPoint bPos,newPos;
 TBasicShapesIterator i(*this);
 for (i; i; i++)
                {
                 TBasicShape* bs = (TBasicShape*)&i.Current();
                 bs->ScaleY(scale);
   bPos = bs->GetPos();
                 newPos.x = 0;
                 newPos.y = nPos.y + (bPos.y - nPos.y)*scale - bPos.y;
                 bs->UpdatePosition(newPos);
                 bs->UpdateBound();
                 }
//  Bound.right  = Bound.left +(Bound.right  - Bound.left)*scale;
  Bound.bottom = Bound.top  +(Bound.bottom - Bound.top)*scale;
  // UpdateBound()
}
```

```
//----------------------------------------------------------------------------
// wdocumen.cpp -- the implementation of TDrawDocument class for
//                      OO Notation Workshop
//                  -- Developed by Bei Zhong (1995)
//----------------------------------------------------------------------------
#include <owl/owlpch.h>
#include <owl/dc.h>
#include <owl/inputdia.h>
#include <owl/chooseco.h>
#include <owl/gdiobjec.h>
#include <owl/docmanag.h>
#include <owl/listbox.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/scroller.h>
#include <classlib/arrays.h>
#include <owl/olemdifr.h>
#include <owl/oledoc.h>
#include <owl/oleview.h>
#include <ocf/automacr.h>
#include <ocf/ocdata.h>
#include <ocf/ocremvie.h>
#include "classes.h"
#include "interfac.h"
#include "views.rc"


DEFINE_AUTOCLASS(TDrawDocument)
// EXPOSE_PROPRW(PenType,   TAutoShort, "PenType",   "Current pen type", 0)
 EXPOSE_PROPRW(PenSize,   TAutoShort, "PenSize",   "Current pen size", 0)
 EXPOSE_PROPRW(PenColor,  TAutoLong,  "PenColor",  "Current pen color", 0)
 EXPOSE_METHOD(AddPoint,  TAutoVoid,  "AddPoint",
                                "Add a point to the current BasicShape", 0)
 REQUIRED_ARG(          TAutoShort, "X")
 REQUIRED_ARG(          TAutoShort, "Y")
 EXPOSE_METHOD(AddBasicShape,   TAutoVoid,  "AddBasicShape",
                                        "Add current BasicShape into drawing", 0)
 EXPOSE_METHOD(ClearBasicShape, TAutoVoid,  "ClearBasicShape",
                                                "Erases current BasicShape", 0)
 EXPOSE_APPLICATION(     TDrawApp,  "Application","Application object", 0)
END_AUTOCLASS(TDrawDocument, tfNormal, "TDrawDoc",  "Draw document class", 0)
TDrawDocument::TDrawDocument(TDocument* parent)
 : TOleDocument(parent), UndoBasicShape(0), UndoState(UndoNone)
{
 BasicShapes     = new TNotation();
 Notations       = new TNotations(20,0,5);
 AutoPenSize     = 1;
 AutoPenColor    = RGB(0, 0, 0);
 AutoBasicShape  = new TBasicShape(AutoPenColor, AutoPenSize);

 TScreenDC dc;
 DocSize.cx = dc.GetDeviceCaps(LOGPIXELSX)* 85 / 10;
 DocSize.cy = dc.GetDeviceCaps(LOGPIXELSY)* 11;
}

TDrawDocument::~TDrawDocument()
{
 delete AutoBasicShape;
 delete BasicShapes;
 delete UndoBasicShape;
 delete Notations;
}

TBasicShape*
TDrawDocument::GetBasicShape(TString& moniker)
{
 int index = atoi(moniker);
 return GetBasicShape(index);
}

bool
TDrawDocument::CommitSelection(TOleWindow& oleWin, void* userData)
{
 TOleDocument::CommitSelection(oleWin, userData);
 TDrawView* drawView = TYPESAFE_DOWNCAST(&oleWin, TDrawView);
 TOutStream* os = OutStream(ofWrite);
 if (!os || !drawView)
  return false;
 // Make the BasicShape usable in a container by adjusting its origin
 TBasicShape* BasicShape = (TBasicShape*)userData;
 int i = BasicShape? 1 : 0;
 TPoint newPos(Margin, Margin);
 if (BasicShape) {
  newPos -= BasicShape->GetBound().TopLeft();
  BasicShape->UpdatePosition(newPos);
```

```
 }
 // Write the number of BasicShapes in the figure
 *os << i;
 // Append a description using a resource string
 *os << ' ' << FileInfo << '\n';
 // Copy the current BasicShape from the iterator and increment the array.
 if (BasicShape)
   *os << *BasicShape;

 delete os;
 // restore BasicShape
 if (BasicShape)
          BasicShape->UpdatePosition(-newPos);
 return true;
}

bool
TDrawDocument::Commit(bool force)
{
 TOleDocument::Commit(force);
 TOutStream* os = OutStream(ofWrite);
 if (!os)
          return false;

// Write the number of notations in the figure
  *os << Notations->GetItemsInContainer();
// Append a description using a resource string
  *os << ' ' << FileInfo << '\n';
// Get an iterator for the array of Notations
  TNotationsIterator j(*Notations);
// While the iterator is valid (i.e. we haven't run out of Notations)
  while (j) {
// Copy the current notation from the iterator and increment the array.
          TNotation Notation = j++;
// Write the number of BasicShapes in the notation
  *os << Notation.GetItemsInContainer();
  *os << ' ' << FileInfo << '\n';
  // Get an iterator for the array of BasicShapes
   *os << Notation;
          }
// Write the number of BasicShapes in the figure
  *os << BasicShapes->GetItemsInContainer();
  *os << ' ' << FileInfo << '\n';
```

```
// Get an iterator for the array of BasicShapes
 TBasicShapesIterator i(*BasicShapes);

 // While the iterator is valid (i.e. we haven't run out of BasicShapes)
 while (i) {
          // Copy the current BasicShape from the iterator and increment the array.
          *os << i++;
 }
 delete os;
 // Commit the storage if it was opened in transacted mode
 TOleDocument::CommitTransactedStorage();
 SetDirty(false);

 return true;
}

bool
TDrawDocument::Open(int mode, const char far* path)
{
 char fileinfo[100];
//  ::MessageBox(0, "open", "open", MB_OK);
 TOleDocument::Open(mode, path);   // normally path should be null
 if (GetDocPath()) {
          TInStream* is = (TInStream*)InStream(ofRead);
          if (!is)
                   return false;

//-----------------------
unsigned  numNotations, numBasicShapes;

          *is >> numNotations;
          is->getline(fileinfo, sizeof(fileinfo));
          while (numNotations--) {
                   *is >> numBasicShapes;
                   is->getline(fileinfo, sizeof(fileinfo));
                   while (numBasicShapes--) {
                            TBasicShape BasicShape;
                            *is >> BasicShape;
                            BasicShape.UpdateBound();
                            BasicShapes->Add(BasicShape);
                    }
                   BasicShapes->UpdateBound();//////
                   Notations->Add(*BasicShapes);
```

```
                                BasicShapes->Flush();
        }

//      is->getline(fileinfo, sizeof(fileinfo));
        *is >> numBasicShapes;
        is->getline(fileinfo, sizeof(fileinfo));
        while (numBasicShapes--) {
                TBasicShape BasicShape;
                *is >> BasicShape;
                BasicShape.UpdateBound();
                BasicShapes->Add(BasicShape);
        }

        delete is;

        FileInfo = fileinfo;
 } else {
        FileInfo = string(*::Module,IDS_FILEINFO);
 }
 SetDirty(false);
 UndoState = UndoNone;
 return true;
}

//
// Read in the BasicShapes from a storage and put them at the location specified
// by where
//
bool
TDrawDocument::OpenSelection(int mode, const char far* path, TPoint far* where)
{
 char fileinfo[100];

 TOleDocument::Open(mode, path);   // normally path should be null
 if (GetDocPath()) {
  TInStream* is = (TInStream*)InStream(ofRead);
  if (!is)
   return false;

        unsigned numBasicShapes;
  *is >> numBasicShapes;
        is->getline(fileinfo, sizeof(fileinfo));
  while (numBasicShapes--) {
   TBasicShape BasicShape;
   *is >> BasicShape;
   if (where) {
    TPoint newPos(where->x, where->y);
    newPos -= BasicShape.GetBound().TopLeft();
    BasicShape.UpdatePosition(newPos);
   }
   BasicShape.UpdateBound();
   BasicShapes->Add(BasicShape);
  }

  delete is;

        FileInfo = fileinfo;
 } else {
        FileInfo = string(*::Module,IDS_FILEINFO);
 }
 SetDirty(false);
 UndoState = UndoNone;
 return true;
}

bool
TDrawDocument::Close()
{
 Notations->Flush();
 BasicShapes->Flush();
 return TOleDocument::Close();
}

TBasicShape*
TDrawDocument::GetBasicShape(uint index)
{
 return index < BasicShapes->GetItemsInContainer() ? &(*BasicShapes)[index] : 0;
}

TNotation*
TDrawDocument::GetNotation(uint index)
{
 return index < Notations->GetItemsInContainer() ? &(*Notations)[index] : 0;
}
```

```
int
TDrawDocument::AddBasicShape(TBasicShape& BasicShape)
{
 int index = BasicShapes->GetItemsInContainer();
 BasicShapes->Add(BasicShape);
 SetDirty(true);
 NotifyViews(vnDrawAppend, index);
 UndoState = UndoAppend;
 return index;
}

int
TDrawDocument::AddNotation(TNotation& Notation)
{
 int index = Notations->GetItemsInContainer();
 Notations->Add(Notation);
 SetDirty(true);
 NotifyViews(vnDrawAppend, index);
// UndoState = UndoAppend;
 return index;
}

void
TDrawDocument::NewNotation()
{
 BasicShapes->Flush();
 NotifyViews(vnRevert, true);
// SetDirty(true);
}

int
TDrawDocument::AddNotation()
{
 BasicShapes->UpdateBound();
 int index = Notations->GetItemsInContainer();
 Notations->Add(*BasicShapes);
// BasicShapes->Flush();
 NotifyViews(vnRevert, true);
 SetDirty(true);
 return index;
}
```

```
void
TDrawDocument::ModifyNotation(TNotation& Notation, uint index)
{
 TPoint newPos = TPoint(15,15)-Notation.GetPos();
 BasicShapes->Flush();
 TBasicShapesIterator j(Notation);
 while(j){
          TBasicShape *bs = (TBasicShape*)&j.Current();
          TBasicShape *BasicShape = new TBasicShape(*bs);
          BasicShape->UpdatePosition(newPos);
          BasicShapes->Add(*BasicShape);
          j++;
 }
 NotifyViews(vnRevert, true);
 SetDirty(true);
}

void
TDrawDocument::GoToDesign()
{
// BasicShapes->Flush();
 NotifyViews(vnRevert, true);
// SetDirty(true);
}

void
TDrawDocument::ModifyBasicShape(TBasicShape& BasicShape, uint index)
{
 delete UndoBasicShape;
 UndoBasicShape = new TBasicShape((*BasicShapes)[index]);
 SetDirty(true);
 (*BasicShapes)[index] = BasicShape;
 NotifyViews(vnDrawModify, index);
 UndoState = UndoModify;
 UndoIndex = index;
}

void
TDrawDocument::ClearBasicShapes()
{
 BasicShapes->Flush();
 NotifyViews(vnRevert, true);
}
```

```
void
TDrawDocument::ClearNotations()
{
 Notations->Flush();
 NotifyViews(vnRevert, true);
}

void
TDrawDocument::Undo()
{
 switch (UndoState) {
  case UndoAppend:
   DeleteBasicShape(BasicShapes->GetItemsInContainer()-1);
   return;
            case UndoDelete:
             UndoBasicShape->Select(false);
                    AddBasicShape(*UndoBasicShape);
   delete UndoBasicShape;
   UndoBasicShape = 0;
   return;
  case UndoModify:
   TBasicShape* temp = UndoBasicShape;
   UndoBasicShape = 0;
   ModifyBasicShape(*temp, UndoIndex);
   delete temp;
 }
}

static char* PropNames[] = {
 "BasicShape Count",    // BasicShapeCount
 "Description",     // Description
};

static int PropFlags[] = {
 pfGetBinary|pfGetText, // BasicShapeCount
 pfGetText,          // Description
};

const char*
TDrawDocument::PropertyName(int index)
{
 if (index <= PrevProperty)
  return TStorageDocument::PropertyName(index);  // OC server change
 else if (index < NextProperty)
  return PropNames[index-PrevProperty-1];
 else
  return 0;
}

int
TDrawDocument::PropertyFlags(int index)
{
 if (index <= PrevProperty)
  return TStorageDocument::PropertyFlags(index); // OC server change
 else if (index < NextProperty)
  return PropFlags[index-PrevProperty-1];
 else
  return 0;
}

int
TDrawDocument::FindProperty(const char far* name)
{
 for (int i=0; i < NextProperty-PrevProperty-1; i++)
  if (strcmp(PropNames[i], name) == 0)
   return i+PrevProperty+1;
 return 0;
}

int
TDrawDocument::GetProperty(int prop, void far* dest, int textlen)
{
 switch(prop) {
  case BasicShapeCount: {
                    int count = BasicShapes->GetItemsInContainer();
   if (!textlen) {
    *(int far*)dest = count;
    return sizeof(int);
   }
   return wsprintf((char far*)dest, "%d", count);
          }
  case Description:
   char* temp = new char[textlen]; // need local copy for medium model
   int len = FileInfo.copy(temp, textlen);
   strcpy((char far*)dest, temp);
   return len;
```

```
 }
 return TStorageDocument::GetProperty(prop, dest, textlen); // OC server change
}

void
TDrawDocument::CopyBasicShape(uint index)
{

 const TBasicShape* oldBasicShape = GetBasicShape(index);
 if (!oldBasicShape)
              return;
 // copy basicshape
              ofstream out("shape");
 if (!out){
              ::MessageBox(0, "Copy operation failed!","Copy BasicShape",  MB_OK);
              return;
              }
 out << *oldBasicShape;
 out.close();

}
void
TDrawDocument::CopyNotation(uint index)
{

 const TNotation* oldNotation = GetNotation(index);
 if (!oldNotation)
              return;
// copy notation
 ofstream out("notation");
 if (!out){
              ::MessageBox(0,  "Copy operation failed!","Copy Notation", MB_OK);
              return ;
              }
 out << oldNotation->GetItemsInContainer();
 out << ' ' << FileInfo << '\n';
 out << *oldNotation;
 out.close();
}


void
TDrawDocument::PasteBasicShape()
{
 TBasicShape BasicShape;
// copy a BasicShape
 ifstream in("shape");
 if (!in){
              ::MessageBox(0,"Paste operation failed!", "Paste BasicShape",  MB_OK);
              return;
              }
//  in >> oldBasicShape;
//  AddBasicShape(oldBasicShape);
 unsigned numPoints;
 in >> numPoints;

 COLORREF color, fillColor;
 int penSize,penType, fillType;
 in >> color >> penSize >> penType  >> fillColor >> fillType;
 BasicShape.SetPen(TColor(color), penSize, penType);
 BasicShape.SetFillType(fillType);
 BasicShape.SetFillColor(fillColor);

 while (numPoints--) {
  TPoint point;
              in >> point;
              BasicShape.Add(point);
 }
 BasicShape.UpdateBound();
 TPoint newPos = TPoint(10,10) - BasicShape.GetPos();
 BasicShape.UpdatePosition(newPos);//TPoint&
 BasicShapes->Add(BasicShape);
 in.close();
 SetDirty(true);
 NotifyViews(vnDrawDelete, 0);// repaint wiew
}

void
TDrawDocument::PasteNotation()
{
TNotation    Notation;
// copy notation
 ifstream in("notation");
 if (!in){
              ::MessageBox(0, "Paste operation failed!","Paste Notation",  MB_OK);
              return;
```

```
        }
                                unsigned numBasicShapes;
                                char fileinfo[100];
                                in >> numBasicShapes;
                                in.getline(fileinfo, sizeof(fileinfo));
                                while (numBasicShapes--) {
                                        TBasicShape BasicShape;
                                        in >> BasicShape;
                                        BasicShape.UpdateBound();
                                        Notation.Add(BasicShape);
                                 }
                                Notation.UpdateBound();//////
                                Notations->Add(Notation); //  *Notation
                                Notation.Flush();
 in.close();
 SetDirty(true);
 NotifyViews(vnDrawDelete, 0);// repaint wiew
}

void
TDrawDocument::DeleteBasicShape(uint index)
{
 const TBasicShape* oldBasicShape = GetBasicShape(index);
 if (!oldBasicShape)
            return;
 delete UndoBasicShape;
 UndoBasicShape = new TBasicShape(*oldBasicShape);
 // copy basicshape
 ofstream out("shape");
 if (!out){
            ::MessageBox(0, "Cut operation failed!","Cut BasicShape",  MB_OK);
            return;
            }
 out << *oldBasicShape;
 out.close();
 BasicShapes->Detach(index);
 SetDirty(true);
 NotifyViews(vnDrawDelete, index);
 UndoState = UndoDelete;
}

void
TDrawDocument::DeleteNotation(uint index)
{
 const TNotation* oldNotation = GetNotation(index);
 if (!oldNotation)
            return;
 ofstream out("notation");
 if (!out){
            ::MessageBox(0,  "Cut operation failed!","Cut Notation", MB_OK);
            return ;
            }
 out << oldNotation->GetItemsInContainer();
 out << ' ' << FileInfo << '\n';
 out << *oldNotation;
 out.close();
 Notations->Detach(index);
 SetDirty(true);
 NotifyViews(vnDrawDelete, index);
// UndoState = UndoDelete;
}

void
TDrawDocument::LoadNotationSet(const char *name)
{
 ifstream in(name);
 if(!in){
            ::MessageBox(0, "Cannot find the selected OO Notation file! ", "Load a selected OO Notation
set", MB_OK);
            return;
            }

unsigned  numNotations, numBasicShapes;

 char fileinfo[100];

            BasicShapes->Flush();
            in >> numNotations;
            in.getline(fileinfo, sizeof(fileinfo));
            while (numNotations--) {
                                in >> numBasicShapes;
                                in.getline(fileinfo, sizeof(fileinfo));
                                while (numBasicShapes--) {
                                        TBasicShape BasicShape;
                                        in >> BasicShape;
```

```
                                        BasicShape.UpdateBound();
                                        BasicShapes->Add(BasicShape);
                                }
                                BasicShapes->UpdateBound();
                                Notations->Add(*BasicShapes);
                                BasicShapes->Flush();

        }
 SetDirty(true);
 NotifyViews(vnDrawDelete,0);
 in.close();
}

void
TDrawDocument::SaveNotationSet(const char *name)
{
 ofstream out(name);
 if (!out){
        ::MessageBox(0, "Cannot open the named file!", "Save Notation Set", MB_OK);
        return;
        }
 out << Notations->GetItemsInContainer();
 out << ' ' << FileInfo << '\n';
 TNotationsIterator j1(*Notations);
 while (j1) {
        TNotation Notation = j1++;
 out << Notation.GetItemsInContainer();
 out << ' ' << FileInfo << '\n';
 out << Notation;
        }
 out.close();
}

void
TDrawDocument::UpdateNotation(TNotation& Notation, uint index)
{
 (*Notations)[index] = Notation;
 SetDirty(true);
 NotifyViews(vnDrawModify, index);
}
```

```
//----------------------------------------------------------------------------
// wview.cpp -- the implementation of the TDrawView class for
//                 OO Notation Workshop
//              -- Developed by Bei Zhong (1995)
//----------------------------------------------------------------------------
#include <math.h>
#include <owl/owlpch.h>
#include <owl/dc.h>
#include <owl/inputdia.h>
#include <owl/chooseco.h>
#include <owl/gdiobjec.h>
#include <owl/docmanag.h>
#include <owl/listbox.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/scroller.h>
#include <classlib/arrays.h>
#include <owl/olemdifr.h>
#include <owl/oledoc.h>
#include <owl/oleview.h>
#include <ocf/automacr.h>
#include <ocf/ocdata.h>
#include <ocf/ocremvie.h>
#include "classes.h"
#include "interfac.h"
#include "views.rc"

const char  DocContent[] = "All";
const char  DrawPadFormat[] = "DrawPad";

BEGIN_REGISTRATION(DocReg)
// REGDATA(progid,    "DrawPad.Drawing.17")
  REGDATA(progid,    "Bshapes.Editing.01")
  REGDATA(description,"BasicNotationFile(*.bzh)")
  REGDATA(menuname,  "Drawing")
  REGDATA(extension, "bzh")
  REGDATA(docfilter, "*.bzh")
  REGDOCFLAGS(dtAutoOpen | dtAutoDelete | dtUpdateDir | dtCreatePrompt | dtRegisterExt)
  REGDATA(insertable, "")
  REGDATA(verb0,     "&Edit")
  REGDATA(verb1,     "&Open")
  REGFORMAT(0, DrawPadFormat,  ocrContent, ocrHGlobal,          ocrGetSet)
  REGFORMAT(1, ocrEmbedSource, ocrContent, ocrIStorage,         ocrGetSet)
  REGFORMAT(2, ocrMetafilePict, ocrContent, ocrMfPict|ocrStaticMed, ocrGet)
  REGFORMAT(3, ocrBitmap,      ocrContent, ocrGDI|ocrStaticMed,    ocrGet)
  REGFORMAT(4, ocrDib,         ocrContent, ocrHGlobal|ocrStaticMed, ocrGet)
END_REGISTRATION

DEFINE_DOC_TEMPLATE_CLASS(TDrawDocument,TDrawView,DrawTemplate);
DrawTemplate drawTpl(DocReg);

// ============= the implementation of the TDrawView class =======

DEFINE_RESPONSE_TABLE1(TDrawView, TOleView)
  EV_WM_LBUTTONDOWN,
  EV_WM_MOUSEMOVE,
  EV_WM_LBUTTONUP,
  EV_WM_RBUTTONDOWN,
  EV_COMMAND(CM_PEN, CmPen),
  EV_COMMAND_ENABLE(CM_PEN, CePen),
  EV_COMMAND(CM_SELECT, CmSelect),
  EV_COMMAND_ENABLE(CM_SELECT, CeSelect),
  EV_COMMAND(CM_LINE, CmLine),
  EV_COMMAND_ENABLE(CM_LINE, CeLine),
  EV_COMMAND(CM_CIRCLE, CmCircle),
  EV_COMMAND_ENABLE(CM_CIRCLE, CeCircle),
  EV_COMMAND(CM_RECT, CmRect),
  EV_COMMAND_ENABLE(CM_RECT, CeRect),
  EV_COMMAND(CM_ROUNDEDRECT, CmRoundedRect),
  EV_COMMAND_ENABLE(CM_ROUNDEDRECT, CeRoundedRect),
  EV_COMMAND(CM_POLYGON, CmPolygon),
  EV_COMMAND_ENABLE(CM_POLYGON, CePolygon),
  EV_COMMAND(CM_POLYLINE, CmPolyline),
  EV_COMMAND_ENABLE(CM_POLYLINE, CePolyline),
  EV_COMMAND(CM_ARROW, CmArrow),
  EV_COMMAND_ENABLE(CM_ARROW, CeArrow),
  EV_COMMAND(CM_DIAMOND, CmDiamond),
  EV_COMMAND_ENABLE(CM_DIAMOND, CeDiamond),
  EV_COMMAND(CM_PARAGRAM, CmParagram),
  EV_COMMAND_ENABLE(CM_PARAGRAM, CeParagram),
  EV_COMMAND(CM_DIAMONDEND, CmDiamondend),
  EV_COMMAND_ENABLE(CM_DIAMONDEND, CeDiamondend),
  EV_COMMAND(CM_SQUAREND, CmSquarend),
  EV_COMMAND_ENABLE(CM_SQUAREND, CeSquarend),
  EV_COMMAND(CM_CIRCLEND, CmCirclend),
  EV_COMMAND_ENABLE(CM_CIRCLEND, CeCirclend),
```

```
EV_COMMAND(CM_LINESEND, CmLinesend),
EV_COMMAND_ENABLE(CM_LINESEND, CeLinesend),
EV_COMMAND(CM_HAS, CmHas),
EV_COMMAND_ENABLE(CM_HAS, CeHas),
EV_COMMAND(CM_IS, CmIs),
EV_COMMAND_ENABLE(CM_IS, CeIs),
EV_COMMAND(CM_PENSIZE, CmPenSize),
EV_COMMAND(CM_PENCOLOR, CmPenColor),
EV_COMMAND(CM_DATAINPUT, CmDataInput),
EV_COMMAND_ENABLE(CM_DATAINPUT, CeDataInput),
EV_COMMAND(CM_EDITCLEAR, CmClear),
EV_COMMAND(CM_EDITUNDO, CmUndo),
EV_COMMAND(CM_EDITCUT, CmEditCut),
EV_COMMAND(CM_EDITCOPY, CmEditCopy),
EV_COMMAND(CM_EDITPASTE, CmEditPaste),
EV_COMMAND_ENABLE(CM_EDITCUT, CeEditCut),
EV_COMMAND_ENABLE(CM_EDITCOPY, CeEditCopy),
EV_COMMAND_ENABLE(CM_EDITPASTE, CeEditPaste),
EV_COMMAND(CM_PSSOLID, CmPsSolid),
EV_COMMAND_ENABLE(CM_PSSOLID, CePsSolid),
EV_COMMAND(CM_PSDASH, CmPsDash),
EV_COMMAND_ENABLE(CM_PSDASH, CePsDash),
EV_COMMAND(CM_PSDASHDOT, CmPsDashDot),
EV_COMMAND_ENABLE(CM_PSDASHDOT, CePsDashDot),
EV_COMMAND(CM_PSDASHDOTDOT, CmPsDashDotDot),
EV_COMMAND_ENABLE(CM_PSDASHDOTDOT, CePsDashDotDot),
EV_COMMAND(CM_PSDOT, CmPsDot),
EV_COMMAND_ENABLE(CM_PSDOT, CePsDot),
EV_COMMAND(CM_OVERLAP,CmOverlap ),
EV_COMMAND_ENABLE(CM_OVERLAP,CeOverlap ),
EV_COMMAND(CM_FILL,CmFill),
EV_COMMAND_ENABLE(CM_FILL,CeFill),
EV_COMMAND(CM_TRANSPARENT,CmTransparent),
EV_COMMAND_ENABLE(CM_TRANSPARENT,CeTransparent ),
EV_COMMAND(CM_FILLCOLOR,CmFillColor),
EV_COMMAND_ENABLE(CM_FILLCOLOR,CeFillColor ),
EV_COMMAND(CM_ORGSIZE, CmOrgSize),
EV_COMMAND(CM_DOUBLESIZE, CmDoubleSize),
EV_COMMAND(CM_HALFSIZE, CmHalfSize),
EV_COMMAND_ENABLE(CM_ORGSIZE, CeOrgSize),
EV_COMMAND_ENABLE(CM_DOUBLESIZE, CeDoubleSize),
EV_COMMAND_ENABLE(CM_HALFSIZE, CeHalfSize),
EV_COMMAND_ENABLE(CM_PENSIZE, CePenSize),
EV_COMMAND_ENABLE(CM_PENCOLOR, CePenColor),
EV_COMMAND(CM_NEWNOTATION, CmNewNotation),
EV_COMMAND_ENABLE(CM_NEWNOTATION, CeNewNotation),
EV_COMMAND(CM_ADDNOTATION, CmAddNotation),
EV_COMMAND_ENABLE(CM_ADDNOTATION, CeAddNotation),
EV_COMMAND(CM_MODIFYNOTATION, CmModifyNotation),
EV_COMMAND_ENABLE(CM_MODIFYNOTATION, CeModifyNotation),
EV_COMMAND(CM_NAMENOTATION, CmNameNotation),
EV_COMMAND_ENABLE(CM_NAMENOTATION, CeNameNotation),
EV_COMMAND(CM_VIEWNOTATIONS, CmViewNotations),
EV_COMMAND_ENABLE(CM_VIEWNOTATIONS, CeViewNotations),
EV_COMMAND(CM_GOTODESIGN, CmGoToDesign),
EV_COMMAND_ENABLE(CM_GOTODESIGN, CeGoToDesign),
EV_COMMAND(CM_SAVENOTATIONSET, CmSaveNotationSet),
EV_COMMAND_ENABLE(CM_SAVENOTATIONSET, CeSaveNotationSet),
EV_COMMAND(CM_LOADNOTATIONSET, CmLoadNotationSet),
EV_COMMAND_ENABLE(CM_LOADNOTATIONSET, CeLoadNotationSet),
EV_COMMAND(CM_SCALE, CmScale),
EV_COMMAND_ENABLE(CM_SCALE, CeScale),
EV_COMMAND(CM_SCALEX, CmScalex),
EV_COMMAND_ENABLE(CM_SCALEX, CeScalex),
EV_COMMAND(CM_SCALEY, CmScaley),
EV_COMMAND_ENABLE(CM_SCALEY, CeScaley),
EV_VN_COMMIT,
EV_VN_REVERT,
EV_VN_DRAWAPPEND,
EV_VN_DRAWDELETE,
EV_VN_DRAWMODIFY,
EV_WM_SETFOCUS,
END_RESPONSE_TABLE;

TDrawView::TDrawView(TDrawDocument& doc, TWindow* parent)
:
 TOleView(doc, parent), DrawDoc(&doc)
{
 Selected        = 0;
 SelectedNotation  = 0;
 Tool            = DrawPen;
 ToolBar         = 0;
 DesignMode      = 1;
 hasCutCopy      = 0;
 CurrentFillMode  = 1;
 ZoomScale       = 100;
```

```
  BasicShape = new TBasicShape(TColor::Black, 1);
  Attr.AccelTable = IDA_DRAWVIEW;
  SetViewMenu(new TMenuDescr(IDM_DRAWVIEW));

  if (DrawDoc->GetNotations()->GetItemsInContainer()>0)
                        if (!(DrawDoc->GetBasicShapes()->GetItemsInContainer()>0))
                        {
                         DesignMode = 0;
                        OldTool =  Tool;
                         Tool      = DrawSelect;                          ////
//                       CmViewNotations();
                        }

  OcApp->AddUserFormatName("DrawPad Native Data", "Owl DrawPad native data", DrawPadFormat);
}

void
TDrawView::SetupWindow()
{
  TOleView::SetupWindow();

  // Scroll bars
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;

  Scroller = new TScroller(this, 1, 1, 0, 0);
  AdjustScroller();

}

// Adjust the Scroller range
void
TDrawView::AdjustScroller()
{
  TDrawDocument *drawDoc = TYPESAFE_DOWNCAST(&GetDocument(), TDrawDocument);
  CHECK(drawDoc);
  TSize range = drawDoc->GetDocSize();
  // Use device unit for scroll range
  //
  TOleClientDC dc(*this);
  dc.LPtoDP((TPoint*)&range);
  range -= GetClientRect().Size();
  Scroller->SetRange(range.cx, range.cy);
}
```

```
void
TDrawView::EvSize(UINT SizeType, TSize& Size)
{
  TOleView::EvSize(SizeType, Size);
  if (SizeType != SIZEICONIC) {
    AdjustScroller();
  }
}

static int
BasicShapeIntersect(const TBasicShape& BasicShape, void* param)
{
  TPoint*  pt = (TPoint*)param;
  TBasicShape &modify = const_cast<TBasicShape&>(BasicShape);
  if (modify.IsSelected()) {
    modify.Where = TUIHandle(modify.GetBound(), TUIHandle::Framed).HitTest(*pt);
  }
  else {
    if (modify.GetBound().Contains(*pt))
      modify.Where = TUIHandle::MidCenter;
    else
      modify.Where = TUIHandle::Outside;
  }

  return BasicShape.Where != TUIHandle::Outside;
}


TBasicShape*
TDrawView::BasicShapeHitTest(TPoint& pt)
{
  return DrawDoc->GetBasicShapes()->LastThat(BasicShapeIntersect, &pt);
}



static int
NotationIntersect(const TNotation& Notation, void* param)
{
  TPoint*  pt = (TPoint*)param;
  TNotation &modify = const_cast<TNotation&>(Notation);
  if (modify.IsSelected()) {
    modify.Where = TUIHandle(modify.GetBound(), TUIHandle::Framed).HitTest(*pt);
```

```
  }
  else {
   if (modify.GetBound().Contains(*pt))
     modify.Where = TUIHandle::MidCenter;
   else
     modify.Where = TUIHandle::Outside;
  }

  return Notation.Where != TUIHandle::Outside;
}

TNotation*
TDrawView::NotationHitTest(TPoint& pt)
{
  return DrawDoc->GetNotations()->LastThat(NotationIntersect, &pt);
}

bool
TDrawView::ShowCursor(HWND /*wnd*/, uint hitTest, uint /*mouseMsg*/)
{
  TPoint pt;
  GetCursorPos(pt);
  ScreenToClient(pt);
  if (Tool == DrawSelect) {
    ::SetCursor(::LoadCursor(0, IDC_ARROW));
    return true;
  }
  else if (Tool == DrawPen && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
  else if (Tool == DrawLine && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
  else if (Tool == DrawArrow && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
  else if (Tool == DrawHas && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
else if (Tool == DrawIs && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
            else if (Tool == DrawDiamondend && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
            else if (Tool == DrawSquarend && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
            else if (Tool == DrawCirclend && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
   else if (Tool == DrawLinesend && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
else if (Tool == DrawDiamond && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
else if (Tool == DrawParagram && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
  }
else if (Tool == DrawPolyline && (hitTest == HTCLIENT)) {
            HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
            ::SetCursor(cur);
            return true;
```

```
}
else if (Tool == DrawPolygon && (hitTest == HTCLIENT)) {
        HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
        ::SetCursor(cur);
        return true;
}
else if (Tool == DrawCircle && (hitTest == HTCLIENT)) {
        HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
        ::SetCursor(cur);
        return true;
}
else if (Tool == DrawRect && (hitTest == HTCLIENT)) {
        HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
        ::SetCursor(cur);
        return true;
}
else if (Tool == DrawRoundedRect && (hitTest == HTCLIENT)) {
        HCURSOR cur = ::LoadCursor(*GetModule(), MAKEINTRESOURCE(IDC_PENCIL));
        ::SetCursor(cur);
        return true;
}
else if (Tool == DrawPen && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
  ::SetCursor(::LoadCursor(0, IDC_ARROW));
  return true;
}
else if (Tool == DrawLine && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawArrow && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawHas && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawIs && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
        else if (Tool == DrawDiamondend && ((hitTest == HTHSCROLL) || (hitTest ==
HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawSquarend && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
 else if (Tool == DrawCirclend && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawLinesend && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawDiamond && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}

 else if (Tool == DrawParagram && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
 else if (Tool == DrawPolyline && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}

 else if (Tool == DrawPolygon && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
        ::SetCursor(::LoadCursor(0, IDC_ARROW));
        return true;
}
else if (Tool == DrawRect && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
  ::SetCursor(::LoadCursor(0, IDC_ARROW));
  return true;
}
else if (Tool == DrawRoundedRect && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
  ::SetCursor(::LoadCursor(0, IDC_ARROW));
  return true;
}
else if (Tool == DrawCircle && ((hitTest == HTHSCROLL) || (hitTest == HTVSCROLL))) {
  ::SetCursor(::LoadCursor(0, IDC_ARROW));
```

```
  return true;
 }

 else return false;
}


// object selection
//
bool
TDrawView::Select(uint modKeys, TPoint& point)
{
 if (Tool != DrawSelect)
        return false;

 // Clicked in BasicShapes?
 if (DesignMode == 1){
        TBasicShape *BasicShape = BasicShapeHitTest(point);
        SetBasicShapeSelection(BasicShape);
        if (Selected) { // there is a selection
                TOleView::SetSelection(0);
                DragRect = BasicShape->GetBound();
                DragRect.right++;
                DragRect.bottom++;
                DragHit = BasicShape->Where;
                DragStart = DragPt = point;
                if (!DragDC)
                        DragDC = new TOleClientDC(*this);

                DragDC->DrawFocusRect(DragRect);
                SetCapture();
                return true;
               }
        else
        // Select OLE object, if any
        return TOleView::Select(modKeys, point);
 }
 else // (DesignMode == 0)
 {
        TNotation *Notation = NotationHitTest(point);
        SetNotationSelection(Notation);
        if (SelectedNotation) { // there is a selection
                TOleView::SetSelection(0);
                DragRect = Notation->GetBound();
                DragRect.right++;
                DragRect.bottom++;
                DragHit = Notation->Where;
                DragStart = DragPt = point;
                if (!DragDC)
                DragDC = new TOleClientDC(*this);
                DragDC->DrawFocusRect(DragRect);
                SetCapture();
                return true;
               }
        else
        // Select OLE object, if any
        return TOleView::Select(modKeys, point);
 }
}

void
TDrawView::EvLButtonDown(uint modKeys, TPoint& point)
{
 TOleView::EvLButtonDown(modKeys, point);
 if (SelectEmbedded() || !DragDC)
  return;

 if (Tool == DrawSelect) { // selection
  Select(modKeys, point);
 }
 else if (Tool == DrawPen) {
        SetCapture();
        BasicShape->Add(TPoint(801,0));
        Pen = new TPen(BasicShape->QueryColor(),BasicShape->QueryPenSize(),
                                                BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->MoveTo(point);
        BasicShape->Add(point);
 }
 else if (Tool == DrawLine) {  //CM_LINE 301
        BasicShape->Add(TPoint(802,0));
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
```

```
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawArrow) {  //CM_ARROW 319
        BasicShape->Add(TPoint(992,0));
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawHas) {  //CM_ARROW 335
        BasicShape->Add(TPoint(987,0));  // 987
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawIs) {  //CM_336
        BasicShape->Add(TPoint(986,0));  // 986
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawDiamondend) {  //CM_DIAMONDEND 331
        BasicShape->Add(TPoint(990,0));  // 990
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawSquarend) {  //CM_SQUAREND  332
        BasicShape->Add(TPoint(989,0)); //989
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);
}
        else if (Tool == DrawCirclend) {  //CM_CIRCLEND 333
        BasicShape->Add(TPoint(991,0));  // 991
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
```

```
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
        else if (Tool == DrawLinesend) {  //CM_LINESEND 334
        BasicShape->Add(TPoint(988,0));  //988
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
}
else if (Tool == DrawPolyline) {  //CM_POLYLINE 318
        if (BasicShape->GetItemsInContainer() > 0){
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = LastPoint;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
        }
        else {
        BasicShape->Add(TPoint(995,0));  // 995
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
        }
}
else if (Tool == DrawPolygon) {  //CM_POLYGON 317
        if (BasicShape->GetItemsInContainer() > 0){
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = LastPoint;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
        }
        else {
        BasicShape->Add(TPoint(996,0));  // 996
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SetROP2(R2_NOT); //
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint);
        DragDC->LineTo(LastPoint);
        BasicShape->Add(FirstPoint);  // my
        }
}
else if (Tool == DrawCircle) {  //CM_CIRCLE 303
        BasicShape->Add(TPoint(998,0));
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                 BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SelectStockObject(NULL_BRUSH);
```

```
        DragDC->SetROP2(R2_NOT);
        FirstPoint = point;
        LastPoint  = point;
        DragDC->Ellipse(FirstPoint, LastPoint);
        BasicShape->Add(FirstPoint);
}
        else if (Tool == DrawRect) {  //CM_RECT 304
        BasicShape->Add(TPoint(999,0));
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                        BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SelectStockObject(NULL_BRUSH);
        DragDC->SetROP2(R2_NOT);
        FirstPoint = point;
        LastPoint  = point;
        DragDC->Rectangle(FirstPoint, LastPoint);
        BasicShape->Add(FirstPoint);
}
        else if (Tool == DrawDiamond) {  //CM_DIAMOND 329
        BasicShape->Add(TPoint(993,0));  //993
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                        BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SelectStockObject(NULL_BRUSH);
        DragDC->SetROP2(R2_NOT);
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
        DragDC->LineTo(LastPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
        DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,LastPoint.y);
        DragDC->LineTo(FirstPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
        DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
}
        else if (Tool == DrawParagram) {  //CM_PARAGRAM 330  parallolgram
        BasicShape->Add(TPoint(994,0));  // 994
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SelectStockObject(NULL_BRUSH);
        DragDC->SetROP2(R2_NOT);
        FirstPoint = point;
        LastPoint  = point;
        DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
        DragDC->LineTo(LastPoint.x,FirstPoint.y);
        DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)*4/5,LastPoint.y);
        DragDC->LineTo(FirstPoint.x,LastPoint.y);
        DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
}
        else if (Tool == DrawRoundedRect) {  //CM_RECT 305
        BasicShape->Add(TPoint(997,0));
        SetCapture();
        Pen = new TPen(BasicShape->QueryColor(), BasicShape->QueryPenSize(),
                                                BasicShape->QueryPenType());
        DragDC->SelectObject(*Pen);
        DragRect.SetNull();
        DragDC->SelectStockObject(NULL_BRUSH);
        DragDC->SetROP2(R2_NOT);
        FirstPoint = point;
        LastPoint  = point;
        DragDC->RoundRect(FirstPoint, LastPoint,TPoint(20,20));
        BasicShape->Add(FirstPoint);
}
}

void
TDrawView::EvMouseMove(uint modKeys, TPoint& point)
{
 TOleView::EvMouseMove(modKeys, point);

 if (SelectEmbedded() || !DragDC)
        return;

 if (DesignMode == 0)// select a notation
        {
        if (SelectedNotation && DragHit == TUIHandle::MidCenter &&
        !InClient(*DragDC, point))
        { // selection
        DragDC->DrawFocusRect(DragRect);  // erase old rect
        // Start drag & drop
        TOcDropAction outAction;
```

```
            TOcDataProvider* ocData = new TOcDataProvider(*OcView, &DocReg, 0,
             (void*)SelectedNotation, CleanUp);
            OcApp->Drag(ocData, daDropCopy | daDropMove | daDropLink, outAction);
 if (outAction == daDropMove) {
            DrawDoc->DeleteNotation(DrawDoc->GetNotations()->Find(*SelectedNotation));
                    SetNotationSelection(0);
             }
             ocData->Release();
             DragHit = TUIHandle::Outside;
}
             return;
            }
// designmode == 1
if (Tool == DrawPen) {
            DragDC->LineTo(point);
            BasicShape->Add(point);
}
else if (Tool == DrawLine) { //CM_LINE 301
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
else if (Tool == DrawArrow) {  //CM_ARROW 319
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
           else if (Tool == DrawHas) {  //CM_HAS  335
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
           else if (Tool == DrawIs) {  //CM_IS 336
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
           else if (Tool == DrawDiamondend) {  //CM_DIAMONDEND 331
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
           else if (Tool == DrawSquarend) {  //CM_SQUEREND  332
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
           else if (Tool == DrawCirclend) {  //CM_CIRCLEND 333
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
 else if (Tool == DrawLinesend) {  //CM_LINESEND  334
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
else if (Tool == DrawPolyline) { //CM_POLYLINE 318
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
}
else if (Tool == DrawPolygon) { //CM_POLYGON317
 DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
            LastPoint  = point;
            DragDC->MoveTo(FirstPoint);
            DragDC->LineTo(LastPoint);
```

```
        else if (Tool == DrawCircle) { //CM_CIRCLE 303
         DragDC->Ellipse(FirstPoint, LastPoint);
         LastPoint  = point;
         DragDC->Ellipse(FirstPoint, LastPoint);
}
        else if (Tool == DrawRect) { //CM_RECT 304
         DragDC->Rectangle(FirstPoint, LastPoint);
         LastPoint  = point;
         DragDC->Rectangle(FirstPoint, LastPoint);
}
        else if (Tool == DrawDiamond) { //CM_DIAMOND 329
         DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
         DragDC->LineTo(LastPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,LastPoint.y);
         DragDC->LineTo(FirstPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
         LastPoint  = point;
         DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
         DragDC->LineTo(LastPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,LastPoint.y);
         DragDC->LineTo(FirstPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y);
}
        else if (Tool == DrawParagram) { //CM_PARAGRAM 330
         DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
         DragDC->LineTo(LastPoint.x,FirstPoint.y);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)*4/5,LastPoint.y);
         DragDC->LineTo(FirstPoint.x,LastPoint.y);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
         LastPoint  = point;
         DragDC->MoveTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
         DragDC->LineTo(LastPoint.x,FirstPoint.y);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)*4/5,LastPoint.y);
         DragDC->LineTo(FirstPoint.x,LastPoint.y);
         DragDC->LineTo(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y);
}
        else if (Tool == DrawRoundedRect) { //CM_RECT 305
         DragDC->RoundRect(FirstPoint, LastPoint,TPoint(20,20));
         LastPoint  = point;
         DragDC->RoundRect(FirstPoint, LastPoint,TPoint(20,20));
}
else if (Selected && DragHit == TUIHandle::MidCenter &&
}
         !InClient(*DragDC, point))
         { // selection
         DragDC->DrawFocusRect(DragRect);   // erase old rect
         // Start drag & drop
         //
         TOcDropAction outAction;
         TOcDataProvider* ocData = new TOcDataProvider(*OcView, &DocReg, 0,
                                                       (void*)Selected, CleanUp);
         OcApp->Drag(ocData, daDropCopy | daDropMove | daDropLink, outAction);
         if (outAction == daDropMove) {
        DrawDoc->DeleteBasicShape(DrawDoc->GetBasicShapes()->Find(*Selected));
                SetBasicShapeSelection(0);
         }
         ocData->Release();
         DragHit = TUIHandle::Outside;
 }
}

void
TDrawView::SetBasicShapeSelection(TBasicShape *BasicShape)
{
 if (Selected) {
  Selected->Select(false);////
         Selected->Invalidate(*this);
//       Selected->Select(false);
 }
 Selected = BasicShape;
 if (Selected) {
         Selected->Select(true);
         Selected->Invalidate(*this);
 }
}

void
TDrawView::SetBasicShapeSelection(int index)
{
 SetBasicShapeSelection(DrawDoc->GetBasicShape(index));
}

void
TDrawView::SetNotationSelection(TNotation *Notation)
{
```

```
 if (SelectedNotation) {
            SelectedNotation->Invalidate(*this);
            SelectedNotation->Select(false);
 }
 SelectedNotation = Notation;
 if (SelectedNotation) {
            SelectedNotation->Select(true);
            SelectedNotation->Invalidate(*this);
 }
}

void
TDrawView::SetNotationSelection(int index)
{
 SetNotationSelection(DrawDoc->GetNotation(index));
}

void TDrawView::EvLButtonUp(uint modKeys, TPoint& point)
{
 TPoint oldPoint = point;
 if (!DragDC)
            return;
 if (DesignMode == 0)// select a notation
 {
 if ((Tool == DrawSelect) && !SelectEmbedded()) {
            if (SelectedNotation && DragHit == TUIHandle::MidCenter) {
                        SelectedNotation->Invalidate(*this);
                        DragDC->DPtoLP(&point);
                        TPoint newPos = point - DragStart;
                        SelectedNotation->UpdatePosition(newPos);
                        SelectedNotation->Invalidate(*this);
                        InvalidatePart(invView);
                        DragHit = TUIHandle::Outside;
                        ReleaseCapture();
            }
 }
 TOleView::EvLButtonUp(modKeys, oldPoint);
 return;
 }
 // designmode == 1
 if ((Tool == DrawSelect) && !SelectEmbedded()) {
            if (Selected && DragHit == TUIHandle::MidCenter) {
                        Selected->Invalidate(*this);
                        DragDC->DPtoLP(&point);
                        TPoint newPos = point - DragStart;
                        Selected->UpdatePosition(newPos);
                        Selected->Invalidate(*this);
                        InvalidatePart(invView);
                        DragHit = TUIHandle::Outside;
                        ReleaseCapture();
            }
} else if ((Tool == DrawPen) && !SelectEmbedded()) {
            ReleaseCapture();
            if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
            }
            BasicShape->Flush();
            delete Pen;
}
            else if ((Tool == DrawLine) && !SelectEmbedded()) {
            DragDC->SetROP2(R2_COPYPEN); //
            DragDC->SelectObject(*Pen);
            BasicShape->Add(LastPoint);
            ReleaseCapture();
            if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
            }
            BasicShape->Flush();
            delete Pen;
}
            else if ((Tool == DrawArrow) && !SelectEmbedded()) {
            DragDC->SetROP2(R2_COPYPEN); //
            DragDC->SelectObject(*Pen);
            BasicShape->Add(LastPoint);
            ReleaseCapture();
            double a1 = LastPoint.x - FirstPoint.x;
            double a2 = LastPoint.y - FirstPoint.y;
            double r  = sqrt(a1*a1+a2*a2);
            if(r > 1 ){
            double px = LastPoint.x - (a1/r)*20;
            double py = LastPoint.y - (a2/r)*20;
            BasicShape->Add(TPoint(px+(a2/r)*6,py+(-a1/r)*6));
            BasicShape->Add(TPoint(LastPoint.x - (a1/r)*16,LastPoint.y - (a2/r)*16));
            BasicShape->Add(TPoint(px-(a2/r)*6,py-(-a1/r)*6));
```

```
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
                }
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawHas) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN);
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        double a1 = LastPoint.x - FirstPoint.x;
        double a2 = LastPoint.y - FirstPoint.y;
        double r  = sqrt(a1*a1+a2*a2);
        if(r > 1 ){
        double px = FirstPoint.x + a1/2;
        double py = FirstPoint.y + a2/2;
        BasicShape->Add(TPoint(px+(a2/r)*15,py+(-a1/r)*15));
        BasicShape->Add(TPoint(px-(a2/r)*15,py-(-a1/r)*15));
        px = px + (a1/r)*30;
        py = py + (a2/r)*30;
        BasicShape->Add(TPoint(px,py));
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
         }
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawIs) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN); //
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        double a1 = LastPoint.x - FirstPoint.x;
        double a2 = LastPoint.y - FirstPoint.y;
        double r  = sqrt(a1*a1+a2*a2);
        if(r > 1 ){
        double px = FirstPoint.x + a1/2;
        double py = FirstPoint.y + a2/2;
        BasicShape->Add(TPoint(px+(a2/r)*15,py+(-a1/r)*15));
        BasicShape->Add(TPoint(px-(a2/r)*15,py-(-a1/r)*15));
        px = px + (a1/r)*30;
        py = py + (a2/r)*30;
        BasicShape->Add(TPoint(px,py));
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
         }
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawDiamondend) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN); //
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        double a1 = LastPoint.x - FirstPoint.x;
        double a2 = LastPoint.y - FirstPoint.y;
        double r  = sqrt(a1*a1+a2*a2);
        if(r > 1 ){
        double px = LastPoint.x - (a1/r)*12;
        double py = LastPoint.y - (a2/r)*12;
        BasicShape->Add(TPoint(px+(a2/r)*6,py+(-a1/r)*6));
        BasicShape->Add(TPoint(LastPoint.x - (a1/r)*24,LastPoint.y - (a2/r)*24));
        BasicShape->Add(TPoint(px-(a2/r)*6,py-(-a1/r)*6));
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
        }
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawSquarend) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN); //
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        BasicShape->Add(TPoint(LastPoint.x-6,LastPoint.y-6));
        BasicShape->Add(TPoint(LastPoint.x+6,LastPoint.y+6));
        if (BasicShape->GetItemsInContainer() > 1) {
```

```
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawCirclend) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN); //
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        BasicShape->Add(TPoint(LastPoint.x-6,LastPoint.y-6));
        BasicShape->Add(TPoint(LastPoint.x+6,LastPoint.y+6));
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawLinesend) && !SelectEmbedded()) {
        DragDC->SetROP2(R2_COPYPEN); //
        DragDC->SelectObject(*Pen);
        BasicShape->Add(LastPoint);
        ReleaseCapture();
        double a1 = LastPoint.x - FirstPoint.x;
        double a2 = LastPoint.y - FirstPoint.y;
        double r  = sqrt(a1*a1+a2*a2);
        if(r > 1 ){
        double px = LastPoint.x - (a1/r)*20;
        double py = LastPoint.y - (a2/r)*20;
        BasicShape->Add(TPoint(LastPoint.x+(a2/r)*6,LastPoint.y+(-a1/r)*6));
        BasicShape->Add(TPoint(px ,py));
        BasicShape->Add(TPoint(LastPoint.x-(a2/r)*6,LastPoint.y-(-a1/r)*6));
        if (BasicShape->GetItemsInContainer() > 1) {
                BasicShape->UpdateBound();
                DrawDoc->AddBasicShape(*BasicShape);
         }
        }
        BasicShape->Flush();
        delete Pen;
}
        else if ((Tool == DrawPolyline) && !SelectEmbedded()) {
                DragDC->SetROP2(R2_COPYPEN); //
                DragDC->SelectObject(*Pen);
                ReleaseCapture();
                delete Pen;
        }
        else if ((Tool == DrawPolygon) && !SelectEmbedded()) {
                DragDC->SetROP2(R2_COPYPEN); //
                DragDC->SelectObject(*Pen);
                ReleaseCapture();
                delete Pen;
        }
else if ((Tool == DrawRect) && !SelectEmbedded()) {
                DragDC->SetROP2(R2_COPYPEN);
                DragDC->SelectObject(*Pen);
                BasicShape->Add(LastPoint);
                ReleaseCapture();
                if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
                }
                BasicShape->Flush();
                delete Pen;
        }
        else if ((Tool == DrawDiamond) && !SelectEmbedded()) {
                DragDC->SetROP2(R2_COPYPEN);
                DragDC->SelectObject(*Pen);
                ReleaseCapture();
                BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y));
                BasicShape->Add(TPoint(LastPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2));
                BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,LastPoint.y));
                BasicShape->Add(TPoint(FirstPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2));
                if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
                }
                BasicShape->Flush();
                delete Pen;
        }
        else if ((Tool == DrawParagram) && !SelectEmbedded()) {
                DragDC->SetROP2(R2_COPYPEN);
                DragDC->SelectObject(*Pen);
                ReleaseCapture();
                BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y));
```

```
            BasicShape->Add(TPoint(LastPoint.x,FirstPoint.y));
            BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)*4/5,LastPoint.y));
            BasicShape->Add(TPoint(FirstPoint.x,LastPoint.y));
            if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
            }
            BasicShape->Flush();
            delete Pen;
}
            else if ((Tool == DrawRoundedRect) && !SelectEmbedded()) {
            DragDC->SetROP2(R2_COPYPEN);
            DragDC->SelectObject(*Pen);
            BasicShape->Add(LastPoint);
            ReleaseCapture();
            if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
            }
            BasicShape->Flush();
            delete Pen;
}
 else if ((Tool == DrawCircle) && !SelectEmbedded()) {
            DragDC->SetROP2(R2_COPYPEN);
            DragDC->SelectObject(*Pen);
            BasicShape->Add(LastPoint);

            ReleaseCapture();
            if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
            }
            BasicShape->Flush();
            delete Pen;
 }
 TOleView::EvLButtonUp(modKeys, oldPoint);
}

void TDrawView::EvRButtonDown(uint modKeys, TPoint& point)
{
 TPoint oldPoint = point;

            if ((Tool == DrawPolyline) && !SelectEmbedded()) {
                        BasicShape->Add(LastPoint);
                        if (BasicShape->GetItemsInContainer() >= 3) {
                                    BasicShape->UpdateBound();
                                    DrawDoc->AddBasicShape(*BasicShape);
                                    }
                        BasicShape->Flush();
 }
            else if ((Tool == DrawPolygon) && !SelectEmbedded()) {
                        BasicShape->Add(LastPoint);
                        if (BasicShape->GetItemsInContainer() >= 3) {
                                    BasicShape->UpdateBound();
                                    DrawDoc->AddBasicShape(*BasicShape);
                                    }
                        BasicShape->Flush();
 }
}

void
TDrawView::CePen(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawPen);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeCircle(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawCircle);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeRect(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawRect);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeRoundedRect(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawRoundedRect);
 ce.Enable(DesignMode==1);
```

```
}
void
TDrawView::CePolygon(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawPolygon);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeArrow(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawArrow);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeDiamond(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawDiamond);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeParagram(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawParagram);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CePolyline(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawPolyline);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeSelect(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawSelect);
}
```

```
void
TDrawView::CePenSize(TCommandEnabler& ce)
{
ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                    ||(DesignMode ==1 && Tool == DrawSelect && Selected != 0)); //
}

void
TDrawView::CePenColor(TCommandEnabler& ce)
{
ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                    ||(DesignMode ==1 && Tool == DrawSelect && Selected != 0)); //
}

void
TDrawView::CeLine(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawLine);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CmPen()
{
 Tool = DrawPen;
 BasicShape->Flush();
 BasicShape->SetFillType(0); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmCircle()
{
 Tool = DrawCircle;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
```

```
 Selected=0;
 }
}

void
TDrawView::CmRect()
{
 Tool = DrawRect;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmPolygon()
{
 Tool = DrawPolygon;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmArrow()
{

 Tool = DrawArrow;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}
```

```
void
TDrawView::CmDiamond()
{
 Tool = DrawDiamond;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmParagram()
{
 Tool = DrawParagram;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmPolyline()
{
 Tool = DrawPolyline;
 BasicShape->Flush();
 BasicShape->SetFillType(0); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}


void
TDrawView::CmRoundedRect()
{
```

```
 Tool = DrawRoundedRect;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmSelect()
{
 Tool = DrawSelect;
}

void
TDrawView::CmLine()
{
 Tool = DrawLine;
 BasicShape->Flush();
 BasicShape->SetFillType(0); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CmPenSize()
{
 if (Selected) {
  GetPenSize(this, *Selected);
            Selected->UpdateBound();
DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
 }
 else
            GetPenSize(this, *BasicShape);
}

void
TDrawView::CmPenColor()
{
 if (Selected) {
            GetPenColor(this, *Selected);
            DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
 }
 else
            GetPenColor(this, *BasicShape);
}

void
TDrawView::CmOrgSize()
{
 ZoomScale = 100;
 SetScale(ZoomScale);
// SetScale(100);
}

void
TDrawView::CmDoubleSize()
{
 ZoomScale += 20;
 SetScale(ZoomScale);
// SetScale(200);
}

void
TDrawView::CmHalfSize()
{
 ZoomScale -= 20;
 SetScale(ZoomScale);
// SetScale(50);
}

void
TDrawView::CeOrgSize(TCommandEnabler& ce)
{
 ce.SetCheck(!Scale.IsZoomed());
 ce.Enable( ZoomScale != 100);
}

void
TDrawView::CeDoubleSize(TCommandEnabler& ce)
{
```

```
        ce.SetCheck(ZoomScale > 100);
        ce.Enable( ZoomScale < 200);
}

void
TDrawView::CeHalfSize(TCommandEnabler& ce)
{
//  ce.SetCheck(Scale.GetScale() == 50);
        ce.SetCheck(ZoomScale < 100);
        ce.Enable( ZoomScale > 20);
}

void
TDrawView::CmClear()
{
 if(DesignMode == 1)
        DrawDoc->ClearBasicShapes();
 else
        DrawDoc->ClearNotations();
}

void
TDrawView::CmUndo()
{
 DrawDoc->Undo();
}

void
TDrawView::CmAddNotation()
{
 DrawDoc->AddNotation();
}

void
TDrawView::CmNewNotation()
{
 DesignMode = 1;
 Tool     = OldTool;
 DrawDoc->NewNotation();
}

void
TDrawView::CmModifyNotation()
{
 DesignMode = 1;
 DrawDoc->ModifyNotation(*SelectedNotation,DrawDoc->GetNotations()->Find(*SelectedNotation));
//  DrawDoc->AddNotation();// delete after implete ModifyNotation
}

void
TDrawView::CmNameNotation()
{
 ;
}

void
TDrawView::CmViewNotations()
{
 DesignMode = 0;
 OldTool =  Tool;
 Tool    =  DrawSelect;
 Invalidate();  // force full repaint
}

void
TDrawView::CmGoToDesign()
{
 DesignMode = 1;
 Tool = OldTool;
 DrawDoc->GoToDesign();
}
void
TDrawView::CeAddNotation(TCommandEnabler& ce)
{
ce.Enable(DesignMode==1&&(DrawDoc->GetBasicShapes()->GetItemsInContainer()>0));
}


void
TDrawView::CeNewNotation(TCommandEnabler& ce)
{
ce.Enable(DesignMode ==0||(DesignMode==1&&
                        (DrawDoc->GetBasicShapes()->GetItemsInContainer()>0)));
}

void
```

```
TDrawView::CeModifyNotation(TCommandEnabler& ce)
{
 ce.Enable(DesignMode == 0 && SelectedNotation != 0);  // and select notation>0
}

void
TDrawView::CeNameNotation(TCommandEnabler& ce)
{
 ce.Enable(DesignMode == 0 && SelectedNotation != 0);  // and select notation>0
}

void
TDrawView::CeViewNotations(TCommandEnabler& ce)
{
 ce.Enable(DesignMode==1);
}

void
TDrawView::CeGoToDesign(TCommandEnabler& ce)
{
 ce.Enable(DesignMode ==0&&(DrawDoc->GetBasicShapes()->GetItemsInContainer()>0));
}

void
TDrawView::CeEditCut(TCommandEnabler& ce)
{
 ce.Enable(DesignMode ==0 && SelectedNotation!=0||
                        DesignMode ==1 && Selected != 0 || SelectEmbedded());
}

void
TDrawView::CeEditCopy(TCommandEnabler& ce)
{
 ce.Enable(DesignMode ==0 && SelectedNotation!=0||
                        DesignMode ==1 && Selected != 0 || SelectEmbedded());
}

void
TDrawView::CeEditPaste(TCommandEnabler& ce)
{
        ce.Enable(DesignMode ==0 ||DesignMode ==1  );
}
```

```
void
TDrawView::CleanUp(void* /*userData*/)
{
 // No data clean up to do here
}

void
TDrawView::CmEditCut()
{
 if (Selected && DesignMode==1) {
        DrawDoc->DeleteBasicShape(DrawDoc->GetBasicShapes()->Find(*Selected));
        Selected =0;
        hasCutCopy=1;
 }
 else if (SelectedNotation && DesignMode==0) {
        // Delete the Notation
        DrawDoc->DeleteNotation(DrawDoc->GetNotations()->Find(*SelectedNotation));
        SelectedNotation =0;
        hasCutCopy=2;
 }
 else
 ;
}

void
TDrawView::CmEditPaste()
{
 if (DesignMode==1) {
        // Paste the BasicShape
        DrawDoc->PasteBasicShape();
 }
 else
        // Paste the Notation
        DrawDoc->PasteNotation();
}

void
TDrawView::CmEditCopy()
{
 if (Selected && DesignMode==1) {
// Copy the selected BasicShape
        DrawDoc->CopyBasicShape(DrawDoc->GetBasicShapes()->Find(*Selected));
        hasCutCopy=1;
```

```
}
else if (SelectedNotation && DesignMode==0) {
        // Copy the selected Notation
        DrawDoc->CopyNotation(DrawDoc->GetNotations()->Find(*SelectedNotation));
        hasCutCopy=2;
}
else
;
//   TOleView::CmEditCopy();
}

void
TDrawView::Paint(TDC& dc, bool /*erase*/, TRect& /*rect*/)
{
 bool metafile = (dc.GetDeviceCaps(TECHNOLOGY) == DT_METAFILE);

 if (DesignMode == 0) //  view notations mode
        {
                if (DrawDoc->GetNotations()->GetItemsInContainer()>0)
                        {
                        // Iterates through the array of TNotation objects.
                        int j = 0;
                        int px=10,py=20,hight=10; //////
                        TNotation* Notation;
        while ((Notation = const_cast<TNotation *>(DrawDoc->GetNotation(j++)))!= 0)
                        {
                        TPoint newPos = TPoint(px,py) - Notation->GetPos(); /////
                        Notation->UpdatePosition(newPos); ////
                        Notation->Draw(dc);
                        TSize size = Notation->GetSize();/////
                        px += size.cx+20;
                        if( size.cy > hight ) hight =size.cy;
                        if (j%4 ==0 ) //////
                                {
                                px=10;
                                py+= hight + 20;
                                hight=10;//////
                                }

                        if (Notation->IsSelected())
                                Notation->DrawSelection(dc);
                        }
                }
        }
        else  if (DesignMode == 1) // design mode
         {
         // Iterates through the array of BasicShape objects.
         int j = 0;
         TBasicShape* BasicShape;
 while ((BasicShape = const_cast<TBasicShape *>(DrawDoc->GetBasicShape(j++))) != 0)
                        {
                                BasicShape->Draw(dc);
                                if (BasicShape->IsSelected() && !metafile)
                                {
                                BasicShape->DrawSelection(dc);
                                Selected = BasicShape;
        }
                        }
                }
}

bool
TDrawView::VnCommit(bool /*force*/)
{
 // nothing to do here, no data held in view
 return true;
}

bool
TDrawView::VnRevert(bool /*clear*/)
{
 Selected =0;   ///my clear all
 Invalidate();  // force full repaint
 InvalidatePart(invView);
 return true;
}

bool
TDrawView::VnAppend(uint index)
{
 if(DesignMode==1){
                                TBasicShape* BasicShape = DrawDoc->GetBasicShape(index);
                                BasicShape->UpdateBound();
                                BasicShape->Invalidate(*this);
                                InvalidatePart(invView);
                                }
```

```
 else  {
                                Invalidate();  // force full repaint
                                InvalidatePart(invView);
                                }
 return true;
}

bool
TDrawView::VnModify(uint /*index*/)
{
 Invalidate();  // force full repaint
 InvalidatePart(invView);
 return true;
}

bool
TDrawView::VnDelete(uint /*index*/)
{
 Invalidate();  // force full repaint
 InvalidatePart(invView);
 return true;

}

void
TDrawView::CmPsSolid()
{
 if (Selected) {
        Selected->SetPenType(PS_SOLID);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//   BasicShape->SetPenType(PS_SOLID);
  }
  else
        BasicShape->SetPenType(PS_SOLID);
}

void
TDrawView::CmPsDash()
{
 if (Selected) {
        Selected->SetPenType(PS_DASH);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//      BasicShape->SetPenType(PS_DASH);
 }
 else
        BasicShape->SetPenType(PS_DASH);
}

void
TDrawView::CmPsDashDot()
{
 if (Selected) {
        Selected->SetPenType(PS_DASHDOT);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//      BasicShape->SetPenType(PS_DASHDOT);
 }
 else
        BasicShape->SetPenType(PS_DASHDOT);
}


void
TDrawView::CmPsDashDotDot()
{
 if (Selected) {
        Selected->SetPenType(PS_DASHDOTDOT);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//      BasicShape->SetPenType(PS_DASHDOTDOT);
 }
 else
        BasicShape->SetPenType(PS_DASHDOTDOT);
}

void
TDrawView::CmPsDot()
{
 if (Selected) {
        Selected->SetPenType(PS_DOT);
        DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//      BasicShape->SetPenType(PS_DOT);
 }
 else
        BasicShape->SetPenType(PS_DOT);
}

void  TDrawView::CePsSolid(TCommandEnabler& ce)
```

```
{
  if ( Selected  != 0)                    //
                    ce.SetCheck(Selected->QueryPenType() == PS_SOLID); //
          else                              //
          ce.SetCheck(BasicShape->QueryPenType() == PS_SOLID);
//        ce.Enable(DesignMode ==1 && Tool != DrawSelect); //
  ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                                    ||(DesignMode ==1 && Tool == DrawSelect && Selected !=
0)); //
//        ce.Enable(DesignMode ==1);
}

void
TDrawView::CePsDash(TCommandEnabler& ce)
{
          if ( Selected  != 0)                //
                    ce.SetCheck(Selected->QueryPenType() == PS_DASH); //
          else                              //
          ce.SetCheck(BasicShape->QueryPenType() == PS_DASH);
//        ce.Enable(DesignMode ==1 && Tool != DrawSelect); //
  ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                                    ||(DesignMode ==1 && Tool == DrawSelect && Selected !=
0)); //

}

void
TDrawView::CePsDashDot(TCommandEnabler& ce)
{  if ( Selected  != 0)                    //
                    ce.SetCheck(Selected->QueryPenType() == PS_DASHDOT); //
          else                              //
          ce.SetCheck(BasicShape->QueryPenType() == PS_DASHDOT);
          ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                    ||(DesignMode ==1 && Tool == DrawSelect && Selected != 0)); //
}

void
TDrawView::CePsDashDotDot(TCommandEnabler& ce)
{  if ( Selected  != 0)                    //
          ce.SetCheck(Selected->QueryPenType() == PS_DASHDOTDOT); //
          else                              //
          ce.SetCheck(BasicShape->QueryPenType() == PS_DASHDOTDOT);
  ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                    ||(DesignMode ==1 && Tool == DrawSelect && Selected != 0)); //
}

void
TDrawView::CePsDot(TCommandEnabler& ce)
{  if ( Selected  != 0)                    //
                    ce.SetCheck(Selected->QueryPenType() == PS_DOT); //
          else                              //
                    ce.SetCheck(BasicShape->QueryPenType() == PS_DOT);
          ce.Enable((DesignMode ==1 && Tool != DrawSelect)
                    ||(DesignMode ==1 && Tool == DrawSelect && Selected != 0)); //
}

void
TDrawView::CmOverlap()
{
 if (Selected) {
          Selected->SetFillType(1);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
//        BasicShape->SetFillType(1);
 }
 else {
          BasicShape->SetFillType(1);
          CurrentFillMode = 1;
          }
}

void
TDrawView::CmFill()
{
 if (Selected) {
          Selected->SetFillType(2);
          Selected->SetFillColor(TColor(0,0,255));
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
   }
 else {
          BasicShape->SetFillType(2);
          BasicShape->SetFillColor(TColor(0,0,255));
          CurrentFillMode = 2;
          }
}
void
TDrawView::CmTransparent()
```

```
{
 if (Selected) {
            Selected->SetFillType(3);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
 }
 else {
            BasicShape->SetFillType(3);
            CurrentFillMode = 3;
            }
}

void
TDrawView::CmFillColor()
{
 if (Selected) {
            GetFillColor(this, *Selected);
 DrawDoc->ModifyBasicShape(*Selected, DrawDoc->GetBasicShapes()->Find(*Selected));
 }
 else
            GetFillColor(this, *BasicShape);
}

void
TDrawView::CeOverlap(TCommandEnabler& ce)
{
  if ( Selected  != 0)                  //
                  ce.SetCheck(Selected->QueryFillType() == 1); //
            else
                  ce.SetCheck(BasicShape->QueryFillType() == 1);
            ce.Enable((DesignMode == 1      &&
                                                Selected  == 0      &&
                                                Tool != DrawSelect   && //
                                                Tool != DrawPolyline &&
                                                Tool != DrawLine     &&
        Tool != DrawLinesend &&
                                                Tool != DrawPen)     ||
                                          (DesignMode == 1      &&
                                                Selected  != 0      &&
                                                Selected->QueryFillType() != 0) );
}

void
TDrawView::CeFill(TCommandEnabler& ce)
{
            if ( Selected  != 0)                  //
                        ce.SetCheck(Selected->QueryFillType() == 2); //
            else
                        ce.SetCheck(BasicShape->QueryFillType() == 2);
            ce.Enable((DesignMode == 1      &&
                                                Selected  == 0      &&
                                                Tool != DrawSelect   && //
                                                Tool != DrawPolyline &&
                                                Tool != DrawLine     &&
                                                Tool != DrawLinesend &&
                                                Tool != DrawPen)     ||
                                          (DesignMode == 1      &&
                                                Selected  != 0      &&
                                                Selected->QueryFillType() != 0) );
}
void
TDrawView::CeTransparent(TCommandEnabler& ce)
{
            if ( Selected  != 0)                  //
                        ce.SetCheck(Selected->QueryFillType() == 3); //
            else                                  //
                        ce.SetCheck(BasicShape->QueryFillType() == 3);
            ce.Enable((DesignMode == 1      &&
                                                Selected  == 0      &&
                                                Tool != DrawSelect   && //
                                                Tool != DrawPolyline &&
                                                Tool != DrawLine     &&
                                                Tool != DrawLinesend &&
                                                Tool != DrawPen)     ||
                                          (DesignMode == 1      &&
                                                Selected  != 0      &&
                                                Selected->QueryFillType() != 0) );
}

void
TDrawView::CeFillColor(TCommandEnabler& ce)
{
ce.Enable((DesignMode == 1      &&
                                    Selected  == 0      &&
                                    Tool != DrawSelect   && //
                                    Tool != DrawPolyline &&
                                    Tool != DrawLine     &&
```

```
                                Tool != DrawLinesend &&
                                Tool != DrawPen      &&
                                CurrentFillMode == 2) ||
                               (DesignMode == 1      &&
                                Selected  != 0       &&
                                Selected->QueryFillType() == 2) );
}

void
TDrawView::CmDiamondend()
{
 Tool = DrawDiamondend;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CeDiamondend(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawDiamondend);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CmSquarend()
{
 Tool = DrawSquarend;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CeSquarend(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawSquarend);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CmCirclend()
{
 Tool = DrawCirclend;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CeCirclend(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawCirclend);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CmLinesend()
{
 Tool = DrawLinesend;
 BasicShape->Flush();
 BasicShape->SetFillType(0); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
 }
}

void
TDrawView::CeLinesend(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawLinesend);
 ce.Enable(DesignMode==1);
```

```
}

void
TDrawView::CmHas()
{
 Tool = DrawHas;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
  }
}

void
TDrawView::CeHas(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawHas);
 ce.Enable(DesignMode==1);
}

void
TDrawView::CmIs()
{
 Tool = DrawIs;
 BasicShape->Flush();
 BasicShape->SetFillType(CurrentFillMode); /////
 if (Selected) {
 Selected->Select(false);
 Selected->Invalidate(*this);
 Selected=0;
  }
}


void
TDrawView::CeIs(TCommandEnabler& ce)
{
 ce.SetCheck(Tool == DrawIs);
 ce.Enable(DesignMode==1);
}
```

```
void
TDrawView::CeDataInput(TCommandEnabler& ce)
{
ce.Enable(DesignMode == 1     &&
                                Tool != DrawSelect);
}

void
TDrawView::CmDataInput()
{
 char inputText[6];
 TPoint FirstPoint,LastPoint;
 BasicShape->Flush();
 wsprintf(inputText, "%d", 0);
 if (Tool == DrawPen )
 BasicShape->Add(TPoint(801,0));
 else if (Tool == DrawLine)
 BasicShape->Add(TPoint(802,0));
 else if (Tool == DrawIs)
 BasicShape->Add(TPoint(986,0));
 else if (Tool == DrawHas)
 BasicShape->Add(TPoint(987,0));
 else if (Tool == DrawLinesend)
 BasicShape->Add(TPoint(988,0));
 else if (Tool == DrawSquarend)
 BasicShape->Add(TPoint(989,0));
 else if (Tool == DrawDiamondend)
 BasicShape->Add(TPoint(990,0));
 else if (Tool == DrawCirclend)
 BasicShape->Add(TPoint(991,0));
 else if (Tool == DrawArrow)
 BasicShape->Add(TPoint(992,0));
 else if (Tool == DrawDiamond)
 BasicShape->Add(TPoint(993,0));
 else if (Tool == DrawParagram)
 BasicShape->Add(TPoint(994,0));
 else if (Tool == DrawPolyline)
 BasicShape->Add(TPoint(995,0));
 else if (Tool == DrawPolygon)
 BasicShape->Add(TPoint(996,0));
 else if (Tool == DrawRoundedRect)
 BasicShape->Add(TPoint(997,0));
 else if (Tool == DrawCircle )
```

```
BasicShape->Add(TPoint(998,0));
else if (Tool == DrawRect)
BasicShape->Add(TPoint(999,0));
if (Tool == DrawPen || Tool == DrawPolyline || Tool == DrawPolygon )
{
FirstPoint.x = atoi(inputText);
FirstPoint.y = atoi(inputText);
while (FirstPoint.x >= 0 && FirstPoint.y >= 0) {
  if (TInputDialog(this, "Input Data",
                                        "Start point X:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    FirstPoint.x = atoi(inputText);
                    if (FirstPoint.x >= 0){
  if (TInputDialog(this, "Input Data",
                                        "Start point Y:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    FirstPoint.y = atoi(inputText);
                    if (FirstPoint.y >= 0)
                    BasicShape->Add(FirstPoint);
                    else {
    if (BasicShape->GetItemsInContainer() > 1) {
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                              BasicShape->Flush();
                    }
                    }
                    else {
  if (BasicShape->GetItemsInContainer() > 1) {
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                              BasicShape->Flush();
                    }
          }
}
else
{
if (TInputDialog(this, "Input Data",
                                        "Start point X:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    FirstPoint.x = atoi(inputText);
if (TInputDialog(this, "Input Data",
                                        "Start point Y:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    FirstPoint.y = atoi(inputText);
                    BasicShape->Add(FirstPoint);
if (TInputDialog(this, "Input Data",
                                        "End point X:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    LastPoint.x = atoi(inputText);
if (TInputDialog(this, "Input Data",
                                        "End point Y:",
                                        inputText,
                                        sizeof(inputText)).Execute() != IDOK)
                                        {
                                        BasicShape->Flush();
                                        return;
                                        }
                    LastPoint.y = atoi(inputText);
                    BasicShape->Add(LastPoint);
```

```
                if (Tool == DrawLine || Tool ==
                DrawRoundedRect || Tool == DrawCircle || Tool == DrawRect)
                        {
if (BasicShape->GetItemsInContainer() > 1) {
                                BasicShape->UpdateBound();
                                DrawDoc->AddBasicShape(*BasicShape);
                                }
                        BasicShape->Flush();
                        }
                else if (Tool == DrawIs){
                        double a1 = LastPoint.x - FirstPoint.x;
                        double a2 = LastPoint.y - FirstPoint.y;
                        double r  = sqrt(a1*a1+a2*a2);
                        if(r > 1 ){
                        double px = FirstPoint.x + a1/2;
                        double py = FirstPoint.y + a2/2;
                        BasicShape->Add(TPoint(px+(a2/r)*15,py+(-a1/r)*15));
                        BasicShape->Add(TPoint(px-(a2/r)*15,py-(-a1/r)*15));
                        px = px + (a1/r)*30;
                        py = py + (a2/r)*30;
                        BasicShape->Add(TPoint(px,py));
                        if (BasicShape->GetItemsInContainer() > 1) {
                                BasicShape->UpdateBound();
                                DrawDoc->AddBasicShape(*BasicShape);
                                }
                        }
                        BasicShape->Flush();
                }
                else if (Tool == DrawHas){
double a1 = LastPoint.x - FirstPoint.x;
                        double a2 = LastPoint.y - FirstPoint.y;
                        double r  = sqrt(a1*a1+a2*a2);
                        if(r > 1 ){
                        double px = FirstPoint.x + a1/2;
                        double py = FirstPoint.y + a2/2;
                        BasicShape->Add(TPoint(px+(a2/r)*15,py+(-a1/r)*15));
                        BasicShape->Add(TPoint(px-(a2/r)*15,py-(-a1/r)*15));
                        px = px + (a1/r)*30;
                        py = py + (a2/r)*30;
                        BasicShape->Add(TPoint(px,py));
if (BasicShape->GetItemsInContainer() > 1) {
                                BasicShape->UpdateBound();
                                DrawDoc->AddBasicShape(*BasicShape);
                                }
                        }
                        BasicShape->Flush();
                }
        else if (Tool == DrawLinesend){
                        double a1 = LastPoint.x - FirstPoint.x;
                        double a2 = LastPoint.y - FirstPoint.y;
                        double r  = sqrt(a1*a1+a2*a2);
                        if(r > 1 ){
                        double px = LastPoint.x - (a1/r)*20;
                        double py = LastPoint.y - (a2/r)*20;
                        BasicShape->Add(TPoint(LastPoint.x+(a2/r)*6,LastPoint.y+(-a1/r)*6));
                        BasicShape->Add(TPoint(px ,py));
                        BasicShape->Add(TPoint(LastPoint.x-(a2/r)*6,LastPoint.y-(-a1/r)*6));
if (BasicShape->GetItemsInContainer() > 1) {
                                BasicShape->UpdateBound();
                                DrawDoc->AddBasicShape(*BasicShape);
                                }
                        }
                        BasicShape->Flush();
                }

                        else if (Tool == DrawSquarend){
                                BasicShape->Add(TPoint(LastPoint.x-6,LastPoint.y-6));
                                BasicShape->Add(TPoint(LastPoint.x+6,LastPoint.y+6));
if (BasicShape->GetItemsInContainer() > 1) {
                                BasicShape->UpdateBound();
                                DrawDoc->AddBasicShape(*BasicShape);
                                }
                                BasicShape->Flush();
                }

        else if (Tool == DrawDiamondend){
                        double a1 = LastPoint.x - FirstPoint.x;
                        double a2 = LastPoint.y - FirstPoint.y;
                        double r  = sqrt(a1*a1+a2*a2);
                        if(r > 1 ){
                        double px = LastPoint.x - (a1/r)*12;
                        double py = LastPoint.y - (a2/r)*12;
                        BasicShape->Add(TPoint(px+(a2/r)*6,py+(-a1/r)*6));
                        BasicShape->Add(TPoint(LastPoint.x - (a1/r)*24,LastPoint.y - (a2/r)*24));
                        BasicShape->Add(TPoint(px-(a2/r)*6,py-(-a1/r)*6));
if (BasicShape->GetItemsInContainer() > 1) {
```

```
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                                }
                                BasicShape->Flush();
                        }
                        else if (Tool == DrawCirclend){
                                BasicShape->Add(TPoint(LastPoint.x-6,LastPoint.y-6));
                                BasicShape->Add(TPoint(LastPoint.x+6,LastPoint.y+6));
if (BasicShape->GetItemsInContainer() > 1) {
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                                        BasicShape->Flush();
                        }
        else if (Tool == DrawArrow){
                        double a1 = LastPoint.x - FirstPoint.x;
                        double a2 = LastPoint.y - FirstPoint.y;
                        double r  = sqrt(a1*a1+a2*a2);
                        if(r > 1 ){
                        double px = LastPoint.x - (a1/r)*20;
                        double py = LastPoint.y - (a2/r)*20;
                        BasicShape->Add(TPoint(px+(a2/r)*6,py+(-a1/r)*6));
                        BasicShape->Add(TPoint(LastPoint.x - (a1/r)*16,LastPoint.y - (a2/r)*16));
                        BasicShape->Add(TPoint(px-(a2/r)*6,py-(-a1/r)*6));
if (BasicShape->GetItemsInContainer() > 1) {
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                                }
                                BasicShape->Flush();
                        }
                        else if (Tool == DrawDiamond){
                                BasicShape->Flush();
BasicShape->Add(TPoint(993,0));
        BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,FirstPoint.y));
        BasicShape->Add(TPoint(LastPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2));
        BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/2,LastPoint.y));
        BasicShape->Add(TPoint(FirstPoint.x,FirstPoint.y+(LastPoint.y-FirstPoint.y)/2));
if (BasicShape->GetItemsInContainer() > 1) {
                                        BasicShape->UpdateBound();
                                        DrawDoc->AddBasicShape(*BasicShape);
                                        }
                                        BasicShape->Flush();
                                }
 else if (Tool == DrawParagram){
                        BasicShape->Flush();
                        BasicShape->Add(TPoint(994,0));
                        BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)/5,FirstPoint.y));
                        BasicShape->Add(TPoint(LastPoint.x,FirstPoint.y));
                        BasicShape->Add(TPoint(FirstPoint.x+(LastPoint.x-FirstPoint.x)*4/5,LastPoint.y));
                        BasicShape->Add(TPoint(FirstPoint.x,LastPoint.y));
                        if (BasicShape->GetItemsInContainer() > 1) {
                        BasicShape->UpdateBound();
                        DrawDoc->AddBasicShape(*BasicShape);
                        }
                        BasicShape->Flush();
                        }
 }
}

void
TDrawView::CmSaveNotationSet()
{
char inputText[20];
 wsprintf(inputText, "%s", "SetName");
 if (TInputDialog(this, "Save Current OO Notation Set",
                                                "Notation Set Name:",
                                                inputText,
                                                sizeof(inputText)).Execute() != IDOK)
                                                return;
 else DrawDoc->SaveNotationSet(inputText);
}

void
TDrawView::CmLoadNotationSet()
{
 char inputText[20];
 wsprintf(inputText, "%s", "SetName");
 if (TInputDialog(this, "Load OO Notation Set",
                                                "Notation Set Name:",
                                                inputText,
                                                sizeof(inputText)).Execute() != IDOK)
                                                return;
 else DrawDoc->LoadNotationSet(inputText);
}
```

```
void
TDrawView::CeLoadNotationSet(TCommandEnabler& ce)
{
 ce.Enable(DesignMode==1 || DesignMode==0);
}

void
TDrawView::CeSaveNotationSet(TCommandEnabler& ce)
{
 ce.Enable(DrawDoc->GetNotations()->GetItemsInContainer()>0);
}

void
TDrawView::CeScale(TCommandEnabler& ce)
{
ce.Enable(DesignMode ==0 && SelectedNotation!=0||
                                DesignMode ==1 && Selected != 0 );
}

void
TDrawView::CeScalex(TCommandEnabler& ce)
{
ce.Enable(DesignMode ==0 && SelectedNotation!=0||
                                DesignMode ==1 && Selected != 0 );
}

void
TDrawView::CeScaley(TCommandEnabler& ce)
{
ce.Enable(DesignMode ==0 && SelectedNotation!=0||
                                DesignMode ==1 && Selected != 0 );
}

void
TDrawView::CmScale()
{
 char inputText[20];
 wsprintf(inputText, "%s", "1");
 if (TInputDialog(this, "Scale the selection",
                                                "Scale:",
                                                inputText,
                                                sizeof(inputText)).Execute() != IDOK)
                                                return;
else {
        if (DesignMode ==0) //&& SelectedNotation!=0
                {
                SelectedNotation->Scale(atof(inputText));
                SelectedNotation->UpdateBound();
                DrawDoc->UpdateNotation(*SelectedNotation,
                        DrawDoc->GetNotations()->Find(*SelectedNotation));
                }
        else  //DesignMode ==1
                {
                Selected->Scale(atof(inputText));
                Selected->UpdateBound();
                DrawDoc->ModifyBasicShape(*Selected,
                        DrawDoc->GetBasicShapes()->Find(*Selected));
                }
 }
}

void
TDrawView::CmScalex()
{
 char inputText[20];
 wsprintf(inputText, "%s", "1");
 if (TInputDialog(this, "Scale X of the selection",
                                                "X scale:",
                                                inputText,
                                                sizeof(inputText)).Execute() != IDOK)
                                                return;
 else {
        if (DesignMode ==0) //&& SelectedNotation!=0
                {
                SelectedNotation->ScaleX(atof(inputText));
                SelectedNotation->UpdateBound();
                DrawDoc->UpdateNotation(*SelectedNotation,
                        DrawDoc->GetNotations()->Find(*SelectedNotation));
                }
        else  //DesignMode ==1
                {
                Selected->ScaleX(atof(inputText));
                Selected->UpdateBound();
                DrawDoc->ModifyBasicShape(*Selected,
                        DrawDoc->GetBasicShapes()->Find(*Selected));
                }
```

```
 }
}

void
TDrawView::CmScaley()
{
 char inputText[20];
 wsprintf(inputText, "%s", "1");
 if (TInputDialog(this, "Scale Y of the selection",
                                                "Y scale:",
                                                inputText,
                                                sizeof(inputText)).Execute() != IDOK)
                                                return;
 else {
        if (DesignMode ==0) //&& SelectedNotation!=0
                {
                SelectedNotation->ScaleY(atof(inputText));
                SelectedNotation->UpdateBound();
                DrawDoc->UpdateNotation(*SelectedNotation,
                        DrawDoc->GetNotations()->Find(*SelectedNotation));
                }

        else  //DesignMode ==1
                {
                Selected->ScaleY(atof(inputText));
                Selected->UpdateBound();
                DrawDoc->ModifyBasicShape(*Selected,
                        DrawDoc->GetBasicShapes()->Find(*Selected));
                }
 }
}
```

GLOSSARY

CASE	Computer-aided Software Engineering.
GUI	Graphical User Interface.
HCI	Human-Computer Interaction.
HTA	Hierarchical Task Analysis — a method of task decomposition.
KAT	Knowledge Analysis of Task — a method of knowledge based task analysis.
Lower-CASE Tools	These tools that automate or support the "lower" or back-end phases of the life cycle; namely, detailed systems design, systems implementation, and systems support.
MIGOCE	Methodology-Independent Graphical OO CASE Environment.
OMT	The Rumbaugh methodology — Object Modeling Technique.
OO	Object-Oriented.
OOA	Object-Oriented Analysis.
OOAD	Object-Oriented Analysis and Design.
OOD	Object-Oriented Aesign.
OOMs	Object-Oriented Methodologies.
OONW	OO Nation Workshop.
OOP	Object-Oriented Programming.
OOSD	Object Oriented Software Development.
TA	Task Analysis.
TAKD	Task Analysis for Knowledge Description —a method of knowledge based task analysis.
TKS	Task Knowledge Structures — a method of knowledge based task analysis.
TM	Task Modeling.
Upper-CASE tools	These tools that automate or support the "upper" or front-end phases of the systems development life cycle; namely, systems planning, systems analysis, and general systems design.
UOODT	Universal OO Diagraming Tool.
WIMP environment	Window, Icon, Menus, Pointer environment.